ARTICLES
https://doi.org/10.1038/s41928-019-0270-x

nature electronics

# A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations

Fuxi Cai[1,3], Justin M. Correll[1,3], Seung Hwan Lee[1,3], Yong Lim[1,2], Vishishtha Bothra[1], Zhengya Zhang[1], Michael P. Flynn[1] and Wei D. Lu[1*]

**Memristors and memristor crossbar arrays have been widely studied for neuromorphic and other in-memory computing applications. To achieve optimal system performance, however, it is essential to integrate memristor crossbars with peripheral and control circuitry. Here, we report a fully functional, hybrid memristor chip in which a passive crossbar array is directly integrated with custom-designed circuits, including a full set of mixed-signal interface blocks and a digital processor for reprogrammable computing. The memristor crossbar array enables online learning and forward and backward vector-matrix operations, while the integrated interface and control circuitry allow mapping of different algorithms on chip. The system supports charge-domain operation to overcome the nonlinear *I–V* characteristics of memristor devices through pulse width modulation and custom analogue-to-digital converters. The integrated chip offers all the functions required for operational neuromorphic computing hardware. Accordingly, we demonstrate a perceptron network, sparse coding algorithm and principal component analysis with an integrated classification layer using the system.**

Memristors are two-terminal resistive devices that have a conductance state that depends on one or more internal state variables and can be modulated by the history of external stimulation[1–5]. Due to their compact device structure and ability to both store and process information at the same physical location, memristors and memristor crossbar arrays have been explored for neuromorphic computing, machine learning and edge computing applications[5–8]. These include single-layer perceptron[9–11] and multi-layer perceptron networks[12,13], image transformation[14,15], sparse coding[16], reservoir computing[17] and principal component analysis (PCA)[18]. In addition, novel approaches using multiple memristor devices or mixed-precision architectures have been developed to effectively mitigate device non-idealities and allow the devices to be used in high-precision computing and training tasks[19–24].

Although the key matrix operations can be performed efficiently with memristor crossbar arrays[14,15,25], previous implementations have largely relied on external printed-circuit boards to provide the required interface and control circuitry[10,14,16], or discrete parameter analysers to generate and collect signals[9,12,15]. In cases where memristor arrays are integrated with periphery circuitry, the circuit's function has been limited to providing access devices (for example, in the form of 1T1R arrays[10,13,14,25]) or address decoding purposes[26,27]. Demonstrating the potential of memristor-based computing hardware requires the development of fully functional systems, where the memristor crossbars are integrated with necessary analogue interface circuitry (including analogue-to-digital converters (ADCs) and digital-to-analogue converters (DACs)), digital buses and ideally a programmable processor to control the digital and analogue components. Integrating all the necessary functions on chip will be key to enabling the practical implementation of memristor-based computing systems and allowing the prototypes to be scaled to larger systems.

In this Article, we report a fully integrated, functional, reprogrammable memristor chip, including a passive memristor crossbar array directly integrated with all the necessary interface circuitry, digital buses and an OpenRISC processor to form a complete hardware system on chip. Thanks to the re-programmability of the memristor crossbar and the integrated complementary metal–oxide–semiconductor (CMOS) circuitry, the system is highly flexible and can be programmed to implement different computing models and network structures. Three widely used models—a perceptron network, a sparse coding algorithm and a bilayer PCA system with an unsupervised feature extraction layer and a supervised classification layer—are demonstrated experimentally on the same chip.

## Fully integrated reprogrammable neuromorphic chip

A memristor crossbar is very efficient at vector-matrix multiplication (VMM), because the values in the matrix can be stored as the analogue device conductances of the crossbar array. When an input vector is applied as voltage pulses with different pulse amplitudes or different pulse widths to the rows of the crossbar, the currents or charges collected at the columns of the crossbar correspond to the resulting VMM outputs, following Ohm's law and Kirchhoff's current law[14,16,25,28]. This approach allows direct computing of this data-intensive task both in memory and in parallel[5,29] in a single step, that is $O(1)$, and has attracted broad interest for implementing neuromorphic computing and machine learning models[28,30,31].

The results from the crossbar are then used to update the neuron outputs, which typically require efficient ADC circuits that convert the analogue current or charge signals collected at the columns to digital signals that can be processed by the artificial neurons. Similarly, neuron outputs need to be supplied to the devices in the crossbar, typically through DAC circuits feeding the rows. Controllers are needed to convert the input signals to pulse

[1]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA. [2]Present address: Samsung Electronics, Yongin, Korea. [3]These authors contributed equally: Fuxi Cai, Justin M. Correll, Seung Hwan Lee. *e-mail: wluee@umich.edu

amplitude or width, and to implement weight update rules during training. To reduce latency and power consumption, all these components need to be integrated together with the crossbar array on a single chip, instead of using discrete components on a board. Integrating a processor on chip also allows the neuron functions and network structures to be reprogrammed though simple software changes, enabling different models to be mapped on the same hardware platform.

In this study we have designed and fabricated a complete, integrated memristor/CMOS system, with a memristor crossbar array integrated on top of CMOS circuits consisting of a full set of ADCs, DACs, digital bus, memory and a processor, allowing the integrated system to take advantage of the efficiency of the matrix operations provided by the memristor crossbar and the flexibility offered by the CMOS system to successfully implement different algorithms on chip. The system architecture is shown in Supplementary Figs. 1 and 2.

We operate the crossbar array in the charge domain to minimize multiplication error due to memristor device $I$–$V$ nonlinearity (Supplementary Note 1). Our approach also simplifies the interface circuit design as the input pulses to the crossbar have fixed amplitude and variable widths. The charge-domain technique is enabled by a current-integrating hybrid ADC design and a pulse-mode DAC scheme (Supplementary Figs. 3 and 4).

Our design features both DACs and ADCs at each row and each column for flexibility and to allow bidirectional and full transpose operation of the crossbar. For example, inputs $x$ can be applied to the rows and the charges collected at the columns correspond to the forward operation $x^{\mathrm{T}}W$. If the output neurons' activities are then applied to the columns, the charges collected at the rows then represent the reconstructed signal, corresponding to multiplication of the output neurons' activities $a$ and the transpose of the weight matrix $aW^{\mathrm{T}}$. With DACs and ADCs at each row and column, both forward and backward operations can be obtained with the same crossbar array[16]. Furthermore, our design allows all DACs and ADCs at each row/column to operate in parallel, thereby supporting high-throughput parallel VMM and transpose VMM operations. More details of the system architecture are provided in Supplementary Fig. 1 and Supplementary Note 2.

The mixed-signal interface is controlled by an on-chip OpenRISC processor that allows the implementation of different computing operations, as shown in Supplementary Fig. 2 and Supplementary Note 2. The processor provides fine-grained programmability, allowing the operation of any set of rows and columns, in either read or write mode and forward or backward direction (Supplementary Figs. 5–8 and Supplementary Note 3). The design also supports online dictionary learning with higher-voltage write DACs to program or update the memristor conductance (Supplementary Figs. 7 and 8).

A 54×108 $WO_x$-based memristor crossbar is fabricated on top of the CMOS circuits. Figure 1a shows a photograph of the integrated chip after packaging, along with the testing set-up. Figure 1b,c are top-view images of the chip, showing the memristor array integrated on top of the chip surface, at different magnifications. Each row and column of the crossbar array is connected to a specific landing pad left open during the CMOS fabrication process, and then connected to the interface circuitry through internal CMOS wiring (Fig. 1d; see also Supplementary Figs. 9 and 10 and Methods for fabrication details). Each row or column of the array is connected to two write DACs, one read DAC and a 13 bit ADC (Fig. 1e). The memristors can be successfully programmed by controlling the number of applied write/erase pulses, then read out by the integrated interface circuitry and controller, even without any current-limiting transistors or external current compliance (Supplementary Fig. 11). Figure 1f shows the on/off ratio distribution of devices in the integrated array after a weight update operation. The relative uniform distribution of devices within the array allows the system to implement a number of computing tasks on chip. We note that device variations can still be observed. This effect is exacerbated by the line resistance effects (Supplementary Figs. 12 and 13). Further device optimizations and the use of a monolithic integration technique that reduces the line resistance can lead to better array and network performance (Supplementary Fig. 14 and Supplementary Note 4).

In the following, we discuss different models and neural network structures implemented in the same chip, by reprogramming the on-chip processor and the functions of the memristor array and the interface circuits.

