ARTICLE

https://doi.org/10.1038/s41586-018-0180-5

Equivalent-accuracy accelerated neural-network training using analogue memory

Stefano Ambrogio[1], Pritish Narayanan[1], Hsinyu Tsai[1], Robert M. Shelby[1], Irem Boybat[2,3], Carmelo di Nolfo[1,3], Severin Sidler[1,3], Massimo Giordano[1], Martina Bodini[1,3], Nathan C. P. Farinha[1], Benjamin Killeen[1], Christina Cheng[1], Yassine Jaoudi[1] & Geoffrey W. Burr[1]*

Neural-network training can be slow and energy intensive, owing to the need to transfer the weight data for the network between conventional digital memory chips and processor chips. Analogue non-volatile memory can accelerate the neural-network training algorithm known as backpropagation by performing parallelized multiply-accumulate operations in the analogue domain at the location of the weight data. However, the classification accuracies of such in situ training using non-volatile-memory hardware have generally been less than those of software-based training, owing to insufficient dynamic range and excessive weight-update asymmetry. Here we demonstrate mixed hardware–software neural-network implementations that involve up to 204,900 synapses and that combine long-term storage in phase-change memory, near-linear updates of volatile capacitors and weight-data transfer with 'polarity inversion' to cancel out inherent device-to-device variations. We achieve generalization accuracies (on previously unseen data) equivalent to those of software-based training on various commonly used machine-learning test datasets (MNIST, MNIST-backrand, CIFAR-10 and CIFAR-100). The computational energy efficiency of 28,065 billion operations per second per watt and throughput per area of 3.6 trillion operations per second per square millimetre that we calculate for our implementation exceed those of today's graphical processing units by two orders of magnitude. This work provides a path towards hardware accelerators that are both fast and energy efficient, particularly on fully connected neural-network layers.

Deep neural networks (DNNs) are a family of neuromorphic computing architectures that have recently made substantial advances in difficult machine-learning problems such as image or object recognition, speech recognition and machine language translation[1]. Computation for DNNs includes both training, during which the weights of the network are optimized on a training dataset, and forward inference, during which the already-learned network is used for classification, prediction or other useful tasks on new, previously unseen 'test' data.

These networks are highly amenable to computation via large and dense matrix–matrix multiplications that can be highly parallelized. This has led to tremendous opportunities for hardware acceleration by using graphical processing units (GPUs)[1,2], which in turn enable large networks with commercially interesting levels of performance. Furthermore, DNNs are highly resilient to numerical inaccuracies[3], especially for forward inference[4]. As a result, reductions in computational precision by using field-programmable gate array (FPGA)[3] and application-specific integrated circuit (ASIC) designs[5,6] offer a path to even higher computational performance and better power efficiency.

Conventional von Neumann hardware is constrained by the time and energy spent moving data back and forth between the memory and the processor (the 'von Neumann bottleneck'). By contrast, in a non-von Neumann scheme, computing is done at the location of the data, with the strengths of the synaptic connections (the 'weights') stored and adjusted directly in memory.

An example of hardware that uses a non-von Neumann scheme is the TrueNorth chip (IBM), a flexible platform for forward inference of large pre-trained DNNs at ultralow power[7,8]. However, for efficient on-chip training, it would be preferable to replace the digital synaptic weights, which are stored in static random access memory (SRAM) arrays on TrueNorth, with high-density analogue devices that encode synaptic weight directly in their conductances. Such analogue systems could achieve substantial speedup and power reduction for both forward inference and training[9–12]. However, it has not been conclusively proven that such analogue approaches can 'do the same job' as current software running on conventional digital hardware, in terms of training DNNs to equivalently high accuracies. There is little point to being faster or more energy-efficient in training a DNN if the resulting classification accuracies are unacceptably low.

Desirable characteristics of analogue devices for training include rapid, low-power programming of multiple analogue levels, dimensional scalability, reasonable retention, high endurance and, most importantly, gradual and symmetric conductance-update characteristics[13]. So far, experimental demonstrations of analogue-memory-based DNN training have suffered from reduced classification accuracies owing to the substantial non-idealities exhibited by existing devices. These demonstrations have featured filamentary resistive RAM (RRAM)[14–16], non-filamentary resistive RAM[17], phase-change memory (PCM)[10,11], conductive-bridging RAM (CBRAM)[18], ferroelectric RAM[19] and hybrid digital–non-volatile memory (NVM) architectures[20]. Electrochemical devices that offer highly symmetric and gradual conductance update—such as ENODe[21] (electrochemical neuromorphic organic device) and LISTA[22] (lithium-ion synaptic transistor for analogue computing)—have been demonstrated, but not in array configurations and so far only with programming pulse durations that are many orders of magnitude too long.

Here, we introduce a synaptic unit-cell design for analogue-memory-based DNN training, which combines non-volatile PCM with volatile weight storage using conventional complementary metal–oxide–semiconductor (CMOS)-based devices. Using this unit cell, we demonstrate software-equivalent DNN accuracies experimentally for various datasets in fully connected networks. These mixed hardware–software experiments combine PCM hardware arrays with

[1]IBM Research–Almaden, San Jose, CA, USA. [2]IBM Research–Zurich, Rueschlikon, Switzerland. [3]EPFL, Lausanne, Switzerland. *e-mail: gwburr@us.ibm.com

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

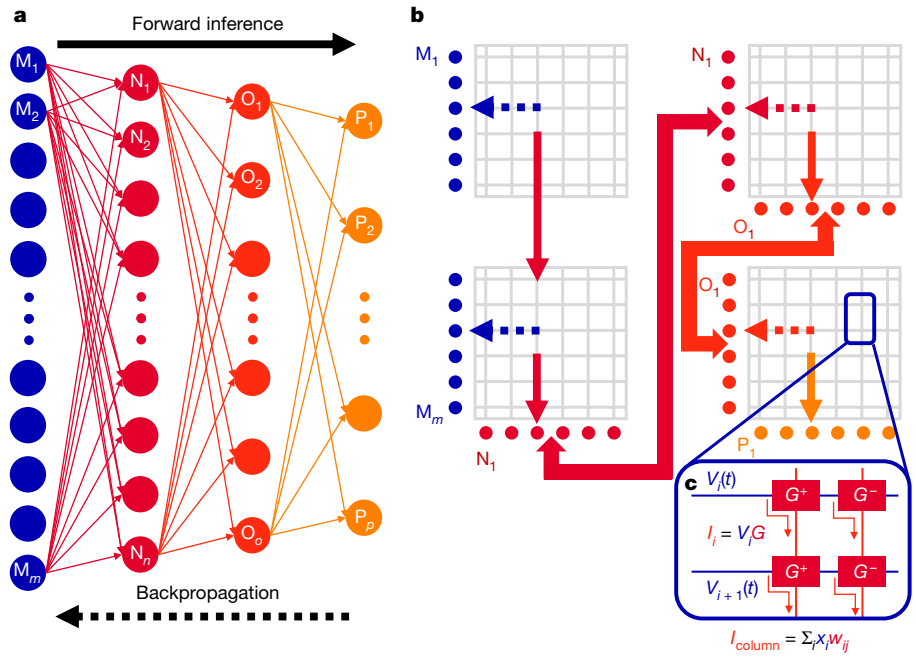


Fig. 1 | Mapping a fully connected neural network onto NVM arrays. **a**, **b**, A fully connected four-layer (M, N, O and P) neural network of size *m*-*n*-*o*-*p* (**a**) can be mapped to multiple blocks of crossbar arrays surrounded by peripheral neuron circuitry (**b**). For wide layers with many neurons, multiple blocks of arrays are combined. Thin solid arrows in **b** show the column-based current integration during forward inference, which corresponds to the logical flow indicated by solid arrows in **a**. Similarly, dotted arrows show the row-based current integration during backpropagation; wide arrows in **b** show the routeing communication needed to connect the two sets of neuron circuitry corresponding to each neuron layer in **a**. **c**, Details of how neuron excitations encoded as voltage pulses ($V_i(t)$, where i corresponds to the row) get multiplied by weight data encoded as conductances (via Ohm's law, $I_i = V_iG$), after which Kirchhoff's current law is applied along the columns j ($I_{column} = \sum_i x_i w_{ij}$).

realistic SPICE (simulation program with integrated circuit emphasis)-based circuit simulations that include full CMOS device variability. We demonstrate neural-network training of the MNIST[23] and MNIST-backrand (https://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations) databases—databases of handwritten digits from the National Institute of Standards and Technology (NIST): MNIST, 'modified' from original NIST database; MNIST-backrand, MNIST with random background noise added to each image—and transfer learning of CIFAR-10 and CIFAR-100[24] datasets—datasets from the Canadian Institute for Advanced Research with 10 or 100 classes of images. We present power estimates for the device arrays and for the requisite analogue peripheral circuitry, with power projections for DNN training as low as 54 mW for a computational energy efficiency of 28,065 billion operations per second per watt (28,065 GOP s^{-1} W^{-1}) and throughput per unit area of 3.6 trillion operations per second per square millimetre (3.6 TOP s^{-1} mm^{-2}). These efficiency and throughput values that we estimate are 280 and 100 times better, respectively, than those achieved using the most-recent GPUs. We believe that these results demonstrate a viable path to low-power hardware acceleration of the training of a wide variety of DNNs using existing analogue memory and computing elements.

Deep learning using NVM

By mapping synaptic layers onto crossbar array blocks (Fig. 1a), non-von Neumann hardware based on NVM could potentially perform all the multiply–accumulate operations for a fully connected neural-network layer in a single parallelized step, without any motion of weight data. Each network layer maps onto one or more identical array blocks (Fig. 1b), each containing 'upstream' (at the end of each row) and 'downstream' (at the end of each column) neuron circuitry, interconnected by a flexible block-to-block routeing network[25]. Neuron circuitry at the west (south) edges generates these pulses on the basis of the accumulated current from the preceding forward (reverse) pass, which drives the crossbar bit and word lines to appropriate voltages on the basis of the mode of operation and the duration of the neuron pulse[25].

The analogue signals used during the forward or reverse pass are fixed in amplitude, but vary in duration. This enables standard CMOS buffering approaches to be used when communicating signals from the outputs of one array to the inputs of another array, and makes it possible to avoid the considerable power- and area-overhead of analogue-to-digital converters. Reconfigurable connectivity between arrays enables mapping of different neural-network topologies to the same physical hardware using only a few configuration bits per array.

During forward inference, upstream neurons i introduce excitations x_i onto row lines. Ohm's law ($I = VG$, where I is the current, V is the voltage and G is the conductance) implements the multiplication between these excitations x_i and the weight values w_{ij} that are encoded in the conductance G of the NVM device. Signed w_{ij} weights can be encoded into the difference between a pair of conductances, $G^+ - G^-$ (Fig. 1c). Kirchhoff's current law sums these contributions along each column line and the downstream neurons integrate the overall signal[25], $\sum_i x_i w_{ij}$.

During backpropagation, the parts played during forward inference are reversed, with the downstream neuron j introducing 'delta' values δ_j onto the columns and the upstream neuron i accumulating integration along each row to implement[10,25,26] $\sum_i \delta_i w_{ij}$. For weight update, each set of neurons fires either a deterministic[10,27] or stochastic[12] series of pulses, with the number of pulses based on the most recent values of x_i and δ_j passed through those neurons during the forward-inference and backpropagation steps. The overlap of these upstream and downstream pulses adjusts the synaptic weights in parallel across the entire array. This scheme corresponds to a mini-batch of size one, with weight updates occurring with every example.

During the DNN training process, a typical weight will receive many thousands of increase requests and almost the same number of decrease requests. In a software implementation, these opposite-sign contributions cancel each other so only a small fraction of weights change substantially. Unfortunately, nonlinearities or other imperfections can cause conductance changes in NVM-based weights to be

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

strongly asymmetric[10], which prevents the opposite-sign contributions from cancelling properly. This update asymmetry is the most important source of poor DNN classification accuracy for NVM-based in-situ neural-network training, followed closely by limited dynamic range (too few conductance steps between the lowest and highest conductance)[11,12,14].

In the next section, we introduce a unit-cell with both more dynamic range and better update symmetry, thus making software-equivalent training accuracies possible despite the imperfections of existing NVM devices.

