# A Memristor-Based Optimization Framework for AI Applications

Sijia Liu, *Member, IEEE,* Yanzhi Wang, *Member, IEEE* Makan Fardad, *Member, IEEE,* and Pramod K. Varshney, *Fellow, IEEE*

**Abstract**

Memristors have recently received significant attention as ubiquitous device-level components for building a novel generation of computing systems. These devices have many promising features, such as non-volatility, low power consumption, high density, and excellent scalability. The ability to control and modify biasing voltages at the two terminals of memristors make them promising candidates to perform matrix-vector multiplications and solve systems of linear equations. In this article, we discuss how networks of memristors arranged in crossbar arrays can be used for efficiently solving optimization and machine learning problems. We introduce a new memristor-based optimization framework that combines the computational merit of memristor crossbars with the advantages of an operator splitting method, alternating direction method of multipliers (ADMM). Here, ADMM helps in splitting a complex optimization problem into subproblems that involve the solution of systems of linear equations. The capability of this framework is shown by applying it to linear programming, quadratic programming, and sparse optimization. In addition to ADMM, implementation of a customized power iteration (PI) method for eigenvalue/eigenvector computation using memristor crossbars is discussed. The memristor-based PI method can further be applied to principal component analysis (PCA). The use of memristor crossbars yields a significant speed-up in computation, and thus, we believe, has the potential to advance optimization and machine learning research in artificial intelligence (AI).

**Keywords**

*Memristor crossbar, mathematical programming, alternating direction method of multipliers (ADMM), principal component analysis (PCA), embedded computation, machine learning*

---

S. Liu is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 48019 USA. E-mail: lsjxjtu@umich.edu. Y. Wang, M. Fardad and P. K. Varshney are with the Department of Electrical Engineering and Computer Science, Syracuse University, E-mail: {makan,ywang393,varshney}@syr.edu.

Submitted version; please do not distribute it.

# I. INTRODUCTION

Memristors, nano-scale devices conceived by Leon Chua in 1971, have been physically realized by scientists from Hewlett-Packard [1], [2]. In contrast with the traditional CMOS technology, memristors can be used as non-volatile memories for building brain-like learning machines with memristive synapses [3]. They offer the ability to construct a dense, continuously programmable, and reasonably accurate cross-point array architecture, which can be used for data-intensive applications [4]. For example, a memristor crossbar array exhibits a unique type of parallelism that can be utilized to perform matrix-vector multiplication and solve systems of linear equations in an astonishing $O(1)$ time complexity [5]–[8]. Discovery and physical realization of memristors has inspired the development of efficient approaches to implement neuromorphic computing systems that can mimic neuro-biological architectures and perform high-performance computing for deep neural networks and optimization algorithms [9].

The similarity between the programmable resistance state of memristors and the variable synaptic strengths of biological synapses facilitates the circuit realization of neural network models [10]. Nowadays, an artificial neural network has become an extremely popular machine learning tool with a wide spectrum of applications, ranging from prediction/classification, computer vision, natural language processing, image processing, to signal processing [11]. Encouraged by its success, many researchers have attempted to design memristor-based computing systems to accelerate neural network training [12]–[22]. In [12], [13], memristor crossbars were used to form an on-chip training circuit for an autoencoder, an artificial neural network with one hidden layer. Training a multi-layer neural network requires the implementation of a back-propagation algorithm [23] for synaptic weight update. Such an implementation using memristor crossbars was discussed in [14]–[18]. In [19], [20], a memristor-based neural network was proposed by using an off-chip training approach where synaptic weights are pre-trained in software. This approach avoided the complexity of mapping the back-propagation algorithm onto memristors but did not fully utilize the computational advantages of memristors. In [21], [22], research efforts were made to overcome hardware restrictions, such as scalability and routing congestion, to design memristor-based large neural networks.

In addition to artificial neural networks, memristor-based computing systems have also been proposed and analyzed for sparse coding, dictionary learning, and compressive sensing [24]–[30]. These applications share a similar sparse learning framework, where a sparse solution is sought to minimize a certain cost function. In [24], a sparse coding algorithm was mapped to memristor crossbars. In [25]–[29], memristors were used to achieve on-chip acceleration of dictionary learning algorithms. However, the algorithms

required the memristor network to be programmed multiple times due to the gradient update step which resulted in computation errors caused by device variations. In [27], redundant memristors were employed to suppress these device variations. Besides sparse learning, memristor crossbars have also been considered for implementing and training a probabilistic graphical model [31] and image learning [32], [33].

Although memristor-inspired AI applications are different, the common underlying theme is the design of a mathematical programming solver for an optimization problem specified by a machine learning or data processing task. Examples include linear programming for portfolio optimization [34], nonlinear programming for regression/classification [35], and regularized optimization for sparse learning [36]. Therefore, a general question to be answered in this context is: *how can one design a general memristor-based computation framework to accelerate the optimization procedure?*

The interior-point algorithm is one of the most commonly-used optimization approaches implemented in software. It begins at an interior point within the feasible region, then applies a projective transformation so that the current interior point is the center of projective space, and then moves in the direction of the steepest descent [37]. However, the inherent hardware limitations prevent the direct mapping from the interior-point algorithm to memristor crossbars. First, a memristor crossbar only allows square matrices with nonnegative entries during computation, since the memristance is always nonnegative. Second, the memristor crossbar suffers from hardware variations, which degrade the reading/writing accuracy of memristor crossbars. To circumvent the first difficulty, additional memristors were used to represent negative elements of a square matrix [8], [38], [39]. In particular, the work [8] presented a memristor-based linear solver using the interior-point algorithm, which, however, requires programming of the resistance state of memristors at every iteration. Consequently, the linear solver in [8] is prone to suffer from hardware variations. Therefore, to successfully design memristor-based optimization solvers, it is crucial to co-optimize algorithm, device and architecture so that the advantages of memristors can be fully utilized and the design complexity and the non-ideal hardware effects can be minimized. Our previous work [7], [30] showed that the alternating direction method of multipliers (ADMM) algorithm can take advantage of the hardware implementation of memristor crossbars. With the aid of ADMM, one can decompose a complex problem into subproblems that require matrix-vector multiplications and solution of systems of linear equations. The decomposed operations are more easily mapped onto memristor crossbars. In this paper, we discuss how to use the idea of ADMM to design memristor-based optimization solvers for solving linear programs, quadratic programs and sparse optimization problems. Different from the interior-point algorithm, memristor crossbars are programmed only once, namely, independent of ADMM iterations. Therefore, the proposed memristor-based optimization framework is of highly resilient to

random noise and process variations.

In addition to designing a memristor-based optimization solver, we also discuss the application of memristors to solve eigenvalue problems. It is worth mentioning that computation of eigenvalues/eigenvectors is the key step in many AI applications and optimization problems, e.g., low-dimensional manifold learning [40], and semidefinite projection in semidefinite programming [41]. In this paper, we present the generalization of the power iteration (PI) method using memristor crossbars. Conventionally, PI only converges when the dominant eigenvalue is unique. Here, we adopt the Gram-Schmidt procedure [42] to handle convergence issues in the presence of multiple dominant eigenvalues. We anticipate that this paper will inspire proliferation of memristor-based technologies, and fully utilize its extraordinary potential in emerging AI applications.

The rest of the paper is organized as follows. In Section II, we review the memristor technology for solving systems of linear equations. In Section III, we discuss the idea of ADMM for convex optimization. In Section IV, we derive memristor-based solvers for linear and quadratic programming. In Section V, we apply the memristor technology for sparse optimization. In Section VI, we extend PI using memristors for eigenvalue/eigenvector computation. In Section VII, we summarize the topics presented in the paper and and discuss future research directions.

## II. Memristors in Solving Systems of Linear Equations

A memristor has the unique property of recording the profile of excitations on the device. That is, the state (memristance) of a memristor changes only when a certain voltage higher than a threshold is applied at its two terminals. This memristive property makes it an ideal candidate for use as non-volatile memory [43], [44]. Physical memristors can be fabricated in a high density grid, and the resulting memristor crossbar structure is attractive for performing matrix-vector operations due to its high degree of parallelism [19]. We elaborate on the memristor technology in the following.

A typical $N \times N$ memristor crossbar structure is illustrated in Fig. 1, where a memristor is connected between each pair of horizontal word-line (WL) and vertical bit-line (BL). This structure can be implemented with a small footprint, and each memristor can be re-programmed to different resistance states by controlling the voltage of WLs and BLs [5], [45], [46]. Let $\mathbf{V}_{\mathrm{I}}$ denote a vector of input voltages on WLs. We obtain the current at each BL by measuring the voltage across a resistor with conductance $g_s$. If the memristor at the connection between $\mathrm{WL}_i$ and $\mathrm{BL}_j$ has a conductance of $g_{i,j}$, then the output

voltage on the $j$th BL $\mathbf{V}_{\mathrm{O},j}$ is given by [5],

$$\mathbf{V}_{\mathrm{O},j} = \left[ \frac{g_{1,j}}{g_s + \sum_{i=1}^{N} g_{ij}} \quad \cdots \quad \frac{g_{N,j}}{g_s + \sum_{i=1}^{N} g_{ij}} \right] \mathbf{V}_{\mathrm{I}},$$

or equivalently,

$$\mathbf{V}_{\mathrm{O}} = \mathbf{C}\mathbf{V}_{\mathrm{I}}, \ \mathbf{C} = \mathrm{diag}\left( \left\{ \frac{1}{g_s + \sum_{i=1}^{N} g_{ij}} \right\}_{j=1}^{N} \right) \mathbf{G}^T, \tag{1}$$

where $\mathrm{diag}(\{x_i\}_{i=1}^N)$ denotes a diagonal matrix with diagonal entries $x_1, x_2, \ldots, x_N$, and $\mathbf{G}$ is the conductance matrix of memristors whose $(i,j)$th entry is given by $g_{i,j}$. In (1), the desired coefficient matrix $\mathbf{C}$ can be realized by adjusting memristor conductivities $\{g_{i,j}\}$ and the bias resistor's conductance $g_s$. In order to avoid out-of-range coefficients in the memristor crossbar, a pre-scaling step is required to scale all matrix coefficients to fall into the memristors' conductance range. In this manner, one can perform matrix-vector multiplications through a pre-configured (or programmed) memristor crossbar.