## Training and classification using a single-layer perceptron

A feed-forward single-layer perceptron (SLP) network was first implemented to verify the operation of the integrated chip; 5×5 binary patterns were used in the SLP training and testing. The SLP has 26 inputs (corresponding to the 25 pixels in the image and a bias term) and 5 outputs, with the input and output neurons fully connected with $26\times5=130$ synaptic weights, where the neuron with the highest output is identified as the winner and used to classify the corresponding class, as schematically shown in Fig. 2a (see Methods).

In our implementation, the original binary input patterns are converted into input voltage pulses through the integrated processor and DAC circuitry and are fed to the rows of the memristor array. Specifically, when a white pixel is present, a pulse is applied to the corresponding row; while black pixels correspond to no pulse. The bias term is fixed at a constant value of 1 (treated as a white pixel) and is applied as an extra input. All the input pulses have the same duration and amplitude in this test, as illustrated in Fig. 2b. Each synaptic weight $w_{ij}$ is implemented with two memristors representing a positive weight and a negative weight, $G_{ij}^{+}$ and $G_{ij}^{-}$, respectively, using the positive memristor conductance values (see equation (10) in Methods).

The integrated chip allows us to perform online learning. Specifically, the synaptic weights are updated during the training phase using the batch gradient descent rule:

$$\Delta w_{ij} = \eta \sum_{n=1}^{N} \left( t_j^{(n)} - y_j^{(n)} \right) x^{(n)} \tag{1}$$

where $x^{(n)}$ is the $n$th training sample of the input dataset, $y^{(n)}$ is the network output and $t^{(n)}$ is the corresponding label. $\eta$ is the learning rate. The update value $\Delta w_{ij}$ for the $i$th element in the $j$th class is then implemented in the memristors by applying programming pulses through the write DACs with a pulse width proportional to the desired weight change (quantized within the range of 0–63 time steps corresponding to 6 bit precision).

We trained and tested the SLP with noisy 5×5 Greek letter patterns, for five distinct classes: 'Ω', 'M', 'Π', 'Σ', 'Φ'. For each Greek letter, we flip one of the 25 pixels of the original image and generate 25 noisy images. Together with the original image they form a set of 26 images for each letter. We randomly select 16 images from the set of each class for training and use the other 10 images for testing (Fig. 2c). Examples of the training set and testing set are shown in Supplementary Figs. 15 and 16. This approach guarantees that the training set and the testing set do not overlap, and therefore improves the robustness of our testing results.

Training and testing results from the experimentally implemented SLP are shown in Fig. 2d,e. After five online training epochs the SLP can already achieve 100% classification accuracy for both the training and testing sets. The average activation of the correct neuron during training is also clearly separated from the others, and the difference in neuron outputs between the winning neuron and the other neurons improves during training, as shown in Fig. 2d, verifying that online learning has been reliably implemented in the experimental set-up.

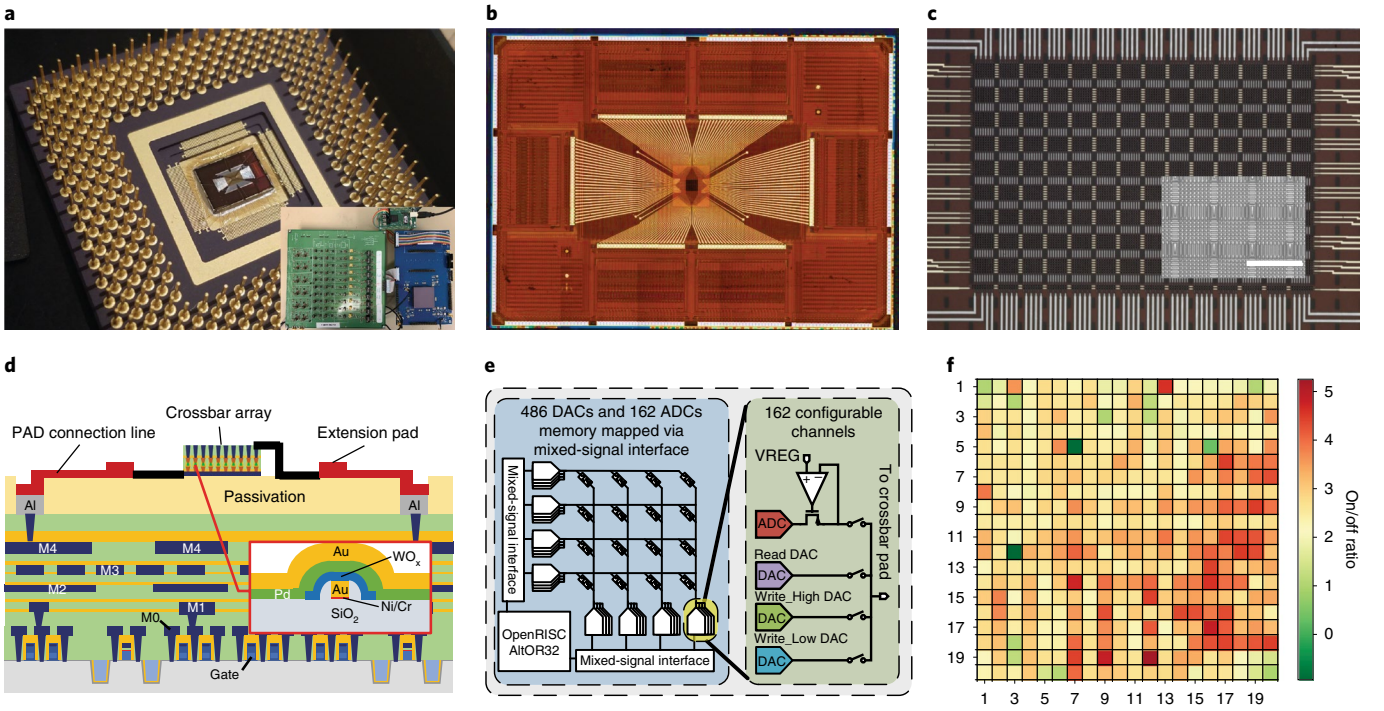


**Fig. 1 | Fully integrated memristor/CMOS chip. a**, Integrated chip wire-bonded on a pin-grid array package. Inset, testing set-up used to power and test the integrated memristor/CMOS chip. **b**, Optical microscopic image of a memristor crossbar array integrated on the chip, with portions of the CMOS circuitry visible underneath the chip surface. **c**, Magnified image of the integrated 54×108 memristor array region. Inset, scanning electron microscope (SEM) image of the $WO_x$ memristor crossbar. Scale bar, 50 μm. **d**, Cross-section schematic of the integrated chip, showing connections of the memristor array with the CMOS circuitry through extension lines and internal CMOS wiring. Inset, cross-section of the $WO_x$ device. **e**, Schematic of the mixed-signal interface to the 54×108 crossbar array, with two write DACs, one read DAC and one ADC for each row and column. **f**, Distribution map for a 20×20 subarray of the memristor crossbar. Colour represents the on/off ratio measured after 50 consecutive programming pulses (1.8 V, 82 μs; see Supplementary Note 1 for more details).

## Sparse coding algorithm implementation

The same hardware system was then used to implement a sparse coding algorithm (see Methods). Sparse coding aims at representing the original data with the activity of a small set of neurons, and can be traced to models of data representation in the visual cortex[32,33]. Sparse coding is an efficient method for feature extraction and information compression, and allows pattern recognition and classification to be performed in the compressed domain[34].

Following our previous work implemented at the board level[16], we mapped the locally competitive algorithm (LCA)[35] on our integrated memristor/CMOS chip. In this approach, the membrane potential of an output neuron is determined by the input, a leakage term and an inhibition term whose strength is proportional to the similarity of the neurons' features; that is, an active neuron will try to inhibit neurons having similar features. It can be shown mathematically that lateral neuron inhibition can be achieved in the memristor crossbar by removing the reconstructed signal from the input (see equation (13) in Methods). With this approach, the LCA algorithm can be implemented in an iterative process through two VMM operations: in the forward direction to obtain neuron activations and in the backward direction to obtain the reconstructed input. The residue term is then obtained by removing the reconstructed input from the original input, and is then fed to the network. The process is repeated until the network stabilizes. Figure 3a illustrates the iterative forward and backward processes employed in the LCA implementation.