CMOS + PCM unit cell

We increase the dynamic range by using two pairs of conductance of varying significance (that is, of different numerical importance). This is implemented by applying different scale factors on the read current from the conductance pairs, so that the total read current per synapse is proportional to $F(G^+ - G^-) + (g^+ - g^-)$, where F is a small gain factor (we typically use $F = 3$; Extended Data Fig. 1), G^+ and G^- are the conductances of the higher-significance conductance pair, and g^+ and g^- are the conductances of the lower-significance pair. During training, only the lower-significance conductance pair is updated, using an open-loop weight-update procedure[10]. After some number of examples have been trained, weight transfer is initiated and the entire synaptic weight is transferred to the higher-significance conductance pair, after which g^+ and g^- are programmed to the same conductance ($g^+ - g^- = 0$). Because this is done only periodically (we typically use a transfer interval of 8,000 examples; Extended Data Fig. 1), while the algorithm is busy doing computations on device arrays that correspond to other layers of the neural network, there is sufficient time to perform this transfer process in a closed-loop, iterative fashion, similarly to previously used processes[28–30], resulting in accurate weight tuning.

A similar 'periodic carry' concept involving additional analogue-to-digital converters was shown to improve the expected performance of TaO_x memristors[22], as demonstrated by network simulations extrapolated from the measured characteristics of a small number of devices. However, training accuracy was still lower than network simulations based on similar extrapolations from the measured characteristics of a few LISTA[22] or ENODe[21] devices. This is because LISTA and ENODe devices, although exceedingly slow to program (from 6 ms for ENODe[21] to 1 s for LISTA[31]), provide a much more linear conductance update than does TaO_x[21,22].

PCM offers a slightly more linear conductance update[10] than filamentary RRAM devices[22], but less linearity than LISTA[22] or ENODe[21]. Our experience using two PCM-based conductance pairs of varying significance was similar to that described for TaO_x[22]: substantial asymmetry in the update of the lower-significance pair, together with a yield of less than 100% and non-ergodic array statistics, which few-device measurements frequently fail to capture, led to improved but not software-equivalent training accuracies.

One analogue memory device that offers extremely linear update characteristics is a CMOS transistor with a capacitor on its gate electrode[32]. The effective conductance of the transistor varies linearly with the voltage on the capacitor, which can be increased (decreased) gradually by briefly connecting current sources to the capacitor node to add (subtract) charge. However, this analogue conductance device is volatile—charge on the capacitor leaks away with a time constant of milliseconds or less. As a result, the neural-network training is tasked not only with improving the weights, but also with maintaining them despite their pervasive and exponential decay. Furthermore, the ideal target of about 1,000 resolvable analogue states[12] and of high linearity calls for as many as 12 transistors in the design of the two current sources[32], making the unit cell quite large. Finally, fabrication variations can easily cause the charge-addition circuitry in any given unit cell to be more effective than the charge-subtraction circuitry, while in the next unit cell the situation is reversed. This re-introduces substantial asymmetries in the conductance update, which again degrade training accuracy. But by using two such volatile analogue conductance

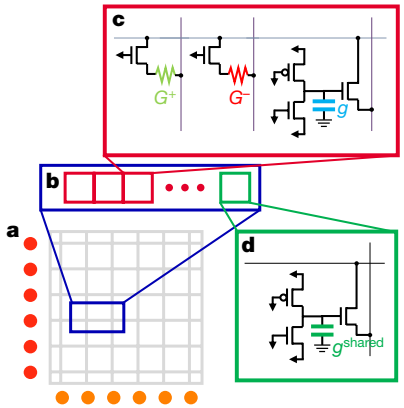


Fig. 2 | Schematic of an analogue-memory unit cell. a, **b**, Rows within each array block (**a**) contain both standard (red) and shared (green) unit cells (**b**). **c**, Each standard unit cell contains a higher-significance pair of PCM devices (labelled G^+ and G^-) and a volatile analogue conductance (labelled g). Horizontal (vertical) arrows indicate which signals are controlled from the same row (column). **d**, Each shared unit cell contains the other volatile analogue conductance (labelled g^{shared}) that completes each lower-significance conductance pair. In our experiments, the peripheral neurons and the g and g^{shared} devices are modelled in software using highly accurate SPICE simulations that include full device variability; the G^+ and G^- devices are real PCM devices programmed and measured on a 200-mm wafer.

circuit modules as the lower-significance pair (referred to as g^+ and g^-) together with a higher-significance non-volatile pair of PCM devices (referred to as G^+ and G^-), we combine the benefits of adjusting weights using cells with both a linear and symmetric response, while still retaining the long-term storage offered by the NVM.

Because the dynamic range of the synapse is only partly dependent on the CMOS cell, we can reduce the number of transistors to three. The volatile conductance (g) circuit module is composed of the read transistor, a PFET (p-type field-effect transistor) for adding charge and an NFET (n-type field-effect transistor) for subtracting charge (Fig. 2), resulting in a 3T1C ('3 transistor, 1 capacitor') circuit module. It can be programmed with shorter pulses and lower asymmetry than can the PCM, leading naturally to rapid training with better characteristics. In contrast to PCM devices, for which the only available gradual programming is conductance increases through partial crystallization, each g device can be programmed bi-directionally. We can gradually increase the weight by adding some charge to the capacitor through short pulses on the PFET, or decrease the weight similarly through the NFET. Therefore, unlike PCM-based conductance pairs, we do not need to independently program a g^- device to tune the weights. We do need a reference current to support negative weight contributions, but this device (g^{shared}) can be shared among many unit cells. We adopted one g^{shared} device—implemented here with three 3T1C units in parallel to reduce variability—for every 128 unit cells. Not counting these g^{shared} components, each dedicated synaptic unit cell contains five transistors, two PCM devices and one capacitor.

During training, only the lower-significance conductance g is updated bi-directionally until it is time for a weight transfer. The length of a transfer interval is chosen to balance the saturation of the capacitor, the leakage of the capacitor and the costs of doing a transfer (in time, energy and accuracy due to incomplete weight transfer). In an eventual chip implementation, transfer would be performed one column (or row) at a time, with weight information transferred from the $g - g^{shared}$ conductance pair to the $G^+ - G^-$ PCM pair.

However, we still have a problem, owing to unavoidable random variations in CMOS devices (such as local dopant fluctuations). For instance, a 3T1C device with a PFET that is more effective than its NFET will tend to report weight increases at every transfer interval. Because we transferred all of the weight from the $g - g^{shared}$ conductance pair, we can choose to invert the effective polarity of this conductance

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

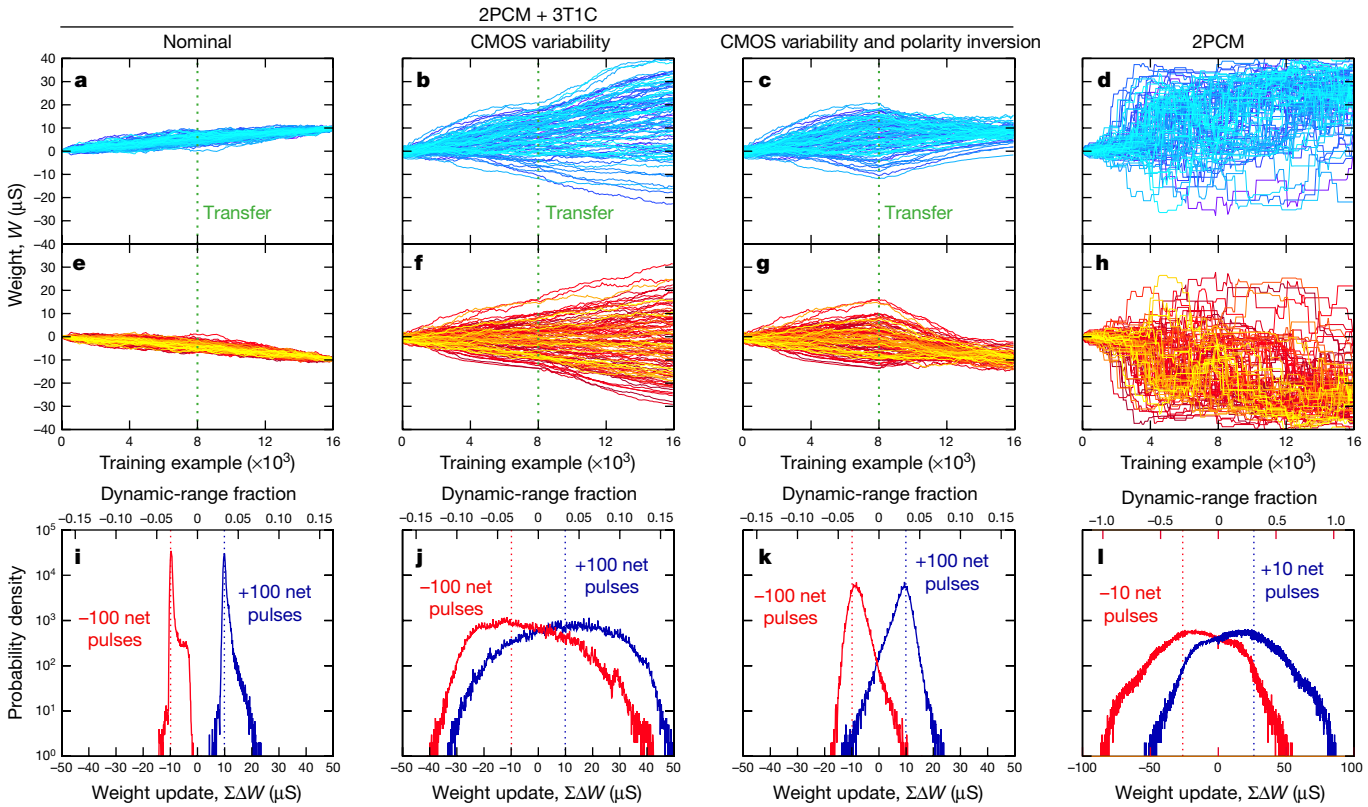


Fig. 3 | Simulated response of different unit cells to nearly offsetting weight-update requests. a–l, Simulation results showing the open-loop conductance-update behaviour of four NVM-based synapses: 2PCM + 3T1C with perfectly nominal CMOS devices (**a**, **e**, **i**), 2PCM + 3T1C with real CMOS variability but no polarity inversion (**b**, **f**, **j**), 2PCM + 3T1C with real CMOS variability and polarity inversion (**c**, **g**, **k**) and 2PCM (**d**, **h**, **l**). **a–c**, Effect of the application of 1,000 update pulses, 550 up and 450 down, randomly permutated and distributed over 16,000 examples (other pulses equal to zero). **d**, The effect of 50 pulses, 30 up and 20 down, distributed over 16,000 examples. Different colours in **a–d** show 100 different random permutations. **e–h**, Results of applying 450 up and 550 down pulses for 2PCM + 3T1C (**e–g**) or 20 up and 30 down (**h**). These pulse sequences are based on MNIST results and reveal typical neural-network requests over 16,000 training examples (an equivalent period of two transfer intervals for 2PCM + 3T1C, Extended Data Fig. 2). For **a–c** and **e–g**, weight transfer occurs after 8,000 examples (indicated by the vertical green dashed lines) and at the end; for **d** and **h**, occasional reset[10] is performed every 100 examples. Polarity inversion strongly reduces the effect of CMOS variability. **i–l**, Distribution of the resulting weight update for 30 initial conditions (10,000 random permutations at each) across the entire dynamic range (−40 μS to +40 μS), not only the initial condition ($W = 0$) shown in **a–h**. The dashed vertical lines show the precise aggregate change in weight that the backpropagation algorithm was ideally seeking with these particular pulse sequences. Tighter distributions correlate well with higher training accuracies (see Figs. 4–6). For **i–k** and **l**, the dynamic-range fraction represents the portion of the dynamic range covered by weights for the application of ±100 pulses (from a total of 1,000 pulses) and of ±10 pulses (from a total of 50 pulses), respectively. Note that the neural network attempts to compensate for the larger steps in the 2PCM synapses by asking for fewer of them (for example, the optimal learning rate is lower).

pair until the next transfer operation. This involves switching to the equation $F(G^+ - G^-) - (g - g^{\text{shared}})$ while reading device currents (for both forward inference and backpropagation), using the PFET to add charge when decreasing the weight and the NFET to subtract charge when increasing the weight. After each transfer interval, we invert the polarity used for $g - g^{\text{shared}}$ during the subsequent training cycle. (See Methods for details.)

