ARTICLES
https://doi.org/10.1038/s42256-019-0089-1
nature machine intelligence

In situ training of feed-forward and recurrent convolutional memristor networks

Zhongrui Wang[1,8], Can Li[1,2,8], Peng Lin[1,8], Mingyi Rao[1], Yongyang Nie[1], Wenhao Song[1], Qinru Qiu[3], Yunning Li[1], Peng Yan[1], John Paul Strachan[2], Ning Ge[2], Nathan McDonald[4], Qing Wu[4], Miao Hu[5], Huaqiang Wu[6], R. Stanley Williams[7], Qiangfei Xia[1*] and J. Joshua Yang[1*]

The explosive growth of machine learning is largely due to the recent advancements in hardware and architecture. The engineering of network structures, taking advantage of the spatial or temporal translational isometry of patterns, naturally leads to bio-inspired, shared-weight structures such as convolutional neural networks, which have markedly reduced the number of free parameters. State-of-the-art microarchitectures commonly rely on weight-sharing techniques, but still suffer from the von Neumann bottleneck of transistor-based platforms. Here, we experimentally demonstrate the in situ training of a five-level convolutional neural network that self-adapts to non-idealities of the one-transistor one-memristor array to classify the MNIST dataset, achieving similar accuracy to the memristor-based multilayer perceptron with a reduction in trainable parameters of ~75% owing to the shared weights. In addition, the memristors encoded both spatial and temporal translational invariance simultaneously in a convolutional long short-term memory network—a memristor-based neural network with intrinsic 3D input processing—which was trained in situ to classify a synthetic MNIST sequence dataset using just 850 weights. These proof-of-principle demonstrations combine the architectural advantages of weight sharing and the area/energy efficiency boost of the memristors, paving the way to future edge artificial intelligence.

Weight sharing is a common feature of state-of-the-art neural networks that considerably reduces the number of free parameters needed to perform feature extractions compared with densely connected networks. Inspired by the fact that the visual cortex neurons of monkeys individually respond to small regions of the visual fields and the neighbouring cells have similar partially overlapping receptive fields[1], weight sharing between locally connected neurons gives rise to convolutional neural networks (CNNs) that not only minimize the pre-processing to handle translation and local distortion of the inputs but also extract and categorize local spatial correlations[2]. CNNs are now the dominant architecture for analysing visual imagery, reflected in the success of VGG[3], GoogLeNet[4] and ResNet[5]. Weight sharing can also exploit both the spatial and temporal translation invariance of patterns, leading to recurrent structures. Convolutional long short-term memory (ConvLSTM), which is capable of learning long-term dependence due to its recurrent architecture, uses convolutional kernels to determine both the input-to-state and state-to-state transitions, exhibiting the capability of intrinsic spatio-temporal feature extraction[6,7]. ConvLSTM provides natural end-to-end trainable building blocks for three-dimensional inputs (such as videos) with applications ranging from precipitation nowcasting[6] to video encoding[8].

Most implementations of shared-weight structures rely on graphics processing units. However, when implemented in such conventional digital hardware, these weight-shared networks suffer from large inference latency and high power consumption. These issues may become less affordable if the computing is performed at the edge in the era of the Internet of Things. Application-specific integrated circuits with optimized multiply-accumulate (MAC) units, such as the tensor processing unit[9], Eyeriss[10] and DaDianNao[11], could potentially boost the area/energy efficiency. However, the von Neumann bottleneck of such architectures and the increasingly cost-ineffective scaling of the transistor limit the ultimate efficiency of those approaches[12–15]. Therefore, fundamental changes to the computing platform and its building blocks are critical to meet the ever-growing demand for computing power.

Memristors are emerging two-terminal electronic devices that show analogue conductance, fast switching, superior scalability, long retention and long endurance[13–22]. In crossbars, memristor arrays naturally parallelize MAC operations where the data are stored[23,24]. Such analogue in-memory computing directly uses intrinsic physics laws to obviate the large energy and time overheads incurred by frequent data shuttling in von Neumann systems[25–31]. This allows memristor crossbar arrays to physically embody the fully connected layers of the networks[21,24,32–40] with a substantially improved area/energy efficiency. However, the practical computing systems are always bounded in terms of computing power and memory. In addition, the typical datasets like images, videos and audios are of spatial or temporal correlations. Given the same amount of hardware (memory) or alternatively the number of trainable parameters (degrees of freedom) of the system, multilayer perceptrons usually show poor functionalities (classification accuracy, for example) compared with the weight-shared networks, which limits their practical applications.

As memristors are most efficiently assembled in crossbars, which differ from the microstructures of the weight-shared artificial neural

[1]Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA. [2]Hewlett Packard Labs, Hewlett Packard Enterprise, Palo Alto, CA, USA. [3]Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA. [4]Information Directorate, Air Force Research Laboratory, Rome, NY, USA. [5]Department of Electrical and Computer Engineering, Binghamton University, Binghamton, NY, USA. [6]Institute of Microelectronics, Tsinghua University, Beijing, China. [7]Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. [8]These authors contributed equally: Zhongrui Wang, Can Li, Peng Lin. *e-mail: qxia@umass.edu; jjyang@umass.edu

networks, it is necessary to remap the high-dimensional trainable parameters of a weight-shared network efficiently to a 2D memristor array. In addition, such mapping should accommodate more stringent requirements for the accuracy of weight representation because the weight-shared architectures are much more susceptible to hardware non-idealities compared with fully connected networks[41]. Contemporary memristors may represent synaptic weights inaccurately with conductance because of their stochastic behaviour, which are associated with ionic transport and can be improved through iterative corrections at the expense of time and energy. This makes the training of weight-shared architectures a demanding task. Memristors have thus only been exploited for fully connected networks[21,24,29,32,34–38], with temporal weight sharing just recently demonstrated in a recurrent structure[42]. Theoretical investigations of how the convolutional layers can be mapped to memristor crossbars[41] and memristor CNNs of binary weights[43] have been conducted, with the former experimentally verified on HfO_x memristors[25,44]. Recently, MAC operations for convolution with 2-bit inputs and 3-bit weights have been demonstrated on megabit binary state resistive switch macros with both 65 nm (ref. [38]) and 55 nm (ref. [39]) CMOS logic process. However, in situ training of memristor-based spatial shared-weight architectures (such as CNNs) are lacking, as is the case for concurrent spatio-temporal weight sharing (ConvLSTM, for example) on memristor arrays.

In situ training stores and updates the weights directly in memristors, and performs computations (for example, forward passes) at the original place where the neural network parameters are stored; this avoids the need to implement a duplicated system in digital computers, such as ex situ training, which substantially enhances the area/energy efficiency of the system by eliminating the processor-memory bottleneck of digital computers. More importantly, in situ training with backpropagation is capable of self-adaptively adjusting the network parameters to minimize the impacts of the inevitable non-idealities of the hardware (such as wire resistance, analogue peripheral asymmetry, non-responsive memristors, conductance drift and variations in the conductance programming) without any prior knowledge of the hardware[45], which is critical to shared-weight networks.