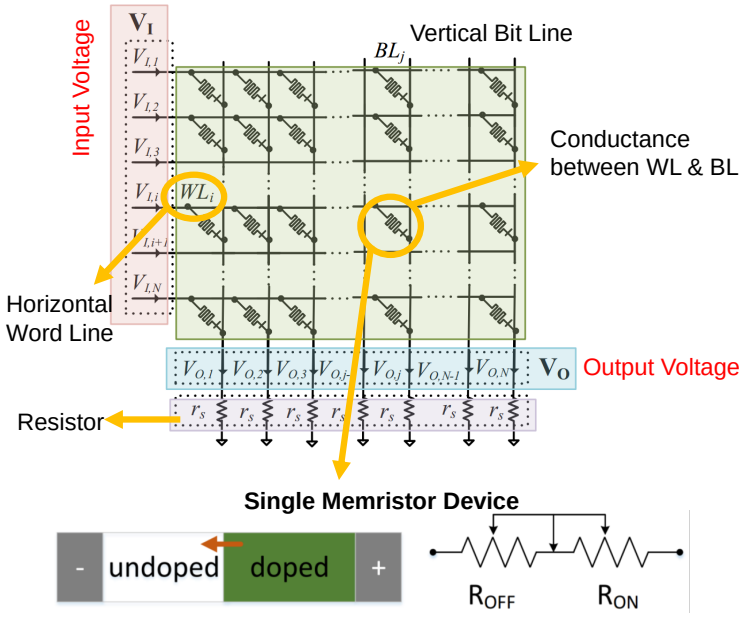



**Fig. 1:** Illustration of a memristor crossbar

Reversing the above operation, the memristor crossbar structure can also solve a system of linear equations [6]. Here, we assume that the solution exists and is unique. It is clear from (1) that if we apply $\mathbf{V}_{\mathrm{O}}$ on BLs, then $\mathbf{V}_{\mathrm{I}}$ on WLs becomes the solution of the linear system described by a pre-configured memristor network. An appealing property of the memristor-based linear equation solver is

its high computational efficiency, an astonishing $O(1)$ time complexity [15], since the matrix-vector multiplication (or its reverse operation) is performed in a parallel fashion. While this structure provides significant computational advantages, there are challenges introduced by the hardware restrictions of memristors. First, in the linear system (1), only a non-negative coefficient matrix can be mapped onto memristors. Second, a memristor crossbar suffers from hardware variations that introduce computational errors while performing matrix-vector operations. In what follows, we elaborate on the aforementioned challenges and present some possible solutions.

Since only non-negative coefficients can be mapped to memristors, it is essential to design a general mechanism that can deal with negative coefficients. In previous work [5], [17] it has been suggested that negative numbers in a memristor system can be represented by using two identical crossbars. Specifically, the weight matrix $\mathbf{C}$ is split into two parts $\mathbf{C}_1$ and $\mathbf{C}_2$ so that $\mathbf{C} = \mathbf{C}_1 - \mathbf{C}_2$, where $\mathbf{C}_1 = (\mathbf{C})_+$, $\mathbf{C}_2 = (-\mathbf{C})_+$, and $(x)_+ = \max\{0, x\}$ is a positive operator taken elementwise for a matrix argument. Given nonnegative matrices $\mathbf{C}_1$ and $\mathbf{C}_2$, the matrix-vector multiplication (1) can be obtained through the subtraction $\mathbf{C}_1\mathbf{V}_\mathrm{I} - \mathbf{C}_2\mathbf{V}_\mathrm{I}$ [38], [39]. Instead of using two identical crossbars, we can eliminate the negative numbers by introducing auxiliary variables in the linear system (1),

$$\mathbf{V}_\mathrm{O} = \mathbf{C}\mathbf{V}_\mathrm{I} \implies \begin{bmatrix} (\mathbf{C})_+ & \mathbf{B} \\ \mathbf{D} & \mathbf{I}_{\bar{N}} \end{bmatrix} \begin{bmatrix} \mathbf{V}_\mathrm{I} \\ \bar{\mathbf{V}}_\mathrm{I} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_\mathrm{O} \\ \mathbf{0}_{\bar{N}} \end{bmatrix}, \tag{2}$$

where $\bar{\mathbf{V}}_\mathrm{I} \in \mathbb{R}^{\bar{N}}$ is a newly introduced variable, $\bar{N}$ is the number of nonzero columns of $(-\mathbf{C})_+$ (namely, the number of columns of $\mathbf{C}$ that contain negative elements), $\mathbf{B} \in \mathbb{R}^{N\times\bar{N}}$ is formed by nonzero columns of $(-\mathbf{C})_+$, $\mathbf{D} \in \mathbb{R}^{\bar{N}\times N}$ is a submatrix of $\mathbf{I}_N$ whose row indices are given by column indices of nonzero columns of $(-\mathbf{C})_+$, and $\mathbf{0}_{\bar{N}}$ is a zero vector of size $\bar{N}$. In Table I, we show that (1) can be recovered from (2) by eliminating $\bar{\mathbf{V}}_\mathrm{I}$. We stress that compared to the use of an identical memristor crossbar (leading to $2N \times 2N$ memristor network), the proposed scheme (2) requires fewer memristors, resulting in the memristor network of size $(N + \bar{N}) \times (N + \bar{N})$, where $\bar{N} \leq N$.

**Table I:** Illustration of linear mapping (2).

- $\mathbf{C} = (\mathbf{C})_+ - (-\mathbf{C})_+$, which yields $\mathbf{V}_\mathrm{O} = (\mathbf{C})_+\mathbf{V}_\mathrm{I} - (-\mathbf{C})_+\mathbf{V}_\mathrm{I}$.
- Let $\{i_1, i_2, \ldots, i_{\bar{N}}\}$ denote the indices of nonzero columns of $(-\mathbf{C})_+$. Definitions of $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_{\bar{N}}]$ and $\mathbf{D} = [\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_N}]^T$ in (2) give

$$(-\mathbf{C})_+ = \sum_{j=1}^{\bar{N}} \mathbf{b}_j \mathbf{e}_{i_j}^T = \mathbf{B}\mathbf{D}.$$

- $\mathbf{V}_{\mathrm{O}} = (\mathbf{C})_{+}\mathbf{V}_{\mathrm{I}} - \mathbf{B}\mathbf{D}\mathbf{V}_{\mathrm{I}} \Longrightarrow \mathbf{V}_{\mathrm{O}} = (\mathbf{C})_{+}\mathbf{V}_{\mathrm{I}} + \mathbf{B}\bar{\mathbf{V}}_{\mathrm{I}}$ with $\bar{\mathbf{V}}_{\mathrm{I}} = -\mathbf{D}\mathbf{V}_{\mathrm{I}}$, which yields (2).

Moreover, parameters of a memristor crossbar may differ from the target values due to variability in the fabrication process, environmental noise, and signal fluctuations from power supplies and neighboring wires [47]. Several methods have been proposed to mitigate these impairments in hardware [19], [27], [28], [30], [48]. In [19], [48], feedback programming techniques were used to improve the writing accuracy in memristor crossbars. In [28], a read peripheral circuitry that functions as an analog-to-digital converter was used to eliminate analog distortions. In [27], multiple memristors were introduced to update a single weight. This method statistically averages out the conductance variations in both time and space. However, it requires more memristors and higher communication overhead. In addition to circuit-level techniques [19], [27], [28], [48], we will show that the non-ideal effects caused by hardware variations can also be mitigated by optimizing the algorithm prior to mapping to memristor crossbars.

## III. Convex Optimization and ADMM

Although memristor-based AI applications are different such as sparse learning and dictionary learning [24]–[30], the principle of designing memristor-based computation accelerators is the same, namely, recognizing the optimization problem underlying the learning task and mapping the corresponding optimization algorithm onto a memristor network. In what follows, we provide some background on mathematical programming and focus on a solver called alternating direction method of multipliers (ADMM).

### *A. Preliminaries on convex optimization and ADMM*

In general, an optimization problem can be cast as

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}), \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{X}, \end{aligned} \tag{3}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the optimization variable, $f(\cdot)$ denotes the cost function to be minimized, and $\mathcal{X}$ denotes a constraint set. In this paper, we focus on the convex version of problem (3), where $f(\cdot)$ is a convex function and $\mathcal{X}$ is a convex set [37]. In convex programming, a local minimum given by a stationary point of (3) implies the global optimality. Convex optimization forms the foundation of many AI applications [35].

There exist many algorithms to solve convex optimization problems, such as gradient-type first-order methods [49], and primal-dual interior-point (second-order) methods [37]. Compared to the conventional optimization methods, ADMM has drawn great attention in the last ten years [41], [50]. The main advantage of ADMM is that it allows us to split the optimization problem into subproblems, each of which can be solved efficiently and, in some cases, analytically.

A standard problem that is suitable for the application of ADMM is given by

$$\begin{aligned} \underset{\mathbf{x},\mathbf{y}}{\text{minimize}} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} + \mathbf{c} = \mathbf{0}, \end{aligned} \tag{4}$$

where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$ are optimization variables, $f(\cdot)$ and $g(\cdot)$ are convex functions, and $\mathbf{A} \in \mathbb{R}^{l\times n}$, $\mathbf{B} \in \mathbb{R}^{l\times m}$, and $\mathbf{c} \in \mathbb{R}^l$ are appropriate coefficients associated with a system of $l$ linear equality constraints. Problem (4) reduces to problem (3) when $\mathbf{A} = \mathbf{I}_n$, $\mathbf{B} = -\mathbf{I}_m$, $\mathbf{c} = \mathbf{0}_l$, and $g(\cdot)$ is an indicator function on the convex set $\mathcal{X}$, namely,

$$g(\mathbf{y}) = \begin{cases} 0 & \text{if } \mathbf{y} \in \mathcal{X} \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

Here $\mathbf{I}_n$ denotes the $n \times n$ identity matrix, and $\mathbf{0}_n$ is the $n \times 1$ vector of all zeros. In what follows, while referring to identity matrices and vectors of all ones (or zeros), their dimensions are omitted for simplicity but can be inferred from the context. ADMM is an iterative algorithm, and its $k$th iteration is given by [41]