In Fig. 3 we compare the operation of four NVM-based synapses when implementing one of the demanding sequences of programming pulses that occur during neural-network training. Matched simulations were performed for a '2PCM + 3T1C' unit cell (which contains two PCM devices, each with its own selection transistor, and one 3T1C circuit module) with perfectly nominal CMOS devices (Fig. 3a, e, i), for 2PCM + 3T1C with full CMOS variability (Fig. 3b, f, j), for 2PCM + 3T1C with CMOS variability and polarity inversion (Fig. 3c, g, k) and, for comparison, two PCM devices without the 3T1C ('2PCM') (Fig. 3d, h, j). First, we used MNIST neural-network training simulations to find the correlation between what the network asks for and what the network actually gets (in terms of weight updates). The former is the effective number of weight-change pulses (or 'net' pulses) over some time interval, for example, the number of weight-increase requests less the number of weight-decrease requests; the latter is the actual number of pulses fired and the resulting weight change (Extended Data Fig. 2). We picked one example combination of a net change of ±100 pulses obtained by firing exactly 1,000 pulses for the 2PCM + 3T1C cell (Fig. 3a–c, e–g) and of ±10 pulses obtained from exactly 50 pulses for 2PCM (Fig. 3d, h). In these simulations, we randomly distribute these 1,000 or 50 pulses across a window of 16,000 training examples (representing two transfer intervals). In Fig. 3i–l, we show weight-change statistics gathered over all possible initial weight conditions.

Each of these pulse sequences should result in exactly the same net number of weight-update pulses and therefore exactly the same weight change (dashed lines in Fig. 3i–l). Although nominal CMOS 3T1C devices provide clearly separated weight increases and decreases (Fig. 3a, e), CMOS variability strongly broadens the weight change (Fig. 3b, f). This occurs because a 3T1C cell in which the PFET is stronger than its NFET always favours weight increases over every transfer interval. By forcing that strong PFET to be responsible for weight decreases in every second transfer interval, a longer-term balance can be restored. A similar argument holds for the cells with a strong NFET. Figure 3c, g illustrates how, across multiple transfer intervals, 'transfer with polarity inversion' cancels out these undesired weight changes that are induced by fixed device asymmetry.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

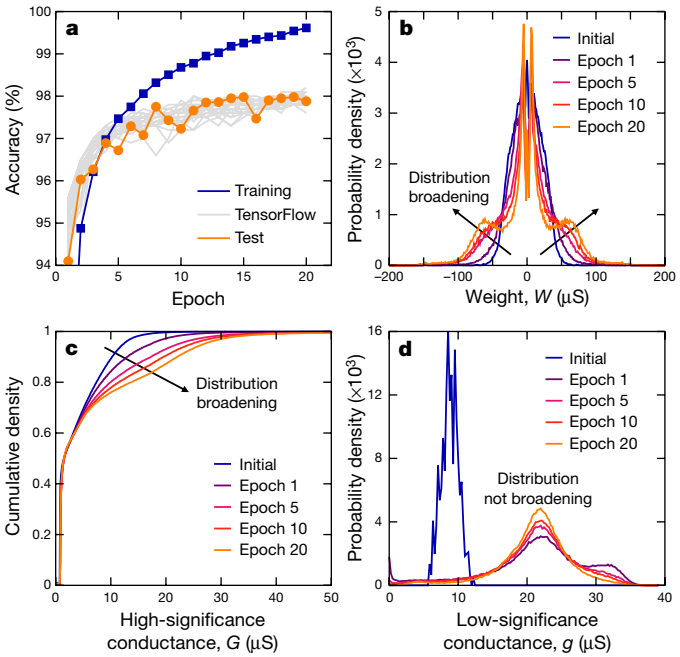


Fig. 4 | Mixed hardware–software results on the MNIST dataset. **a**, Training (blue) and test (orange) accuracies for our mixed hardware–software experiment, which combines hardware-based PCM devices and SPICE-modelled 3T1C devices with full CMOS variability, on the MNIST dataset closely match those achieved for the same size network using TensorFlow (grey). The TensorFlow curves correspond to ten different initial network conditions and sequencing of training images, illustrating the modest run-to-run deviations that are inherent to neural-network training. **b**, **c**, The distribution of weights for the initial state and epochs 1, 5, 10 and 20 (**b**), and the cumulative distributions of all array conductances (G^+ and G^- together; **c**). Half of the G^+ (G^-) values are almost zero, corresponding to negative (positive) weights. **d**, Initial g distribution and successive distributions just before weight transfer to the hardware-based PCM devices.

In comparison to the 3T1C-based synapses, 2PCM synapses offer much less dynamic range.

Although Fig. 3 demonstrates that both the 2PCM + 3T1C concept and transfer with polarity inversion should lead to better training accuracies, it is extremely difficult to specify how tight these distributions should be to achieve a given training accuracy. It is therefore critical to perform actual neural-network training experiments, which we implement below with real PCM devices. This enables us to demonstrate that transfer with polarity inversion results in high training accuracy even in the presence of substantial variability of the CMOS device.

Mixed hardware–software implementation

We conducted mixed hardware and SPICE model experiments using PCM device arrays as hardware (identical to those used previously[10]) and realistic software-based CMOS device models, including highly accurate modelling of the variability of the CMOS device. Training within a transfer interval was performed in software, including forward propagation, backpropagation and weight update of the modelled 3T1C devices. Each software synapse contains the measured conductances from two PCM devices, the instantaneous voltage of the software-modelled capacitor and the indices for four SPICE models (one for charge addition, one for charge subtraction, one for the read transistor and one for the aggregate charge leakage) that encapsulate full CMOS variability. (See Methods for details; in Extended Data Fig. 1 we compare a fully hardware implementation to the mixed software–hardware experiment demonstrated here.)

To speed up training, we applied an 'example triage' method after each forward propagation to decide whether to perform the backpropagation step. If the network was already 'good' at classifying the example, then training was skipped. This helped to reduce the number of examples trained and markedly decreased the wall-clock time of the experiment. By the end of training, approximately 78% of the training examples in the MNIST, 5% in the MNIST-backrand, 47% in the CIFAR-10 and 13% in the CIFAR-100 datasets were being skipped.

Results

We performed a series of experiments on different benchmark datasets to compare the performance of networks based on our unit cell against the accuracies obtained with TensorFlow, a widely used machine-learning toolkit (https://www.tensorflow.org). Our goal was to achieve comparable test accuracy against software baselines that use exactly the same network size as our experiment, but take advantage of software techniques such as unbounded rectified linear unit activation function and cross-entropy training, and optimizers such as AdaGrad, ideal momentum and ADAM[33]. All experiments described below incorporate the full extent of variability and non-ideality seen in the PCM devices and the SPICE-simulated 3T1C cells.

We first tested our NVM-based unit cell on the MNIST dataset[23], which is composed of 60,000 training and 10,000 test images of handwritten digits, cropped to 22 × 24 pixels. We trained a four-layer network with 528 neurons (plus 1 bias neuron) in the first layer, 250 (plus 1 bias) in the second, 125 (plus 1 bias) in the third and 10 in the fourth (denoted as 528-250-125-10), involving 164,885 weights and therefore 329,770 PCMs. Training and test accuracies over 20 epochs are shown in Fig. 4a, compared against the fully software simulations performed in TensorFlow for the same network. Experimental results closely match the mean software test accuracy of 97.94%.

To track the behaviour of weights, in Fig. 4b we show the evolution of the weight distribution, combining contributions from higher-significance PCM conductances and lower-significance SPICE-simulated $g - g^{\text{shared}}$ conductances. We also show the PCM conductance distributions during training (Fig. 4c) and the distributions for g during training at the instant before a weight-transfer operation (Fig. 4d). As training proceeds, weight distributions broaden, as reflected in the broader cumulative distributions of the G values (Fig. 4c). However, distributions of g at later epochs do not differ from each other (Fig. 4d), meaning that weight evolution is implemented mainly by the PCM devices. The role of g is then to support the proper tuning of PCM rather than to encode important long-term information.

The MNIST-backrand dataset poses a much more difficult recognition problem than does MNIST, by adding uniform random noise to each handwritten-digit image (Fig. 5a) and by providing fewer training examples (12,000) but requiring generalization across many more test examples (50,000). We trained on a network comprising 784-180-125-10 neurons. Our NVM-based experimental accuracy of 82.13% is only slightly below the TensorFlow accuracy of 83.3% (Fig. 5b).

State-of-the-art image classification systems use a combination of convolution layers that act as feature extractors with one or more fully connected classification layers. Although our work on efficiently mapping convolution networks to analogue array architectures is ongoing, a well-known approach (transfer learning[34]) repurposes convolution layers that are pre-trained on one dataset for new datasets, by re-training only the last fully connected classification layers. Here, we use the weights from the Google Inception-v3 network (https://github.com/tensorflow/models/tree/master/research/inception) (more than 70 layers) trained on ImageNET[35] and re-train a single software–hardware fully connected layer on either the CIFAR-10 or CIFAR-100 dataset.

We used an image re-training script (https://www.tensorflow.org/tutorials/image_retraining) to rescale the 32 × 32 CIFAR images to the 299 × 299 ImageNET input size and to compute the 2,048 neuron excitations at the input of the fully connected layer of Inception-v3 using forward inference. These vectors were then used to train a 2,048-10 fully connected network with two neuron layers with CIFAR-10 labels and a 2,048-100 network with CIFAR-100 labels. Our experimental results are again compared against software-based (TensorFlow) training of these two networks.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

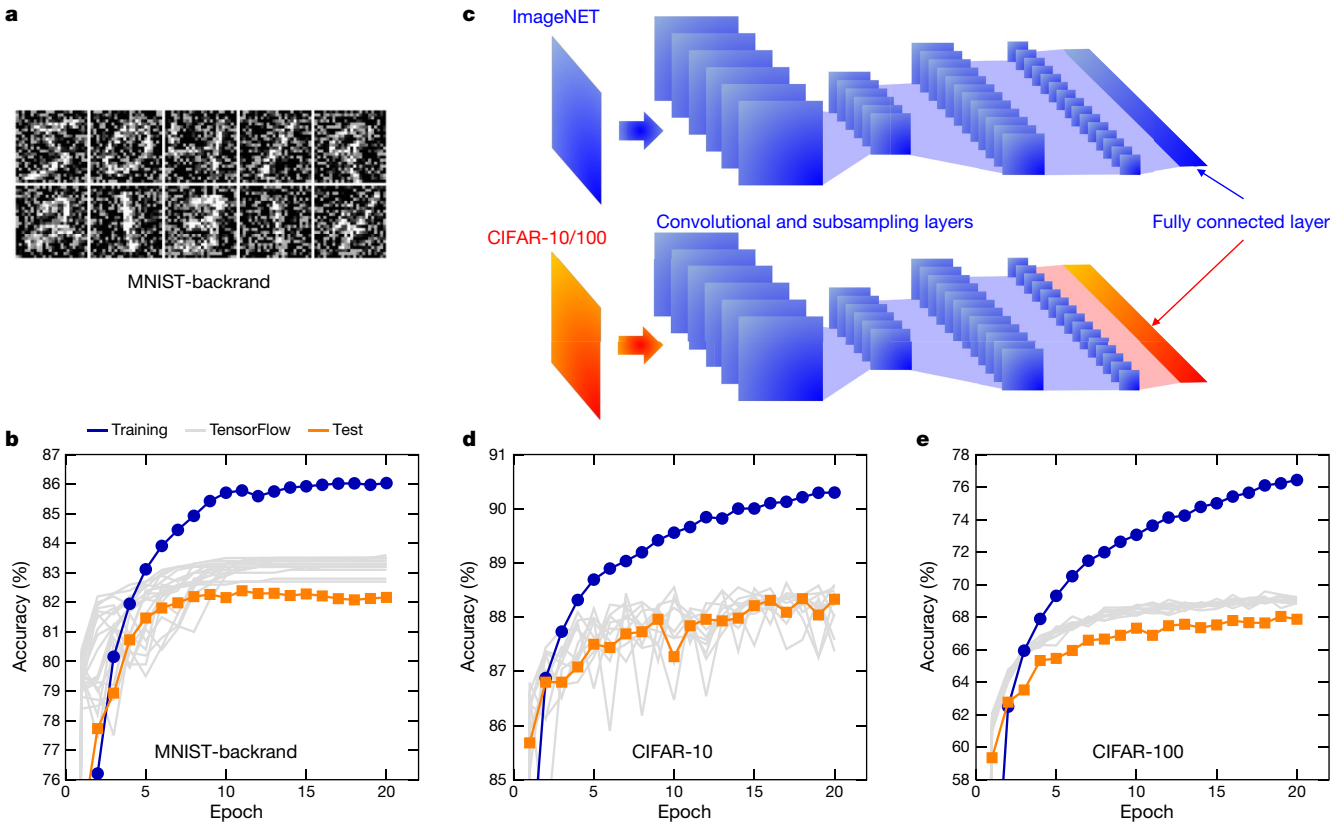


Fig. 5 | Mixed hardware–software results on the MNIST-backrand and CIFAR-10/100 datasets. a, **b**, When trained on the MNIST-backrand dataset, which consists of handwritten digits plus uniform random noise[23] (**a**), our mixed hardware–software experiment shows similar test accuracies to fully software-based training using TensorFlow (**b**). **c**, For our transfer-learning experiments, the weights from the Inception-v3 network as originally trained on ImageNET (top) are used to learn CIFAR-10/100 images by re-training only the last fully connected layer (bottom). **d**, **e**, The experimental test accuracies for CIFAR-10 (**d**) and CIFAR-100 (**e**) closely match the expected results from software-based (TensorFlow) training. See Extended Data Fig. 3 for corresponding experimental cumulative distribution functions.