Here we show that in situ training of shared-weight neural networks tolerates the hardware non-idealities of the one-transistor one-memristor (1T1R) array using a simple dense mapping of convolution kernels to the memristor crossbar. We achieved 92.13% accuracy in classifying MNIST handwritten digits using just ~1,000 weights—that is, only one-quarter of the number of a memristor multilayer perceptron with similar performance. In addition, we demonstrate that the advantages of the weight sharing can be further extended by cascading the convolutional kernels in a recurrent ConvLSTM network with intrinsic 3D inputs, which identified the spatial and temporal correlations in both input-to-state and state-to-state transitions. The experimentally demonstrated reduction of trainable parameters to 850 by spatio-temporal memristor weight sharing represents a promising approach for using memristor-based hardware to efficiently implement advanced network topologies for edge applications.

Results

1T1R CNN with spatial weight sharing. Figure 1a shows an optical micrograph of the 1T1R memristor array, which comprises Pt/TaO_x/Ta memristors with analogue programming capability, long data retention and large endurance due to the chemically stable Ta–O phases[46], and its partition that physically implements all trainable parameters of the neural network. The trainable parameters, or weights of different layers, are interfaced with upstream and downstream neurons that are implemented by conventional transistor circuits off-chip (as depicted in Supplementary Fig. 1). In contrast to employing dedicated 1T1R arrays for different neural network layers, the partitioning of a single large array essentially shares the pre-synaptic neurons between different layers of the same network, maximizing the utility of the peripheral circuits and benefitting edge applications.

The zoomed scanning electron micrograph in Fig. 1 shows that 1T1R cells of the same column share common sources of the transistors, connected by the same bit lines. Cells of the same row share common gates of the transistors, wired on the same word lines, and common top electrodes (TE) of the memristors. During the inference stage, all transistors were operated in the deep triode region so the 1T1R array became a pseudo-1R-crossbar capable of performing vector–matrix multiplications directly using Ohm's law and Kirchhoff's current law, negating the need for transferring data back and forth between memories and processors. In the weight-updating phase, the transistors imposed current compliance, which programmed the memristors at an acceptable accuracy with a single-shot blind-update that is faster than applying a train of identical pulses. (see Fig. 1b and Supplementary Fig. 2)

Figure 1c illustrates the computation flow of the CNN during a forward pass. The network was a modified five-level CNN derived from the pioneering LeNet-5 with a reduced weight population to fit the 128×64 1T1R array[2]. As our primary goal is to show the advantage of weight sharing, this simple CNN features a relatively large ratio between the accuracy and the number of weights (see Supplementary Note 1 for the experimental comparison with another LeNet-5-like network). The example input to the network was a resized MNIST handwritten digit image with 8×8 pixels and 8-bit resolution (see the Methods for the generation of the resized MNIST dataset, and Supplementary Note 2 for the downsampling impact on the network performance). Here we collapsed both the input patch for convolution (highlighted in the purple box) and the parameters of a single kernel to column vectors. The different unrolled kernels of the same layer were placed horizontally, forming a parameter matrix that stored all the trainable weights of a single convolutional layer. Such a dense mapping scheme, in contrast to the Toeplitz matrix or sparse mapping scheme, performs convolution by scanning the receptive field over the entire input volume, leading to multiple MAC operations that yield the full volume of the output map, featuring a large MAC-operation/weight-update ratio[24] (the weight updates cannot be fully parallelized on the memristor array, and thus should be performed as infrequently as possible). Alternatively, sparse mapping using duplicate kernels with a step shift could further parallelize the convolution over different input patches. However, the effective throughput per unit area drops rapidly as the data/kernel ratio scales up, in addition to the challenge of duplicating identical kernels and having ideal 0s in the sparse matrix (due to the intrinsic variability of memristors) and increased memristor programming energy[24]. We therefore chose the dense mapping scheme over sparse mapping.

The input was first physically convolved with the 15 3×3 kernels characterized by zero padding and unitary stride on the memristor array. The signed weight was encoded into the conductance difference between one pair or multiple pairs (here we used two pairs to further improve the programming accuracy[37]) of memristors on the same row. The analogue convolution outputs, digitized by 16-bit analogue-to-digital converters (ADCs), were activated by rectified linear units (ReLU) in software element-wise before being sent to the next layer, producing 15 feature maps of size 8×8 (see Methods). These feature maps were again convolved by four 2×2 kernels followed by a downsampling layer that implemented the max pooling function over non-overlapping pooling windows of size 2×2. Finally, the feature maps were flattened as the inputs into a 10-way softmax output layer.

The computational complexity for a convolution stride is usually $O(k_{dim1} \times k_{dim2} \times k_{dim3} \times k_{num})$ where k_{dim1}, k_{dim2}, k_{dim3} and k_{num} are the height, width, depth and population of the kernels, respectively. On the other hand, the complexity becomes $O(1)$ once the weights are physically mapped to the 1T1R array because all MAC operations within the same kernel and across all kernels of the entire layer are performed in a single time step. Alternatively, the first (second) con-