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}} \left\{ f(\mathbf{x}) + (\boldsymbol{\mu}^k)^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}^k + \mathbf{c}) + \frac{\rho}{2}\left\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}^k + \mathbf{c}\right\|_2^2 \right\} \tag{6}$$

$$\mathbf{y}^{k+1} = \arg\min_{\mathbf{y}} \left\{ g(\mathbf{y}) + (\boldsymbol{\mu}^k)^T(\mathbf{A}\mathbf{x}^{k+1} + \mathbf{B}\mathbf{y} + \mathbf{c}) + \frac{\rho}{2}\left\|\mathbf{A}\mathbf{x}^{k+1} + \mathbf{B}\mathbf{y} + \mathbf{c}\right\|_2^2 \right\} \tag{7}$$

$$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \rho(\mathbf{A}\mathbf{x}^{k+1} + \mathbf{B}\mathbf{y}^{k+1} + \mathbf{c}), \tag{8}$$

where $\boldsymbol{\mu}$ is the Lagrangian multiplier (also known as the dual variable), $\rho$ is a positive weight to penalize the augmented term associated with the equality constraint of (4), and $\|\cdot\|_2$ denotes the $\ell_2$ norm. The ADMM algorithm terminates when an $\epsilon$-accuracy is achieved, namely, $\|\mathbf{x}^k - \mathbf{y}^k\|_2 \leq \epsilon$, and $\|\mathbf{x}^k - \mathbf{x}^{k-1}\|_2 \leq \epsilon$. ADMM has a linear convergence rate $O(1/K)$ for general convex optimization problems [51], where $K$ is the number of iterations. In other words, given the stopping tolerance $\epsilon$, ADMM requires $O(1/\epsilon)$ iterations to converge. We remark that ADMM has a faster convergence rate than the gradient decent algorithm, which has the convergence rate of $O(1/\sqrt{K})$. In the next section, we will show that ADMM provides a suitable framework for mapping to a memristor network.

## IV. Memristor-Based Linear and Quadratic Optimization Solvers

In this section, we employ memristor crossbars to solve linear and quadratic programs. Linear programs (LPs) and quadratic programs (QPs) are the most common optimization problems that are encountered in many applications such as resource scheduling, intelligent transportation, portfolio optimization, smart grid and signal processing [52]–[55]. The interior-point algorithm is a standard method to solve LPs as well as QPs [37], with $O(n^3 \sim n^{3.5})$ time complexity [56], where $n$ is the number of optimization variables. The conventional interior-point algorithm running on CPUs/GPUs has low degree of parallelism. By contrast, as we next demonstrate, ADMM breaks up optimization problems into subproblems involving the solution of linear equations, which lend themselves to the use of memristors for efficient computation.

### A. Linear optimization with memristors

The standard form of LP is expressed as follows,

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{d}^T\mathbf{x} \\ \text{subject to} & \mathbf{G}\mathbf{x} = \mathbf{h}, \quad \mathbf{x} \geq \mathbf{0}, \end{array} \tag{9}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the optimization variable, $\mathbf{d} \in \mathbb{R}^n$, $\mathbf{G} \in \mathbb{R}^{l\times n}$ and $\mathbf{h} \in \mathbb{R}^l$ are given parameters, and the last inequality constraint represents the elementwise inequalities $x_i \geq 0$ for $i = 1, 2, \ldots, n$. In this paper, we assume that $\mathbf{G}$ is of full row rank.

We begin by reformulating problem (9) as the canonical form (4) that is amenable to the use of ADMM algorithm,

$$\begin{array}{ll} \underset{\mathbf{x},\mathbf{y}}{\text{minimize}} & \mathbf{d}^T\mathbf{x} + p(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & \mathbf{x} = \mathbf{y}, \end{array} \tag{10}$$

where $\mathbf{y} \in \mathbb{R}^n$ is a newly introduced optimization variable, and similar to (5), $p$ and $g$ are indicator functions, with respect to constraint sets $\{\mathbf{x} \,|\, \mathbf{G}\mathbf{x} = \mathbf{h}\}$ and $\{\mathbf{y} \,|\, \mathbf{y} \geq \mathbf{0}\}$, respectively. If we set $f(\mathbf{x}) = \mathbf{d}^T\mathbf{x} + p(\mathbf{x})$, $\mathbf{A} = \mathbf{I}$, $\mathbf{B} = -\mathbf{I}$ and $\mathbf{c} = \mathbf{0}$, then problem (10) is the same as problem (4).

Based on (10), the ADMM steps (6)-(8) become

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}} \left\{ \mathbf{d}^T\mathbf{x} + p(\mathbf{x}) + (\boldsymbol{\mu}^k)^T(\mathbf{x} - \mathbf{y}^k) + \frac{\rho}{2}\left\|\mathbf{x} - \mathbf{y}^k\right\|_2^2 \right\} \tag{11}$$

$$\mathbf{y}^{k+1} = \arg\min_{\mathbf{y}} \left\{ g(\mathbf{y}) + (\boldsymbol{\mu}^k)^T(\mathbf{x}^{k+1} - \mathbf{y}) + \frac{\rho}{2}\left\|\mathbf{x}^{k+1} - \mathbf{y}\right\|_2^2 \right\} \tag{12}$$

$$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \rho(\mathbf{x}^{k+1} - \mathbf{y}^{k+1}). \tag{13}$$

As we show next, the primary advantage of employing ADMM here is that problem (11) can be readily solved using memristor crossbars, and problem (12) yields a closed-form solution that only involves elementary vector operations.

Problem (11) is equivalent to

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \frac{\rho}{2}\|\mathbf{x}-\boldsymbol{\alpha}\|_2^2 \\ \text{subject to} \quad & \mathbf{G}\mathbf{x}=\mathbf{h}, \end{aligned} \tag{14}$$

where $\boldsymbol{\alpha} := \mathbf{y}^k - (1/\rho)(\boldsymbol{\mu}^k + \mathbf{d})$. The solution of problem (14) is determined by its Karush-Kuhn-Tucker (KKT) conditions [37], $\rho(\mathbf{x}-\boldsymbol{\alpha}) + \mathbf{G}^T\boldsymbol{\lambda} = \mathbf{0}$, and $\mathbf{G}\mathbf{x} = \mathbf{h}$, where $\boldsymbol{\lambda} \in \mathbb{R}^l$ is the Lagrangian multiplier. The KKT conditions imply a system of linear equations

$$\mathbf{C}\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\rho\boldsymbol{\alpha}\\ \mathbf{h}\end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix}\rho\mathbf{I} & \mathbf{G}^T\\ \mathbf{G} & \mathbf{0}\end{bmatrix}. \tag{15}$$

Based on (2), the linear system (15) can be efficiently mapped to memristor crossbars by configuring their memristance values according to the matrix $\mathbf{C}$.

On the other hand, problem (12) is equivalent to

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \frac{\rho}{2}\|\mathbf{y}-\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \mathbf{y}\geq\mathbf{0}, \end{aligned} \tag{16}$$

where $\boldsymbol{\beta} := \mathbf{x}^{k+1} + (1/\rho)\boldsymbol{\mu}^k$. The solution of problem (16) is determined by the projection of $\boldsymbol{\beta}$ onto the nonnegative orthant,

$$\mathbf{y}^{k+1} = (\boldsymbol{\beta})_+. \tag{17}$$

Note that the positive part operator $(\cdot)_+$ in (17) can be readily implemented using elementary logical or digital operations.

We summarize the memristor-based LP solver in Fig. 2. Although LP is a relatively simple optimization problem, the LP solver illustrates our general idea and paves the way for numerous memristor-based applications in optimization problems. Our solution framework offers two major advantages. First, in the linear system (15), the coefficient matrix $\mathbf{C}$ is independent of the ADMM iteration so that memristors need to be configured only once. This feature makes it more attractive than gradient-type and interior-point algorithms, where memristors have to be reconfigured at each iteration [27]. Second, ADMM splits a complex problem into subproblems, each of which is easier to solve and implement in hardware.

We remark that a memristor crossbar is size-limited (e.g., $1024 \times 1024$) due to manufacturing and performance considerations [10]. To improve its scalability, analog network-on-chip (NoC) communication structures can be adopted to effectively coordinate multiple memristor crossbars for supporting

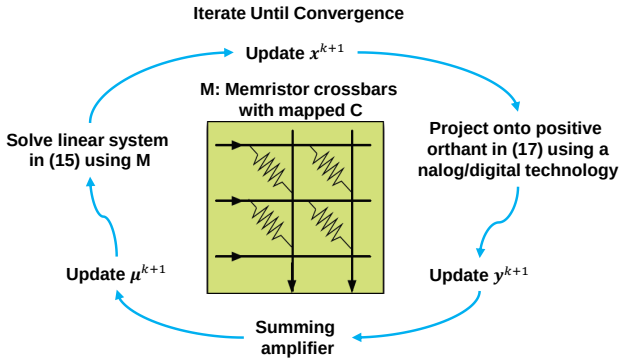
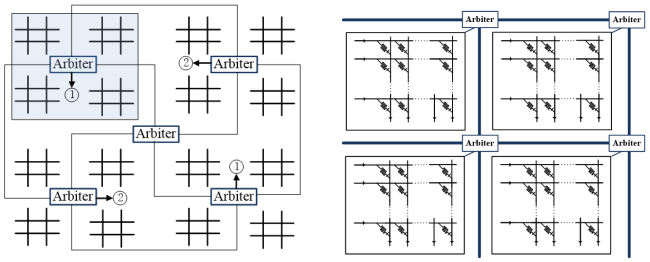



**Fig. 2:** Memristor crossbar based solution framework in linear programming.

large-scale applications [10], [20], [57], [58]. Data transfers within the NoC structure maintain analog form and are managed by the NoC arbiters. Two potential analog NoC structures for multiple memristor crossbars are presented in Fig. 3. Fig. 3(a) shows a hierarchical structure of memristor crossbars [10], where four crossbar arrays are grouped and controlled by one arbiter, and those groups again form a higher-level group controlled by a higher-level arbiter. Fig. 3(b) shows a mesh network-based structure of memristor crossbars, which resembles a mesh network-based NoC structure in multi-core systems [58].




(a) (b)

**Fig. 3:** Examples of NoC structures coordinating multiple memristor crossbars. (a) Four crossbar arrays are grouped and controlled by one arbiter. The resulting higher-level group is controlled by a higher-level arbiter. (b) Mesh network-based structure of memristor crossbars.

