ARTICLES
PUBLISHED ONLINE: 22 MAY 2017 | DOI: 10.1038/NNANO.2017.83
nature nanotechnology

Sparse coding with memristor networks

Patrick M. Sheridan†, Fuxi Cai†, Chao Du, Wen Ma, Zhengya Zhang and Wei D. Lu*

Sparse representation of information provides a powerful means to perform feature extraction on high-dimensional data and is of broad interest for applications in signal processing, computer vision, object recognition and neurobiology. Sparse coding is also believed to be a key mechanism by which biological neural systems can efficiently process a large amount of complex sensory data while consuming very little power. Here, we report the experimental implementation of sparse coding algorithms in a bio-inspired approach using a 32 × 32 crossbar array of analog memristors. This network enables efficient implementation of pattern matching and lateral neuron inhibition and allows input data to be sparsely encoded using neuron activities and stored dictionary elements. Different dictionary sets can be trained and stored in the same system, depending on the nature of the input signals. Using the sparse coding algorithm, we also perform natural image processing based on a learned dictionary.

Memristors are two-terminal devices whose resistance values depend on an internal state variable and can be modulated by the history of external stimulation[1–4]. Unlike conventional charge-based electronic devices, a memristor's state is determined by the internal ion (either cation or anion) configuration, and the redistribution of oxygen ions or metal cations inside the device modulates the local resistivity and overall device resistance[2–4]. Memristors have been extensively studied for both digital memory and analog logic circuit applications[3–7]. At the device level, memristors have been shown to be able to emulate synaptic functions by storing the analog synaptic weights and implementing synaptic learning rules[8–12]. When constructed into a crossbar form, memristor networks offer the desired density and connectivity that are required for hardware implementation of neuromorphic computing systems[13–15]. Recently, memristor arrays and phase-change memory devices have been used as artificial neural networks to perform pattern classification tasks[16–18]. Other studies have shown memristors can be used in recurrent artificial neural networks for applications such as analog-to-digital convertors[19].

Memristor-based architectures have also been proposed and analysed for tasks such as sparse coding and dictionary learning[20,21]. The ability to sparsely encode data is believed to be a key mechanism by which biological neural systems can efficiently process large amounts of complex sensory data[22–24] and can enable the implementation of efficient bio-inspired neuromorphic systems for data representation and analysis[25–28]. In this Article, we experimentally demonstrate the implementation of a sparse coding algorithm in a memristor crossbar, and show that this network can be used to perform applications such as natural image analysis using learned dictionaries.

Memristor crossbar array and system set-up

The hardware system used in our study is based on a 32 × 32 crossbar array of WO_x-based analog memristors, formed at each intersection in the crossbar (Fig. 1a). The devices were fabricated using electron-beam lithography following previously developed procedures[29]; a completed array is shown in Fig. 1b (see Methods). After fabrication, the array was wire-bonded and integrated onto a custom-built testing board (Fig. 1b, lower inset and Supplementary Fig. 1), enabling random access to single or multiple memristors simultaneously, for sending and retrieving signals from the array. The memristors can be programmed into different conductance states and can be used to modulate signals in either the forward (the read voltage is applied to the rows and current is measured at each column) or backward (the read voltage is applied to the columns and current is measured at each row) directions.

The original input, such as an image, is fed to the rows of the memristor crossbar, and the columns of the crossbar are connected to output neurons. The memristor network performs critical pattern matching and neuron inhibition operations to obtain a sparse, optimal representation of the input. Once the memristor network stabilizes, the re-constructed image can be obtained based on the (sparse) output neuron activities and the features stored in the crossbar array[30]. A fundamental requirement of sparse coding is the ability to exert inhibition among neurons to re-construct the input using an optimized set of features (out of many possible solutions). In our approach, lateral inhibition is achieved using iterations of forward and backward passes in the same network in discrete time domain, without having to physically implement inhibitory synaptic connections between the output neurons.

To verify the operation of the memristor array, a 32 × 32 greyscale image (a chequerboard pattern with 2 × 2 patch size) was written and read out from the system (Fig. 1c). A single programming pulse was used to program each device without a read-verify procedure, demonstrating the system's capability to program and store analog weights in the memristor array. Details on the programming procedure and more examples of patterns stored in the same array are provided in Supplementary Figs 2 and 3.

Mapping sparse coding onto memristor network

Sparse representation reduces the complexity of the input signals and enables more efficient processing and storage, as well as improved feature extraction and pattern recognition functions[25,27]. Given a signal x, which may be a vector (for example, representing the pixel values in an image patch), and a dictionary of features D, the goal of sparse coding is to represent x as a linear combination of features from D using a sparse set of coefficients a, while minimizing the number of features used. A schematic of sparse coding is shown in Fig. 1d, where an input (for example, the image of a clock) is formed by a few features selected from a large dictionary[24,27]. The

Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109, USA. †These authors contributed equally to this work. *e-mail: wluee@eecs.umich.edu