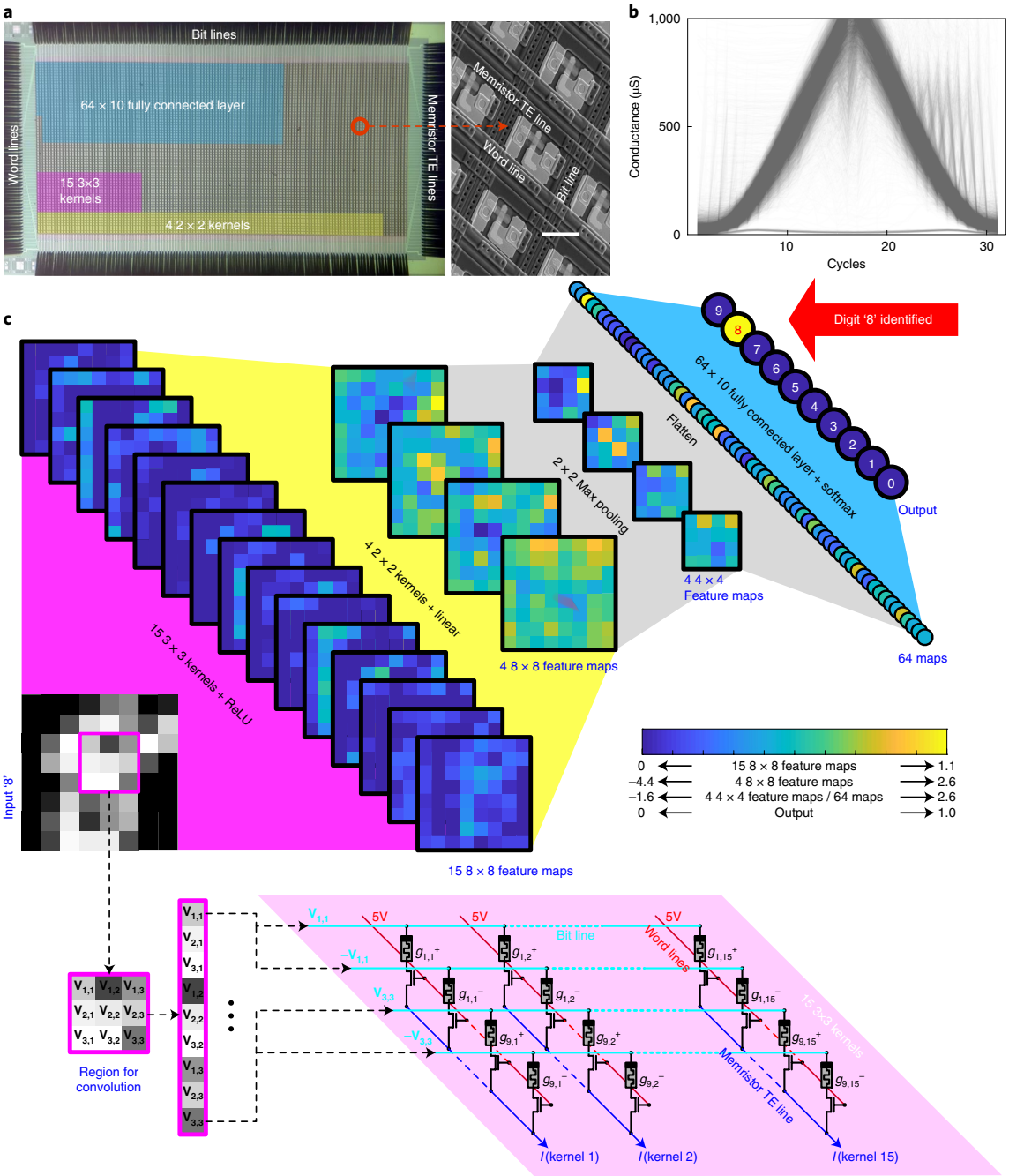

Fig. 1 | 1T1R implementation of the 5-level convolutional neural network (CNN). a, Optical and scanning electron micrographs of the 128 × 64 1T1R array with probe card (black tips on the four edges) landed. The colour blocks illustrate the partitions of the 1T1R array to implement the trainable parameters of the two convolutional layers (15 3 × 3 kernels of the first convolutional layer and 4 2 × 2 kernels of the second convolutional layer) and the fully connected layer (weight matrix size 64 × 10). The differential conductance pairs were formed between adjacent bit lines (for example, $g_{i,j}^+$ and $g_{i,j}^-$ where i and j are integers). Each weight is represented by two differential pairs here. The scanning electron micrograph shows a zoomed 1T1R cell (right). Scale bar, 20 μm. **b**, Analogue SET and RESET for all 128 × 64 1T1R cells with linearly varying common gate voltages for all compliance transistors. All 1T1R cells receive 15 consecutive SET operations with the gate voltage linearly increasing from 1.0 V to 2.4 V, followed by 15 RESET operations with the gate voltage linearly decreasing from 2.4 V to 1.0 V (for each RESET operation, a full RESET is applied before the 1T1R cell is SET again with current compliance). **c**, The network structure of the five-level CNN for classifying MNIST handwritten digits. The 8-bit grayscale input of size 8 × 8 was convolved by sliding the convolution region in the purple box. The selected convolution region was unrolled (for example, column vector elements $V_{1,1}$, $V_{2,1}$, $V_{3,1}$, $V_{1,2}$, …, $V_{3,3}$) and fed to the crossbar network of differential pairs as voltage signals (two duplicated differential pairs were used to represent a single weight in the experiment). The summed output currents I(kernel 1), I(kernel 2),…, I(kernel 15) were collected for all kernels. The input image was first physically convolved by the 15 memristor kernels of size 3 × 3 with zero padding and unitary stride. The analogue convolving outputs of the first convolutional layer were activated by the ReLU element-wise before being sent to the next layer, producing 15 feature maps of size 8 × 8. These feature maps were again convolved by the four kernels of size 2 × 2 of the second convolutional layer, followed by a subsampling layer that implemented the max pooling function over non-overlapping pooling windows of size 2 × 2. Then those feature maps were flattened as the inputs into a 10-way softmax output layer. The winner neuron of the output layer predicted the class of the input image (see Methods). The values of the feature maps and output are as indicated by the colour bar.

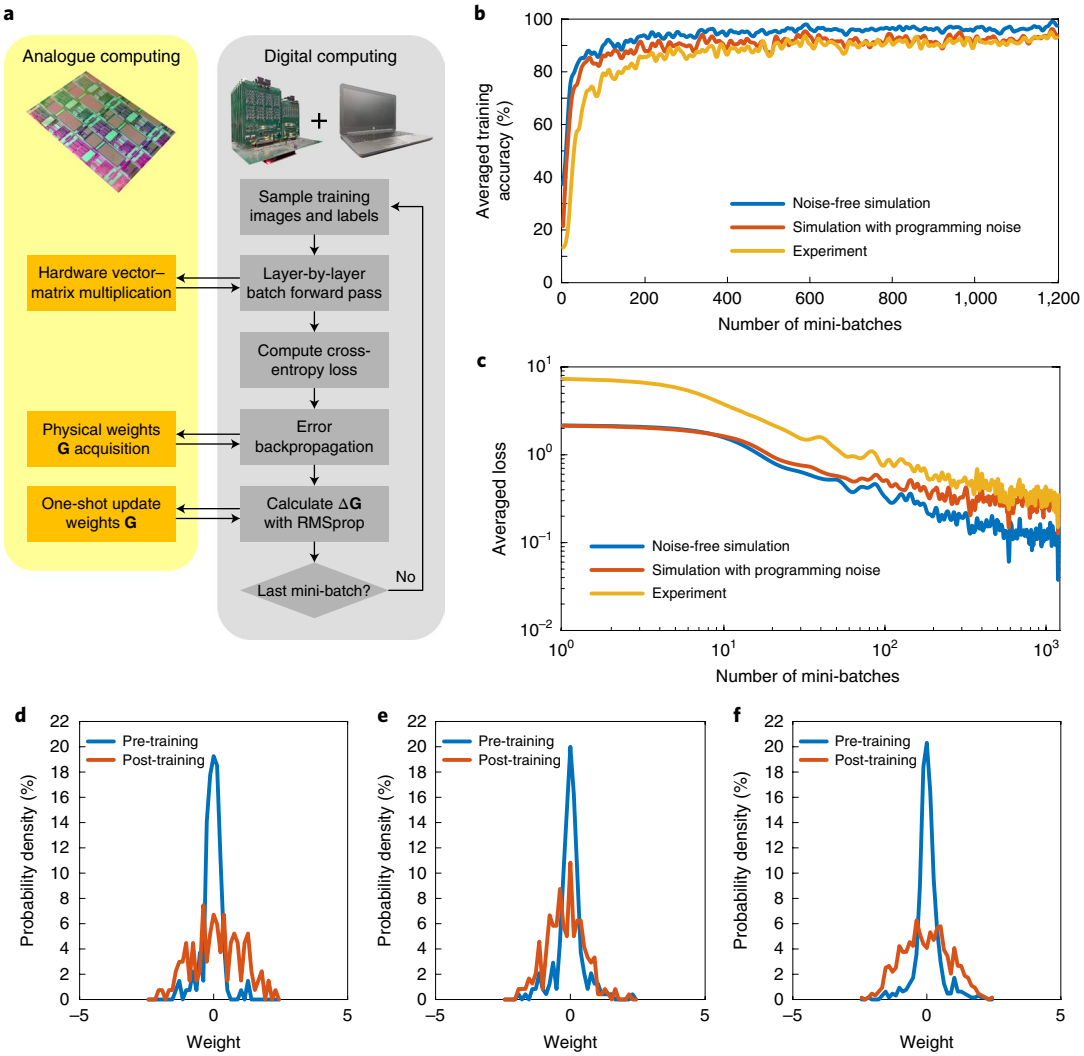

Fig. 2 | In situ training of the 1T1R-based five-level CNN. a, Schematic of the hybrid analogue–digital training of the CNN. The analogue 1T1R memristor array physically performed the computationally expensive vector–matrix multiplications, while the digital logic calculated the cross-entropy loss and the weight updates (the new conductance matrix *G*) using the RMSprop optimizer. **b**,**c**, The smoothed experimental in-batch accuracy increased (**b**) and loss decresed (**c**) over the course of in situ training. The experimental curves are indistinguishable from the simulation that includes programming noise, closely following the defect-free simulation with a ~4% gap in accuracy during the second epoch of the training. **d**–**f**, The distribution of weights before and after the in situ training of the 15 3×3 kernels (**d**), the four 2×2 kernels (**e**) and the fully connected layer (**f**). The Gaussian-like distributions were broadened by the training.