### B. Quadratic optimization with memristors

QP is an optimization problem whose objective and constraint functions involve quadratic and/or linear terms. There exist many variants of QP, such as a second-order cone program (SOCP) and a quadratically constrained quadratic program (QCQP) [37]. In this section, we focus on the design of a memristor-based solver for SOCP, since it is possible to convert a QCQP into a SOCP, e.g., homogeneous QCQP that excludes linear terms [7].

SOCP is a convex program for minimizing a linear cost function subject to linear and second-order cone constraints,

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{d}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{G}\mathbf{x} = \mathbf{h}, \quad \|\mathbf{x}_{1:(n-1)}\|_2 \leq x_n, \end{aligned} \tag{18}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the optimization variable, $\mathbf{G}$ and $\mathbf{h}$ are given parameters, $\mathbf{x}_{1:(n-1)}$ denotes a vector that consists of the first $n-1$ entries of $\mathbf{x}$, and $x_n$ is the $n$th entry of $\mathbf{x}$. The last constraint in (18) is known as the second-order cone constraint.

Similar to (10), we can rewrite problem (18) in the canonical form (4) that is amenable to the ADMM algorithm

$$\begin{aligned} \underset{\mathbf{x},\mathbf{y}}{\text{minimize}} \quad & \mathbf{d}^T\mathbf{x} + p(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & \mathbf{x} = \mathbf{y}, \end{aligned} \tag{19}$$

where $\mathbf{y} \in \mathbb{R}^n$ is the newly introduced optimization variable, and $p$ and $g$ are indicator functions with respect to constraint sets $\{\mathbf{x} \,|\, \mathbf{G}\mathbf{x} = \mathbf{h}\}$ and $\{\mathbf{y} \,|\, \|\mathbf{y}_{1:(n-1)}\|_2 \leq y_n\}$, respectively.

Following (6)-(8), the ADMM algorithm for solving problem (19) includes subproblem (14) with respect to the variable $\mathbf{x}$, step (13) for updating dual variables $\boldsymbol{\mu}$, and a specific $\mathbf{y}$-minimization problem (7),

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \frac{\rho}{2}\|\mathbf{y} - \boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \|\mathbf{y}_{1:(n-1)}\|_2 \leq y_n, \end{aligned} \tag{20}$$

where recall from (16) that $\boldsymbol{\beta} = \mathbf{x}^{k+1} + (1/\rho)\boldsymbol{\mu}^k$. The solution of problem (20) is given by projecting $\boldsymbol{\beta}$ onto a second-order cone [50],

$$\mathbf{y}^{k+1} = \begin{cases} 0 & \|\boldsymbol{\beta}_{1:(n-1)}\|_2 \leq -\beta_n \\ \boldsymbol{\beta} & \|\boldsymbol{\beta}_{1:(n-1)}\|_2 \leq \beta_n \\ \frac{1}{2}\left(1 + \frac{\beta_n}{\|\boldsymbol{\beta}_{1:(n-1)}\|_2}\right)\left[\boldsymbol{\beta}_{1:(n-1)}^T, \|\boldsymbol{\beta}_{1:(n-1)}\|_2\right]^T & \|\boldsymbol{\beta}_{1:(n-1)}\|_2 \geq |\beta_n|. \end{cases} \tag{21}$$

Similar to the memristor-based LP solver, the ADMM step (6) reduces to the solution of a system of linear equations that can be mapped onto memristor crossbars. In the ADMM step (20), we can use

peripheral circuits including analog multipliers and summing amplifiers to evaluate the vector norm in (21) [59], [60]; see schematic illustration in Fig 4.

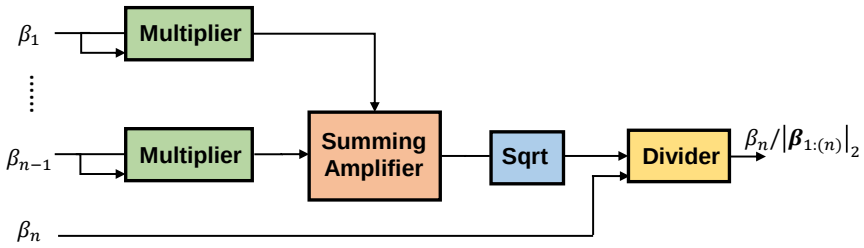



**Fig. 4:** Schematic illustration of the hardware system for calculating $\mathbf{y}^{k+1}$ in (21).

To summarize, one may exploit the alternating structure of ADMM to design memristor-based optimization solvers. The crucial property to enable this is that ADMM helps in extracting parallel operations of matrix/vector multiplication/addition which can be implemented using memristor crossbars and elementary hardware elements.

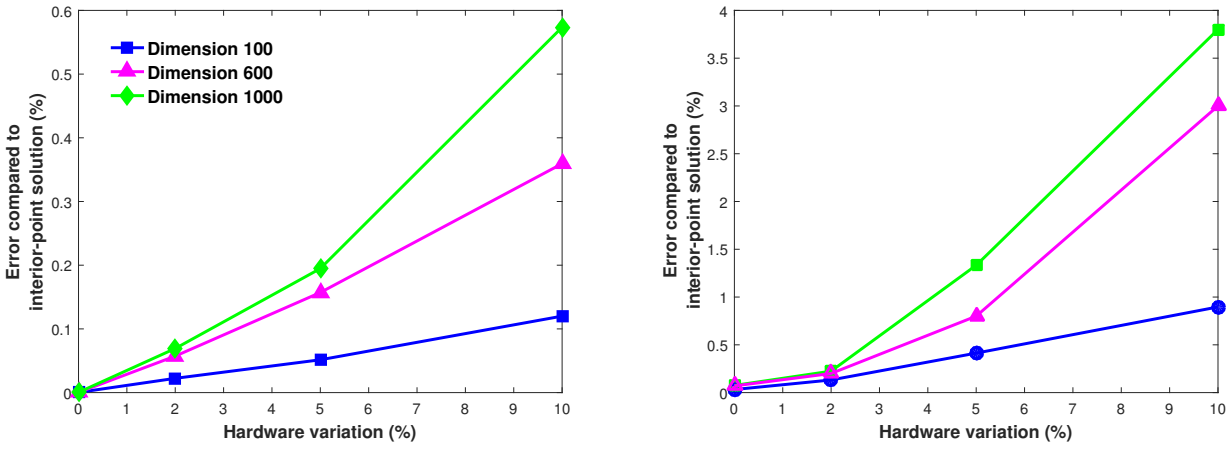
### *C. Performance evaluation*

In what follows, we present empirical results that show the effectiveness of the proposed memristor-based optimization framework to solve LPs and QPs[1]. Since the presence of hardware variations leads to a reduced configuration accuracy on memristor crossbars, the matrix $\mathbf{C}$ in (15) is actually modified to $\tilde{\mathbf{C}} = \mathbf{C} + \mathbf{\Sigma}$, where $\mathbf{\Sigma}$ denotes a random matrix whose elements are i.i.d. zero-mean Gaussian random variables. The quantity $\|\mathbf{\Sigma}\|_F/\|\mathbf{C}\|_F$ then provides the level of hardware variations, where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. In the presence of hardware variations, we compare the solution $\mathbf{x}$ above to the optimal solution $\mathbf{x}^*$ obtained from the off-the-shelf interior-point solver CVX [61], that excludes the effect of hardware variation. We adopt $\|\mathbf{x}-\mathbf{x}^*\|_2/\|\mathbf{x}^*\|_2$ (averaged over 50 random trials) to measure the error between $\mathbf{x}$ and $\mathbf{x}^*$. In ADMM, the augmented parameter and the stopping tolerance are set to be $\rho \in \{0.1, 1, 10, 100\}$ and $\epsilon = 10^{-3}$.

In Fig. 5, we present the difference between the memristor-based solution and the variation-free interior-point solution as a function of the level of hardware variations for problems with dimension

[1]All the codes will be made public once the paper is accepted.

$n \in \{100, 600, 1000\}$. When the hardware variation is excluded, the memristor-based solution yields the same accuracy as the interior-point solution. As the problem size or the hardware variation increases, the difference from the interior-point solution increases. However, the induced error is always below $5\%$. In Fig. 6, we further show the convergence of the memristor-based solution framework as a function of the choice of the ADMM parameter $\rho$. For each value of $\rho$, $50$ random trials were performed, each of which involved $10\%$ hardware variation. We find that the convergence of the memristor-based approach (to achieve $\epsilon$-accuracy solution) is robust to hardware variations and the choice of ADMM parameter $\rho$. Compared to LP, QP requires more iterations to converge due to its higher complexity. Moreover, a moderate choice of $\rho$, e.g., $\rho = 1$ in this example, improves the convergence of the memristor-based approach.

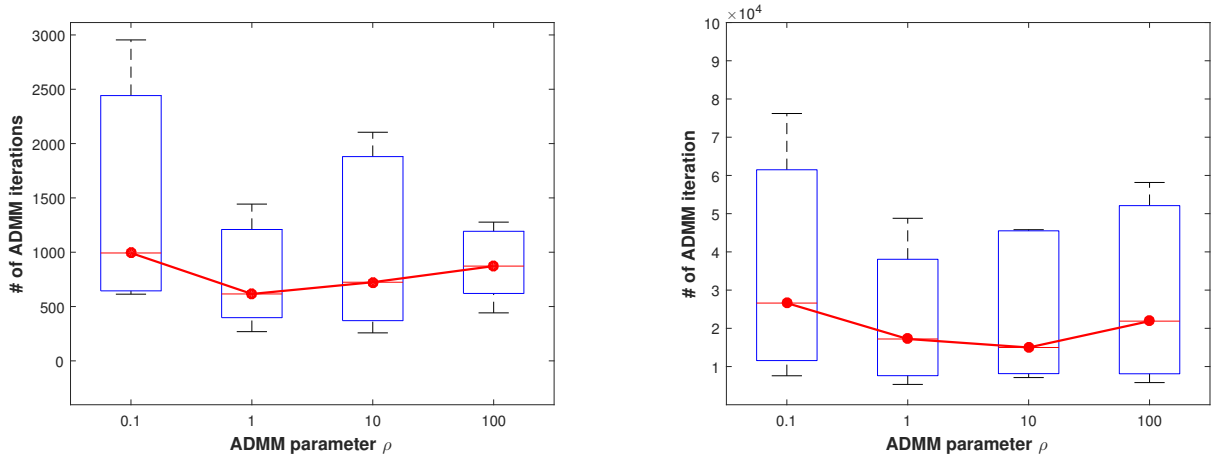



(a) (b)

**Fig. 5:** Solution accuracy versus level of hardware variations for different problem sizes $n \in \{100, 600, 1000\}$. (a) Memristor-based LP solver. (b) Memristor-based QP solver with the same legend as (a).