Training and test accuracies are shown in Fig. 5d for CIFAR-10 (2048-10 neurons, 20,490 weights) and in Fig. 5e for CIFAR-100 (2048-100 neurons, 204,900 weights, corresponding to 409,800 PCM devices). Accuracy is equivalent for CIFAR-10, whereas CIFAR-100 shows a small difference of just over 1% (Extended Data Fig. 3 shows weight and PCM conductance distributions for MNIST-backrand and CIFAR-10/100 experiments). In Fig. 6a we summarize all of our experimental results against the expectations of software-based (TensorFlow) training.

In Fig. 6b we provide a detailed comparison of the MNIST test accuracy after 20 epochs for different unit cells according to a matched simulator that mimics our PCM hardware[10], as compared to our final mixed hardware–software experiment (Fig. 4 and unfilled bar in Fig. 6b). Adding the 3T1C devices to the flawed 2PCM devices improves the accuracy (purple bar in Fig. 6b) until we consider the strong effect of real CMOS variability (left-most orange bar). As predicted by Fig. 3, polarity inversion greatly reduces the undesirable effects of CMOS variability (red bar in Fig. 6b). We also use this matched simulator to determine how training would have proceeded if we had left out only one of the techniques that we used in the full experiment (orange bars; see also Methods and Extended Data Fig. 4).

Finally, we estimated the average power required to fully process a single MNIST example on the 528-250-125-10 network. We include dissipated power for one full forward and reverse pass, the associated weight updates and a prorated number of occasional reset events and/or transfers per example, on the basis of the activity levels observed within our experiments and on circuit simulations of a full mixed-signal design in 90-nm node. We do not attempt to include the input–output power associated with bringing example and label data onto a chip. Separate estimates were performed for the 2PCM and 2PCM + 3T1C designs, including power consumption in the three crossbar arrays, and in the peripheral analogue and digital circuitry (see Methods for details).

Under the assumption that an MNIST example can be processed in 240 ns, the average power consumption for the 2PCM + 3T1C design was calculated to be 54 mW, compared to 22 mW for the PCM design. The increase in power of a factor of 2.5 is primarily due to the large forward- and reverse-propagate currents in the 3T1C array, wherein each cell could contribute several microamps of current. To compare to a modern GPU, the training of a single example on our small network performs 362,405 multiply–accumulate operations and requires 12.9 nJ of power for a computational energy efficiency of 28,065 GOP s^{-1} W^{-1}. At the specified processing time of 240 ns, and extrapolating our 90-nm circuit designs to 14 nm, the throughput per unit area is 3,582 GOP s^{-1} mm^{-2}. For comparison, a Tesla V100 GPU offers[36] 30.0 trillion floating-point operations per second (30.0 TFLOPs) of 16-bit floating-point numbers in a footprint of 300 W and 815 mm^2, or 100 GOP s^{-1} W^{-1} and 37 GOP s^{-1} mm^{-2}. Therefore, our analogue NVM-based approach could potentially provide more than two orders of magnitude in energy efficiency while accelerating the backpropagation algorithm for fully connected layers by nearly two orders of magnitude.

We have established that our approach can also deliver software-equivalent training accuracies, despite the imperfections of existing analogue memory devices. The next steps will be to demonstrate this same software equivalence on larger networks that require large fully connected layers—such as recurrently connected long short-term memory[37] and gated recurrent networks[38]—and to design, implement and refine these analogue techniques on prototype NVM-based hardware accelerators.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

a

	Accuracy (%)			
	MNIST	MNIST-backrand	CIFAR-10	CIFAR-100
Training	99.61	86.03	90.3	76.43
Test	97.95	82.13	88.29	67.96
TensorFlow	97.94	83.30	88.16	69.02

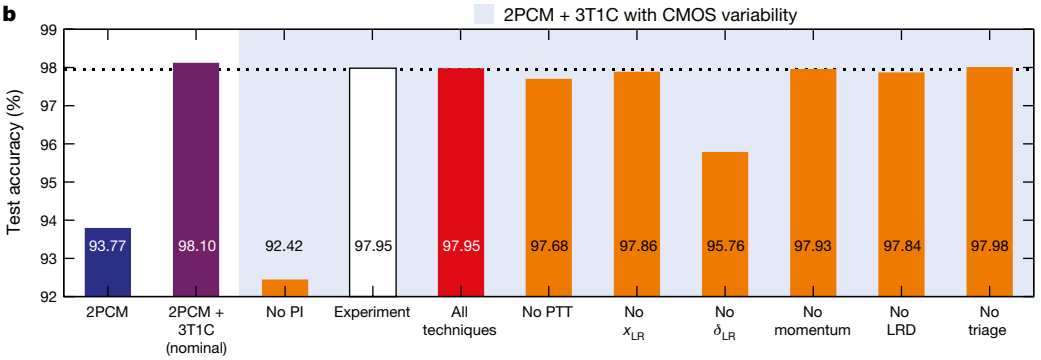


Fig. 6 | Accuracy comparison and effect of different techniques. **a**, Training and test accuracies from our mixed hardware–software experiments on different datasets are compared to expected results from purely software-based (TensorFlow) training. **b**, Using a matched simulator that mimics how our PCM hardware trains on the MNIST dataset, we evaluate the relative effect of the various imperfections present in our experiments and of each mitigation technique that we used within our final experiment by removing that technique during simulated training (orange bars). The matched simulator when no techniques are removed (red bar) closely matches our experimental results (unfilled bar and dotted horizontal line). The accuracy with only our flawed PCM devices (2PCM) would be insufficient (blue bar); although nominal 3T1C devices could improve this greatly (purple bar), the effect of CMOS variability (bars on light-blue background) is severe without polarity inversion (PI; left-most orange bar). Additional techniques that are important include post-transfer tuning (PTT) of g after weight transfer and modulation of upstream (x_{LR}) and downstream (δ_{LR}) weight-update pulses (see Methods and Extended Data Fig. 4). Other techniques, which are less relevant include momentum, learning-rate decay (LRD) and triage.

We anticipate that the 3T1C cell proposed here will eventually be phased out in favour of a more compact, modestly linear NVM, and that these NVM devices will be stacked above each other in the metal back-end using non-silicon access devices[39]. Although the periodic carry concept[22] has already relaxed device requirements for effective dynamic range, our approach for multiple significant device pairs further relaxes and differentiates these requirements. Our approach calls for a new type of analogue memory device that can provide high linearity and endurance, but that does not need long retention, a huge resistance window or even tightly constrained device-to-device variability. For the higher-significance conductance-pair, we need an NVM device that offers good retention and fast, low-power programming for high-precision closed-loop tuning, but we do not need to impose requirements on ultra-linear conductance update, high endurance or wide dynamic range as well.

Therefore, there remains a strong incentive to develop compact NVM devices that are capable of gentle, symmetric conductance change[12,13]. Even in such an advanced design, we anticipate that the technique used here (and proposed independently elsewhere[22]) of using multiple conductances of varying significance will be necessary to synthesize a synapse with high dynamic range from individual devices of lower dynamic range, and that polarity inversion on transfer (introduced here to suppress the highly undesirable effects of fixed device asymmetries) will be essential.

Online content

Any Methods, including any statements of data availability and Nature Research reporting summaries, along with any additional references and Source Data files, are available in the online version of the paper at https://doi.org/10.1038/s41586-018-0180-5.

Received: 26 January 2018; Accepted: 29 March 2018;
Published online 6 June 2018.

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
2. Coates, A. et al. Deep learning with COTS HPC systems. In *Proc. 30th International Conference on Machine Learning* 1337–1345 (Association for Computing Machinery, 2013).
3. Gupta, S., Agrawal, A., Gopalakrishnan, K. & Narayanan, P. Deep learning with limited numerical precision. In *Proc. 30th International Conference on Machine Learning* 1737–1746 (Association for Computing Machinery, 2015).
4. Merolla, P., Appuswamy, R., Arthur, J., Esser, S. K. & Modha, D. Deep neural networks are robust to weight binarization and other non-linear distortions. Preprint at https://arxiv.org/abs/1606.01981 (2016).
5. Nurvitadhi, E. et al. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *Proc. 2017 ACM/SIGSA International Symposium of Field-Programmable Gate Arrays* 5–14 (Association for Computing Machinery, 2017).
6. Jouppi, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. 2017 International Symposium on Computer Architecture* 1–12 (Association for Computing Machinery, 2017).
7. Merolla, P. A. et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **345**, 668–673 (2014).
8. Esser, S. K. et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl Acad. Sci. USA* **113**, 11441–11446 (2016).
9. Morie, T. & Amemiya, Y. An all-analog expandable neural network LSI with on-chip backpropagation learning. *IEEE J. Solid-State Circuits* **29**, 1086–1093 (1994).
10. 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. In *2014 IEEE International Electron Devices Meeting* T29.5 (IEEE, 2014).
11. Burr, G. W. et al. Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element. *IEEE Trans. Electron Dev.* **62**, 3498–3507 (2015).
12. Gokmen, T. & Vlasov, Y. Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Front. Neurosci.* **10**, 333 (2016).
13. Burr, G. W. et al. Neuromorphic computing using non-volatile memory. *Adv. Physics X* **2**, 89–124 (2017).
14. Yu, S. et al. Scaling-up resistive synaptic arrays for neuro-inspired architecture: challenges and prospect. In *2015 IEEE International Electron Devices Meeting* 17.3 (IEEE, 2015).
15. Gao, L. et al. Fully parallel write/read in resistive synaptic array for accelerating on-chip learning. *Nanotechnology* **26**, 455204 (2015).
16. Prezioso, M. et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015).
17. Jang, J.-W., Park, S., Burr, G. W., Hwang, H. & Jeong, Y.-H. Optimization of conductance change in $Pr_{1-x}Ca_xMnO_3$-based synaptic devices for neuromorphic systems. *IEEE Electron Device Lett.* **36**, 457–459 (2015).
18. Jeong, Y. J., Kim, S. & Lu, W. D. Utilizing multiple state variables to improve the dynamic range of analog switching in a memristor. *Appl. Phys. Lett.* **107**, 173105 (2015).
19. Kaneko, Y., Nishitani, Y. & Ueda, M. Ferroelectric artificial synapses for recognition of a multishaded image. *IEEE Trans. Electron Dev.* **61**, 2827–2833 (2014).

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

20. Nandakumar, S. R. et al. Mixed-precision training of deep neural networks using computational memory. Preprint at https://arxiv.org/abs/1712.01192 (2017).
21. van de Burgt, Y. et al. A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing. *Nat. Mater.* **16**, 414–418 (2017).
22. Agarwal, S. et al. Achieving ideal accuracies in analog neuromorphic computing using periodic carry. In *2017 Symposium on VLSI Technology* T13.2 (IEEE, 2017).
23. Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
24. Krizhevsky, A. *Learning Multiple Layers of Features From Tiny Images*. Ch. 3, https://www.cs.toronto.edu/~kriz/cifar.html (2009).
25. Narayanan, P. et al. Towards on-chip acceleration of the backpropagation algorithm using non-volatile memory. *IBM J. Res. Develop.* **61**, 11 (2017).
26. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by backpropagating errors. *Nature* **323**, 533–536 (1986).
27. Xu, Z. et al. Parallel programming of resistive cross-point array for synaptic plasticity. *Procedia Comput. Sci.* **41**, 126–133 (2014).
28. Papandreou, N. et al. Programming algorithms for multilevel phase-change memory. In *2011 IEEE International Symposium on Circuits and Systems* 329–332 (IEEE, 2011).
29. Alibart, F., Gao, L., Hoskins, B. D. & Strukov, D. B. High-precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* **23**, 075201 (2012).
30. Hu, M. et al. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Proc. 53rd Annual Design Automation Conference* 19 (Association for Computing Machinery, 2016).
31. Fuller, E. J. et al. Li-ion synaptic transistor for low power analog computing. *Adv. Mater.* **29**, 1604310 (2017).
32. Kim, S., Gokmen, T., Lee, H.-M. & Haensch, W. E. Analog CMOS-based resistive processing unit for deep neural network training. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems* 422–425 (IEEE, 2017).
33. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* Ch. 8 (MIT Press, 2016).
34. Donahue, J. et al. DeCAF: a deep convolutional activation feature for generic visual recognition. In *Proc. 31st International Conference on Machine Learning* 647–655 (Association for Computing Machinery, 2014).
35. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. Rethinking the Inception architecture for computer vision. Preprint at https://arxiv.org/abs/1512.00567 (2015).
36. Mujtaba, H. Nvidia Volta GV100 12nm FinFET GPU detailed – Tesla V100 specifications include 21 billion transistors, 5120 CUDA cores, 16 GB HBM2 with 900 GB/s bandwidth. *Wccftech* https://wccftech.com/nvidia-volta-gv100-gpu-tesla-v100-architecture-specifications-deep-dive/ (2017).
37. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput.* **9**, 1735–1780 (1997).
38. Cho, K., van Merrienboer, B., Bahdanau, D. & Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. Preprint at https://arxiv.org/abs/1409.1259 (2014).
39. Burr, G. W. et al. Access devices for 3D crosspoint memory. *J. Vac. Sci. Technol. B* **32**, 040802 (2014).