The bidirectional operation of the memristor array in the integrated memristor/CMOS chip allowed us to experimentally implement the sparse coding algorithm on chip. Similar to the SLP case, we use the crossbar array to perform VMM operations, here in both forward and backward directions. Given that the chip offers full flexibility to implement different algorithms by re-programming the integrated processor, the LCA algorithm was implemented in the same chip used in the SLP study, through simple software changes.

We used 4×4 inputs to test the experimental implementation of the LCA algorithm. By using linear combinations of horizontal and vertical bar patterns, the input dimension is reduced to 7. To satisfy the over-completeness requirement of the LCA algorithm, a dictionary containing 14 features of horizontal and vertical bar patterns is used, as shown in Fig. 3b. This set-up produces a 2× over-complete dictionary[35] that enables the network to find a sparse, optimal solution out of several possible solutions.

The LCA algorithm was mapped to a 16×14 subarray in the memristor/CMOS chip, using the corresponding interface circuitry and the processor that provide the neuron functions. An example of the LCA network operation is shown in Fig. 3c,d. The experimentally implemented network correctly reconstructs the input image while minimizing the number of activated neurons. For example, it identifies the optimized solution with two neurons 6 and 13, instead of using three neurons 2, 4 and 6 in this case. The dynamics of the LCA network operation can also be correctly captured, as shown in Fig. 3e, where the effects of lateral neuron inhibition that lead to a more sparse solution than the initial solution can be clearly observed (see Methods).

To verify the system's performance for other input patterns, an exhaustive test of all 24 possible patterns consisting of two horizontal bars and one vertical bar was performed using the on-chip memristor network, resulting in 100% success rate (Fig. 3f), measured by the network's ability to correctly identify the sparse solutions.

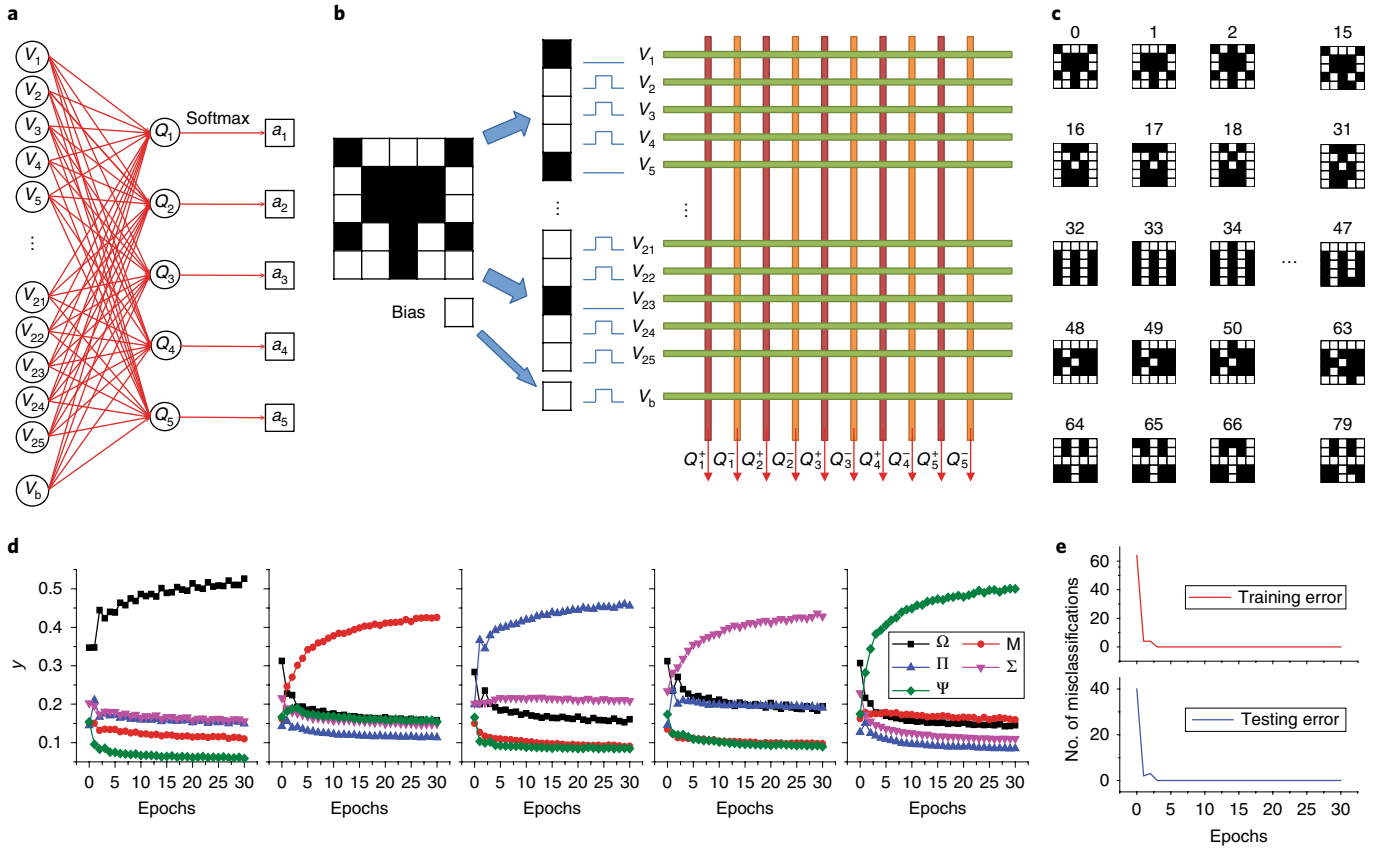


**Fig. 2 | Experimental demonstration of the single-layer perceptron on the integrated memristor chip. a**, Schematic of the single-layer perceptron for classification of 5×5 images. **b**, Implementation of the SLP using a 26×10 memristor subarray through the integrated chip. Input data (for example, the Greek letter 'Ω') are converted to voltage pulses of $V_{read}$ or 0 through the on-chip circuitry, depending on the pixel value. **c**, Samples from the training data set. **d**, Evolution of the output neuron signals during training. Each data point represents the average output value for a specific neuron during a training epoch for a given input class. **e**, Classification error of the training and testing data set as a function of training epochs. The network can achieve 100% classification for both the training set and the testing set after five training epochs.

## Implementation of a multi-layer neural network

Finally, we demonstrate a bilayer neural network with the same integrated chip, using two subarrays in the memristor crossbar implementing unsupervised and supervised online learning to achieve feature extraction and classification functions, respectively. The bilayer network is used to analyse and classify data obtained from breast cancer screening based on PCA. Specifically, the first layer of the system is a 9×2 network that performs PCA of the original data, which reduces the nine-dimensional (9D) raw input data to a 2D space based on the learned principal components (PCs). The second layer is a 3×1 SLP layer (with differential weights and a bias term), which performs classification using the reduced data in the 2D space for the two classes (benign or malignant). The schematic and crossbar implementation of the bilayer network are shown in Fig. 4a,b.

PCA reduces data dimensionality by projecting data onto lower dimensions along the PCs, with the goal of finding the best summary of the data using a limited number of PCs[36]. The conventional approach to PCA is to solve the eigenvectors of the covariance matrix of the input data, which can be computationally expensive in hardware. A more hardware-friendly approach is to find the PCs through unsupervised, online learning.

Specifically, following our previous study[18], Sanger's rule, also known as the generalized Hebbian algorithm, is implemented in the integrated chip to obtain the PCs (Supplementary Note 5). The desired weight change for the $j$th PC is determined by

$$\delta g_{ij} = \eta y_j \left( x_i - \sum_{k=1}^{j} g_{ij} y_j \right) \quad (2)$$

In the experiment, the weights of the first and second PCs, $g_{ij}$, are mapped onto the memristor conductances through a linear transformation[18] (Supplementary Note 5). The network is trained online, using a subset of the original database consisting of 100 data points. During the training process, the 9D breast cancer data are converted into input voltage pulses with pulse widths proportional to the data values, within the range of 0–63 time units. The output charge collected at column $j$ then corresponds to the dot product of the input vector and the conductance vector stored in column $j$, projecting data from the original 9D space to a 2D output space (when only two PCs are used). During training, the weights are then updated following equation (2), using programming voltage pulses generated through the write DACs with pulse widths proportional to $\delta g_{ij}$.