## V. Memristor-Based Sparse Learning

Sparse learning is concerned with the problem of finding intrinsic sparse patterns of variables to be optimized. This problem is central to machine learning and big-data processing. Examples of applications include model selection in regression/classification, dictionary learning, matrix completion in recommendation systems, image restoration, graphical modelling, natural language processing, resource management in sensor networks, and compressive sensing [36], [62]–[64]. It is often the case that we can




(a) (b)

**Fig. 6:** Number of ADMM iterations to obtain an $\epsilon$-accuracy solution for different values of ADMM parameter $\rho$ under $10\%$ hardware variation. (a) Memristor-based LP solver; (b) Memristor-based QP solver.

cast sparse learning as an optimization problem that involves sparsity-inducing regularizers, such as the $\ell_1$ norm, mixed $\ell_1$ and $\ell_2$ norms, and the nuclear norm [36]. In this section, we focus on the problem of robust compressive sensing (CS), which recovers sparse signals from noisy observations [65]. We remark that CS yields a problem formulation similar to LASSO [66], sparse coding [24] and sensor selection problems [67]. Previous research efforts [65], [68]–[73] focused on software-based approaches for sparse signal recovery, with the support of CPUs/GPUs. Here we discuss approaches to employ memristor crossbars to design CS solvers.

### *A. Preliminaries on CS*

Let $\mathbf{z}_* \in \mathbb{R}^p$ be a sparse or compressible vector, e.g., a digital signal or image, to be recovered. We have access to measurements $\mathbf{h} = \mathbf{H}\mathbf{z}_* + \mathbf{v}$, where $q \ll p$, $\mathbf{H} \in \mathbb{R}^{q \times p}$ is a given measurement matrix, such as a random Gaussian matrix, and $\mathbf{v} \in \mathbb{R}^q$ is a stochastic or deterministic error with bounded energy $\|\mathbf{v}\|_2 \leq \xi$.

The main goal of CS is to stably recover the unknown sparse signal $\mathbf{z}_*$ from noisy measurements $\mathbf{h}$. It has been shown in [74] that stable recovery can be achieved in polynomial time by solving the convex

optimization problem for robust CS

$$\begin{aligned} \underset{\mathbf{z}}{\text{minimize}} \quad & \|\mathbf{z}\|_1 \\ \text{subject to} \quad & \|\mathbf{H}\mathbf{z}-\mathbf{h}\|_2 \leq \xi, \end{aligned} \tag{22}$$

where $\mathbf{z} \in \mathbb{R}^n$ is the optimization variable, and $\|\cdot\|_1$ denotes the $\ell_1$ norm of a vector. In problem (22), the $\ell_1$ norm is introduced to promote the sparsity of $\mathbf{z}$ [69]. Note that problem (22) can also be formulated in the form of LASSO or sparse coding [24], [66]

$$\underset{\mathbf{z}}{\text{minimize}} \quad \|\mathbf{H}\mathbf{z}-\mathbf{h}\|_2^2 + \gamma\|\mathbf{z}\|_1,$$

where $\gamma$ is a regularization parameter that governs the tradeoff between the least square error and the sparsity of $\mathbf{z}$. In what follows, we focus on the problem formulation in (22).

### *B. Memristor-based accelerator for solving CS problems*

Similar to memristor-based linear and quadratic optimization solvers, the key step to successfully applying memristor crossbar arrays to CS problems is to extract subproblems, with the aid of ADMM, that solve systems of linear equations. By introducing three new optimization variables $\mathbf{s} \in \mathbb{R}^q$, $\mathbf{w} \in \mathbb{R}^p$ and $\mathbf{u} \in \mathbb{R}^q$, problem (22) can be reformulated in a way that lends itself to the application of ADMM,

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{z},\mathbf{s}) + \|\mathbf{w}\|_1 + p(\mathbf{u}) \\ \text{subject to} \quad & \mathbf{z}-\mathbf{w}=\mathbf{0}, \quad \mathbf{s}-\mathbf{u}=\mathbf{0}, \end{aligned} \tag{23}$$

where $\mathbf{z}$, $\mathbf{s}$, $\mathbf{w}$ and $\mathbf{u}$ are optimization variables, and $f$ and $p$ are indicator functions corresponding to the constraints of problem (22), namely,

$$f(\mathbf{z},\mathbf{s}) = \begin{cases} 0 & \mathbf{H}\mathbf{z}-\mathbf{s}=\mathbf{h} \\ \infty & \text{otherwise}, \end{cases} \tag{24}$$

and

$$p(\mathbf{u}) = \begin{cases} 0 & \|\mathbf{u}\|_2 \leq \xi \\ \infty & \text{otherwise}. \end{cases} \tag{25}$$

In (23), the introduction of new variables $\mathbf{s}$, $\mathbf{w}$ and $\mathbf{u}$ together with the indicator functions (24) – (25) allows us to split the original constrained problem into subproblems for solving systems of linear equations, and elementary proximal operations related to the $\ell_1$ norm and the Euclidean ball constraint [50].

We recall from the standard form of ADMM given by (4) that if we set $\mathbf{x} = [\mathbf{z}^T, \mathbf{s}^T]^T$, $\mathbf{y} = [\mathbf{w}^T, \mathbf{u}^T]$, $g(\cdot) = \|\cdot\|_1 + g'(\cdot)$, $\mathbf{A} = \mathbf{I}$, $\mathbf{B} = -\mathbf{I}$ and $\mathbf{c} = \mathbf{0}$, then problem (4) reduces to the CS problem (23). As a result, the ADMM step (6) with respect to $\mathbf{z}$ and $\mathbf{s}$ can be written as

$$\begin{aligned} \underset{\mathbf{z},\mathbf{s}}{\text{minimize}} \quad & \frac{\rho}{2}\|\mathbf{z} - \boldsymbol{\alpha}_1\|_2^2 + \frac{\rho}{2}\|\mathbf{s} - \boldsymbol{\alpha}_2\|_2^2 \\ \text{subject to} \quad & \mathbf{H}\mathbf{z} - \mathbf{s} = \mathbf{h}, \end{aligned} \tag{26}$$

where $\boldsymbol{\alpha}_1 := \mathbf{w}^k - (1/\rho)\boldsymbol{\mu}_1^k$, $\boldsymbol{\alpha}_2 := \mathbf{u}^k - (1/\rho)\boldsymbol{\mu}_2^k$, $\boldsymbol{\mu} = [\boldsymbol{\mu}_1^T, \boldsymbol{\mu}_2^T]^T \in \mathbb{R}^{p+q}$ is the vector of dual variables corresponding to problem (23), and $k$ is the ADMM iteration number. The solution of problem (26) is given by KKT conditions: $\rho\mathbf{z} + \mathbf{H}^T\boldsymbol{\lambda} = \rho\boldsymbol{\alpha}_1$, $\rho\mathbf{s} - \boldsymbol{\lambda} = \rho\boldsymbol{\alpha}_2$, and $\mathbf{H}\mathbf{z} - \mathbf{s} = \mathbf{h}$, where $\boldsymbol{\lambda} \in \mathbb{R}^q$ is the Lagrangian multiplier corresponding to problem (26). These form a system of linear equations

$$\mathbf{C}\begin{bmatrix} \mathbf{z} \\ \mathbf{s} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \rho\boldsymbol{\alpha}_1 \\ \rho\boldsymbol{\alpha}_2 \\ \mathbf{h} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \rho\mathbf{I}_p & \mathbf{0} & \mathbf{H}^T \\ \mathbf{0} & \rho\mathbf{I}_q & -\mathbf{I}_q \\ \mathbf{H} & -\mathbf{I}_q & \mathbf{0} \end{bmatrix}. \tag{27}$$

Based on (2), the linear system (27) can be mapped onto a memristor network by configuring its memristance values. Recall that a programmed memristor crossbar only requires a constant-time complexity $O(1)$ to solve problem (27).

The ADMM step (7) with respect to $\mathbf{w}$ and $\mathbf{u}$ becomes

$$\underset{\mathbf{w},\mathbf{u}}{\text{minimize}} \quad \|\mathbf{w}\|_1 + p(\mathbf{u}) + \frac{\rho}{2}\|\mathbf{w} - \boldsymbol{\beta}_1\|_2^2 + \frac{\rho}{2}\|\mathbf{u} - \boldsymbol{\beta}_2\|_2^2, \tag{28}$$

where $\boldsymbol{\beta}_1 := \mathbf{z}^{k+1} + (1/\rho)\boldsymbol{\mu}_1^k$ and $\boldsymbol{\beta}_2 := \mathbf{s}^{k+1} + (1/\rho)\boldsymbol{\mu}_2^k$. Note that problem (28) can be decomposed into two problems with respect to $\mathbf{w}$ and $\mathbf{u}$:

$$\begin{cases} \underset{\mathbf{w}}{\text{minimize}} \;\; \|\mathbf{w}\|_1 + \dfrac{\rho}{2}\|\mathbf{w} - \boldsymbol{\beta}_1\|_2^2, \\ \underset{\mathbf{u}}{\text{minimize}} \;\; \|\mathbf{u} - \boldsymbol{\beta}_2\|_2^2, \;\; \text{subject to} \;\; \|\mathbf{u}\|_2 \le \xi. \end{cases} \tag{29}$$

Both problems in (29) can be solved analytically [30]

$$\begin{cases} \mathbf{w}^{k+1} = (\boldsymbol{\beta}_1 - 1/\rho\mathbf{1})_+ - (-\boldsymbol{\beta}_1 - 1/\rho\mathbf{1})_+, \\ \mathbf{u}^{k+1} = \min\{\xi, \|\boldsymbol{\beta}_2\|_2\}\frac{\boldsymbol{\beta}_2}{\|\boldsymbol{\beta}_2\|_2}, \end{cases} \tag{30}$$

where recall that $(\cdot)_+$ is the positive part operator.