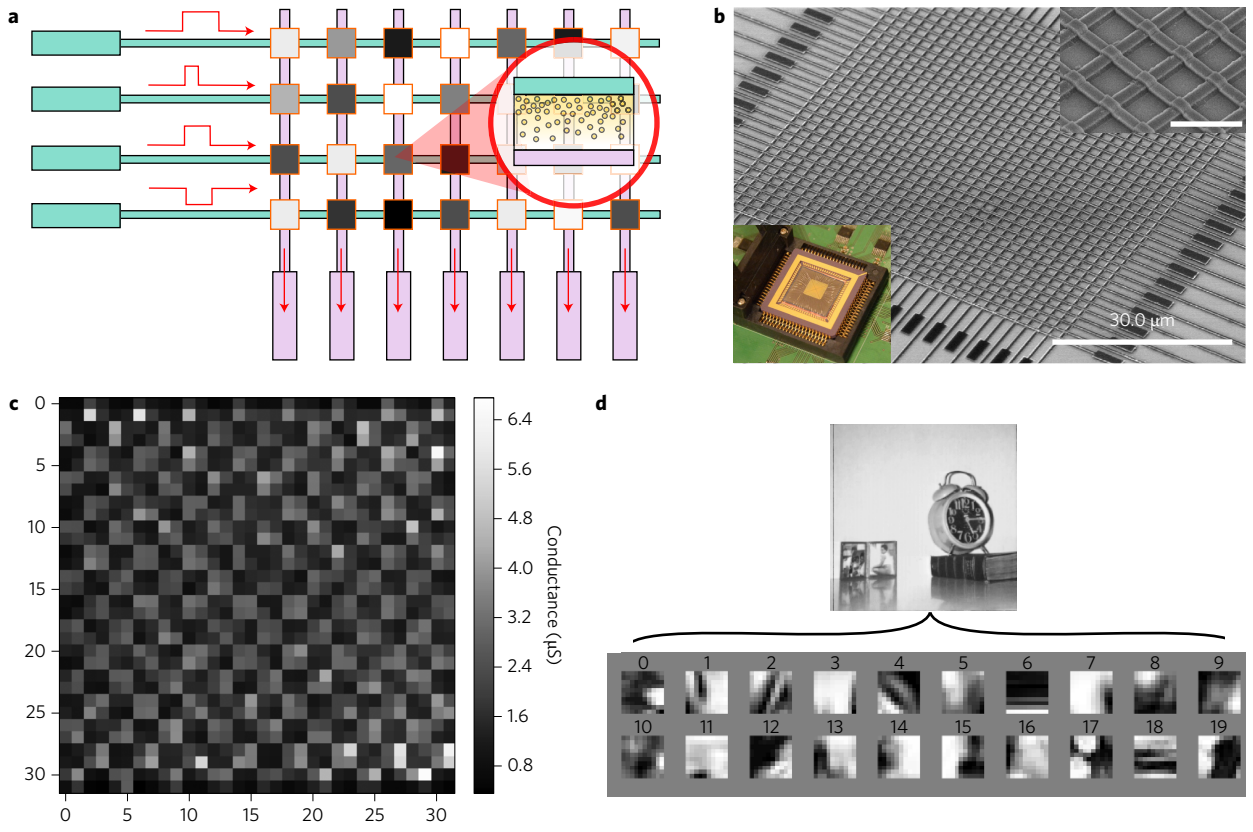

Figure 1 | Memristor crossbar array-based computing hardware system. a, Schematic of a memristor crossbar-based computing system, showing the input neurons (green), the memristor crossbar array and the leaky integrating output neurons (pink). A memristor is formed at each crosspoint, and can be programmed to different conductance states (represented in greyscale) by controlling the internal ion redistribution (inset). **b**, Scanning electron micrograph (SEM) image of a fabricated memristor array used in this study. Upper right inset: magnified SEM image of the crossbar. Scale bar, 3 μm. Lower left inset: memristor chip integrated on the test board after wire-bonding. **c**, A 32 × 32 chequerboard pattern (with 2 × 2 patch size) programmed into the memristor array and subsequently read back using the hardware system shown in **b**. Results from a higher-density array are provided in Supplementary Figs 4 and 5. **d**, Schematic of the sparse coding concept, where an input (for example, the image of a clock) can be decomposed into and represented with a minimal number of dictionary elements.

objective of sparse coding can be summarized mathematically as minimizing an energy function, defined as

$$\min_a(|x - Da^T|_2 + \lambda|a|_0) \quad (1)$$

where $|\cdot|_2$ and $|\cdot|_0$ are the L^2- and the L^0-norm, respectively. Here, the first term measures the reconstruction error, which is the difference between the original signal x and the sparse representation Da^T, while the second term measures the sparsity, which is reflected by the number of active elements in a used to reconstruct the input. Unlike many compression algorithms that focus on reconstruction error only, sparse coding algorithms reduce the complexity by assuming that real signals lie in only a few dimensions (of a high-dimensional space) and attempt to find an optimal representation that also reduces dimensionality. As a result, sparse coding not only enables more efficient representation of the data, but may also be more likely to identify the 'hidden' constituent features of the input and thus can lead to improved data analyses such as pattern recognition[24,25,27].

Several sparse coding algorithms have been developed[26], and this work focuses on the 'locally competitive algorithm'[30] for its advantages in encoding spatiotemporal signals, biological plausibility and compatibility with the crossbar architecture. In this approach, the membrane potential of an output neuron is determined by the input, a leakage term, and an inhibition term that helps achieve sparsity by preventing multiple neurons with similar receptive fields from firing simultaneously. Mathematically, it can be shown that lateral neuron inhibition can be achieved through an iterative approach by removing the reconstructed signal from the input to the network (see equations (2) and (3) in the Methods).

We experimentally implemented the sparse coding algorithm in the memristor array-based artificial neural network. Memristor crossbars are particularly suitable for implementing neuromorphic algorithms, because the vector-matrix multiplication operations can be performed through a single read operation in the memristor array[14,17]. In this approach, the dictionary D is directly mapped element-wise onto the memristor crossbar with each memristor at row i and column j storing the corresponding synaptic weight element D_{ij}. The input vector x (for example, pixel intensities of an input image) is implemented with read pulses with a fixed amplitude and variable width proportional to the input data value. As a result, the total charge Q_{ij} passed by a memristor at crosspoint (i,j) is linearly proportional to the product of the input data x_i and the conductance D_{ij} of the memristor, $Q_{ij} = x_iD_{ij}$, and the charge passed by all memristors sharing column j is summed via Kirchhoff's current law (Fig. 1a) $Q_j = \sum_i x_iD_{ij} = x^TD_j$. In other words, the total charge accumulated at neuron j is proportional to the dot product of input x with the neuron's receptive field D_j. Because the dot product of vectors measures how close the input vector is matched with the stored vector, the ability to implement this operation in a single read process allows the memristor network to conveniently and efficiently perform this important