Initially, the weights of the first and second PCs are randomized in the memristor array (Fig. 4c). Projection of the input along these vectors leads to severe overlapping of the benign and malignant cases in the 2D space, as shown in Fig. 4d,e. After 30 training epochs (an epoch is defined as a training cycle through the 100 training data), the PCs are correctly learned (Fig. 4f) and the 2D projected data can be clearly separated into two clusters (Fig. 4g,h). Note that the ground truth (benign or malignant) is not used in the PCA training or clustering process. It is included in the plots

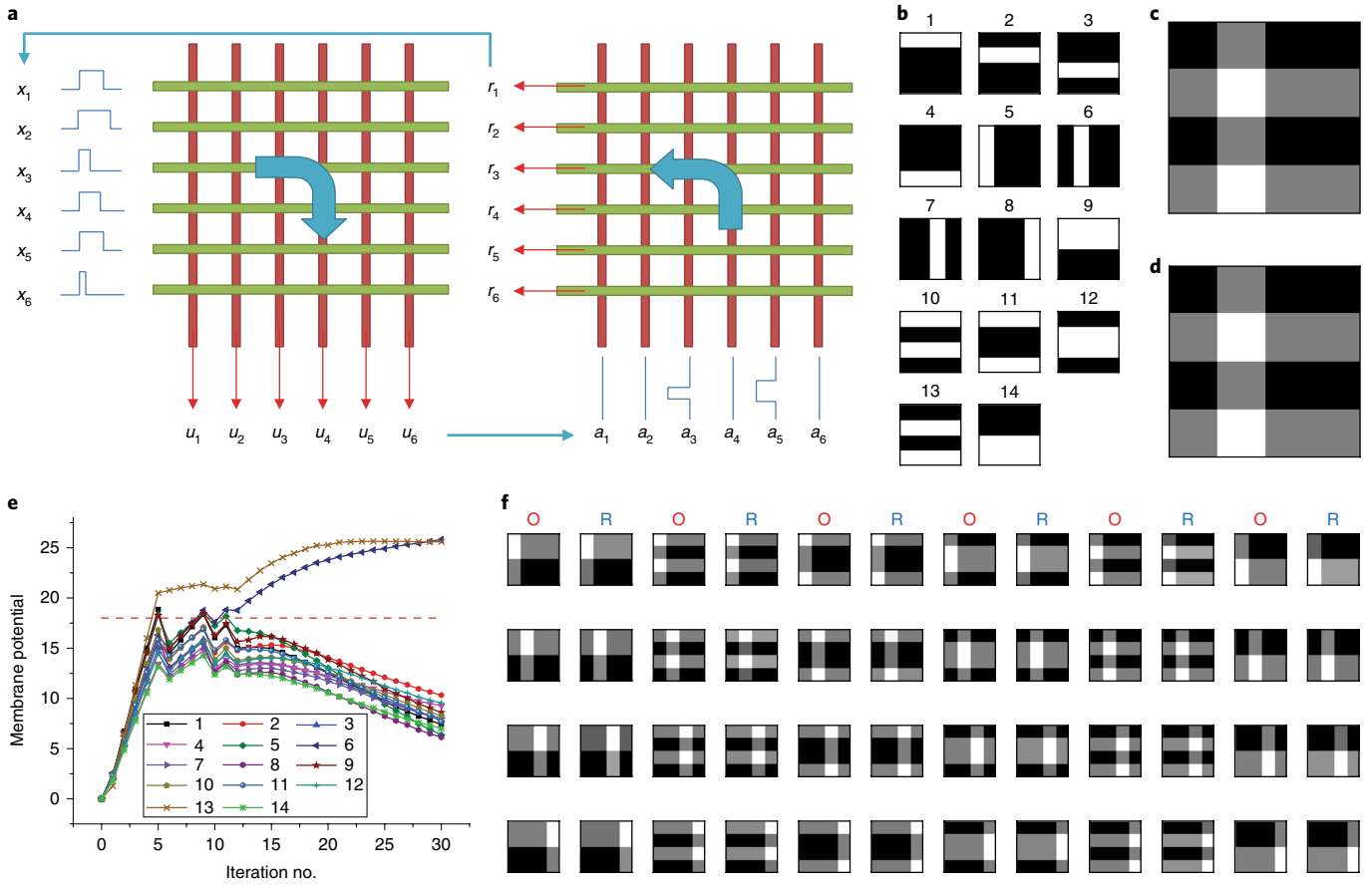


**Fig. 3 | Experimental demonstration of sparse coding using the integrated memristor chip. a**, Schematic of the experimental implementation of the LCA algorithm using on-chip memristor arrays. In each iteration a forward pass is performed to update the membrane potential *u* of the output neurons based on the inputs *x*, followed by a backward pass to update the residual *r* based on the neuron activities *a*. The residual *r* becomes the input for the next iteration. **b**, Dictionary elements based on horizontal and vertical bars. **c**, Original test image. **d**, Experimentally reconstructed image based on the neuron activities from the memristor chip. **e**, Experimentally obtained neuron membrane potentials as a function of iteration number during LCA analysis. The horizontal red dashed line marks the threshold parameter $\lambda$. **f**, Additional examples of original input images (O) and reconstructed images (R). The same threshold $\lambda = 18$ is used in all experiments in **d**–**f**.

(represented as blue and red colours in Fig. 4g,h) only to highlight the effectiveness of the clustering before and after learning the PCs.

The PCA layer separates the original data into clusters, but does not classify them. To achieve classification, we implement a second layer, an SLP, in the same hardware system. The SLP processes outputs from the PCA layer and generates a label (benign or malignant). Given that there are only two classes to distinguish, the SLP is trained online using logistic regression. A $3 \times 2$ subarray is used in the second layer to account for the two inputs, the bias term and the differential weights (see Methods).

After learning the PCs in the PCA layer, the original 9D data are fed through the PCA layer, and the clustered 2D data are used as inputs for the SLP layer. The same 100 training data used for the PCA layer training are used for the SLP layer training (Fig. 5a) in a supervised fashion, using the ground truth (the label associated with the original data, Supplementary Note 6). Training is completed after 30 epochs.

Afterwards, the 500 test data not included in the training set are applied to the network, passing first through the PCA layer then as 2D data into the SLP layer. After online training of the PCA and the SLP layers, the experimentally implemented two-layer network can achieve 94% and 94.6% classification accuracy during training and testing (Fig. 5b,c). The values are slightly lower than those obtained from software implementation (95% during training and 96.8% during testing, Fig. 5d,e), due to the non-ideality in the memristor weight update that results in a decision boundary that differs from that obtained from software (which assumes ideal linear weight updates) after the online training process (Supplementary Figs. 17 and 18). Other statistical parameters, including the sensitivity, specificity and accuracy for the experimentally implemented PCA + classifier network, were calculated to be 93.1%, 99.0% and 94.6%, respectively, along with excellent receiver operating characteristic and $F_1$ scores (Supplementary Fig. 19 and Supplementary Note 7).

## Performance analysis of the integrated memristor chip

An important design goal in this study is to have the complete algorithms run on-chip, without having to access off-chip storage for weights and instructions. To achieve this goal, we first program all necessary instructions in the algorithm in C code, and compile the C code into binary machine code. This process only needs to be performed once for each application. Afterwards, the binary code is loaded into the static random-access memory on chip by a bootloader, and all operations are performed on chip. During training and inference, the binary program instructions are executed through the OpenRISC processor without the need to access