Similar to LPs and QPs, the hardware design of the memristor-based CS solver mainly consists of two parts. The first part is the memristor-based linear system solver, in which memristor crossbars are only programmed once since the coefficient matrix $\mathbf{C}$ in (27) is independent of ADMM iterations. The second part is the digital or analog implementation of the solution to problem (30). This requires the calculation

of the $\ell_2$ norm of a vector that can be realized using elementary logic or digital operations; similar to Fig. 4. The ADMM-based solution exhibits low hardware complexity.

We finally remark that one can adjust the ADMM parameter $\rho$ to avoid the hardware variation-induced singularity for $\mathbf{C}$ in (27). This is supported by the invertibility of the Schur complement of $\mathbf{C}$ [75], $(-1/\rho)(\mathbf{I} + \mathbf{H}\mathbf{H}^T)$. Specifically, if $\rho$ is too large, the Schur complement approaches zero (towards singularity). If $\rho$ is too small, the effect of hardware variations on $\mathbf{H}$ is magnified. Therefore, an appropriate choice of $\rho$ enhances the robustness of memristor-based optimization solvers to hardware variations.

### *C. Performance evaluation*

Next, we empirically show the effectiveness of the proposed solution framework for sparse signal recovery. Assume that the original signal $\mathbf{z}^*$ is of dimension $p = 1024$ with $s \in \{10, 50, 100, 150, 200\}$ nonzero elements. These nonzero spike positions are chosen randomly, and their values are chosen independently from the standard normal distribution. To specify the CS problem (22), a measurement matrix $\mathbf{H} \in \mathbb{R}^{500\times 1024}$ with i.i.d. entries from the standard normal distribution is generated, and set $\xi = 10^{-3}$. The vector of measurement noises $\mathbf{v}$ is drawn from the normal distribution $\mathcal{N}(\mathbf{0}, 0.01\mathbf{I})$. To evaluate the recovery performance, the following two measures are employed a) the difference between the recovered signal $\mathbf{z}$ and the true sparse signal $\mathbf{z}^*$, namely, $\|\mathbf{z} - \mathbf{z}^*\|$, and b) the sparse pattern difference between $\mathbf{z}$ and $\mathbf{z}^*$. All the performance measures are obtained by averaging over $50$ random trials. For ADMM, unless specified otherwise, we set $\rho \in \{0.1, 1, 10, 100\}$ and $\epsilon = 10^{-3}$ for its augmented parameter and stopping tolerance.

In Fig. 7, we present the performance of sparse signal recovery by using the memristor-based solution framework. Fig. 7(a) shows the signal recovery error as a function of the sparsity level $s$ under different levels of hardware variations. We compare the resulting solution with the solution obtained from the orthogonal matching pursuit (OMP) algorithm [76], a commonly used software-based CS solver. We observe that the recovery accuracy improves as the signal becomes sparser, namely, $s$ is smaller. This is not surprising, since a sparser signal can be more stably recovered at the rate much smaller than what is commonly prescribed by Shannon-Nyquist theorem [69]. By fixing $s$, we observe that the recovery accuracy decreases while increasing the level of hardware variations. Although the presence of hardware variations negatively affects the recovery accuracy, the sparse pattern error shown by Figs. 7(b) and (c) is acceptable, as it is below $6\%$. In particular, in Fig. 7(c) the recovered signal yields almost the same sparse support as that of the original signal even in the presence of $10\%$ hardware variation. These promising

results show that the memristor-based CS solver is quite robust to hardware variations, and is able to provide reliable recovered sparse patterns. Lastly, we investigate the convergence of the memristor-based approach against different values of the ADMM parameter $\rho$. Similar to Fig. 6, a moderate choice of $\rho$, namely, $\rho = 10$ in this example, is preferred over others as shown in Fig. 7(d).

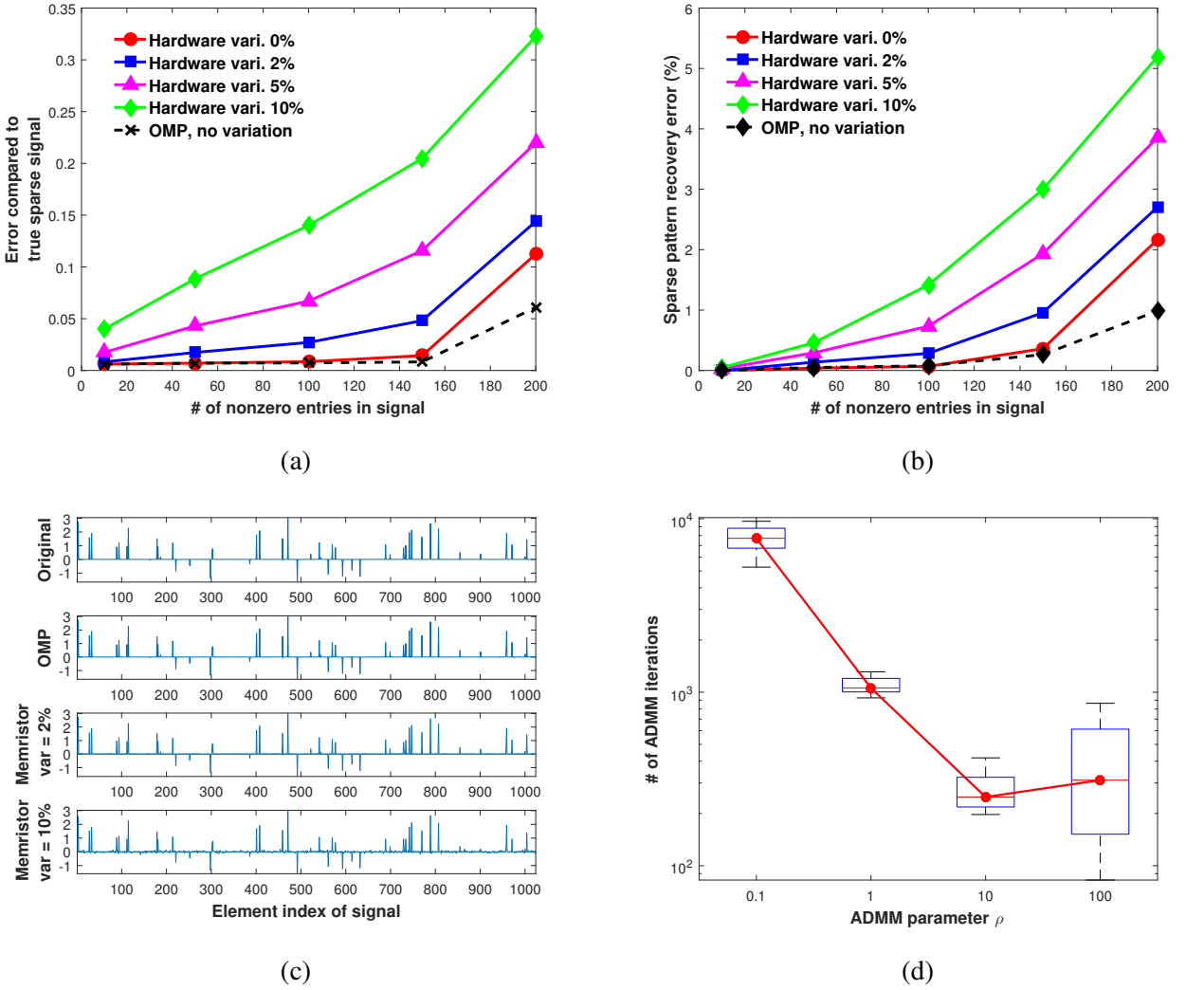



**Fig. 7:** Sparse signal recovery performance under different levels of hardware variation. (a) Error with respect to the true signal versus the sparsity level $s$. (b) Sparse pattern recovery error versus the sparsity level $s$. (c) Recovered signals with $s = 50$ nonzero entries. (d) Number of iterations required for convergence versus the ADMM parameter $\rho$.

## VI. Power Iteration via Memristors: Application to PCA

Principal component analysis (PCA) is the best-known dimensionality-reduction technique to find intrinsic low-dimensional manifolds from high-dimensional data [40]. The implementation of PCA requires the computation of the principal eigenvalues and the corresponding eigenvectors of a symmetric matrix. The calculation of eigenvalues and eigenvectors is also motivated by optimization problems, e.g., a projection onto semidefinite cones in semidefinite programming [77]. Since power iteration (PI) is a widely-used algorithm for eigenvalue analysis [78], here we describe a memristor-based PI framework.

### A. Preliminaries on PI

PI is an iterative algorithm that converges to the eigenvector associated with the largest eigenvalue of a matrix. Let $\{(\lambda_i, \mathbf{u}_i)\}_{i=1}^n$ denote a set of eigenvalue-eigenvector pairs for matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, where we refer to $\lambda_1$, regardless of its multiplicity, as the dominant eigenvalue. The $k$th iteration of PI is given by [42]

$$\mathbf{x}^k = \frac{\mathbf{A}\mathbf{x}^{k-1}}{\|\mathbf{A}\mathbf{x}^{k-1}\|_2}, \tag{31}$$

where $\mathbf{x}^0$ is an arbitrary starting vector. If $k \to \infty$, then by (31), $\mathbf{x}^k$ converges to the eigenvector $\mathbf{u}_1$, and thus $\frac{(\mathbf{x}^k)^T\mathbf{A}\mathbf{x}^k}{(\mathbf{x}^k)^T\mathbf{x}^k}$ converges to the largest eigenvalue $\lambda_1$. The convergence of PI is geometric, with ratio $\frac{|\lambda_2|}{|\lambda_1|}$ [42]. Therefore, PI converges slowly if there is an eigenvalue close in magnitude to the dominant eigenvalue. Moreover, if the largest eigenvalue is not unique, say $\lambda_1 = \lambda_2$ with multiplicity 2, the limiting point $\mathbf{x}^k$ fails to converge to $\mathbf{u}_1$, and instead converges to a linear combination of eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$ [79]. Thus, it is required that the memristor-based PI be able to address the issue of repeated eigenvalues.

### B. Memristor-based PI

It is clear from (31) that the PI algorithm involves a) matrix-vector multiplication $\mathbf{A}\mathbf{x}^{k-1}$, and b) evaluation of a vector norm. Based on (2), the first operation is easily implemented using memristor crossbars. And the second operation can be realized using elementary digital (or analog) circuits [30]. The major challenge of customizing PI for memristor implementation is to determine the multiplicity of the dominant eigenvalue and to find the corresponding eigenvectors. In what follows, we show that with the aid of Gram-Schmidt process such a problem can be addressed via elementary matrix-vector operations.