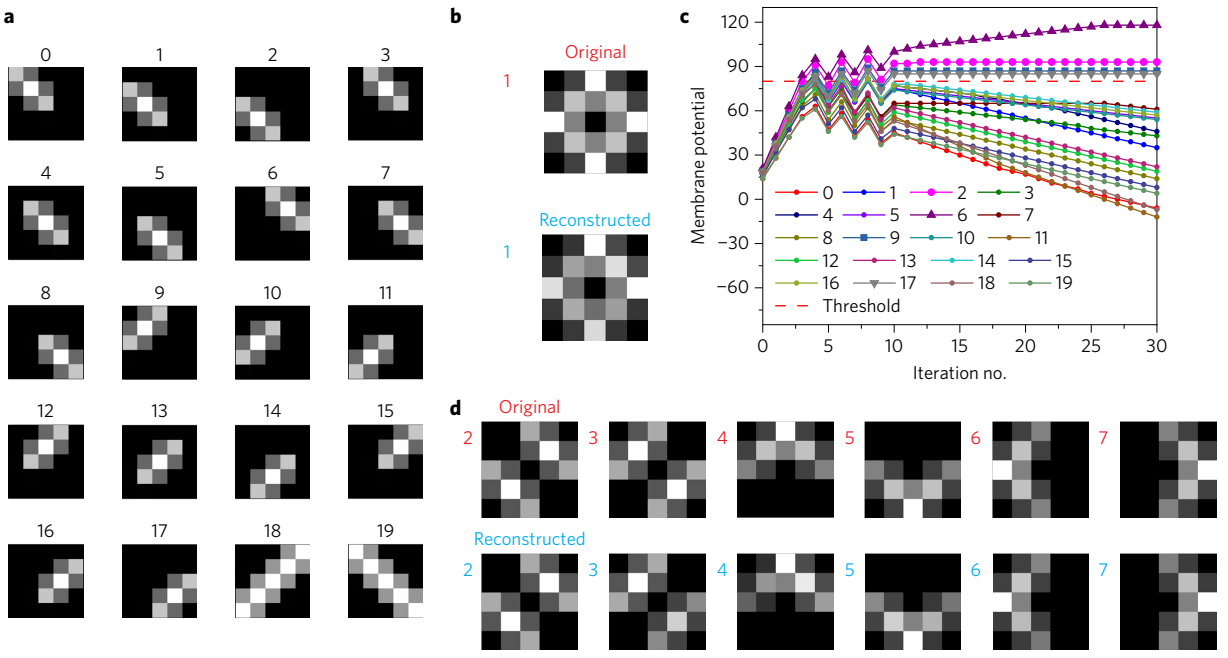

Figure 2 | Experimental demonstration of sparse coding using memristor network. a, Dictionary elements programmed into the memristor crossbar array. Each dictionary element is stored in a single column and connected to an output neuron. The different greyscales represent four different levels. The neuron number is listed above each element. **b**, The original image to be encoded and the reconstructed image after the memristor network settles. **c**, Membrane potentials of the neurons as a function of iteration number during locally competitive algorithm (LCA) analysis. The red dashed horizontal line marks the threshold parameter λ. **d**, Additional examples of input images and reconstructed images. The same threshold, $\lambda = 80$, is used in all experiments in **b**–**d**.

pattern-matching task. This term ($x^T D_j$) is then added to the neuron's membrane potential. If the membrane potential is above threshold λ following equation (2b), the neuron is active for the next phase.

In the second phase, the input image is reconstructed using the currently active neurons and compared with the original input. This is accomplished by performing a 'backward read' (Supplementary Fig. 6): variable-width read pulses, proportional to the neurons' activities a_j, are applied on the columns while the charge is collected on each row i to obtain $Q_i = \sum_j D_{ij} a_j = D_i a^T$. This backward read has the effect of performing a weighted sum of the receptive fields of the active neurons directly through the transpose of the weight matrix, and the total integrated charge on the rows is proportional to the intermediate reconstructed signal $\hat{x} = Da^T$ in vector form. The difference of x and $\hat{x}$, referred to as the residual, is used as the new input to the array to obtain an updated membrane potential based on equation (3). The forward and backward processes are repeated, alternately updating the neuron activities and then the residual. Experimentally, after collecting charges from the memristor array in each step, the neuron activities and membrane potentials are updated by a field-programmable gate array (FPGA) board in the measurement set-up. After the network has stabilized, a sparse representation of the input, represented by the final output activity vector a, is obtained. By performing these forward and backward passes in the same memristor network in discrete time domain, we can effectively achieve the lateral inhibition required by the sparse coding algorithm without having to implement physical inhibitory synaptic connections between all the output neurons.