volution layer performs 270 (480) operations (multiplications and additions) per cycle (that is, one matrix–vector multiplication with a 1T1R sub-array), 64 (64) cycles per sample and 17,280 (30,720) operations per sample in the forward pass. Similarly, the fully connected layer performs 1,280 operations per cycle, 1 cycle per sample (see Supplementary Table 1). As a result, 1T1R array-based hybrid analogue–digital computing platforms show clear advantages in area and energy efficiencies over transistor-based implementations; these advantages are also scalable to state-of-the-art neural networks (see Supplementary Notes 3 and 4 on the projected area/energy efficiency of hybrid analogue–digital systems based on a 128×64 1T1R array and multiple large-scale arrays for running AlexNet).

In situ training of the 1T1R CNN. The in situ training of the network, consisting of the forward pass, error backpropagation and weight update, was implemented on the hybrid analogue–digital system. The training comprised 1,200 mini-batches in 2 epochs using a mini-batch size of 100 for all 60,000 images in the MNIST training dataset. As shown in Fig. 2a, the hybrid analogue–digital system combined the advantages of the area/energy efficiency of the memristors in performing the computationally expensive vector–matrix multiplications and digital logic for implementing the rest of the training algorithm (computing the cross-entropy loss and calculating the weight change using the RMSprop optimizer, for example; see Supplementary Fig. 1). The backpropagation was implemented in a MATLAB environment (R2018b, version 9.5) using the physically acquired weights. The weight gradients were then used to calculate how the array would be updated by the RMSprop, a variant of stochastic gradient descent with weight- and gradient-history dependency[47]. Instead of an iterative write–read scheme, we used the one-shot blind-update, which was enabled by the transistor current compliance. The conductance was updated column by column with limited parallelism $O(n_{\mathrm{bitlines}})$, where n_{bitlines} denotes the number of bit lines of the 1T1R array, rather than the $O(1)$ complexity of the vector–matrix multiplication (see Methods).

Smoothed training accuracy and loss over the epochs are presented in Fig. 2b,c, with test-set accuracies of 92.13% after two epochs (see Supplementary Fig. 3 for the statistics of the inference and Supplementary Fig. 4 for the representative classification and misclassification of all classes in the inference). The illustrative

training and inference are shown in the Supplementary Videos 1 and 2. The inaccuracy of memristor programming (the limiting factor for the performance) could be improved by further engineering the memristor material stack[48]. However, stochastic programming of the memristors may naturally mitigate overfitting as adding noise to weights[49,50] is one of the widely used methods of suppressing the generalization error associated with small datasets (see Supplementary Fig. 5 for the simulated impact of the programming noise over the generalization error).

To capture the impact of inevitable device imperfections on programming, we simulated the in situ training by presuming cycle-to-cycle and device-to-device variation in memristor programming. The programming noise was set to follow a normal distribution with a standard deviation of 10 μS in the simulated update of memristor conductance. The experimental curve, particularly in the second epoch, is nearly indistinguishable from the simulated curve with the assumed programming noise (see Fig. 2b,c). Moreover, we observed a consistent trend in the training of noise-free memristors (or equivalently double-precision floating point simulations), with an in-batch accuracy gap of around 4% in the second epoch of the training and a test-set accuracy difference of around 4%. Using just ~1,000 weights, this accuracy was equivalent to our previously reported multilayer perceptron using ~4,000 weights, with 11% non-responsive devices in both cases[45], illustrating the robust learning capability of the hybrid analogue–digital computing platform. The reduction of the free parameters without compromising performance hinges on the shared weights of the memristor-based CNN. The performance of the hardware network can approach that of the software with improved memristor fabrication yield and programming accuracy, which are being actively pursued by the materials community[48].

Figure 2d–f shows the weight distributions of the convolutional and fully connected layers before and after training (see Supplementary Fig. 6 for the conductance and weights after training). The distribution of the weights is nearly normal, as the training algorithm placed no limitation on the weight values. The weight distributions broadened as the training proceeded, verifying robust in situ training without requiring extra regularizing techniques.

1T1R ConvLSTM with spatio-temporal weight sharing. In addition to spatial weight sharing, the 1T1R array could concurrently implement spatio-temporal weight sharing—with a ConvLSTM network, for example. A ConvLSTM block is essentially a layer with two multidimensional inputs: in our case, the frame of the synthetic MNIST-sequence video of size 8×8 and the recurrent input of size 6×6 (×5), based on the output of the last time step (see Methods for the generation of the MNIST-sequence dataset). The recurrent nature, or temporal structure, of ConvLSTM was expressed by a sequence of feed-forward operations, as shown in Fig. 3a, linking one time step to the next. Sharing of the trainable parameters occurs between the time steps as in an LSTM, effectively reusing the same weights across all time steps of the calculation. In implementing spatial correlations by convolving both inputs, ConvLSTM shares the weights both spatially and temporally, which could model general-purpose spatio-temporal patterns in 3D inputs[6].

Both of the inputs were subsequently convolved by the self-parameterized cell input and control gates of the ConvLSTM. The cell input and control gates were mapped to the memristor arrays as shown in Fig. 3b. A new frame of the input arrived with each time step, and its information was incorporated into the cell under the control of the gates. The previous cell status would be partially removed in this process if the forget gate was on. Whether the latest cell output propagates to the next time frame was determined by the output gate. Before the first frame of the input, all the states of the ConvLSTM layer were initialized to zero, corresponding to 'total ignorance' of the future[6].

The final output, five 6×6 feature maps of the ConvLSTM layer, was downsampled by a max pooling layer. The pooled feature maps were then flattened as the inputs to a six-neuron softmax layer, to classify the six classes of the MNIST-sequences (see Methods).

A single convolution stride of ConvLSTM has computational complexity $O(\mathrm{ki}_{\mathrm{dim1}} \times \mathrm{ki}_{\mathrm{dim2}} \times \mathrm{ki}_{\mathrm{dim3}} \times \mathrm{ki}_{\mathrm{num}}) + O(\mathrm{kr}_{\mathrm{dim1}} \times \mathrm{kr}_{\mathrm{dim2}} \times \mathrm{kr}_{\mathrm{dim3}} \times \mathrm{kr}_{\mathrm{num}})$, where variables ki and kr correspond to the kernels of the input and recurrent input, respectively. Similar to the CNN case, the complexity was reduced to $O(1)$ by mapping the weights to the 1T1R array. The ConvLSTM layer performs 1,160 operations per cycle, 108 cycles per sample in the forward pass, which consists of 3 time steps, and 125,280 operations per sample. Similarly, the fully connected layer performs 540 operations per cycle, 1 cycle per sample (see Supplementary Table 2). All other computations shown in the computational graph were element-wise operations performed infrequently, once per completion of all strides of a convolution, using the conventional logic circuitry of the hybrid analogue–digital system.