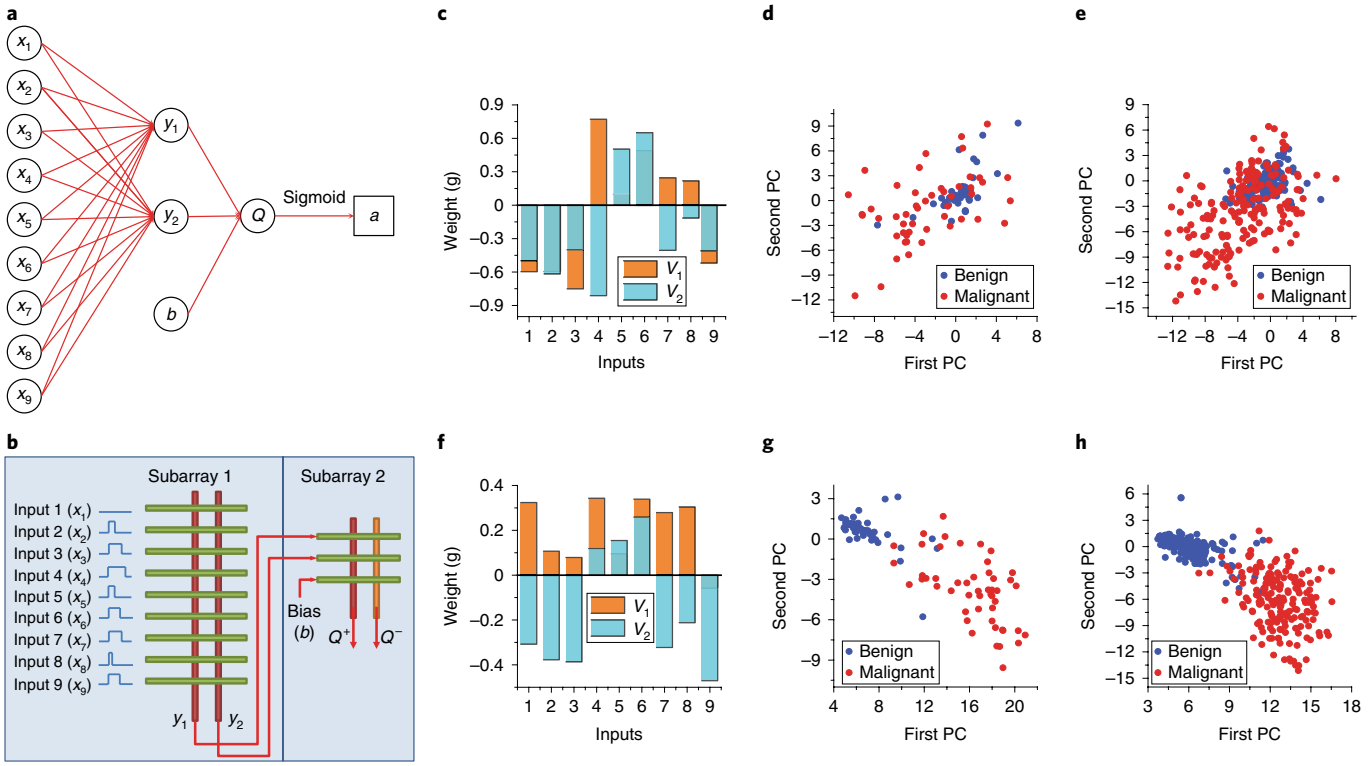


**Fig. 4 | Experimental demonstration of PCA using the integrated chip. a**, Schematic of the bilayer neural network for PCA analysis and classification. **b**, The bilayer network is mapped onto the integrated memristor chip, using a 9×2 subarray for the PCA layer and a 3×2 subarray for the classification layer. **c**, Initial weights for the two PCs in the network. **d**,**e**, Before training, linear separation is not possible in the projected 2D space, for both training (**d**) and testing (**e**) data. **f**, Weights for the two PCs after unsupervised online training obtained from the memristor network, using Sanger's learning rule and 30 training cycles. **g**,**h**, Clear separation can be observed in the 2D space for both training (**g**) and testing (**h**) data after projection along the trained PCs. Note that the colour labels (representing the ground truth) are used only to highlight the effect of clustering in the plots, and are not used in the PC training or PCA analysis.

external controllers. The inputs to the memristor array are supplied through the DACs following the instructions, and the VMM results are read as charge values from the ADCs and processed by the OpenRISC processor for batch gradient decent calculations or for running other algorithms. The results (for example, batch update values) are then sent to the on-chip registers, and the DACs are reconfigured for the next operations. After the process is complete, the final output can be transferred to a computer and accessed by the users. Details of the data path are provided in Supplementary Fig. 20 and Supplementary Note 8.

The integrated chip suggests that different computing tasks can be efficiently mapped on the memristor-based computing platform by taking advantage of the bidirectional VMM operations in the memristor crossbars and the flexibility of the CMOS interface and control circuitry. In our prototype, the supporting analogue interfaces, as well as digital control and the OpenRISC processor, are implemented in 180 nm CMOS technology. The entire mixed-signal interface with independent ADCs and DACs supporting the 54×108 crossbar and operating at the maximum frequency of 148 MHz consumes 64.4 mW, obtained from experimental measurements. This corresponds to an energy consumption of 6.53 nJ per inner product or 1.12 pJ per operation for the mixed-signal interface, where an operation is defined as the multiplication and accumulate (MAC) process of a 4 bit input with a stored analogue weight in the memristor array. At the maximum operating frequency of 148 MHz, the OpenRISC core consumes 235.3 mW and the system supports 9.87 M VMMs per second, corresponding to a throughput of 57.5 GOPS. Along with an average power consumption of 7 mW experimentally measured from the 54×108 crossbar, this results in a total system power of 306.7 mW and a power efficiency of 187.62 GOPS per W for the experimentally demonstrated memristor/CMOS chip based on 180 nm CMOS technology. Simply scaling the design to a more advanced process node such as 40 nm CMOS technology would reduce the total system power consumption to 42.1 mW, corresponding to a power efficiency of 1.37 TOPS per W (Supplementary Fig. 21 and Supplementary Note 9). Further optimizations of the system design, for example by replacing the generic processor with a custom-designed controller or field-programmable gate array, and by replacing the fast and high-precision 13 bit ADC with simpler interface circuits (for example, 8 bit neuron activations may be sufficient for common neural networks) and more optimized ADC designs, along with memristor device optimizations that reduce power consumption in the memristor crossbar, can further improve the system's performance and power efficiency (see Supplementary Figs. 22–27 and Supplementary Note 10 for details about circuit optimizations).

During batch training, the memory needed to store the batch information may become a bottleneck as the task becomes complex and the pattern size increases. Batch gradient descent has also been associated with overfitting due to the low stochasticity of the process. For complicated tasks and large datasets, mini-batch training[37,38] may be an attractive option by splitting the training dataset into small batches, with typical batch sizes between 2 and 32. Additionally, recently proposed hybrid techniques that utilize CMOS capacitors to store the lower-significance weight updates could also provide a realistic solution to the batch information storage challenge[22].