Sparse coding of simple inputs

Figure 2 shows an example of encoding an image composed of diagonally oriented stripe features using the algorithm given above. The dictionary, shown in Fig. 2a, contains 20 stripe features, with each feature consisting of 25 weights. In this experiment, a 25×20 sub-array from the 32×32 memristor array was used. The 20 features were written into the 20 columns (with each weight represented as a memristor conductance) and the inputs were fed into the 25 rows. An image consisting of a combination of four features, shown in Fig. 2b, was used as a test input to the system. A total of 30 forward–backward iterations, as described already, were performed to stabilize the sparse-coding network, and the final reconstructed signal is shown in Fig. 2b. The input image was correctly reconstructed using neurons 2, 6, 9 and 17, corresponding to the native features of the input, weighted by their activities. Additionally, the experimental set-up allows us to study the network dynamics during the sparse-coding analysis, as shown in Fig. 2c, which plots the membrane potential values for all 20 neurons during the iterations. For the first two iterations, all neurons are charging (at different rates depending on how well the input is matched with the stored receptive fields), and none is above threshold. After the fourth iteration, the membrane potentials of 11 neurons (numbers 1, 2, 4, 5, 6, 9, 10, 13, 14, 16 and 17) have exceeded the threshold. Of these 11 neurons, the receptive fields of neurons 2, 6, 9 and 17 match the features in the input, and those of neurons 1, 4, 5, 10, 13, 14 and 16 are not perfect matches but still overlap enough with the input image to allow these neurons to be charged at reasonable rates. In the subsequent backward read, all the active neurons contribute their receptive fields to the reconstruction and result in a reduced, or even negative residual input in the next forward pass following equation (3). As a result, the charging rates and the neurons' membrane potentials evolve accordingly. Over the next few iterations, the lateral inhibition between neurons eventually drives the membrane potentials of neurons 1, 4, 5, 10, 13, 14 and 16 below the threshold in the 10th iteration and keeps these neurons inactive in subsequent iterations. The inactive neurons' membrane potentials continue to decay due to the leakage term, but because they are below threshold, their values have no impact on the final sparse code. In the end, a correct and sparse representation of the input is obtained in Fig. 2b based on the active neurons 2, 6, 9 and 17 after the network stabilizes. This experiment demonstrates an important feature of the sparse-coding algorithm: lateral neuron

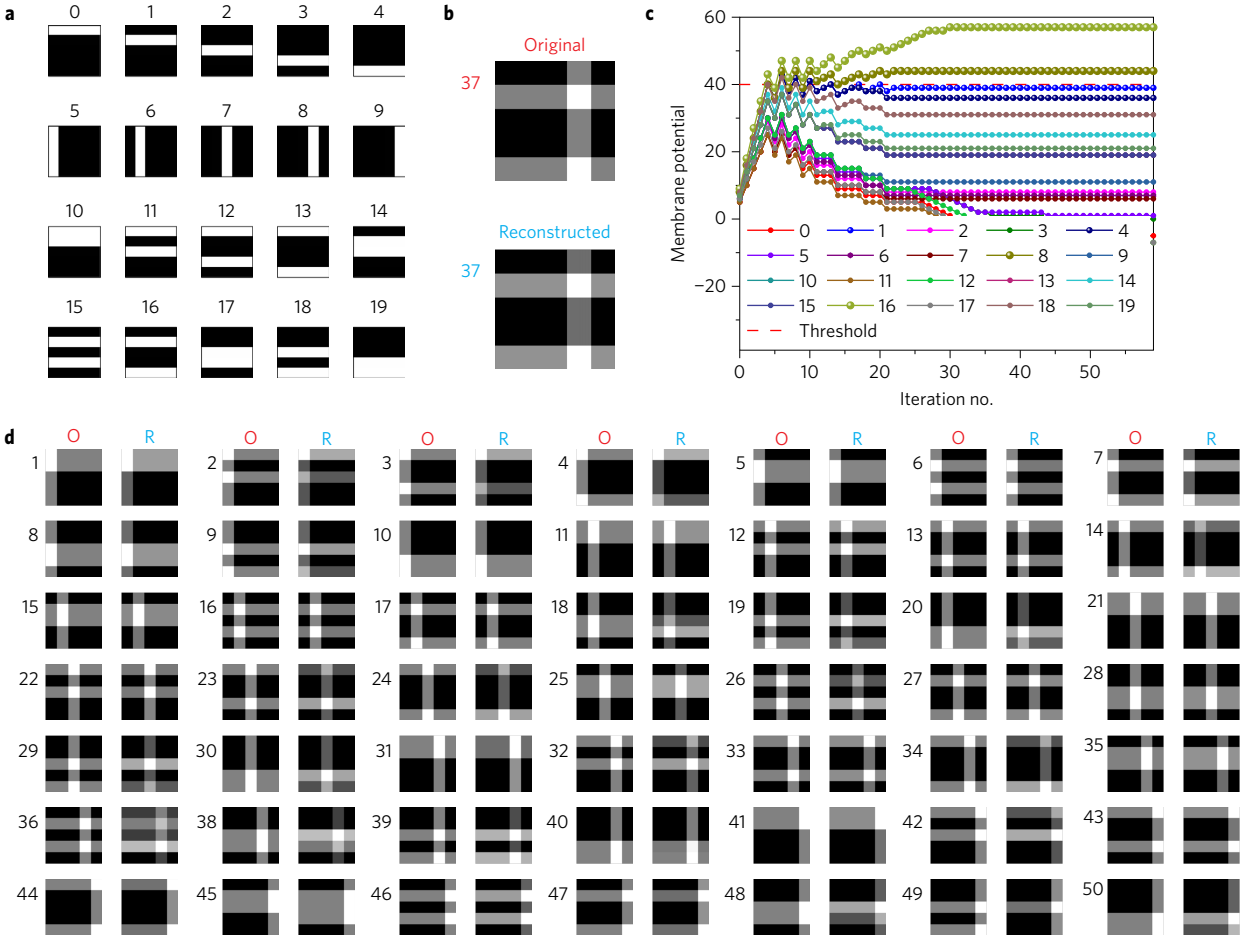

Figure 3 | Sparse coding of different inputs using a more overcomplete dictionary. a, Dictionary elements based on horizontal and vertical bars programmed into the memristor crossbar array. **b**, The original image to be encoded, and the reconstructed image after the memristor network settles. **c**, Membrane potentials of the neurons as a function of iteration number during LCA analysis. The red horizontal line marks threshold parameter λ. **d**, Additional examples of input images and reconstructed images. The same threshold, $\lambda = 40$, is used in all experiments in **b**–**d**.

inhibition drives the system to identify the native features of the input. Non-idealities in the memristor network may temporarily lead to incorrect behaviour (as in the case of the fourth iteration or the eighth iteration when neurons 4, 14 and 16 exceed the threshold), but inhibition can effectively address these errors and allow correct features to be identified. Additional sparse-coding examples of inputs based on stripe features using the memristor crossbar are provided in Fig. 2d.