In situ training of the 1T1R ConvLSTM. The temporal recurrence of ConvLSTM makes the error signal backpropagate both spatially (within the same ConvLSTM block) and temporally (across all ConvLSTM blocks at different times). Extra memory is thus needed to record the neural activities of each time step in a forward pass, which differs from the training of a feed-forward network in Fig. 2a. Such a memory was implemented digitally here, but potential analogue sample and hold memories could be employed to develop fast backpropagation in the analogue domain.

The animated training and inference are shown in Supplementary Videos 3 and 4. Similar to the case in Fig. 2b, the intrinsic variations of the memristors lead to small differences between the training (95.97%) and test-set (96.44%) accuracies, confirming the built-in regularizing capability of the memristor array (see Supplementary Fig. 7 for the statistics of the inference) . This proof-of-concept video classification used only 850 weights, in contrast to the 3,675 weights of a functional equivalent with a five-kernel convolution layer followed by five LSTM units and a dense layer, manifesting the drastic parameter reduction of the spatio-temporal weight sharing of ConvLSTM. The experimental accuracy reveals small differences between both predictions, assuming ideal and noisy memristor programming, as shown in Fig. 4a,b.

Figure 4c,d shows the broadening of the Gaussian-like weight distributions of the ConvLSTM and fully connected layers before and after training, demonstrating robust in situ training with simultaneous spatial and temporal weight sharing (see Supplementary Fig. 8 for the conductance and weights after training). Figure 4e depicts representative examples of valid and invalid classifications of the MNIST-sequences. The invalid classifications may also be challenging for human beings to identify at the downsampled resolution (see Supplementary Fig. 9 for the representative classification and misclassification of all classes in the inference).

Discussion

In summary, we demonstrated in situ training of both the spatial shared-weight CNN and the spatio-temporal shared-weight ConvLSTM using memristor crossbar arrays. The in situ training self-adaptively mitigated the impact of hardware non-idealities, which is vital to meeting the stringent convergence requirements of weight-sharing structures. The accuracy was equivalent to that of a fully connected memristive neural network in classifying MNIST handwritten digits with CNN, but required only ~25% of the weights. In addition, we demonstrated simultaneous spatial and temporal weight sharing, for example, in the ConvLSTM network that features intrinsic 3D input processing for classifying the MNIST-sequence videos. The reduction in the number of trainable parameters as a result of weight sharing, as well as the area/energy efficiency boost

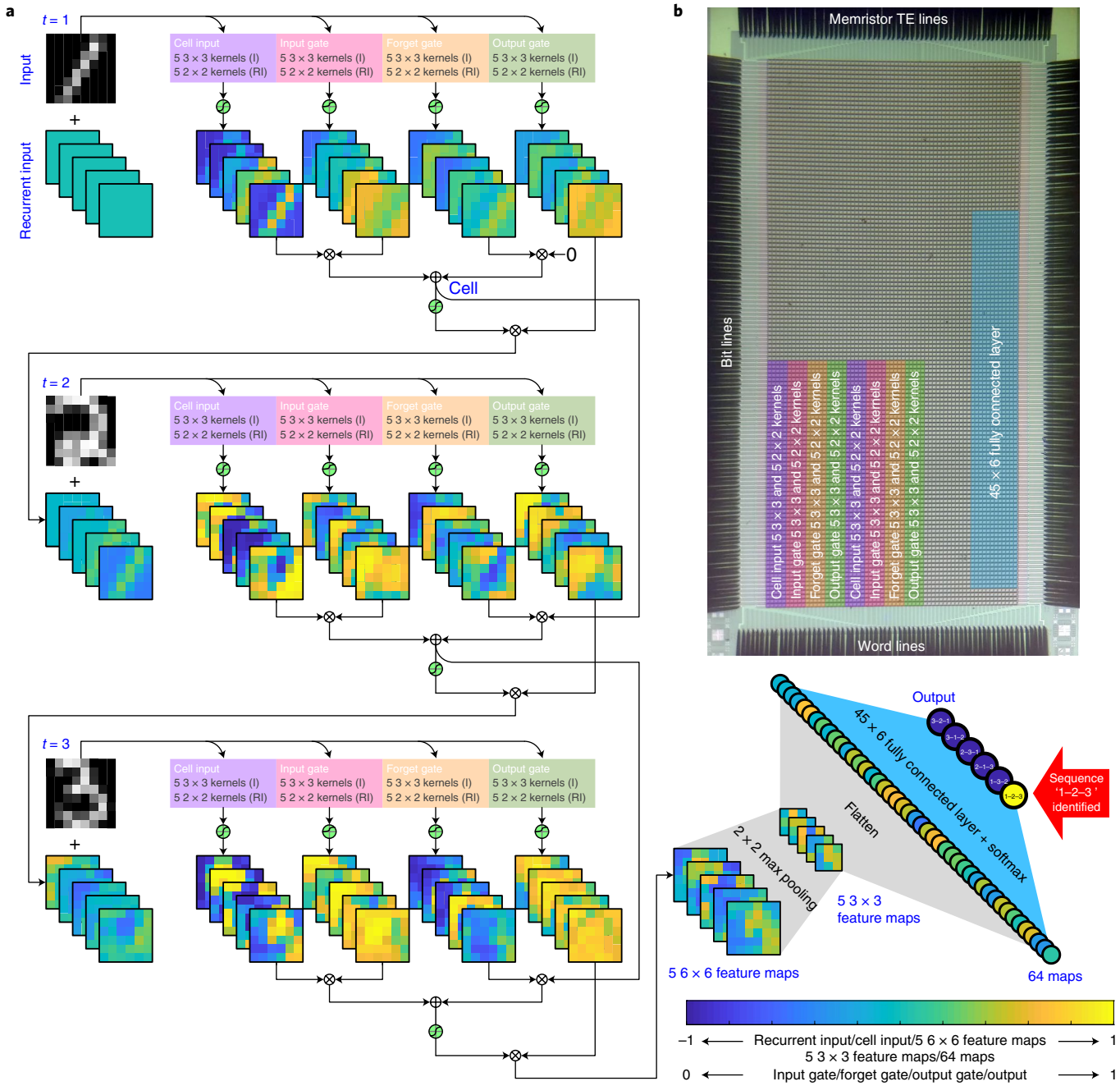

Fig. 3 | 1T1R implementation of the ConvLSTM network. a, The network structure of ConvLSTM for classifying the MNIST sequence. The temporal recurrence is expanded as the operations proceed in sequence from one time step (*t*) to the next. The 8-bit grayscale frame input (I) of size 8×8 was fed to the ConvLSTM layer at each time step, together with the recurrent input (RI) based on the output of the same layer in the last time step. Both inputs were physically convolved by the 20 kernels of the self-parameterized cell input and controlling gates (for example input gate, forget gate and output gate), which were mapped to the memristor array. The single cell input or gate consisted of five input kernels of size 3×3 and five recurrent input kernels of size 2×2. The 1T1R output currents were then nonlinearly activated and gated to update the cell and yield the output. The output of the current time step would serve as one of the inputs to the same ConvLSTM layer in the next time step. Similarly, the cell of the current time step would be passed to the next time step to be gated by the forget gate of the same ConvLSTM layer. Note that all the states of the ConvLSTM were initialized to zero before the first input, corresponding to a total ignorance of the future. The output of the last time step of the ConvLSTM, five feature maps of size 6×6, was downsampled by a max pooling layer. The pooled feature maps were then flattened as the inputs to a six-neuron softmax layer, to classify the six possible MNIST sequences (see Methods). All trainable parameters of the ConvLSTM layer are shared spatially between different convolution regions, besides the weights are also effectively reused across all time steps. **b**, Optical micrograph of the 128×64 1T1R array with colour blocks illustrating the partitions of the 1T1R array used to implement the trainable parameters of the ConvLSTM layer and the fully connected layer (weight matrix size 45×6). The values of the feature maps and output are as indicated by the colour bar.