Acknowledgements We acknowledge management support from B. Kurdi, C. Lam, W. Wilcke, S. Narayan, T. C. Chen, W. Haensch, R. Divakaruni, J. Welser and D. Gil, and discussions with P. Solomon, S. Kim, A. Sebastian, K. Hosokawa and S. C. Lewis. This work was performed as part of the 'Neuromorphic Devices & Architectures' project under the auspices of the IBM Research Frontiers Institute (https://www.research.ibm.com/frontiers). We acknowledge advice and support from H. Riel, S. Gowda, D. Maynard and the member companies of the IBM RFI.

Reviewer information *Nature* thanks G. C. Adam, R. Legenstein and the other anonymous reviewer(s) for their contribution to the peer review of this work.

Author contributions G.W.B. developed the multiple-conductances-of-varying-significance and polarity-inversion techniques; P.N. and G.W.B. designed the 3T1C unit cell; G.W.B., R.M.S., C.d.N., I.B. and P.N. developed the neural-network simulation software; R.M.S., I.B., C.d.N., S.S., M.B., N.C.P.F. and S.A. used the simulator to develop insights key to the success of the experiment; G.W.B. designed and S.A. extended the experimental apparatus; S.A. performed the experiments; H.T. designed the transfer learning experiment and performed the TensorFlow software training; P.N., G.W.B. and S.A. developed the SPICE modelling approach; P.N. performed the power analysis; M.G., S.A. and G.W.B. developed the triage approach used in the experiment; and all authors contributed to the writing and editing of the manuscript.

Competing interests The authors declare no competing interests.

Additional information
Extended data is available for this paper at https://doi.org/10.1038/s41586-018-0180-5.
Reprints and permissions information is available at http://www.nature.com/reprints.
Correspondence and requests for materials should be addressed to G.W.B.
Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

METHODS

Analogue-memory-based neural networks. The crossbar-memory approach (Fig. 1) is most efficient for fully connected layers. Training of convolutional layers[23], in which many neurons share and re-use a small set of weight kernels, is much less straightforward to implement using these crossbar-array techniques. Such networks also pose reduced memory-size and memory-bandwidth demands on conventional digital hardware. Given the reduced memory-bandwidth demands, the recent trend in convolutional networks has been to replace fully connected layers with convolutional layers. Interestingly, the fully parallel computation offered by an analogue crossbar-memory approach would call for exactly the opposite strategy—restricting the number of convolutional layers to the absolute minimum necessary and using transfer learning when possible, especially for the first few layers where the depth dimension is much smaller than the input size. By contrast, for forward inference only, the analogue-memory approach can deliver high speed (TOP s^{-1}) and good energy efficiency (TOP s^{-1} W^{-1}) for convolutional layers, by replicating kernel weights across multiple large arrays (with some reduction in TOP s^{-1} mm^{-2}).

Example triage. Example triage is a technique for accelerating any supervised learning algorithm of DNNs by skipping over training examples that the network already 'knows' relatively well, as quantified by the 'safety margin'. The safety margin is the difference between the excitation level of the correct output neuron and the highest excitation level of the other output neurons (Extended Data Fig. 5). A positive safety margin denotes a correct classification and a negative safety margin denotes an incorrect classification. After forward inference, each example is assigned a focus probability, depending on the safety margin and the stage of training (Extended Data Fig. 5). Any training example with a safety margin below 'acceptable safety margin' is included for training with 100% probability. As the network trains, example triage selects fewer images for backpropagation, as shown by cumulative distributions of the safety margin (Extended Data Fig. 6). Thus, the acceleration boost is large when training accuracy is high (our 528-250-125-10 network trained on MNIST), but smaller when training accuracy is low (our 784-180-125-10 network trained on MNIST-backrand).

Neural-network implementation. Although we use a logistic squashing function at each neuron layer in these experiments, we have previously shown that piecewise-linear or 'hard' logistic functions are effectively equivalent[40]. In the final layer, we train against the raw difference between ground truth and the output activation, but omit the derivative of the logistic function. This secures the primary benefit of training against cross-entropy loss without the need for expensive softmax exponential functions to explicitly estimate cross-entropy. In our experiments and simulations, no discrepancy in test accuracy could be traced to this minimization choice. Our multiple-conductances-of-varying-significance concept, although similar in high-level description to previous work[22], was developed independently and therefore provides some very different characteristics. Because we weight the read currents in the analogue rather than the digital domain, we do not need to increase the number of analogue-to-digital converters. (Compared to the array itself, analogue-to-digital converters are particularly power- and area-inefficient.) Our circuitry for integrating charge and for transferring excitation data to the next layer is completely independent of the number of conductances encoding the weights. Finally, as mentioned in the main text, by introducing the concept of using a different device for each tier of significance, we can relax the device constraints on these memory elements much further than with the previous approach[22].

Hardware set-up. A single array diagnostic monitor on a 200-mm wafer was contacted with a 94-pin probe card mounted on a Cascade microprobe station, enabling access to a 512 × 1,024 1T–1PCM device array (180-nm technology). A 10-bit digital bus selects one of 1,024 rows; a second 9-bit bus selects which of the 512 active columns connects to a single master bitline connected to the single read/write head. Both the probe station and the Nextest Magnum 2EV tester were controlled by a single computer running custom neural-network software, the tester code for accessing the PCM arrays through the Magnum and the Matlab interface code for connecting these two software components. Each experiment uses a single large contiguous rectangular block of devices from the entire array, within which all synapses were assigned in order. No remapping to avoid dead or defective devices was performed. Typically, 0.01% of the PCM devices were stuck 'on' (persistently remained in conductance states higher than 50 μS) and 1% of the devices were stuck 'off' (persistently remained in conductance states lower than 0.5 μS). In this mixed hardware–software experiment, we serially read the array, load the values into the software and perform the summation in software. However, in an eventual chip implementation, the vector–matrix multiplication would be completely parallel within each array core.

Before beginning an experiment, weights were initialized to a uniform distribution of raw conductances between 1 μS and 12 μS. The neural-network software reads the PCM array and maps measured conductances into weights within software memory by multiplying the conductance differences by $F = 3.0$. Roughly 8,000 images are then trained fully in software, including reads and writes of each 3T1C device (from 0 to 7 programming pulses of 300 ps each) according to its set of three SPICE variability models: one for charge addition, one for charge subtraction and one for voltage-to-conductance mapping.

The software keeps track of incurred time (80 ns per forward propagation; 160 ns per backpropagation/weight-update step), decays each capacitor voltage (5.16-ms time constant) and triggers a transfer event after every 1.92 ms of incurred time. The total weight information trained onto each software-based 3T1C device is used to compute a new target weight for the hardware-based PCM devices. After an initial reset pulse, the hardware-based PCM conductances are tuned using an iterative sequence of partial-set pulses varying between 13.6 ns and 4.4 μs. Given the serial hardware read/write, the wall-clock time between two transfer events is 7–20 min, depending on the size of the network and the corresponding number of PCM devices used. PCM drift[41] is present, and contributes to inaccurate PCM conductance tuning, but on a timescale different from that of the internal clock of the software training. After hardware tuning is complete, all hardware-based PCM conductances are measured and reported to the software, which then completes the transfer process. The operational polarity of the software-based 3T1C conductance g is inverted. Because the PCM programming operation is inherently noisy, the transferred weight is different from the expected weight. To minimize this error, post-transfer tuning is performed: each g device is programmed using a small number of 300-ps pulses to correct any residual weight error left after hardware-based PCM tuning.

Weight transfer. Weight transfer converts large differences in the lower-significance conductance pair into smaller changes in the higher-significance conductance-pair, allowing the network to protect larger weights effectively from the effects of nonlinearity and asymmetry. Any weight residue that cannot be transferred completely to the higher-significance conductance pair can be restored to the lower-significance conductance pair. An eventual chip implementation could implement weight transfer on a column-by-column basis, implemented while the chip is executing a later (or earlier) portion of the network. However, because the hardware described above is most efficient when performing the same operation sequentially (a read comparison or a write operation) on all devices within the array, here weight transfer is performed on the entire array in the same PCM tuning process.

Given the target PCM weights (supplied by the neural-network software in units of raw PCM conductance, after dividing by $F = 3.0$), the closed-loop PCM conductance-tuning operation begins with a measurement of all conductances. Any weights already within ±240 nS of their targets (typically 2% of the array) are removed from consideration. All remaining devices are first reset with three high-amplitude 500-ns pulses. PCM devices corresponding to G^+ (for positive weights) or G^- (for negative weights) are then programmed with a series of eight SET rampdown pulses and measured. The initial pulse sequence contains eight 50-ns steps with decreasing amplitude. If the conductances obtained after a write operation result in a PCM weight within ±1.2 μS of its target, then we stop programming that device (the verify stage of the closed-loop tuning); otherwise, we reset that PCM device again (three high-amplitude 500-ns pulses) and try another pulse length. At each stage, many PCM devices are removed from consideration (for the remainder of that transfer operation), so that by the last iteration only a small number of devices are being programmed and measured. Eight-step SET rampdowns ranged from a step-width of 1.7 ns to 550 ns, with a median of 50 ns. Extended Data Fig. 7 shows cumulative conductance distributions after these various set pulses are applied to freshly reset devices. Rampdowns with fewer steps show similar results. At each iteration, we choose a new pulse-step duration on the basis of measured conductances and previously used pulses, tracking the shortest duration that was too long and the longest duration that was too short. This bisectional search results in a maximum of six pulse sequences on any given device, with 70% of devices finishing after three iterations. Extended Data Figs. 8 and 9 show histograms and correlation maps for various quantities before and after the transfer operation, taken from the last transfer operation during the MNIST experiment (Fig. 4).

Even though the volatile 3T1C conductances are critical to reach software-equivalent accuracies, the non-volatile PCM devices are more than sufficient for preserving the trained weights permanently. After performing a typical transfer onto PCM devices without any post-transfer tuning (and thus no contribution from 3T1C devices), test accuracy for the MNIST array after epoch 18 slips from 97.98%, but to only 97.25%. By taking more care during this final transfer, we were able to retain even higher accuracy (97.48%). Further improvements should be possible. After the CIFAR-10 experiment, accuracy actually increases from 88.04% to 88.17% when the weights are preserved solely on the PCM devices. Even higher non-volatile dynamic range should be possible with additional tiers of significance (for example, two non-volatile conductance pairs and one volatile lowest-significance conductance pair).

Transfer learning. In the hardware-based PCM experiments, the 2,048 neuron activations to the last fully connected layer are remapped to the 8-bit integer

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

neuron excitations used in our network implementation. Although activations across the 2,048 neurons ranged from 0 to 10.0, we clipped all excitations above 3.0. Software re-training confirmed that this clipping has a minimal effect on the resulting accuracies.

SPICE modelling. Random variation in FET threshold voltages introduces substantial asymmetry in both the up and down conductance change of 3T1C cells and their conductance–voltage characteristics and can therefore degrade neural-network accuracy. Unlike modelling of PCM and other emerging NVM devices, modelling of transistor characteristics and variability is both well-established and highly accurate. We created 1,000 different fitted models for each of these three effects from data generated through Monte Carlo circuit simulations in SPICE (Extended Data Fig. 10). The threshold voltage offset of each FET was sampled independently, with a standard deviation inversely proportional to the square-root of FET area[42]. These models were assigned randomly to all synapses in the array, allowing the mixed hardware–software experiment to account for the effect of this variability on neural-network training. Because each transistor within a 3T1C circuit chooses independently, there are 10^9 possible model combinations.