The reprogrammability of memristors allows the dictionary set to be readily adapted to the type of signal to be encoded, so the same memristor hardware system can process different types of inputs using a single general approach. To demonstrate this point, we reprogrammed a new dictionary composed of horizontally and vertically oriented bars (Fig. 3a) into the same array used in Fig. 2. By using this new dictionary, images consisting of bar patterns can be efficiently analysed using the same algorithm. Additionally, in the examples shown in Fig. 2, the dictionary is minimally over-complete (with inputs restricted to be combinations of diagonal stripe features and corresponds to an input dimensionality of 18, determined from the linear span of the features). By using the bar patterns in Fig. 3a and restricting the input images to combinations of horizontal and vertical bars, the input dimensionality is reduced to 9. With a total of 20 stored dictionary elements, the system now achieves greater than 2× over-completeness[30], which allows it to better highlight the capability of sparse coding to find an optimal solution out of several possible solutions.

An example (input pattern no. 37) using this over-complete dictionary is shown in Fig. 3b,c. The network not only correctly reconstructed the input image, but, as expected, it picked the more efficient solution—a solution based on neurons 8 and 16, over another solution based on neurons 1, 4 and 8. As can be seen from Fig. 3c, in the first five iterations, all neurons are charging and the membrane potentials of neurons 1, 4, 8 and 16 first cross the threshold at the sixth iteration. Even though the receptive fields of all four neurons (1, 4, 8 and 16) are correct features in the input, neurons 8 and 16 (consisting of two bars) represent a more sparse representation. As a result, inhibition implemented in the system eventually suppresses the membrane potentials of neurons 1 and 4 to be below the threshold after the 11th iteration and keeps them below the threshold after the network stabilizes. The activities of these two neurons are precisely 0 (equation (2b)), and an optimal solution based only on neurons 8 and 16 is obtained, compared to other possible, but less-sparse solutions. These features of the network dynamics have also been confirmed through detailed simulations of the memristor crossbar-based sparse-coding system (Supplementary Fig. 10).

Our experimental set-up allows us to directly study the role of the sparsity coefficient λ on network dynamics and sparse-coding outcome. We verified that, when changing λ, the network will naturally adapt, and often the same set of active neurons whose receptive fields optimally represent features in the input will prevail in the end (Supplementary Fig. 11).