from analogue in-memory computing, illustrates the potential of memristor-based computing platforms for implementing advanced topologies of neural networks directed towards edge applications.

The proof-of-concept in situ training of a neural network using the spatial or spatio-temporal weight-sharing technique on emerging memories not only demonstrated that the advantages of the

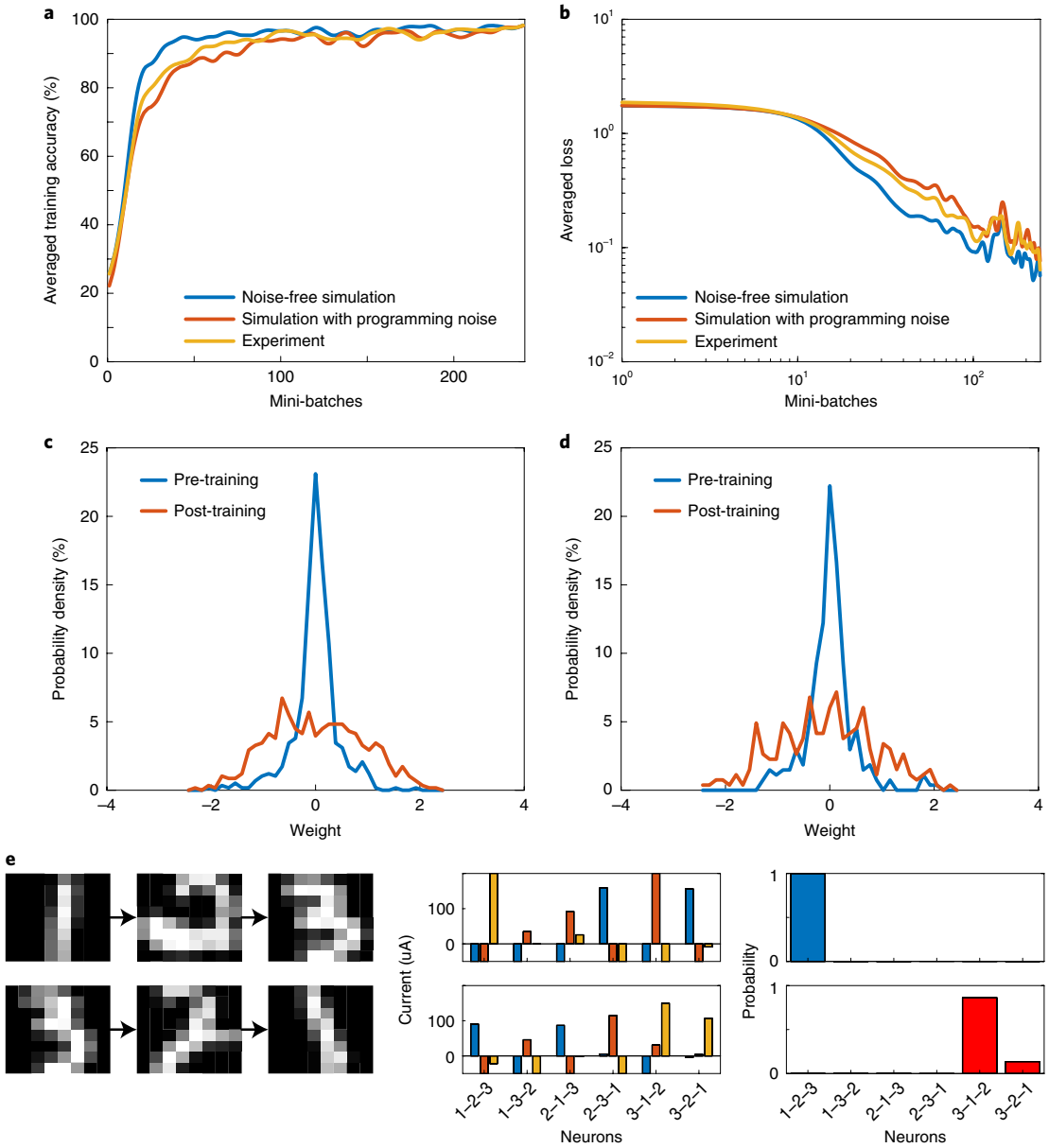

Fig. 4 | In situ training of the 1T1R based ConvLSTM. a,**b**, The smoothed experimental mini-batch accuracy increased (**a**) and loss decreased (**b**) over the course of in situ training in classifying the MNIST sequence. The experimental curves, the simulation with programming noise and the ideal programming simulation are close to each other. **c**,**d**, The Gaussian-like distribution of weights before and after the in situ training. The distributions were broadened by the training of the ConvLSTM (**c**) and fully connected layers (**d**), consistent with those shown in Fig. 2d–f. **e**, Representative inference examples of the correctly classified sequence (top) and misclassified sequence (bottom) after the in situ training. The middle panels show the raw currents collected by the six output neurons (blue, red and orange bars correspond to time steps 1, 2 and 3, respectively). The right panels show the associated Bayesian probabilities calculated by the softmax activation (of the last time step only). All inference examples are with Supplementary Video 4.

weight sharing could be combined with the area/energy efficiency of memristor crossbars for post-von Neuman computing, but also revealed that the inherent stochasticity of memristor programming does not always negatively impact the in situ training because reasonable amount of stochasticity could serve as a natural regularization to avoid overfitting.

Methods

1T1R array integration. The 1T1R arrays were fabricated by integrating Pt/TaO_x/Ta memristors on the transistor array. The front-end and part of the back-end processes to build the transistor arrays, word lines, bit lines and the memristor TE lines were done in a commercial fab using the 2 μm technology node. The transistors were used as selecting devices to mitigate the sneak path currents in programming the memristors and to enable precise conductance tuning by limiting the current. The memristor integration began with Ar plasma etching of the native oxide of the metal vias, followed by the deposition of 8 nm Ag, 2 nm Ti and 200 nm Pd in sequence using a sputter with Ar ambience. The chip was then annealed at 300 °C for 36 minutes to improve the metallic contacts with memristor electrodes. The bottom electrode of memristors consisted of 5 nm Ta and 60 nm Pt deposited by sputtering in sequence. The 10 nm TaO_x switching layer was deposited by sputtering a Ta_2O_5 target at room temperature in Ar plasma, and patterned by reactive ion etching using CHF_3 and O_2. The memristor top electrodes comprised 50 nm Ta and 20 nm Pd, deposited by sputtering in sequence. The memristor integration was done in our lab. All metals deposited in the lab were patterned by lift-off, with a memristor junction size of 4 × 4 μm^2.