Circuit simulations of the 3T1C cell were carried out using LTSPICE (https://www.linear.com/solutions/1066). NMOS and PMOS FET model files for 90-nm technology were obtained from the Predictive Technology Model website (http://ptm.asu.edu/)[43], which provides free-to-use BSIM models for multiple technology nodes. Oxide thickness was increased to 1.9 nm for lower gate leakage, but the model was otherwise unchanged.

From circuit simulations, we extracted (Extended Data Fig. 10): (i) the dependence of the conductance (sensed as a read current through the read FET at a constant read voltage) on capacitor voltage V_g; (ii) the increase in V_g as a function of instantaneous voltage for multiple 'up' pulses; (iii) the decrease in V_g as a function of instantaneous voltage for multiple 'down' pulses; and (iv) the decay of V_g over time. The decay is mostly unaffected by variations in threshold voltage, so all 3T1C devices were modelled with the same decay constant of 5.16 ms.

Power assessment. The average power per example of the neural network was assessed by first estimating the total energy across all of the individual components and then dividing by the time per example. The core peripheral circuitry required for neural-network training was designed in IBM CMOS 9FLP (90-nm) technology; all relevant static and switching energies were obtained for all required circuitry from circuit-level simulations within the Cadence design environment. Although the capacitances of the short interconnect wiring between FETs in the peripheral circuits are not considered, we include parasitic and gate capacitances of the transistors themselves and all static and leakage power in the peripheral circuitry. Our implementation of the capacitive unit cell in 90-nm node uses the gate capacitance of a MOSFET. Because we need to use thick oxide gates for low leakage, the effective capacitance per unit area is lower ($4.8\,\mathrm{fF}\,\mu\mathrm{m}^{-2}$), leading to a gate area of $2.2\,\mu\mathrm{m}^2$ for the capacitor in this initial design (10.5 fF).

Extrapolating to a future, yet feasible, chip solution, we assume that a complete forward, reverse and weight-update operation could be completed in 240 ns. Because forward propagation must be completed in 80 ns for a three-layer network, each layer is assumed to have a maximum allowed pulse duration of 20 ns plus 6.7 ns of circuit set-up time. Read current from g^{shared} devices—in extra columns for forward propagation and in extra rows for reverse propagation—is scaled down before being shared. The relative significance of PCM versus 3T1C is implemented with different scaling factors on the peripheral circuitry, with associated energy fully represented in these estimates.

Energy is consumed in the array when driving the long word lines and bit lines to their appropriate voltage levels (intrinsic and loading capacitances on the wires), and in the synaptic unit cells (the read/write currents and the time for which it flows). For forward and reverse propagation, we know from our experiment both the average conductance for the PCM cells (3 μS) and the average read current for the 3T1C cells (3 μA). On the basis of data collected from our neural-network experiments, we compute pulse widths of the scaled average neuron excitation (for example, x) of 6.0 ns and of the average neuron error (for example, δ) of 0.6 ns, relative to the maximum pulse duration of 20 ns. Our read energy results are cycle-accurate to the exact training runs in our experiment, with the only assumed parameter being the 20-ns absolute duration of the maximum-length pulse in an eventual hardware implementation. We also calculate the total number of weight updates, occasional resets and transfer operations across 20 epochs, with triage included, and then prorate these numbers per training example.

The effective number of transfers per example was calculated on the basis of statistics collected over 20 epochs of 60,000 examples each. Each transfer operation operates on a single column at a time and is assumed to involve up to nine read operations on all synapses in the column and up to eight write (partial-set) pulses on some of the devices in that column. Set pulses are assumed to be 4 ns in duration, with the set current 10 times the PCM read current. The energy for initializing the g and g^{shared} cells and for post-transfer tuning of g is calculated on the basis of a similar prorated approach. No analogue-to-digital converters are used in the transfer process, or in the forward/reverse propagation. The same circuitry that generates pulse widths on the basis of neuron activation is used to generate a pulse that dictates the strength of the partial-set pulse used in transfer. The energy of these circuits is included in the calculations.

We assume that all transfer operations in the 2PCM + 3T1C design can be hidden within this processing time (such as while the network is performing operations on a different layer of the network). This would preclude the use of the relatively long rampdown pulses used here, mandating a closed-loop PCM conductance-tuning procedure that emphasizes short, low-amplitude partial-set and partial-reset pulses. Transfer would be distributed in time along different columns, rather than performed all at once over the entire array, as was necessary in our experiments. This choice relaxes the time constraint. Networks with a large number of layers would provide substantial time for such operation because the signal needs to pass through a large number of layers, so many cores would be idling. Considering our MNIST network (528-250-125-10 neurons) and a transfer interval of 8,000 images, the time available for transferring weights to PCM would be $240\,\mathrm{ns} \times 8{,}000/(250+125+10) = 5\,\mu\mathrm{s}$, where $250+125+10$ represents the number of columns in the network. Although larger networks enable even longer programming times and relax the timescale for update, any scheme for pipelining computation so that all arrays are computing all the time would require some accommodation for weight transfer. The optimization of such a tuning procedure will be an important subject of future work.

We find the total dynamic energy per example to be 12.9 nJ for the 2PCM + 3T1C design. The primary contribution is from the first-stage forward propagate (6.3 nJ). This is consistent with the fact that in the 3T1C cell the read currents are high (of the order of microamps); the write currents are an order of magnitude lower, owing to the different current paths for write and read. This contribution is followed by the energy to route signals between stages (2 nJ) and the energy for transferring conductance information from the lower- to higher-significance pair (1.8 nJ). The corresponding average dynamic power is 53.75 mW. Static power is 0.12 mW and so the total power is 53.87 mW. Dynamic energy for the 2PCM design is 5.25 nJ, which corresponds to an average dynamic power of 21.9 mW. Static and leakage power is about the same as for the 2PCM + 3T1C design, yielding a net average power of 22.02 mW.

The effective number of multiply–accumulate operations performed is $2(529 \times 250) + 3(251 \times 125 + 126 \times 10) = 362{,}405$. The factor of 3 comes from the three passes through the hidden and output layers, first for forward inference, then for backpropagation and finally for weight update. The first layer has a pre-factor of only 2 because backpropagation through the first layer is not needed. This corresponds to a computational energy efficiency of $28{,}065\,\mathrm{GOP\,s^{-1}\,W^{-1}}$.

We use here a communication of neuron excitations between the device arrays that represents network layers that appears to be much more power and time efficient than is analogue-to-digital conversion, incurring approximately 2.2 nJ per training example (already included in the above analysis). Analogue-to-digital conversion and digital routeing would instead incur 17.6 nJ per training example[12], given the specified 12.5 million samples per second at 9 bits per samples incurring 240 μW or 19.2 fJ per channel, which would substantially increase the total energy costs of training. The analogue-to-digital converter is needed $250+125+10+125+250 = 760$ times to train each example. Similarly, digital-to-analogue or, more accurately, 9-bit-to-duration conversion incurs only 0.369 fJ per channel, but is used $528+250+125+10+125+250 = 1{,}288$ times to train each example. This results in 0.475 nJ per example for a digital-to-analogue converter to reinsert digital data back into the crossbar arrays.

Area usage is estimated from the full layout of a 512×512 array at 90 nm including all peripheral and routeing circuits needed to perform all operations, including transfer, polarity inversion, neuromorphic read and write and array-to-array data routeing. This layout incurs $5.8 \times 10^6\,\mu\mathrm{m}^2$ at 90 nm (57% area efficiency). Assuming usage of 262,144 operations once each 240 ns (for example, 20-ns integrations occurring with an average duty cycle of about 10% for other layers) and extrapolating the area of the design to the 14-nm node through the square of the ratio of minimum wire half-pitches $(140\,\mathrm{nm}/32\,\mathrm{nm})^2$, we estimate a performance-per-area metric of $3{,}582\,\mathrm{GOP\,s^{-1}\,mm^{-2}}$. (Scaling to the 14-nm node necessitates the implementation of PCM or some other suitable NVM in that technology, posing considerable hurdles for device scaling and for delivering switching currents and voltages to the analogue memory devices.) For the V100 GPU, we assume that the best-case performance available for training is the full 30 TOP of FP16 compute. This is both pessimistic (probably only a portion of the V100 area is used in FP16 compute) and optimistic (GPU and digital accelerator performance on the fully connected layers of interest here tend to be limited by memory bandwidth to well below the peak specified performance[6]). In comparison, the analogue approach discussed here could potentially reduce the off-chip communication to only the incoming training examples and labels and occasional weight updates and overrides. This elimination of back-and-forth off-chip traffic for all weights, for weight-update data and for intermediate neuron excitations for all neural-network layers is the source of much

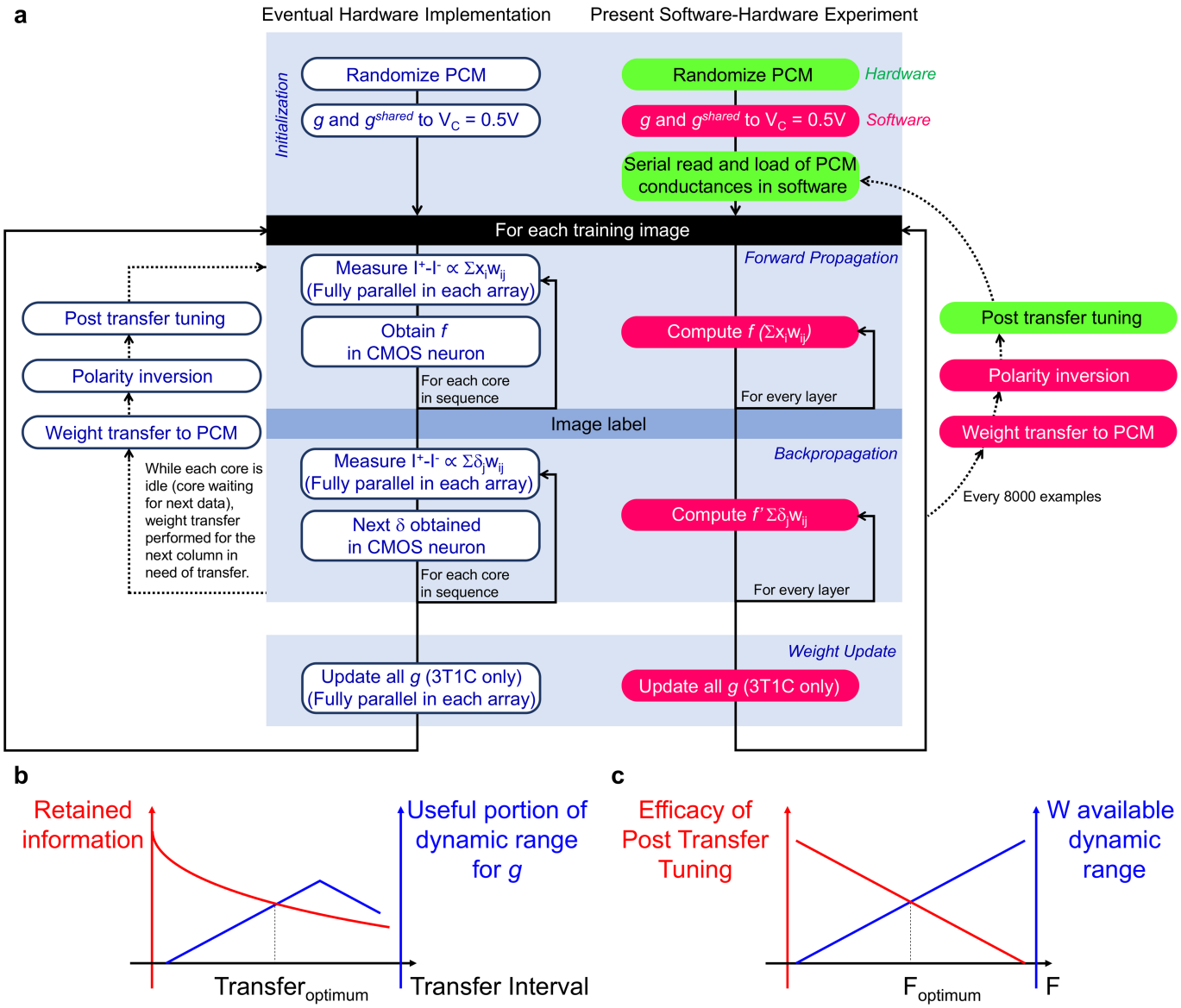
© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

of the energy advantage over GPUs. However, our energy estimates do not include the costs of delivering input data from off-chip to the first layer (and of delivering the label data to the last layer). The importance of these costs will depend strongly on the size of the network. If a wide but shallow network with many inputs and not much computation is implemented with these techniques, then these off-chip input–output costs will probably dominate. For a large network with a reasonable number of inputs and labels, we expect that the attractive energy efficiencies described above will greatly improve the overall energy efficiency of the system.