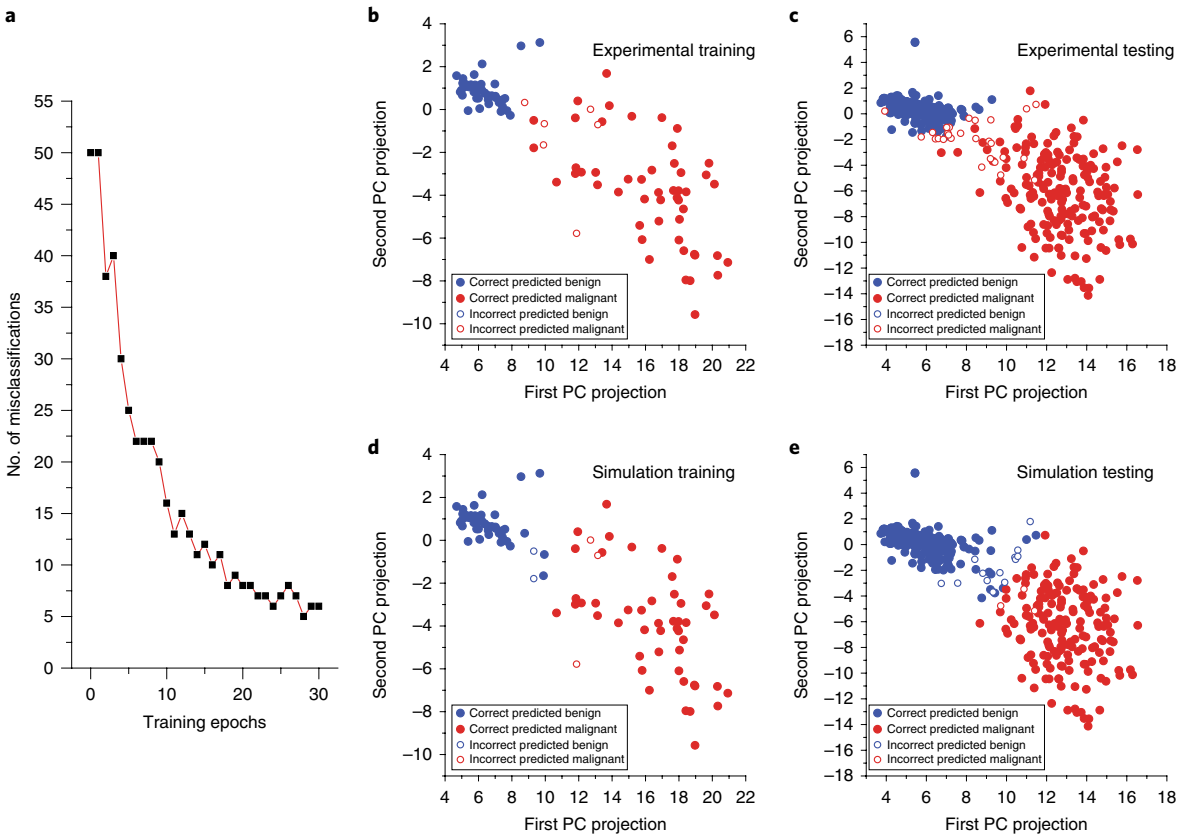


**Fig. 5 | Classification results in the bilayer network. a**, Evolution of the classification error during the online training process, from the experimentally implemented bilayer network on chip. **b**,**c**, Classification results experimentally obtained from the memristor chip, for the training (**b**) and testing (**c**) data. Blue and red colours represent the predicted benign and malignant data, respectively. The incorrectly classified results are marked as open circles. Classification rates of 94% and 94.6% are obtained for the training and testing data, respectively. **d**,**e**, Classification results of the bilayer network implemented in software. Blue and red colours represent the predicted benign and malignant data, respectively. Incorrectly classified results are marked as open circles. Classification rates of 95% and 96.8% are obtained for the training (**d**) and testing (**e**) data in software, respectively.

To achieve high accuracy with more complex and larger datasets, the memristor device properties should be further improved. The $WO_x$ memristor device can be reliably programmed over $10^7$ times. Although this level of endurance can support certain online training algorithms, longer endurance may be desirable. Additionally, the cycle to cycle variation during the $10^7$ programming cycles is ~3.4–4.2% (Supplementary Fig. 28a) and the device to device variability is ~4.5% (Supplementary Fig. 28b). The small models we implemented in the current study can tolerate these cycle-to-cycle and device-to-device variations; however, device nonlinearity and variability need to be reduced to implement larger networks[39]. To this end, future device optimizations that can improve device uniformity and weight update linearity[13,40], along with architecture innovations such as hybrid non-volatile memory (NVM)–CMOS neural-network implementations[22], mixed-precision[19], multi-memristive device architectures[21] and other precision extension techniques[24] can be employed. To scale up the system for larger networks, rather than simply increasing the crossbar size, a promising approach may be to tile small crossbars together in a modular fashion[24,41,42], as schematically shown in Supplementary Figs. 29 and 30. In this approach, each tile is a self-contained, integrated memristor–CMOS unit (macro); these units are then tiled together using digital interfaces to construct larger systems[5] (Supplementary Note 11).

## Conclusions

We have reported the design and fabrication of a fully functional, programmable neuromorphic computing chip, in which a passive memristor crossbar array is integrated with a complete set of analogue and digital components and an on-chip processor. The integrated chip allows mapping of different neuromorphic and machine learning algorithms on chip through simple software changes. Three different and commonly used models, perceptron, sparse coding and PCA with an integrated classification layer, have been demonstrated. A classification accuracy of 100% was achieved for 5×5 noisy Greek letters in the SLP implementation, reliable sparse coding analysis was obtained from an exhaustive test set using 4×4 bar patterns, and 94.6% classification rate was experimentally obtained from the breast cancer screening dataset using the same integrated chip.

The integrated memristor–CMOS systems potentially offer efficient hardware solutions for different network sizes and applications[6–10,13,14,16,19,20,22,25,43]. An initial application of such systems may be edge computing, for example as used in the Internet of Things (IoT) to process data near its source, allowing real-time data processing with high speed and low energy consumption[44,45]. Continued device, circuit and architecture innovations as discussed above, along with algorithm advances such as quantized neural networks[46,47], can allow the system to be scaled up for more complex and more demanding tasks.

## Methods

**Crossbar array fabrication and integration.** The memristor crossbar array used in this work was directly fabricated on top of the CMOS circuits. First, bottom electrode (BE) patterns with 500 nm width were defined by electron-beam lithography; the 80-nm-thick Au BEs were then deposited (with Ni/Cr adhesion

layer underneath) by electron-beam evaporation and liftoff processes. Next, 300 nm of $SiO_2$ was deposited by plasma-enhanced chemical vapour deposition, followed by a reactive-ion etch back process to form a spacer structure along the sidewalls of the BEs. The spacer structure allows better step coverage for the $WO_x$ switching layer and the top electrodes (TEs), and also restricts the active device regions to the flat exposed top surface of the BEs, as shown in Fig. 1d. To prevent leakage through the switching layer among adjacent devices, the switching layer was only deposited at the crosspoint regions through electron-beam lithography defined patterns. The switching layer was formed by first depositing 20-nm-thick W by d.c. sputtering and liftoff processes in the electron-beam patterned regions, then through rapid thermal annealing of the patterned W islands with oxygen gas at 400 °C for 60 s to form the $WO_x$ switching material. Afterwards, TEs (Pd (40 nm)/Au (90 nm)) with 500 nm width were patterned and deposited by electron-beam lithography, electron-beam evaporation and liftoff processes. Finally, metallization processes were performed by photolithography to connect the crossbar electrodes with the CMOS landing pads that are left open during the CMOS circuit fabrication process. An in situ etch process was performed to remove the native aluminium oxide on the CMOS landing pads, followed by deposition of 800 nm thick Al with d.c. sputtering and liftoff processes to ensure step coverage of the deeply recessed landing pads.

**Experimental set-up for chip measurement.** The integrated chip was wire-bonded onto a pin-grid-array package and mounted on a printed circuit board (PCB). The PCB provides the needed power signals and the global system clock for the integrated chip. No active circuitry (DACs, ADCs, matrix switches and so on) are implemented on the PCB, as these functions are all provided directly on chip. A UART-to-serial (UART, universal asynchronous receiver–transmitter) board was used to convert the input/ouput data from the chip into serial data and communicate with a desktop computer through a USB cable.

**On-chip analogue-to-digital and digital-to-analogue interfaces.** A charge-based mixed-signal interface was used to support the VMM and programming operations of the integrated crossbar array. We operated the memristor array in the charge domain instead of the voltage/current domain to avoid the nonlinear voltage-to-current multiplication issue (for example, 2 × voltage does not produce 2 × current because of the nonlinear current–voltage characteristics in the device). Specifically, we applied a discrete-time pulse-train input and measured the accumulated charge from each column (row). During VMM operations the row (column) DACs applied 6 bit programmable pulses at fixed amplitude (600 mV with respect to the virtual ground) to the crossbar. The integrating ADCs provided a virtual ground (at 1.2 V) and measured the collected charge over the input period.

A hybrid incremental charge-integrating ADC was used to tackle the challenge of digitizing a broad range of column (row) outputs while minimizing the area and energy consumption. The required large number of ADCs (162), charge domain inputs and large dynamic ranges all present unique design challenges. Supplementary Fig. 3 shows the 13 bit ADC design used for this study, which balances performance and area consumption. Accurate and linear time-domain DACs were implemented with a pulsed, return-to-zero DAC architecture. A simple time-domain DAC is inherently nonlinear because of the finite rise and fall time of the output waveform (Supplementary Fig. 4). Instead, we applied duty-cycled (75%) pulsed inputs where the number of pulses represented the input amplitude. The return-to-zero 'read' DACs toggled between 1.8 V and 1.2 V (virtual ground), creating 600 mV pulses for VMM operations without changing the memristor conductance. Two additional 'write' DACs were connected for each column and row to create 1.8 V amplitude pulses for weight updates during online learning.

**Logistic regression.** A supervised learning algorithm, logistic regression, was used to train the SLP layer of the bilayer network to classify benign and malignant cells.

Logistic regression is commonly used for the classification of two classes. Suppose an $N$ training sample dataset $\mathbf{x} = \left(x^{(1)}, \ldots, x^{(N)}\right)^{\mathrm{T}}$ with label $\mathbf{t} = \left(t^{(1)}, \ldots, t^{(N)}\right)^{\mathrm{T}}$, where the $n$th training sample can be written as $x^{(n)}$ and the $n$th label as $t^{(n)} \in \{0, 1\}$.