The hybrid analogue–digital platform. A customized hybrid analogue–digital platform was used to electrically read and write the 1T1R chip[24,25,51]. The system provides 128 + 64 + 64 ways of concurrent analogue voltages with a minimum pulse width of ~100 ns and an amplitude of up to ±10 V. It could also sense 64 ways

Table 1 | List of the hyperparameters used for the CNN and ConvLSTM (see Code)

	Hyperparameters	CNN	ConvLSTM	Description
Neural network	Learning rate	10^{-2}	10^{-2}	The learning rate used by RMSprop
	Momentum	0	0	The gradient momentum used by RMSprop
	Decay	0.9	0.9	The decay factor of the previous moving average of the squared gradient used by RMSprop
	eps	10^{-8}	10^{-8}	A small floating point number to avoid division by zero used by RMSprop
Hardware interfacing	R_{gw}	10^{-4}	10^{-4}	The ratio between the change in conductance and the change in weight
	R_{vg}	1.5×10^{4}	1.5×10^{4}	The ratio between the change in gate voltage and the change in conductance
	$V_{g,min}$	0.5	0.5	Minimal gate voltage used to SET a memristor
	$V_{g,max}$	2.5	2.5	Maximum gate voltage used to SET a memristor
	V_{SET}	2.5	2.5	Positive voltage applied to the memristor TE line to SET a memristor
	RESET gate voltage	5	5	Word line voltage used to RESET a memristor
	RESET voltage	1.7	1.7	Bit line voltage used to RESET a memristor
	Read voltage	0.2	0.2	Bit line voltage used to read the conductance of a memristor

of analogue current signals at the same time (see Supplementary Fig. 1 for the functional block diagram of the system).

Each bit line or word line could be independently configured for voltage biasing, ground or high impedance. For voltage biasing, each line was driven by a channel of a digital-to-analogue converter (LTC2668, Analog Devices, 16-bit resolution), which communicated with the microcontroller via digital I/O ports.

Each memristor TE line could be independently configured for voltage biasing, ground, high-impedance or current sensing. The voltage biasing was implemented in the same way with that of the bit or word lines. For current sensing, each memristor TE line connected to one of the four transimpedance amplifiers (LTC6268, Analog Devices) with different gains. The voltage outputs of the transimpedance amplifiers were read by the analogue-to-digital converters (MAX11046, Maxim Integrated, 16-bit resolution) and retrieved by the microcontroller.

Basic 1T1R operations. The basic electrical operations of the 1T1R memristor array include analogue SET programming, conductance readout and vector–matrix multiplication.

For analogue SET programming, the 1T1R array was programmed row-by-row (see the flowchart in Supplementary Fig. 10a). To program a selected row, all bit lines were floating, except the selected one, which was grounded. Each word line was assigned a different voltage on the basis of the targeted conductance. All memristor TE lines were biased with the same V_{SET} voltages. The analogue SET programming of memristors of the same row is parallel. For weight readout, the row electrodes output a scaled identity voltage matrix in 128 time steps (that is, the only non-zero input V_{Read} at time step i is with the ith bit line). Within each time step, the currents of all column wires were read by the microcontroller (see the flowchart in Supplementary Fig. 10b). For vector–matrix multiplications, the bit lines output the vector $\mathbf{V}$. The currents of all column wires were read by the microcontroller (see the flowchart in Supplementary Fig. 10c). More advanced functions such as hidden neurons were implemented by employing multiple basic 1T1R operations (see Supplementary Fig. 1).

Training method. For both the CNN and the ConvLSTM networks, the forward passes were physically implemented on the hybrid analogue–digital computing platforms (see Supplementary Fig. 11). The cross-entropy loss function was used to define the error. To minimize the error on a set of training samples, the error derivatives with the trainable parameters were computed by the standard backpropagation[52] (for CNN) or iterative backpropagation through time[53] (for ConvLSTM) for each sample and accumulated over the mini-batch. The error derivatives over the weights were used to calculate the required change in each weight using the RMSprop optimizer[47] for every mini-batch.

The calculated weight update was first mapped to the changes of the memristor conductance by multiplying a constant factor R_{gw} (see Table 1). Each weight, $(g_+ - g_-)/R_{gw}$, is proportional to the mean conductance difference $(g_+ - g_-)$ between one or more pairs of memristors (here we used two pairs of memristors from the same row to further improve the programming accuracy[37]). Note that the conductance changes in the two memristors in the same differential pair are of the same magnitude but opposite sign. The changes in the memristor conductance were then converted to the changes in the transistor gate voltages by multiplying by the constant factor R_{vg} (see Table 1). The new gate voltages for SET programming were the sums of the calculated changes of the gate voltages and the very last gate voltages, bounded to the interval $[V_{g,min}, V_{g,max}]$ (see Table 1). The new gate voltages were subsequently used to program the 1T1R memristor array.

Resized MNIST dataset and the MNIST sequence dataset. To fit the dimension of the 1T1R array, the original MNIST dataset images were resized from 28×28 to 20×20 pixels by trimming the edges of the original images and keeping the central zones. The chopped images were then downsampled to 8×8 pixels using bicubic interpolation for all 60,000 training samples and 10,000 test samples.

The MNIST-sequence was generated by first creating a random sequence of three digits 1, 2 or 3 (that is, one of the following six sequences: 1–2–3, 1–3–2, 2–1–3, 2–3–1, 3–1–2, and 3–2–1). The first frame would be the next unused resized MNIST image corresponding to the first digit in the generated sequence. The same process would apply to the generation of the second and the third frames. The used images were not reused, so each sequence comprised exclusive frames. This generation process was repeated until the images of a certain digit were used up, resulting in an MNIST-sequence dataset with 5,958 training sequences and 1,010 testing sequences.

All input images are linearly mapped to the voltage range [−0.2 V, 0.2 V]; these voltages were applied to the bit lines of the 1T1R array.

Hyperparameters. Table 1 shows the hyperparameters used for the in situ training, including the hyperparameters for the neural network and the hardware parameters for 1T1R crossbar operation.

Data availability

The data that support the plots within this paper and other finding of this study are available in a Zenondo repository at https://doi.org/10.5281/zenodo.3273475.

Code availability

The code that support the plots within this paper and other finding of this study is available in a Zenondo repository at https://doi.org/10.5281/zenodo.3277298 and https://github.com/zhongruiwang/memristorCNN. The code that supports the communication between the custom-built measurement system and the integrated chip is available from the corresponding author on reasonable request.

Received: 17 December 2018; Accepted: 24 July 2019;
Published online: 9 September 2019

References

1. Hubel, D. H. & Wiesel, T. N. Receptive fields and functional architecture of monkey striate cortex. *J. Physiol.* **195**, 215–243 (1968).
2. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
3. Simonyan, K. & Zisserman, A. Very deep convolutional networks for large-scale image recognition. Preprint at https://arxiv.org/abs/1409.1556 (2014).
4. Szegedy, C. et al. Going deeper with convolutions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2015).
5. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 770–778 (IEEE, 2016).
6. Shi, X. et al. Convolutional LSTM network: a machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems* 802–810 (NIPS, 2015).
7. Buonomano, D. V. & Maass, W. State-dependent computations: spatiotemporal processing in cortical networks. *Nat. Rev. Neurosci.* **10**, 113–125 (2009).
8. Patraucean, V., Handa, A. & Cipolla, R. Spatio-temporal video autoencoder with differentiable memory. Preprint at https://arxiv.org/abs/1511.06309 (2015).