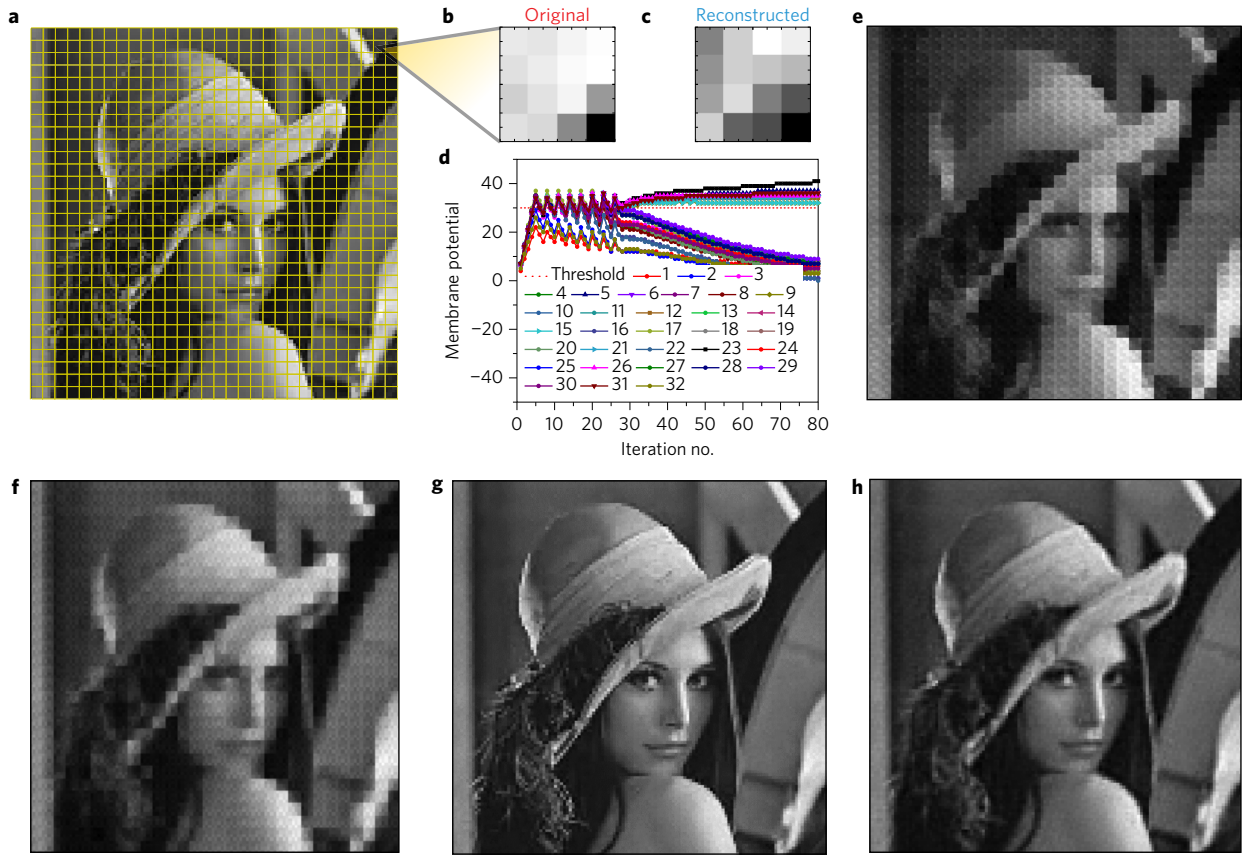

Figure 4 | Natural image processing using the memristor crossbar. a, Original 120 × 120 image. The image is divided into non-overlapping 4 × 4 patches for processing. **b**, A 4 × 4 patch from the original image. **c**, The experimentally reconstructed patch from the 16 × 32 memristor crossbar using the LCA algorithm and an offline-learned dictionary based on WTA. **d**, Membrane potentials of the neurons as a function of iteration number during LCA analysis. The red horizontal line marks the threshold parameter λ. Neurons 5, 6, 9, 12, 14, 15, 21, 23, 26 and 31 stay active after 30 iterations while the other neurons are suppressed below the threshold. **e**, Experimentally reconstructed image based on the reconstructed patches. **f**, Simulated reconstructed image using an offline trained dictionary based on WTA. **g**, Simulated reconstructed image using larger patches and an ideal dictionary learned via sparse coding and gradient descent. **h**, Simulated reconstructed image by considering realistic device variabilities during online learning. 8 × 8 patches were used during training and image reconstructions in **g** and **h**.

Additional examples showing the network dynamics in picking a sparse representation for different input patterns are provided in Supplementary Figs 7–9. In total, all 50 patterns consisting of two horizontal bars and one vertical bar were tested using the experimental set-up (Fig. 3d), with a success rate of 94% (measured by the network's ability to correctly identify the sparse solutions), despite variabilities inherent in the memristor devices.

Sparse coding of natural images

Finally, we applied the prototype memristor network to experimentally process natural images using the sparse-coding algorithm. In this study, a 16 × 32 sub-array from the 32 × 32 memristor array was used, corresponding to a 2× over-complete dictionary with 16 inputs and 32 output neurons and dictionary elements. The dictionary elements were learned offline using 4 × 4 patches randomly sampled from a training set consisting of nine natural images (with sizes of 128 × 128 pixels), using a realistic memristor model and an algorithm based on the 'winner-take-all' (WTA) approach and Oja's learning rule[14]. After training, the obtained dictionary elements were programmed into the physical 16 × 32 crossbar array (Supplementary Figs 12–14).

Using the trained dictionary, we successfully performed reconstruction of greyscale images experimentally using the 16 × 32 memristor crossbar. During the process, the input image (for example, Fig. 4a) was divided into 4 × 4 patches and each patch was experimentally processed using the memristor crossbar based on the sparse-coding algorithm (Fig. 4b,c). Once the memristor network stabilized (typically after 80 forward–backward iterations, Fig. 4d), the patch was reconstructed using the neuron activities and the corresponding receptive fields, as shown in Fig. 4c. The complete image was then composed from the individual patches, shown in Fig. 4e.

To verify the experimental results, we performed detailed simulations of the memristor crossbar network. Effects of device variations were carefully considered during the initialization of the matrix and during the weight updates (Supplementary Figs 15 and 16). In the case of Fig. 4e, non-idealities during the dictionary storage were simulated based on the weight update equation from our device model[29]. Image reconstructions were then analysed using the simulated memristor network, following the same procedure as the experimental processes. The simulation results consistently reproduced the experimental results (Fig. 4f) for this image-processing task.

We note the current experimental results are limited by the network size, so only 4 × 4 patches are processed. Additionally, sparse-coding analysis works better if the dictionary is also learned via sparse coding instead of simple WTA. Indeed, analysis based on larger receptive fields (for example, 8 × 8, corresponding to a 64 × 128 memristor array with 2× over-completeness) and using a sparse-coding trained dictionary produces excellent

reconstruction results, as shown in Fig. 4g. Detailed simulations further show that high-quality image reconstruction can still be obtained even in the presence of realistic device variations (Fig. 4h) if the dictionary is learned online using the memristor crossbar. This effect can be explained from the fact that the learning algorithm is self-adaptive and adjusts to device variabilities during the training stage. As a result, online learning can effectively handle device variations and is particularly suitable for emerging devices such as memristor-based systems where large device variations are expected.

Conclusions

In this work, we have successfully demonstrated a sparse-coding hardware system in a memristor crossbar architecture. This approach, based on pattern matching and neuron lateral inhibition, is an important milestone in the development of large-scale, low-power neuromorphic computing systems. The use of a crossbar architecture allows matrix operations, including vector-matrix multiplication and matrix transpose operations, to be performed directly and efficiently in the analog domain without the need to read each stored weight. Image reconstruction was also demonstrated using the memristor system, and online dictionary learning was shown to be feasible even in the presence of realistic device variations. Future studies, aimed at integrating (ideally larger) memristor crossbar arrays with complementary metal–oxide–semiconductor (CMOS) circuitry that can perform the necessary periphery functions on chip, should provide significant speed improvements and enable online learning implementation. Online learning was found to be able to efficiently tolerate device variations, even for simple algorithms using WTA (Supplementary Figs 17 and 18). Image pre-processing techniques such as whitening can also be implemented to further improve the network's performance (Supplementary Figs 19–21). Our benchmarking analysis against an efficient digital approach shows that an integrated memristor system based on devices similar to the prototype system can already offer significant energy advantages when performing data-intensive tasks such as real-time video analysis (Supplementary Figs 22–25 and Supplementary Tables 1 and 2). Continued optimization of the devices and the architecture can lead to future computing systems that can help eliminate the 'von Neumann bottleneck' that is present in conventional computing designs, and produce efficient computing hardware with low energy consumption, small footprint and high throughput.

Methods

Methods and any associated references are available in the online version of the paper.