A cross-entropy energy function can be defined as

$$E(w) = -\sum_{n=1}^{N}\left\{t^{(n)}\ln y^{(n)} + \left(1 - t^{(n)}\right)\ln\left(1 - y^{(n)}\right)\right\} \tag{3}$$

where $y^{(n)} = \sigma\left(z^{(n)}\right)$ and $z^{(n)} = w^{\mathrm{T}}x^{(n)}$

$\sigma(z)$ is the logistic sigmoid function defined by

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{4}$$

The likelihood of a training data $x^{(n)}$ belonging to class $t^{(n)} = 1$ is determined by the sigmoid function output $y^{(n)}$, with larger $y^{(n)}$ meaning $x^{(n)}$ is more likely to belong to the class.

Taking the gradient of the error function leads to

$$\nabla E(w) = \sum_{n=1}^{N}\left(y^{(n)} - t^{(n)}\right)x^{(n)} \tag{5}$$

To minimize the energy function, the network is trained using batch gradient descent defined as[48]

$$w = w - \eta\sum_{n=1}^{N}\left(y^{(n)} - t^{(n)}\right)x^{(n)} \tag{6}$$

**Softmax regression.** Softmax regression is a generalized logistic regression for multi-class classification, usually used when more than two classes need to be classified. The activation function is defined as

$$y_j^{(n)}\left(z^{(n)}\right) = \frac{\exp\left(z_j^{(n)}\right)}{\sum_k \exp\left(z_k^{(n)}\right)} \tag{7}$$

where $z_k^{(n)}$ is given by $z_k^{(n)} = w_k^{\mathrm{T}}x^{(n)}$, representing the likelihood of training data $x^{(n)}$ belonging to class $C_k$ $\left(t^{(n)} = (0, \ldots, \underbrace{1}_{k\mathrm{th}}, \ldots, 0)\right)$

The network ends up with a similar form of gradient as in the logistic regression case:

$$\nabla_{w_j}E(w_1, \ldots, w_K) = \sum_{n=1}^{N}\left(y_j^{(n)} - t_j^{(n)}\right)x^{(n)} \tag{8}$$

To minimize the energy function, the network is trained using batch gradient descent defined as[48]

$$w_j = w_j - \eta\nabla_{w_j}E = w_j - \eta\sum_{n=1}^{N}\left(y_j^{(n)} - t_j^{(n)}\right)x^{(n)} \tag{9}$$

In general, the weighted sums of the inputs can be anything ranging from $-\infty$ to $+\infty$. To bound the neuron output values and perform proper classifications, activation functions play a key role. Nonlinear activation functions such as the Sigmoid and the Softmax functions are among the most widely used for binary and multi-class classification, respectively. In this case, we chose the Softmax function for the multi-class classification task, which allows us to compute the probabilities for all classes. The Softmax function produces neuron output values in the range 0–1, where the neuron with the highest output is identified as the winner and used to classify the corresponding class.

In the memristor crossbar array, the charge collected at an output neuron $j$ is

$$Q_j = \sum_i w_{ij}x_i = V\sum_i G_{ij}t_i = V\sum_i\left(G_{ij}^{+} - G_{ij}^{-}\right)t_i = Q_j^{+} - Q_j^{-} \tag{10}$$

where $x_i$ is the input at row $i$ and is represented by a voltage pulse with amplitude $V$ and width $t_i$. The charges are measured at the output columns and digitized by the ADCs, then converted to the neuron output $y_j$ through the Softmax function:

$$y_j\left(Q_j\right) = \frac{\exp\left(\beta Q_j\right)}{\sum_k \exp\left(\beta Q_k\right)} \tag{11}$$

where $\beta$ is a scaling factor of the ADC output and $k$ represents the output neuron index.

Compared with earlier SLP implementations[9] that used the Manhattan rule and required on average 23 epochs to achieve perfect classification for a similar database, the batch gradient descent rule used here not only considers the direction of the weight update (which represents the Manhattan rule), but also the value of the weight update, so much faster training convergence can be obtained.

**LCA.** The LCA is a sparse coding algorithm that uses a dictionary of feature vectors to encode an input signal with a small number of output coefficients, while minimizing the reconstruction error.

The concept of sparse coding is as follows. Given an input signal $x$ and a dictionary of features $D$, sparse coding aims to represent $x$ as a linear combination of features from $D$ using a set of sparse coefficients $a$, with a minimum number of features. Mathematically, the objective of sparse coding can be summarized as minimizing an energy function containing both the reconstruction error term as well as a sparsity penalty term, defined as

$$\min_a\left(\left|x - Da^{\mathrm{T}}\right|_2 + \lambda|a|_0\right) \tag{12}$$

where $|\cdot|_2$ and $|\cdot|_0$ are the $L^2$- and $L^0$-norm, respectively, and $\lambda$ is a sparsity parameter that determines the relative weights of the reconstruction error (first term) and the sparsity penalty (the number of neurons used, second term).

The mathematical form of the LCA can be expressed as follows: $x$ is an $m$-element ($m \times 1$) input vector, $D$ is an $m \times n$ matrix, where each column of $D$ represents an $m$-element feature vector (that is, a dictionary element) and $a$ is an $n$-element ($1 \times n$) row vector representing the neuron activity coefficients, where the $i$th element of $a$ corresponds to the activity of the $i$th neuron. After feeding input $x$ to the network and allowing the network to stabilize through lateral inhibition, a reconstruction of $x$ can be obtained as $Da^{\mathrm{T}}$, that is, the linear

combination of the neuron activities and the corresponding neurons' feature vectors. In a sparse representation, only a few elements in $a$ are non-zero, while the other neurons' activities are suppressed to be precisely zero.

The neuron dynamics during LCA analysis can be summarized by the following equation[35]:

$$\frac{du}{dt} = \frac{1}{\tau}\left(-u + x^{T}D - a(D^{T}D - I_n)\right) \tag{13a}$$

$$a_i = \begin{cases} u_i \text{ if } u_i > \lambda \\ 0 \text{ otherwise} \end{cases} \tag{13b}$$

where $u_i$ is the membrane potential of neuron $i$, $\tau$ is a time constant and $I_n$ is the $n \times n$ identity matrix.

Implementing the inhibition effect $D^{T}D$ can be very computation-intensive. To implement the algorithm in memristor hardware, the original equation (13a) can be rewritten as

$$\frac{du}{dt} = \frac{1}{\tau}\left(-u + (x - \hat{x})^{T}D + a\right) \tag{14}$$

where $\hat{x} = Da^{T}$ is the reconstructed signal. Equation (14) shows that the inhibition term between neurons can be interpreted as a neuron removing its feature from the input when it becomes active, thus suppressing the activity of other neurons having similar features. The matrix–matrix operation $D^{T}D$ in equation (13a) is thus reduced to two sequential matrix-vector operations, one used to calculate $\hat{x} = Da^{T}$ and the other used to calculate the neuron activity from the updated input $r^{T}D$, where $r = x - \hat{x}$ is the residue term. This approach allows us to implement the LCA in memristor crossbars without physical inhibitory synaptic connections between the neurons.

The dynamics of the LCA network operation are correctly captured by the memristor chip, as shown in Fig. 3e. In this example, all neurons are charging up in the first four iterations. At the fifth iteration, neuron 13 first crosses the threshold, as it consists of two horizontal bars which result in a larger output value in the membrane potential update. As a result, the lateral inhibition effect in the system suppresses the membrane potentials of other neurons (2 and 4) sharing part of the features of neuron 13, even though they also represent features of the input. Meanwhile neuron 6, which represents the vertical bar feature, continues to charge up. At iteration 11, the membrane potential of neuron 6 crosses the threshold, and all other neurons' membrane potentials are suppressed below the threshold, leading to the optimal solution. The neurons' membrane potentials continue to evolve, but those of neurons 6 and 13 remain above the threshold and those of other neurons continue to decrease due to the inhibition term and the leaky term in the membrane potential equation. The solution from the network was read out after 30 iterations.

**Breast cancer dataset and classification.** A standard breast cancer dataset from the University of Wisconsin Hospital[49] was used as the input for the PCA network, which is available through the University of Californian, Irvine Machine Learning Repository[50].

The dataset consists of breast cell mass properties measured in nine categories; each property is scored from 1 to 10. Each input to the memristor network is thus a 9D vector consisting of scores from the nine measurements. The bilayer network was trained using 100 training data (containing 50 benign and 50 malignant cases) from the dataset, and tested with 500 data (containing 312 benign and 188 malignant cases) not in the training set.

## Data availability

The data that support the plots within this paper and other findings of this study are available from the corresponding author upon reasonable request.