Data and code availability. The training and test datasets used here are publicly available, as described in the text and the relevant references[23,24]. TensorFlow (https://www.tensorflow.org), the SPICE simulator and the PTM transistor models used here are publicly available, as per the text and relevant reference[43]. The specific neural-network models and parameters are provided in the text. The specific SPICE and Tensorflow decks are available on request. However, the code for our custom neural-network simulator cannot be publicly released without IBM management approval and is restricted for export by the US Export Administration Regulations under Export Control Classification Number 3A001.a.9.

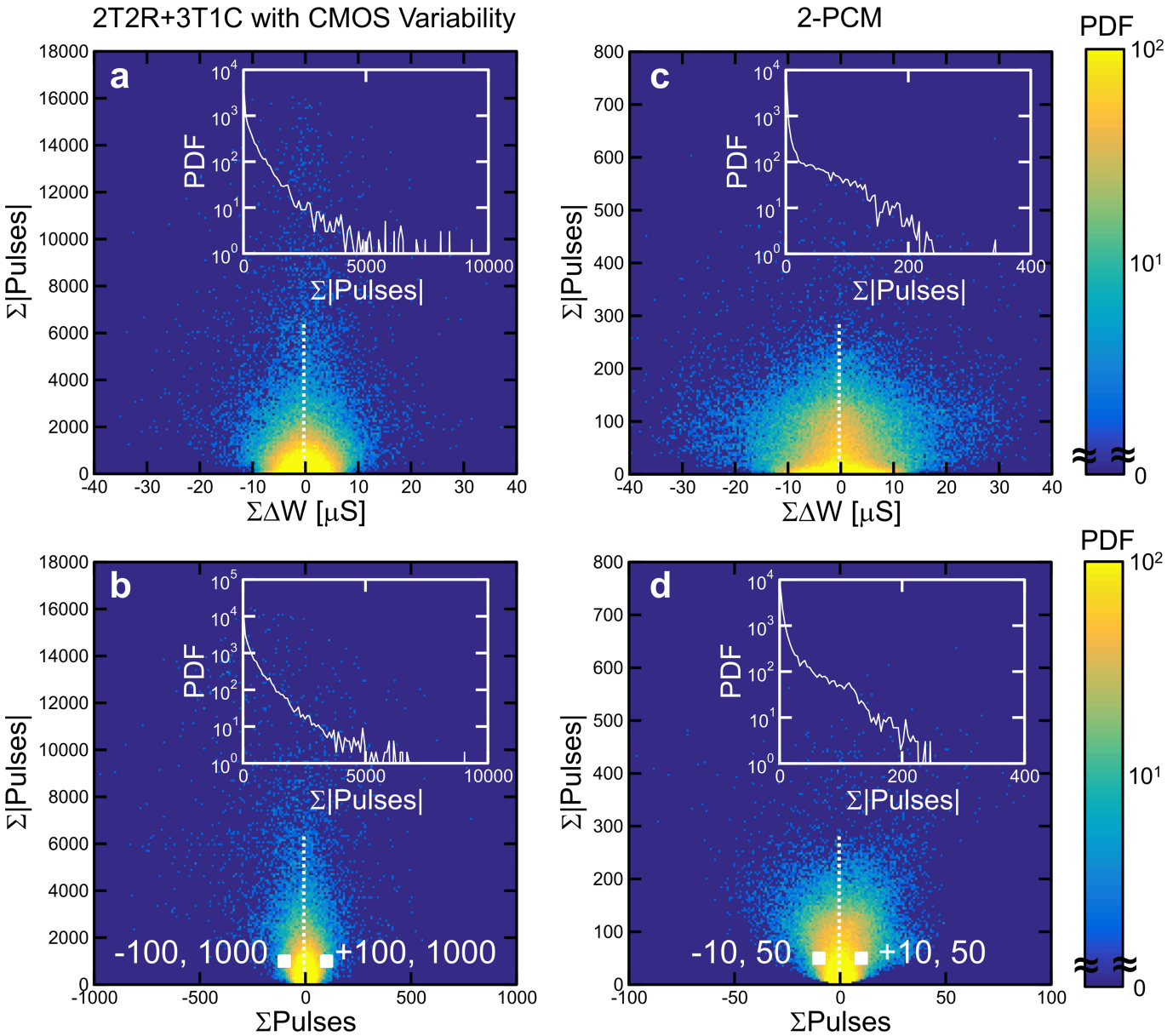
40. Narayanan, P. et al. Reducing circuit design complexity for neuromorphic machine learning systems based on non-volatile memory arrays. In *2017 IEEE International Symposium on Circuits and Systems* 1–4 (IEEE, 2017).
41. Ielmini, D., Lacaita, A. L. & Mantegazza, D. Recovery and drift dynamics of resistance and threshold voltages in phase-change memories. *IEEE Trans. Electron Dev.* **54**, 308–315 (2007).
42. Pelgrom, M. J. M., Duinmaijer, A. C. J. & Welbers, A. P. G. Matching properties of MOS transistors. *IEEE J. Solid-State Circuits* **24**, 1433–1439 (1989).
43. Cao, Y. What is predictive technology model (PTM)? *SIGDA Newsl.* **39**, 1 (2009).
44. Bengio, Y. Louradour, J. Collobert, R. & Weston, J. Curriculum learning. In *Proc. 26th Annual International Conference on Machine Learning* 41–48 (ACM, 2009).

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



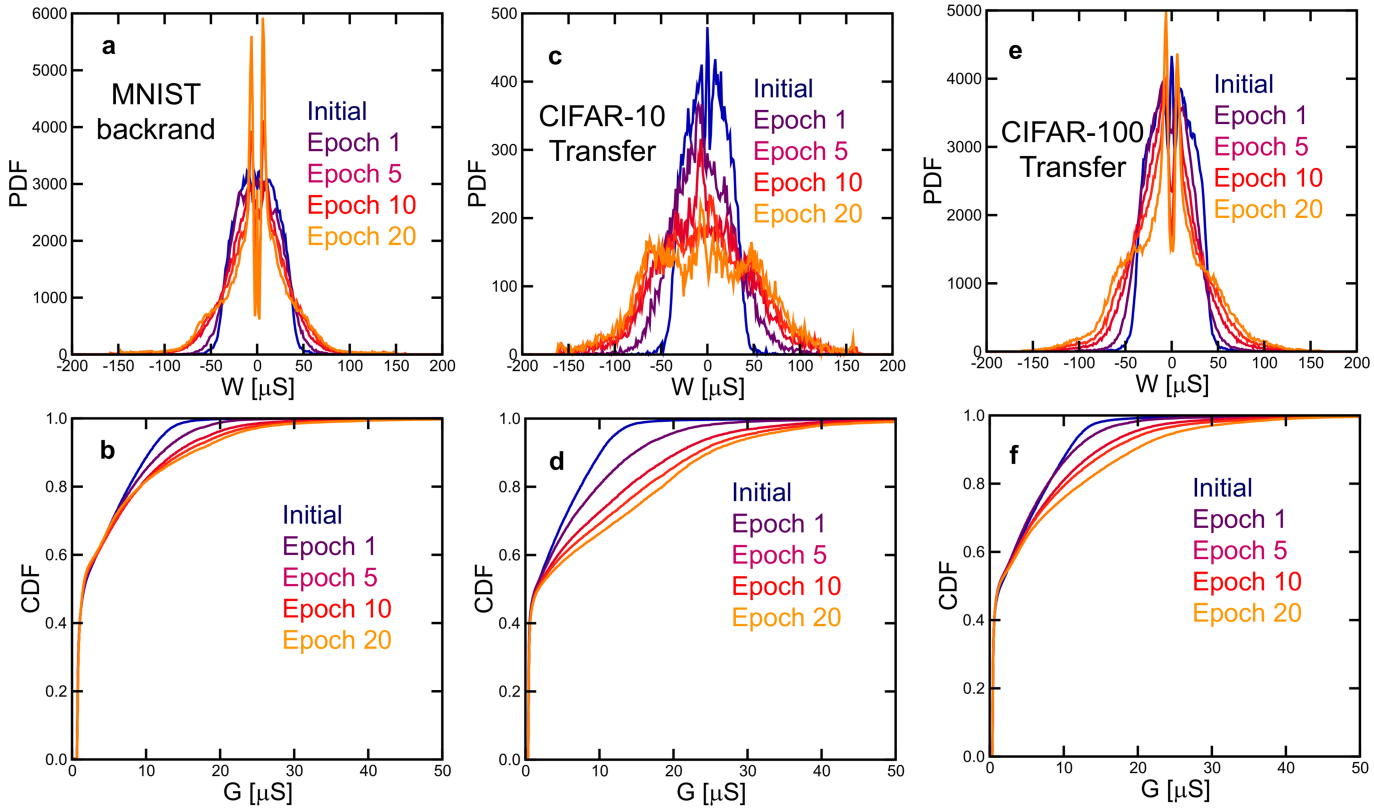
Extended Data Fig. 1 | Flow chart comparing eventual and currently implemented DNN acceleration approaches. a, Comparison between an eventual analogue-memory-based hardware implementation and our mixed software–hardware experiment. Although we do not implement CMOS neurons, we mimic their behaviour closely. In both schemes, weight update is performed on only the 3T1C g devices, and these contributions are later transferred to the PCM devices (G^+ and G^-). Owing to wall-clock throughput issues in our experiment, we have to perform all of the weight transfers at once. By contrast, in an eventual hardware implementation, weight transfer would take place on a distributed, column-by-column basis. Ideally, transfer for any weight column would be performed at a point in time when the neural-network computation, focused on some other layer, leaves that particular array core temporarily idle. **b**, Guidelines for optimizing the choice of transfer interval, depending on the time constant of the capacitor and the dynamic range of g. Because training of one image is performed in 240 ns, training of 8,000 images is performed in $8{,}000 \times 240\,\text{ns} = 1.92\,\text{ms}$, which is a substantial fraction of the time-constant of the capacitor (5.16 ms). Despite allowing more of the dynamic range of g to be used, a longer transfer interval would probably suffer from poor retention of information in any volatile g device. However, even in the ideal case of an infinitely-long time constant, the transfer interval would still need to be limited, owing to the finite dynamic range of g. A long transfer interval would probably result in g values saturating owing to weight updates, leading to loss of training information before transfer. **c**, Guidelines for optimizing the choice of gain factor F. We define 'efficacy of post-transfer tuning' as the inverse of the overall residual error after g tuning. Bcause a larger gain factor F means more available dynamic range for each weight, larger F is desirable. However, large F also amplifies any programming errors on the PCM devices due to intrinsic device variability and limits the correction that g can provide during post-transfer tuning. The efficacy would definitely decrease monotonically, although perhaps not linearly as is sketched here. The value we chose ($F = 3$) represents a reasonable trade-off for the PCM and 3T1C devices used here. For other situations, F can be initially estimated as $F = \text{DR}_g/\sigma$, where DR_g is the g dynamic range and σ is the standard deviation of the PCM programming error. Additional optimization comes with neural-network training, which includes the weak effect of drift contribution.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