9. Jouppi, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. 44th Annual International Symposium on Computer Architecture* (ACM/IEEE, 2017).
10. Chen, Y.-H., Krishna, T., Emer, J. S. & Sze, V. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid St. Circ.* **52**, 127–138 (2017).
11. Chen, Y. et al. Dadiannao: a machine-learning supercomputer. In *Proc. 47th Annual IEEE/ACM International Symposium on Microarchitecture* 609–622 (IEEE/ACM, 2014).
12. Tsai, H., Ambrogio, S., Narayanan, P., Shelby, R. M. & Burr, G. W. Recent progress in analog memory-based accelerators for deep learning. *J. Phys. D* **51**, 283001 (2018).
13. Ielmini, D. & Wong, H. S. P. In-memory computing with resistive switching devices. *Nat. Electron* **1**, 333–343 (2018).
14. Zidan, M. A., Strachan, J. P. & Lu, W. D. The future of electronics based on memristive systems. *Nat. Electron* **1**, 22–29 (2018).
15. Yu, S. Neuro-inspired computing with emerging nonvolatile memorys. *Proc. IEEE* **106**, 260–285 (2018).
16. Strukov, D. B., Snider, G. S., Stewart, D. R. & Williams, R. S. The missing memristor found. *Nature* **453**, 80–83 (2008).
17. Jo, S. H. et al. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* **10**, 1297–1301 (2010).
18. 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. Elect. Dev.* **58**, 2729–2737 (2011).
19. Eryilmaz, S. B. et al. Brain-like associative learning using a nanoscale non-volatile phase change synaptic device array. *Front. Neurosci.* **8**, 205 (2014).
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. Elect. Dev.* **62**, 3498–3507 (2015).
21. Prezioso, M. et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015).
22. Ambrogio, S. et al. Unsupervised learning by spike timing dependent plasticity in phase change memory (PCM) synapses. *Front. Neurosci.* **10**, 56 (2016).
23. Hu, M. et al. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *53rd ACM/EDAC/IEEE Design Automation Conference* (ACM/IEEE, 2016).
24. Hu, M. et al. Memristor-based analog computation and neural network classification with a dot product engine. *Adv. Mater.* **30**, 1705914 (2018).
25. Li, C. et al. Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* **1**, 52–59 (2018).
26. Nili, H. et al. Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors. *Nat. Electron.* **1**, 197–202 (2018).
27. Le Gallo, M. et al. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
28. Zidan, M. A. et al. A general memristor-based partial differential equation solver. *Nat. Electron.* **1**, 411–420 (2018).
29. Jeong, Y., Lee, J., Moon, J., Shin, J. H. & Lu, W. D. K-means data clustering with memristor networks. *Nano Lett.* **18**, 4447–4453 (2018).
30. Shin, J. H., Jeong, Y. J., Zidan, M. A., Wang, Q. & Lu, W. D. Hardware acceleration of simulated annealing of spin glass by RRAM crossbar array. In *2018 IEEE International Electron Devices Meeting* 3.3.1–3.3.4 (IEEE, 2018).
31. Sun, Z. et al. Solving matrix equations in one step with cross-point resistive arrays. *Proc. Natl Acad. Sci. USA* **116**, 4123–4128 (2019).
32. Sheridan, P. M. et al. Sparse coding with memristor networks. *Nat. Nanotechnol.* **12**, 784–789 (2017).
33. 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).
34. Yao, P. et al. Face classification using electronic synapses. *Nat. Commun.* **8**, 15199 (2017).
35. Ambrogio, S. et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018).
36. Bayat, F. M. et al. Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. *Nat. Commun.* **9**, 2331 (2018).
37. Boybat, I. et al. Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* **9**, 2514 (2018).
38. Chen, W.-H. et al. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *2018 IEEE International Solid-State Circuits Conference* 494–496 (IEEE, 2018).
39. Xue, C.-X. et al. A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In *2019 IEEE International Solid-State Circuits Conference* 388–390 (IEEE, 2019).
40. Mochida, R. et al. A 4M synapses integrated analog ReRAM based 66.5 TOPS/W neural-network processor with cell current controlled writing and flexible network architecture. In *2018 IEEE Symposium on VLSI Technology* 175–176 (IEEE, 2018).
41. Gokmen, T., Onen, M. & Haensch, W. Training deep convolutional neural networks with resistive cross-point devices. *Front Neurosci.* **11**, 538 (2017).
42. Li, C. et al. Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intell.* **1**, 49–57 (2019).
43. Sun, X. et al. XNOR-RRAM: a scalable and parallel resistive synaptic architecture for binary neural networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition* 1423–1428 (IEEE, 2018).
44. Gao, L., Chen, P.-Y. & Yu, S. Demonstration of convolution kernel operation on resistive cross-point array. *IEEE Elect. Dev. Lett.* **37**, 870–873 (2016).
45. Li, C. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018).
46. Yang, J. J. et al. High switching endurance in TaO_x memristive devices. *Appl. Phys. Lett.* **97**, 232102 (2010).
47. Tieleman, T. & Hinton, G. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* **4**, 26–31 (2012).
48. Choi, S. et al. SiGe epitaxial memory for neuromorphic computing with reproducible high performance based on engineered dislocations. *Nat. Mater.* **17**, 335–340 (2018).
49. Graves, A., Mohamed, A.-r. & Hinton, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* 6645–6649 (IEEE, 2013).
50. An, G. The effects of adding noise during backpropagation training on a generalization performance. *Neural Comput.* **8**, 643–674 (1996).
51. Wang, Z. et al. Reinforcement learning with analogue memristor arrays. *Nat. Electron.* **2**, 115–124 (2019).
52. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).
53. Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proc. IEEE* **78**, 1550–1560 (1990).

Acknowledgements

This work was supported in part by the US Air Force Research Laboratory (grant no. FA8750-18-2-0122) and the Defense Advanced Research Projects Agency (contract no. D17PC00304). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of US Air Force Research Laboratory. H.W. was supported by the Beijing Advanced Innovation Center for Future Chip and National Science Foundation of China (grant no. 61674089 and 61674092). Part of the device fabrication was conducted in the clean room of the Center for Hierarchical Manufacturing, an National Science Foundation Nanoscale Science and Engineering Center located at the University of Massachusetts Amherst.

Author contributions

J.J.Y. conceived the idea. J.J.Y., Q.X. and Z.W. designed the experiments. Z.W., C.L., P.L., Y.N. and W.S. performed the programming, measurements, data analysis and simulation. M.R., P.Y, C.L. and N.G. built the integrated chips. P.L., Y.L., M.H. and J.P.S. designed the measurement system and firmware. Q.Q., H.W., N.M., Q.W. and R.S.W. helped with experiments and data analysis. J.J.Y. and Z.W 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/s42256-019-0089-1.

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

Correspondence and requests for materials should be addressed to Q.X. or J.J.Y.

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