Received: 18 October 2018; Accepted: 12 June 2019;
Published online: 15 July 2019

## References

1. Chua, L. Memristor—the missing circuit element. *IEEE Trans. Circuit Theory* **18**, 507–519 (1971).
2. Strukov, D. B., Snider, G. S., Stewart, D. R. & Williams, R. S. The missing memristor found. *Nature* **453**, 80–83 (2008).
3. Jo, S. H. et al. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* **10**, 1297–1301 (2010).
4. Yu, S., Wu, Y., Jeyasingh, R., Kuzum, D. & Wong, H. S. P. An electronic synapse device based on metal oxide resistive switching memory for neuromorphic computation. *IEEE Trans. Electron Devices* **58**, 2729–2737 (2011).
5. Zidan, M. A., Strachan, J. P. & Lu, W. D. The future of electronics based on memristive systems. *Nat. Electron.* **1**, 22–29 (2018).
6. Krestinskaya, O., James, A. P. & Chua, L. O. Neuro-memristive circuits for edge computing: a review. Preprint at https://arxiv.org/abs/1807.00962 (2018)
7. Xia, Q. & Yang, J. J. Memristive crossbar arrays for brain-inspired computing. *Nat. Mater.* **18**, 309–323 (2019).
8. Ielmini, D. & Wong, H.-S. P. In-memory computing with resistive switching devices. *Nat. Electron.* **1**, 333–343 (2018).
9. Prezioso, M. et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015).
10. Yao, P. et al. Face classification using electronic synapses. *Nat. Commun.* **8**, 15199 (2017).
11. Alibart, F., Zamanidoost, E. & Strukov, D. B. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nat. Commun.* **4**, 2072 (2013).
12. Bayat, F. M. et al. Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. *Nat. Commun.* **9**, 2331 (2018).
13. Li, C. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018).
14. Li, C. et al. Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* **1**, 52–59 (2017).
15. Gao, L., Chen, P.-Y. & Yu, S. Demonstration of convolution kernel operation on resistive cross-point array. *IEEE Electron Device Lett.* **37**, 870–873 (2016).
16. Sheridan, P. M. et al. Sparse coding with memristor networks. *Nat. Nanotechnol.* **12**, 784–789 (2017).
17. Du, C. et al. Reservoir computing using dynamic memristors for temporal information processing. *Nat. Commun.* **8**, 2204 (2017).
18. Choi, S., Shin, J. H., Lee, J., Sheridan, P. & Lu, W. D. Experimental demonstration of feature extraction and dimensionality reduction using memristor networks. *Nano Lett.* **17**, 3113–3118 (2017).
19. Le Gallo, M. et al. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
20. Burr, G. W. et al. Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Electron Devices* **62**, 3498–3507 (2015).
21. Boybat, I. et al. Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* **9**, 2514 (2018).
22. Ambrogio, S. et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018).
23. Gao, B. et al. Ultra-low-energy three-dimensional oxide-based electronic synapses for implementation of robust high-accuracy neuromorphic computation systems. *ACS Nano* **8**, 6998–7004 (2014).
24. Zidan, M. A. et al. A general memristor-based partial differential equation solver. *Nat. Electron.* **1**, 411–420 (2018).
25. Hu, M. et al. Memristor-based analog computation and neural network classification with a dot product engine. *Adv. Mater.* **30**, 1705914 (2018).
26. Xia, Q. et al. Memristor–CMOS hybrid integrated circuits for reconfigurable logic. *Nano Lett.* **9**, 3640–3645 (2009).
27. Kim, K.-H. et al. A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. *Nano Lett.* **12**, 389–395 (2012).
28. Yang, J. J., Strukov, D. B. & Stewart, D. R. Memristive devices for computing. *Nat. Nanotechnol.* **8**, 13–24 (2013).
29. Chen, B. et al. Efficient in-memory computing architecture based on crossbar arrays. In *Proceedings of 2015 IEEE International Electron Devices Meeting (IEDM)* 17.5.1–17.5.4 (IEEE, 2015).
30. Pershin, Y. V. & Di Ventra, M. Neuromorphic, digital and quantum computation with memory circuit elements. *Proc. IEEE* **100**, 2071–2080 (2012).
31. Jeong, D. S. & Hwang, C. S. Nonvolatile memory materials for neuromorphic intelligent machines. *Adv. Mater.* **30**, 1704729 (2018).
32. Olshausen, B. A. & Field, D. J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature* **381**, 607–609 (1996).
33. Olshausen, B. A. & Field, D. J. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vis. Res.* **37**, 3311–3325 (1997).
34. Sheridan, P. M., Du, C. & Lu, W. D. Feature extraction using memristor network. *IEEE Trans. Neural Netw. Learn. Syst.* **27**, 2327–2336 (2016).
35. Rozell, C. J., Johnson, D. H., Baraniuk, R. G. & Olshausen, B. A. Sparse coding via thresholding and local competition in neural circuits. *Neural Comput.* **20**, 2526–2563 (2008).
36. Lever, J., Krzywinski, M. & Altman, N. Points of significance: principal component analysis. *Nat. Methods* **14**, 641–642 (2017).
37. Masters, D. & Luschi, C. Revisiting small batch training for deep neural networks. Preprint at https://arxiv.org/abs/1804.07612 (2018).
38. Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. & Tang, P. T. P. On large-batch training for deep learning: generalization gap and sharp minima. Preprint at https://arxiv.org/abs/1609.04836 (2016).
39. Chen, P.-Y., Peng, X. & Yu, S. NeuroSim: a circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **37**, 3067–3080 (2018).
40. Choi, S. et al. SiGe epitaxial memory for neuromorphic computing with reproducible high performance based on engineered dislocations. *Nat. Mater.* **17**, 335–340 (2018).

41. Zidan, M. A. et al. Field-programmable crossbar array (FPCA) for reconfigurable computing. *IEEE Trans. Multi-scale Comput. Syst.* **4**, 698–710 (2017).
42. Mikhailenko, D., Liyanagedera, C., James, A. P. & Roy, K. M2CA: modular memristive crossbar arrays. In *Proceedings of 2018 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5 (IEEE, 2018).
43. Xu, X. et al. Scaling for edge inference of deep neural networks. *Nat. Electron.* **1**, 216–222 (2018).
44. Shafiee, A. et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of 43rd International Symposium on Computer Architecture* 14–26 (IEEE, 2016).
45. Gokmen, T. & Vlasov, Y. Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Front. Neurosci.* **10**, 33 (2016).
46. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. & Bengio, Y. Quantized neural networks: training neural networks with low precision weights and activations. Preprint at https://arxiv.org/abs/1609.07061 (2016).
47. Jacob, B. et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. Preprint at https://arxiv.org/abs/1712.05877 (2017)
48. Bishop, C. M. *Pattern Recognition and Machine Learning* Vol. 4 (Springer, 2006).
49. Mangasarian, O. L., Street, W. N. & Wolberg, W. H. Breast cancer diagnosis and prognosis via linear programming. *Oper. Res.* **43**, 570–577 (1995).
50. Dheeru, D. & Karra Taniskidou, E. *Machine Learning Repository* (Univ. California–Irvine, 2017).

## Acknowledgements

The authors acknowledge inspiring discussions with C. Liu, T. Chou, P. Brown, M.A. Zidan and P.M. Sheridan. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) through award HR0011-13-2-0015, the National Science Foundation (NSF) through awards CCF-1617315 and 1734871, and the Applications Driving Architectures (ADA) Research Centre, a JUMP Centre co-sponsored by SRC and DARPA.

## Author contributions

F.C., J.M.C., S.H.L., Z.Z., M.P.F. and W.D.L. conceived the project and constructed the research frame. S.H.L. prepared the memristor arrays and performed device integration. J.M.C., Y.L., V.B., Z.Z. and M.P.F. designed the CMOS chip. F.C. and J.M.C. prepared the test hardware and software platform. F.C. and S.H.L. performed the network measurements and software simulations. W.D.L. directed the project. F.C., J.M.C., S.H.L., Z.Z., M.P.F. and W.D.L. analysed the experimental data and wrote the manuscript. All authors discussed the results and implications and commented on the manuscript at all stages.

## Competing interests

The authors declare no competing interests.

## Additional information

**Supplementary information** is available for this paper at https://doi.org/10.1038/s41928-019-0270-x.

**Reprints and permissions information** is available at www.nature.com/reprints.

**Correspondence and requests for materials** should be addressed to W.D.L.

**Publisher's note:** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.