We assume that the largest eigenvalue has multiplicity $s$, namely, $\lambda_1 = \lambda_2 = \ldots = \lambda_s$. Under $s$ random initial vectors, we denote by $\{\mathbf{y}_i\}_{i=1}^s$ the converging vectors of PI. It is known from [79] that $\{\mathbf{y}_i\}_{i=1}^s$

are linear combinations of eigenvectors $\{\mathbf{u}_i\}_{i=1}^s$. This implies two facts. First, given $p$ initial vectors, the resulting $\{\mathbf{y}_i\}_{i=1}^p$ are linearly independent if $p \leq s$ and linearly dependent if $p > s$. Therefore, we are able to determine the number of repeated dominant eigenvalues by adding new columns to $\mathbf{Y}_p$ until its rank stops increasing where $\mathbf{Y}_p := [\mathbf{y}_1, \ldots, \mathbf{y}_p]$, and its rank can be determined by the singularity of $\mathbf{Y}_p\mathbf{Y}_p^T$. Second, given the number of repeated eigenvalues, finding the eigenvectors $\{\mathbf{u}_i\}_{i=1}^s$ is equivalent to seeking an orthogonal subspace spanned by $\{\mathbf{y}_i\}_{i=1}^s$. This procedure is precisely described by the Gram-Schmidt process. Given a sequence of vectors $\{\mathbf{y}_i\}_{i=1}^s$, the Gram-Schmidt process generates a sequence of orthogonal vectors $\{\mathbf{u}_i\}_{i=1}^s$ [42],

$$\mathbf{u}_i = \mathbf{y}_i - \sum_{j=1}^{i-1} \frac{\mathbf{y}_i^T\mathbf{u}_j}{\mathbf{u}_j^T\mathbf{u}_j}\mathbf{u}_j, \ i = 2, \ldots, s, \tag{32}$$

where $\mathbf{u}_1 = \mathbf{y}_1$.

By incorporating the Gram-Schmidt process (32), the generalized PI algorithm is able to calculate the dominant eigenvalue even if it is not unique. Once the dominant eigenvalue $\lambda_1$ is found, the second largest eigenvalue $\lambda_2$ can then be found by performing PI to a new matrix $\mathbf{A} - \lambda_1\mathbf{u}_1\mathbf{u}_1^T$, known as a matrix deflation [42]. Since both (31) and (32) only involve elementary matrix-vector operations, it is possible to accelerate PI by using memristors.

### *C. Performance evaluation*

In what follows, we demonstrate the empirical performance of the proposed PI method to compute the dominant eigenvalues/eigenvectors based on a synthetic dataset and to perform PCA based on the Iris flower dataset [80]. To specify the eigenvalue problem, let $\mathbf{A}$ be a symmetric matrix of dimension $n = 50$. We assume that the dominant eigenvalue is repeated $k$ times, where $k \in [1, 10]$. The proposed algorithm continues until a $10^{-4}$-accuracy solution is achieved. Such an experiment is performed over $50$ independent trials. In Fig. 8, we present the computation error, success rate, and the number of iterations of PI against the multiplicity of the dominant eigenvalue. Here the computation error is averaged over $50$ trials, and given by the difference between the memristor-based solution $\lambda$ and the optimal solution $\lambda^*$ obtained from the eigenvalue decomposition. As we can see, the proposed PI solver is of high accuracy with error less than $10^{-6}$. Moreover, at each trial, the proposed solver correctly recognizes the number of repeated dominant eigenvalues. And it converges fast, within $1000$ iterations.

In Fig. 9, we apply the proposed PI solver to find the principal components (PCs) of the Iris flower dataset, which contains $150$ iris flowers, and each flower involves $4$ measurements, sepal length, sepal width, petal length and petal width. These flowers belong to three different species: setosa, versicolor,

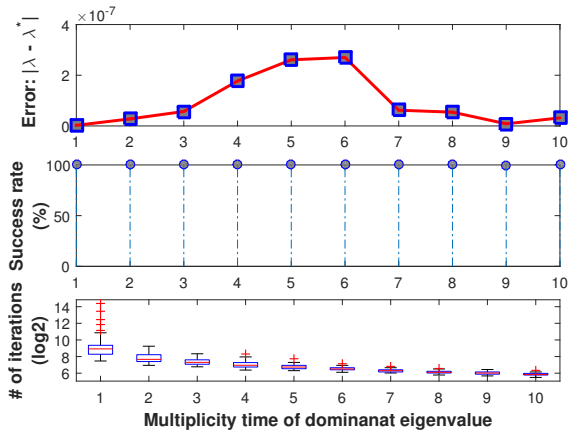



**Fig. 8:** Performance of the proposed PI solver against the multiplicity of the dominant eigenvalue.

and virginica. We compare the memristor-based approach with the standard *pca* function in MATLAB. As we can see, both methods yield the same 2D data distribution and the same variance of each PC. These results imply that the application of memristor crossbars is of feasible for this problem.

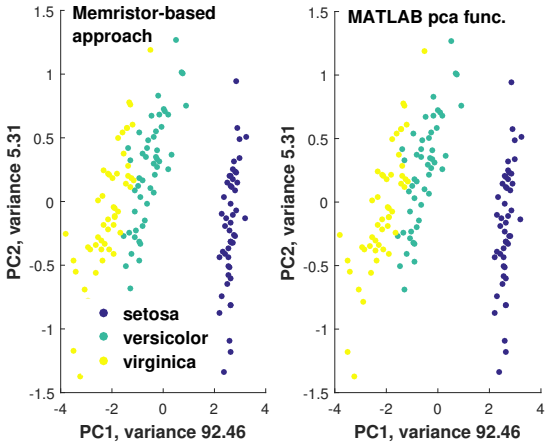



**Fig. 9:** PCA results for the Iris flower dataset. (Left) memristor-based approach; (Right) MATLAB *pca* function.

## VII. Conclusion and Future Directions

In this paper, we presented an overview of a memristor-based optimization/computation framework that exploits both memristors' properties and algorithms' structures. Popularly used algorithms, ADMM

and PI, were selected to illustrate memristor crossbar-based implementations. We showed that ADMM is able to decompose a complex problem into matrix-vector multiplications and subproblems for solving systems of linear equations, which then facilitates memristor-based computing architectures. To solve the eigenvalue problem using memristor crossbars, we presented a generalized version of the PI algorithm in the presence of repeated dominant eigenvalues. The effectiveness of memristor-based framework was illustrated via examples involving LP, QP, compressive sensing and PCA. The framework showed a great deal of promise with low computational complexity and high resiliency to hardware variations.

Although there has been a great deal of progress on the design of memristor-based computation accelerators, many questions and challenges still remain to enable its adoption in real-life applications, e.g., enhancing memristor-based computing precision, co-optimizing algorithm and hardware for nonconvex optimization, and determining the feasibility of other problems that can benefit from memristor-based hardware implementation. Some specific future directions are discussed below.

First, memristor-based computing systems have not yet demonstrated a competitively high computation accuracy for solving practical problems in the presence of hardware variations. To enhance precision, extra hardware resources would be needed. It is thus essential to optimize a full hardware system under given hardware resources. Problems of interest include selection of device-level components in hardware implementation, and design of energy-efficient on-chip communication infrastructure.

Second, the convergence of ADMM for nonconvex optimization is not guaranteed. Therefore, new optimization algorithms, appropriate for hardware design, are desired to address nonconvex problems, e.g., artificial neural network based applications. Traditional algorithms to train neural networks, such as back-propagation or other gradient-based approaches, require updating of the gradient information at each iteration. This leads to frequent writing/reading operations on memristor crossbars and thus an increasing amount of energy consumption. Motivated by that, innovation beyond the existing algorithms is encouraged to co-optimize algorithm and hardware for nonconvex optimization.

Third, in many scenarios, it is assumed that certain solutions exist for the considered optimization and machine learning problems. However, it is possible that the mapped problems on memristor crossbars are infeasible, e.g., no solution exists for an overdetermined linear system. Therefore, a robust memristor crossbar-based solver should be capable of identifying the feasibility of problems. This identification procedure should be implemented by using device-level components subject to limited hardware resources.

Fourth, there is much work to be done to expand the applications of memristor crossbars from the end-user perspective. Some potential lucrative applications include memristor-based smart sensors, small footprint intelligent controllers in wearable devices, and on-chip training platforms in autonomous vehicles

and Internet of Things.

To sum up, the memristor technology has the potential to revolutionize computing, optimization and machine learning research due to its orders-of-magnitude improvement in energy efficiency and computation speed. Moving forward, engineers and scientists in different fields, such as, machine learning, signal processing, circuits and systems, and materials should collaborate with each other to make significant progress on this exciting research topic.

## REFERENCES


[1] Leon Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 05 2008.

[3] R. Kozma, R. E. Pino, and G. E. Pazienza, "Are memristors the future of AI?," in *Advances in neuromorphic memristor science and applications*, pp. 9–14. Springer, 2012.

[4] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs, L. Xie, N. Wald, S. Joshi, H. M. Elsayed, H. Corporaal, and K. Bertels, "Memristor for computing: Myth or reality?," in *Proc. IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 722–731.

[5] M. Hu, H. Li, Q. Wu, G. S. Rose, and Y. Chen, "Memristor crossbar based hardware realization of bsb recall function," in *Proc. International Joint Conference on Neural Networks (IJCNN)*, June 2012, pp. 1–7.

[6] I. Richter, K. Pas, X. Guo, R. Patel, J. Liu, E. Ipek, and E. G. Friedman, "Memristive accelerator for extreme scale linear solvers," in *Government Microcircuit Applications & Critical Technology Conference (GOMACTech)*, 2015.

[7] A. Ren, S. Liu, R. Cai, W. Wen, P. K. Varshney, and Y. Wang, "Algorithm-hardware co-optimization of the memristor-based framework for solving socp and homogeneous qcqp problems," in *Proc. 22nd IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 788–793.

[8] R. Cai, A. Ren, Y. Wang, S. Soundarajan, Q. Qiu, B. Yuan, and P. Bogdan, "A low-computation-complexity, energy-efficient, and high-performance linear program solver using memristor crossbars," in *Proc. 29th IEEE International System-on-Chip Conference (SOCC)*, 2016, pp. 317–322.