Received 16 September 2016; accepted 28 March 2017; published online 22 May 2017

References

1. Chua, L. O. 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. Waser, R. & Aono, M. Nanoionics-based resistive switching memories. *Nat. Mater.* **6,** 833–840 (2007).
4. Yang, Y., Chang, T. & Lu, W. in *Memristors and Memristive Systems* 195–221 (Springer, 2014).
5. 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).
6. Xia, Q. *et al.* Memristor–CMOS hybrid integrated circuits for reconfigurable logic. *Nano Lett.* **9,** 3640–3645 (2009).
7. Pershin, Y. V. & Di Ventra, M. Practical approach to programmable analog circuits with memristors. *IEEE Trans. Circuits Syst. I Regul. Pap.* **57,** 1857–1864 (2010).
8. Jo, S. H. *et al.* Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* **10,** 1297–1301 (2010).
9. Pershin, Y. V. & Di Ventra, M. Experimental demonstration of associative memory with memristive neural networks. *Neural Networks* **23,** 881–886 (2010).
10. Du, C., Ma, W., Chang, T., Sheridan, P. & Lu, W. D. Biorealistic implementation of synaptic functions with oxide memristors through internal ionic dynamics. *Adv. Funct. Mater.* **25,** 4290–4299 (2015).
11. Kuzum, D., Jeyasingh, R. G. D., Lee, B. & Wong, H.-S. P. Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Lett.* **12,** 2179–2186 (2012).
12. Ohno, T. *et al.* Short-term plasticity and long-term potentiation mimicked in single inorganic synapses. *Nat. Mater.* **10,** 591–595 (2011).
13. Yang, J. J., Strukov, D. B. & Stewart, D. R. Memristive devices for computing. *Nat. Nanotech.* **8,** 13–24 (2013).
14. Sheridan, P. M., Du, C. & Lu, W. D. Feature extraction using memristor networks. *IEEE Trans. Neural Networks Learn. Syst.* **27,** 2327–2336 (2016).
15. Legenstein, R. Computer science: nanoscale connections for brain-like circuits. *Nature* **521,** 37–38 (2015).
16. 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).
17. Prezioso, M. *et al.* Training and operation of an integrated neuromorphic network based on metal–oxide memristors. *Nature* **521,** 61–64 (2015).
18. Burr, G. W. *et al.* in *2014 IEEE International Electron Devices Meeting* 29.5.1–29.5.4 (IEEE, 2014).
19. Guo, X. *et al.* Modeling and experimental demonstration of a Hopfield network analog-to-digital converter with hybrid CMOS/memristor circuits. *Front. Neurosci.* **9,** 488 (2015).
20. Agarwal, S. *et al.* Energy scaling advantages of resistive memory crossbar based computation and its application to sparse coding. *Front. Neurosci.* **9,** 484 (2016).
21. Kadetotad, D. *et al.* in *Proceedings of the Biomedical Circuits and Systems Conference* (*BioCAS*) 536–539 (IEEE, 2014).
22. Földiák, P. & Young, M. P. Sparse coding in the primate cortex. *Handb. Brain Theory Neural Netw.* **1,** 1064–1068 (1995).
23. Vinje, W. E. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science.* **287,** 1273–1276 (2000).
24. 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).
25. Wright, J. *et al.* Sparse representation for computer vision and pattern recognition. *Proc. IEEE* **98,** 1031–1044 (2010).
26. Lee, H., Battle, A., Raina, R. & Ng, A. Y. in *Proceedings of the 19th International Conference on Neural Information Processing Systems* 801–808 (MIT Press, 2006).
27. Olshausen, B. A. & Field, D. J. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Res.* **37,** 3311–3325 (1997).
28. Lee, D. D. & Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature* **401,** 788–791 (1999).
29. Chang, T. *et al.* Synaptic behaviors and modeling of a metal oxide memristive device. *Appl. Phys. A* **102,** 857–863 (2011).
30. 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).
31. Hubel, D. H. & Wiesel, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* **160,** 106–154 (1962).

Acknowledgements

The authors thank G. Kenyon, P. Knag, T. Chen, Z. Zhang, Y. Jeong and M. Zidan for discussions and help. This work was support by the Defense Advanced Research Projects Agency (DARPA) through award no. HR0011-13-2-0015, by the Air Force Office of Scientific Research (AFOSR) through MURI grant FA9550-12-1-0038 and by the National Science Foundation (NSF) through grant CCF-1617315.

Author contributions

P.M.S. and W.D.L. conceived and directed the project. P.M.S., F.C., W.M., Z.Z. and W.D.L analysed the experimental data. P.M.S. and F.C. constructed the circuitry and performed the network measurements. C.D. and W.M. prepared the memristor arrays. P.M.S., F.C. and W.D.L. constructed the research frame. All authors discussed the results and implications and commented on the manuscript at all stages.

Additional information

Supplementary information is available in the online version of the paper. Reprints and permissions information is available online at www.nature.com/reprints. Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations. Correspondence and requests for materials should be addressed to W.D.L.

Competing financial interests

The authors declare no competing financial interests.

Methods

Locally competitive algorithm. The locally competitive algorithm (LCA) solves the minimization problem (equation (1)) using a network of leaky-integrator neurons and connection weights. In this implementation, x is an m-element column vector, with each element corresponding to an input element (for example, the intensity of a pixel in an image patch). D is an $m \times n$ matrix, where each column of D represents an m-element feature vector (that is, a dictionary element) and is connected to a leaky-integrator output neuron (Fig. 1a). a is an n-element row vector representing the neuron activity coefficients, where the ith element of a corresponds to the activity of the ith 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^T, that is, linear combination of the neuron activities and corresponding neurons' feature vectors. In a sparse representation, only a few elements in a are non-zero, and the other neurons' activities are suppressed to be precisely zero.