[9] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[10] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Yu Wang, H. Jiang, M. Barnell, Q. Wu, and J. Yang, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Proc. 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] R. Hasan and T. M. Taha, "Memristor crossbar based unsupervised training," in *2015 National Aerospace and Electronics Conference (NAECON)*, June 2015, pp. 327–332.

[13] R. Hasan, T. Taha, and M. Z. Alom, "A reconfigurable low power high throughput streaming architecture for big data processing," *arXiv preprint https://arxiv.org/abs/1603.07400*, 2016.

[14] R. Hasan, T. M. Taha, and C. Yakopcic, "On-chip training of memristor crossbar based multi-layer neural networks," *Microelectronics Journal*, vol. 66, pp. 31–40, 2017.

[15] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: design considerations," *Frontiers in neuroscience*, vol. 10, 2016.

[16] S. Agarwal, S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James, and M. J. Marinella, "Resistive memory device requirements for a neural algorithm accelerator," in *Proc. IEEE International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 929–938.

[17] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, "Training itself: Mixed-signal training acceleration for memristor-based neural network," in *Proc. 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 361–366.

[18] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 10, pp. 2408–2421, 2015.

[19] C. Yakopcic, R. Hasan, and T. M. Taha, "Flexible memristor based neuromorphic system for implementing multi-layer neural network algorithms," *International Journal of Parallel, Emergent and Distributed Systems*, pp. 1–22, 2017.

[20] X. Liu, M. Mao, B. Liu, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, J. Yang, H. Li, and Y Chen, "Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 5, pp. 617–628, 2016.

[21] Y. Wang, W. Wen, B. Liu, D. Chiarulli, and H. H. Li, "Group scissor: Scaling neuromorphic computing design to large neural networks," in *Proc. 54th Annual Design Automation Conference*, 2017, p. 85.

[22] L. Ni, Z. Liu, H. Yu, and R. Joshi, "An energy-efficient digital reram-crossbar based cnn with bitwise parallelism," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 2017.

[23] L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[24] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, and W. D. Lu, "Sparse coding with memristor networks.," *Nature nanotechnology*, 2017.

[25] S. Yu and Y. Cao, "On-chip sparse learning with resistive cross-point array architecture," in *Proc. 25th edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 195–197.

[26] J.-S. Seo, B. Lin, M. Kim, P.-Y. Chen, D. Kadetotad, Z. Xu, A. Mohanty, S. Vrudhula, S. Yu, J. Ye, and Y. Cao, "On-chip sparse learning acceleration with cmos and resistive synaptic devices," *IEEE Transactions on Nanotechnology*, vol. 14, no. 6, pp. 969–979, 2015.

[27] P.-Y. Chen, B. Lin, I.-T. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J.-S. Seo, Y. Cao, and S. Yu, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 194–199.

[28] P. Y. Chen, D. Kadetotad, Z. Xu, A. Mohanty, B. Lin, J. Ye, S. Vrudhula, J. S. Seo, Y. Cao, and S. Yu, "Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip," in *Proc. 2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 854–859.

[29] D. Kadetotad, Z. Xu, A. Mohanty, P.-Y. Chen, B. Lin, J. Ye, S. Vrudhula, S. Yu, Y. Cao, and J. Seo, "Neurophysics-

inspired parallel architecture with resistive crosspoint array for dictionary learning," in *Proc. IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Oct 2014, pp. 536–539.

[30] S. Liu, A. Ren, Y. Wang, and P. K. Varshney, "Ultra-fast robust compressive sensing based on memristor crossbars," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 1133–1137.

[31] S. B. Eryilmaz, E. Neftci, S. Joshi, S. Kim, M. BrightSky, H.-L. Lung, C. Lam, G. Cauwenberghs, and H.-S. P. Wong, "Training a probabilistic graphical model with resistive switching electronic synapses," *IEEE Transactions on Electron Devices*, vol. 63, no. 12, pp. 5004–5011, 2016.

[32] L. Chen, C. Li, T. Huang, Y. Chen, and X. Wang, "Memristor crossbar-based unsupervised image learning," *Neural Computing and Applications*, vol. 25, no. 2, pp. 393–400, 2014.

[33] L. Chen, C. Li, T. Huang, S. Wen, and Y. Chen, "Memristor crossbar array for image storing," in *International Symposium on Neural Networks*. Springer, 2015, pp. 166–173.

[34] R. Mansini, W. Ogryczak, and M. G. Speranza, "Twenty years of linear programming based portfolio optimization," *European Journal of Operational Research*, vol. 234, no. 2, pp. 518–535, 2014.

[35] C. M. Bishop, *Pattern recognition and machine learning*, springer, 2006.

[36] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *Found. Trends Mach. Learn.*, vol. 4, no. 1, pp. 1–106, Jan. 2012.

[37] S. Boyd and L. Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

[38] M. Hu, H. Li, Q. Wu, and G. Rose, "Hardware realization of neuromorphic bsb model with memristor crossbar network," in *IEEE Design Automation Conference (DAC)*, 2012, pp. 554–559.

[39] D. Kadetotad, Z. Xu, A. Mohanty, P.-Y. Chen, B. Lin, J. Ye, S. Vrudhula, S. Yu, Y. Cao, and J.-S. Seo, "Neurophysics-inspired parallel architecture with resistive crosspoint array for dictionary learning," in *Proc. IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2014, pp. 536–539.

[40] I. K. Fodor, "A survey of dimension reduction techniques," Tech. Rep., Lawrence Livermore National Lab., CA (US), 2002.

[41] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[42] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3, JHU Press, 2012.

[43] M. Di Ventra, Y. V. Pershin, and L. O. Chua, "Circuit elements with memory: Memristors, memcapacitors, and meminductors," *Proceedings of the IEEE*, vol. 97, no. 10, pp. 1717–1724, Oct. 2009.

[44] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[45] L. Ni, H. Huang, Z. Liu, R. V. Joshi, and H. Yu, "Distributed in-memory computing on binary rram crossbar," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 36, 2017.

[46] A. Heittmann and T. G. Noll, "Limits of writing multivalued resistances in passive nanoelectronic crossbars used in neuromorphic circuits," in *Proc. great lakes symposium on VLSI*, 2012, pp. 227–232.

[47] S. H. Jo, K.-H. Kim, and W. Lu, "High-density crossbar arrays based on a si memristive system," *Nano letters*, vol. 9, no. 2, pp. 870–874, 2009.

[48] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, 2012.

[49] D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont, MA, 1999.

[50] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[51] B. He and X. Yuan, "On the $O(1/n)$ convergence rate of the douglas-rachford alternating direction method," *SIAM Journal on Numerical Analysis*, vol. 50, no. 2, pp. 700–709, 2012.

[52] J. S. Aronofsky, "Growing applications of linear programming," *Communications of the ACM*, vol. 7, no. 6, pp. 325–332, 1964.

[53] H. Dahrouj and W. Yu, "Coordinated beamforming for the multicell multi-antenna wireless system," *IEEE Transactions on wireless communications*, vol. 9, no. 5, 2010.

[54] S. Liu, S. Kar, M. Fardad, and P. K. Varshney, "Sparsity-aware sensor collaboration for linear coherent estimation," *IEEE Transactions on Signal Processing*, vol. 63, no. 10, pp. 2582–2596, 2015.

[55] Y. J. A. Zhang and A. M.-C. So, "Optimal spectrum sharing in mimo cognitive radio networks via semidefinite programming," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 2, pp. 362–373, 2011.

[56] A. Nemirovski, "Interior point polynomial time methods in convex programming," *Lecture Notes*, 2004.

[57] C. Yakopcic, R. Hasan, and T. M. Taha, "Hybrid crossbar architecture for a memristor based cache," *Microelectronics Journal*, vol. 46, no. 11, pp. 1020–1032, 2015.

[58] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *IEEE Proc. Design Automation Conference*, 2001, pp. 684–689.

[59] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose, "Bsb training scheme implementation on memristor-based circuit," in *Proc. IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, April 2013, pp. 80–87.

[60] W. Wen, C. R. Wu, X. Hu, B. Liu, T. Y. Ho, X. Li, and Y. Chen, "An eda framework for large scale hybrid neuromorphic computing systems," in *Proc. 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[61] Inc. CVX Research, "CVX: Matlab software for disciplined convex programming, version 2.0," *http://cvxr.com/cvx*, Aug 2012.

[62] S. Liu, *Resource management for distributed estimation via sparsity-promoting regularization*, Ph.D. thesis, Syracuse University, 2016.

[63] K. Slavakis, G. B. Giannakis, and G. Mateos, "Modeling and optimization for big data analytics:(statistical) learning tools for our era of data deluge," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 18–31, 2014.

[64] S. P. Chepuri and G. Leus, "Sparse sensing for statistical inference," *Foundations and Trends® in Signal Processing*, vol. 9, no. 3–4, pp. 233–368, 2016.

[65] E. J. Candes and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, March 2008.

[66] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[67] S. Liu, E. Masazade, M. Fardad, and P. K. Varshney, "Sparsity-aware field estimation via ordinary Kriging," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 3948–3952.

[68] S. Qaisar, R. M. Bilal, W. Iqbal, M. Naureen, and S. Lee, "Compressive sensing: From theory to applications, a survey," *Journal of Communications and Networks*, vol. 15, no. 5, pp. 443–456, Oct. 2013.

[69] E. Candès, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Comm. Pure and Applied Math.*, vol. 59, no. 8, pp. 1207–1223, 2006.

[70] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, April 2006.

[71] E. J. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 489–509, Feb. 2006.

[72] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2230–2249, May 2009.

[73] W. Xu and B. Hassibi, "Efficient compressive sensing with deterministic guarantees using expander graphs," in *IEEE Information Theory Workshop*, Sept. 2007, pp. 414–419.

[74] E. Candès, "Compressive sampling," in *Proc. International Congress of Mathematicians*, 2006.

[75] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," *Technical University of Denmark*, vol. 7, pp. 15.

[76] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on information theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[77] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.

[78] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23, Prentice hall Englewood Cliffs, NJ, 1989.

[79] M. Panju, "Iterative methods for computing eigenvalues and eigenvectors," *The Waterloo Mathematics Review*, 2011.

[80] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, "Support vector clustering," *Journal of machine learning research*, vol. 2, pp. 125–137, 2001.