The neuron dynamics during LCA analysis can be summarized by

$$\frac{du}{dt} = \frac{1}{\tau}(-u + x^T D - a(D^T D - I_n)) \tag{2a}$$

$$a_i = \begin{cases} u_i & \text{if } u_i > \lambda \\ 0 & \text{otherwise} \end{cases} \tag{2b}$$

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

During LCA analysis, each neuron i integrates its input $x^T D$, leakage $-u$ and inhibition $a(D^T D - I_n)$ terms and updates its membrane potential u_i (equation (2a)). If and only if u_i exceeds a threshold (set by parameter λ), neuron i will produce an output $a_i = u_i$, otherwise the neuron's activity a_i is kept at 0 (equation (2b)). Specifically, the input to neuron i originates from the input signal x scaled by weights D_{ji} connected to the neuron (second term in equation (2a)). In this regard, the collection of synaptic weights D_{ji} associated with neuron i is also referred to as the receptive field of neuron i, analogous to the receptive fields of biological neurons in the visual cortex[24,31]. A key feature of the LCA is that the neurons also receive inhibition from other active neurons (the last term in equation (2a)), an important observation in biological neural systems[24]. The LCA incorporates this competitive effect through the inhibition term, which is proportional to the similarity of the neurons' receptive fields[30] (measured by $D^T D$ in equation (2a)). By doing so, it prevents multiple neurons from representing the same input pattern and allows the network to dynamically evolve to find an optimal output. Note that when a neuron becomes active, all other neurons' membrane potentials will be updated through the inhibition term (to different degrees depending on how similar the neurons' receptive fields are). As a result, an initially active neuron may become suppressed and a more optimal representation that better matches the input may be found. In the end, the network evolves to a steady state where the energy function (equation (1)) is minimized and an optimized sparse representation (out of many possible solutions) of the input data is obtained, from a combination of the stored features and the activities of the (sparse) active neurons.

Note, however, that implementing the inhibition effect $D^T D$ can be very computationally intensive. On the other hand, the original equation (2a) can be rewritten as

$$\frac{du}{dt} = \frac{1}{\tau}(-u + (x - \hat{x})^T D + a) \tag{3}$$

where $\hat{x} = Da^T$ is the signal estimation (that is, the reconstructed signal). Equation (3) shows that the inhibition term between neurons can be reinterpreted as a neuron removing its feature from the input when it becomes active, thus suppressing the activity of other neurons with similar features. By doing so, the matrix–matrix multiplication operation $D^T D$ in equation (2a) is reduced to two sequential vector-matrix multiplication operations (one used to calculate $\hat{x} = Da^T$ and the other used to calculate the contribution from the updated input $(x - \hat{x})^T D$, which we show can be efficiently implemented in memristor crossbars in a discrete time domain without physical inhibitory synaptic connections between all the output neurons.

WO_x memristor array fabrication. To form the crossbar array of WO_x devices, 60 nm of W was first sputter-deposited on a Si carrier wafer with a 100 nm thermally grown oxide. The bottom electrodes (BEs, 500 nm width) were patterned by electron-beam lithography and reactive ion etching (RIE) using Ni as a hard mask. Afterwards, the Ni hard mask was removed by wet etching. SiO_2 (250 nm) was then deposited by plasma-enhanced chemical vapour deposition followed by etch back to form a spacer structure along the sidewalls of the BEs. The spacer structure allows better step coverage of the top electrodes (TEs) at the crosspoints. The resistive switching WO_x layer was formed by rapid thermal annealing of the exposed W electrode surface with oxygen gas at 425 °C for 60 s. Afterwards, the TEs (Pd (90 nm)/Au (50 nm)) were patterned by electron-beam lithography, electron-beam evaporation and liftoff processes. Another RIE process was used to remove excess WO_x between the TEs and to expose the BEs for electrical contacts. Finally, photolithography, electron-beam evaporation and liftoff processes were performed to deposit 150 nm of Au as wire-bonding pads. The completed WO_x memristor crossbar chip was then wire-bonded to an 80-pin chip carrier and integrated on the test board.

Memristor array test board and software set-up. A custom board was designed to test memristor arrays for neuromorphic computing applications including the sparse-coding tasks. The board can apply timed voltage pulses and measure currents at both row and column terminals, with an integrated controller system to perform these tasks in an automated manner. It can measure arrays of up to 32 rows and 32 columns. There are four digital-to-analog converters (DACs) capable of independently producing voltage pulses with amplitude ranges from −5 to 5 V. Typically, two voltage sources are connected to the rows through the matrix switches and two to the columns. The matrix switches are connected in such a way as to perform 2×32 routing, with a 32-bit binary word used to configure which of the rows (columns) is connected to DAC0 (DAC2) while the remaining rows (columns) are connected to DAC1 (DAC3). The board can perform an array of tests to characterize memristor devices including d.c. sweeps, pulse measurements and, importantly, random read and write procedures for memristor crossbar arrays. A virtual ground with negative feedback is used to convert the current flowing to an output electrode to a voltage that can be read by analog-to-digital converters. A variable resistor in the path is used to control the amplification of the current signal. A multiplexer is included in the signal path to allow connection of either the virtual ground or the DAC. All control and data signals are passed through logic-level shifters so that the signals can be communicated between the board (at 5 V level) and off-board (at 3.3 V) (Supplementary Fig. 1).

The algorithm is programmed onto the board with a mixture of Python and C code. The Python functions direct the pre-processing and compilation of C routines and download the compiled binaries to the board. The generated data are received using Python functions and displayed with the Matplotlib library. Algorithm execution is directed by the Python code to reduce the processing load on the soft microcontroller, while board control C routines benefit the real-time execution of the microcontroller.

Low-level board tasks such as setting the output voltages and configuring the matrix switches were written exclusively in C using memory-mapped control registers, while higher-level functions such as reading an array or programming a pattern were written in a mixture of C and Python. C code templates were developed to execute generic tasks. The Python code acted as a preprocessor for these templates, filling in parameters such as hexadecimal values corresponding to a voltage or 32-bit configurations for the matrices. A soft microprocessor was implemented on the Opal Kelly XEM6010 FPGA on the test board using the AltOR32 OpenRISC architecture. The SConstruct build tools were used to control compilation and linking C codes into binaries, which was performed by the or1knd-toolchain developed for AltOR32. The binaries were then downloaded onto the board and executed.

Data availability. The data that support the findings of this study are available from the corresponding author upon request.