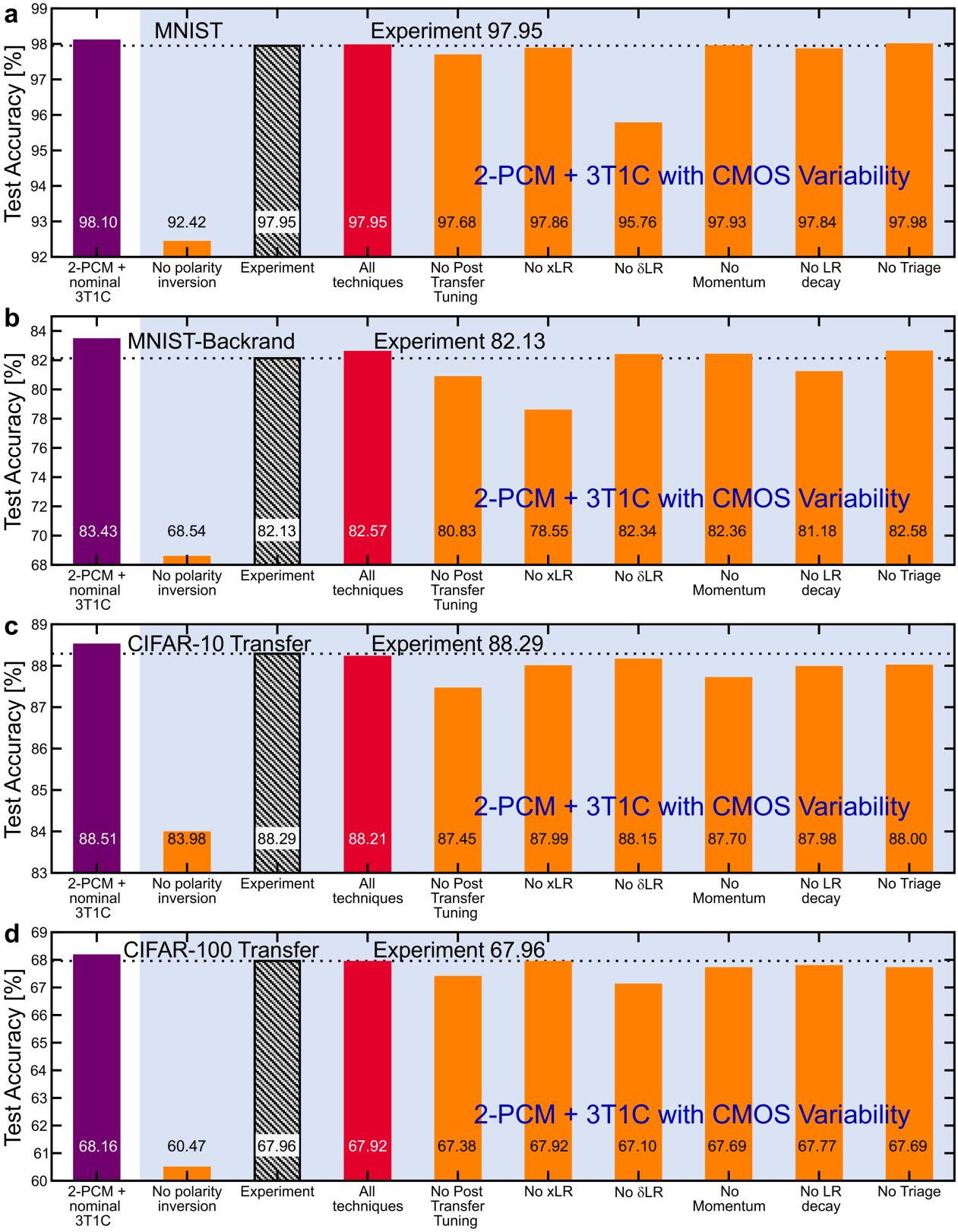
Extended Data Fig. 2 | Weight-update requests and resulting net weight change observed during neural network training. a–**d**, Simulation results based on MNIST 20-epoch simulations for the 2PCM + 3T1C cell with full CMOS variability and transfer polarity inversion (matched with the experimental results; **a**, **b**) and for the 2PCM cell (**c**, **d**). **a**, **c**, Correlation between the aggregate weight update across 16,000 training images (for 2PCM + 3T1C, this corresponds to two consecutive transfer intervals) and the total number of pulses applied to obtain this weight update. **b**, **d**, Correlation between the aggregate number of pulses and the total number of programming pulses applied. The points chosen for Fig. 3 (±100, 1,000 for 2PCM + 3T1C and ±10, 50 for 2PCM) represent typical values requested by the backpropagation algorithm. Insets show vertical cross-sections at $\sum \Delta W = 0$, where the aggregate sum of all individual weight changes ΔW is zero (sum of pulses is zero).

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



Extended Data Fig. 3 | Experimental distributions for different datasets. (Extension of Fig. 5.) **a**–**f**, Weight probability density functions (PDFs) and cumulative distribution functions (CDFs) of device conductances for MNIST-backrand (**a**, **b**), CIFAR-10 transfer learning (**c**, **d**) and CIFAR-100 transfer learning (**e**, **f**). Results are shown for the initial condition and increasing epochs, from 1 to 20. For the CIFAR-100 experiment only, we increased the transfer interval to 16,000 images to reduce the overall wall-clock time.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

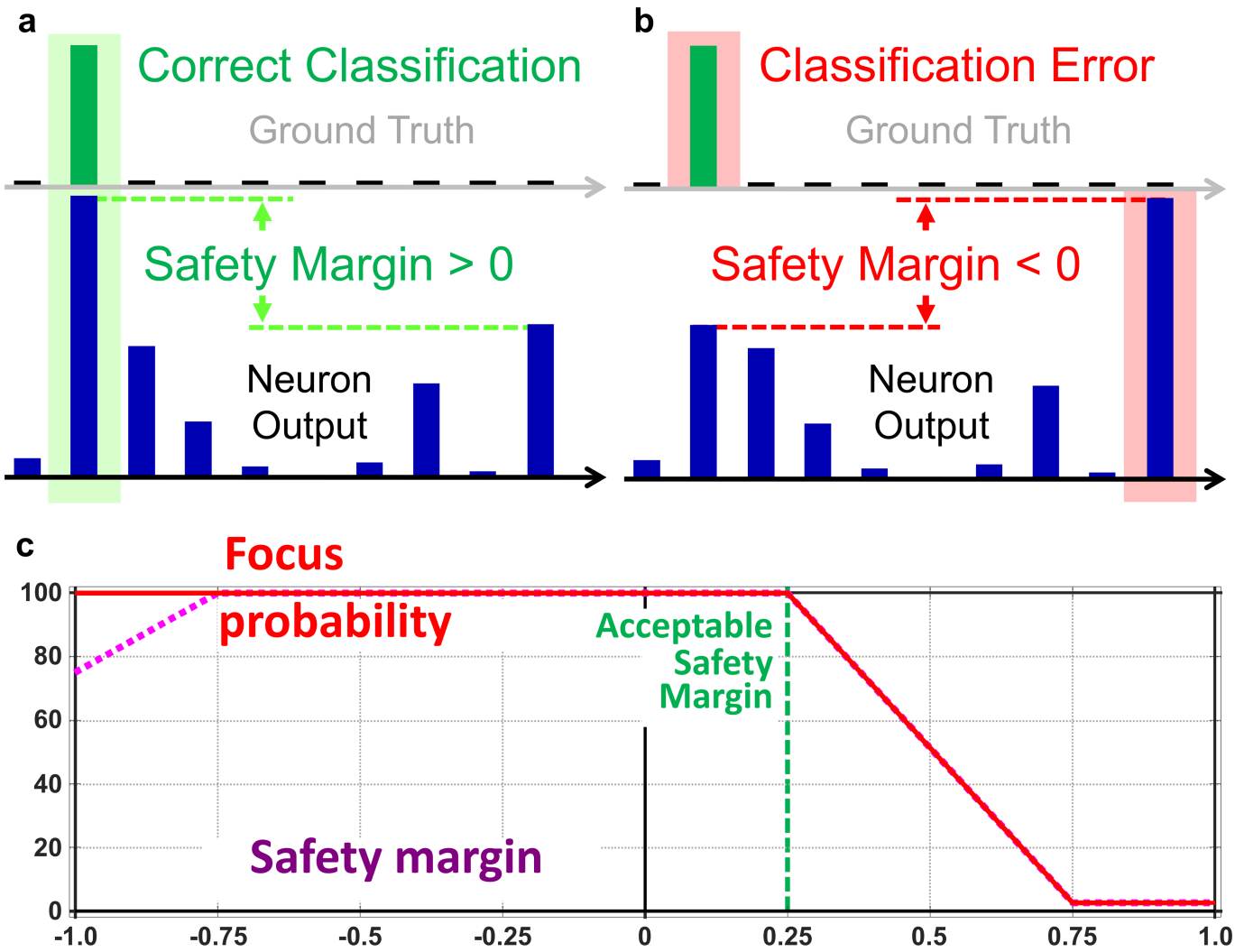


Extended Data Fig. 4 | See next page for caption.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

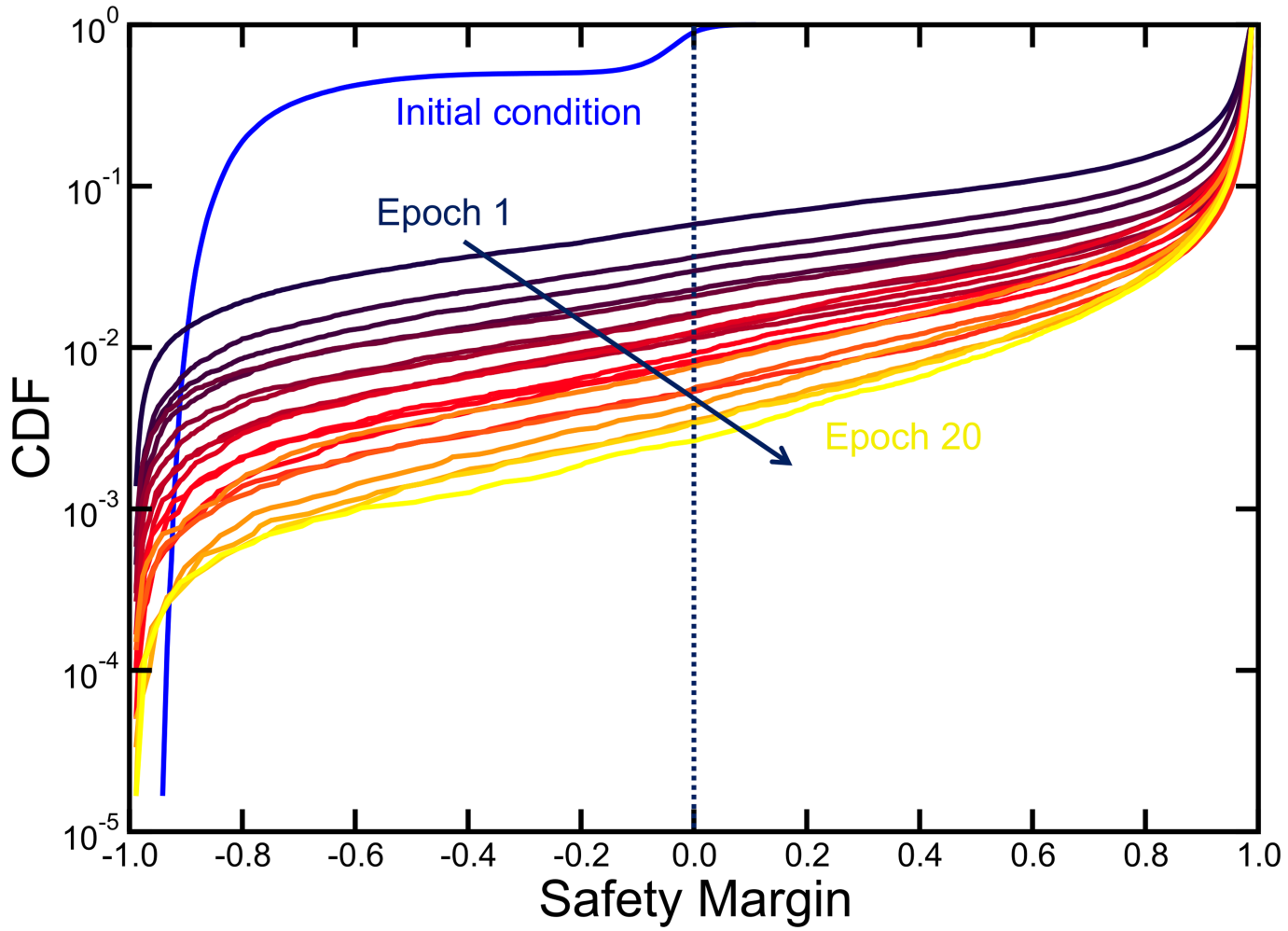
Extended Data Fig. 4 | Effect of different techniques on neural-network training. (Extension of Fig. 6.) **a**–**d**, Simulation results as in Fig. 6b, extended to all experiments performed: MNIST results (as in Fig. 6b; **a**), MNIST-backrand (**b**), CIFAR-10 transfer (**c**) and CIFAR-100 transfer (**d**). We introduce two parameters, x_{LR} and δ_{LR}, to modify the crossbar-compatible weight-update scheme from its original conception[10]. The upstream neurons fire a number of weight-update pulses based on the x input signal, the global learning rate η and the x_{LR} coefficient; downstream neurons fire pulses depending on the error signal, the global η and new δ_{LR} coefficient. x_{LR} and δ_{LR} are both constant throughout training: x_{LR} enables differentiation between upstream and downstream pulsing, but is constant across all layers; δ_{LR} enables careful tuning of the importance of δ for each weight layer. x_{LR} modulation can provide substantial accuracy benefits for MNIST-backrand (**b**) and δ_{LR} modulation is beneficial for CIFAR-100 and particularly for MNIST (**a**, **d**). Although momentum and learning-rate (LR) decay are commonly used techniques[33], their absence would not have greatly affected our experimental results. Example triage mostly provides a wall-clock advantage, but also a slight improvement in accuracy for CIFAR-10/100 transfer learning by avoiding 'useless' weight updates.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



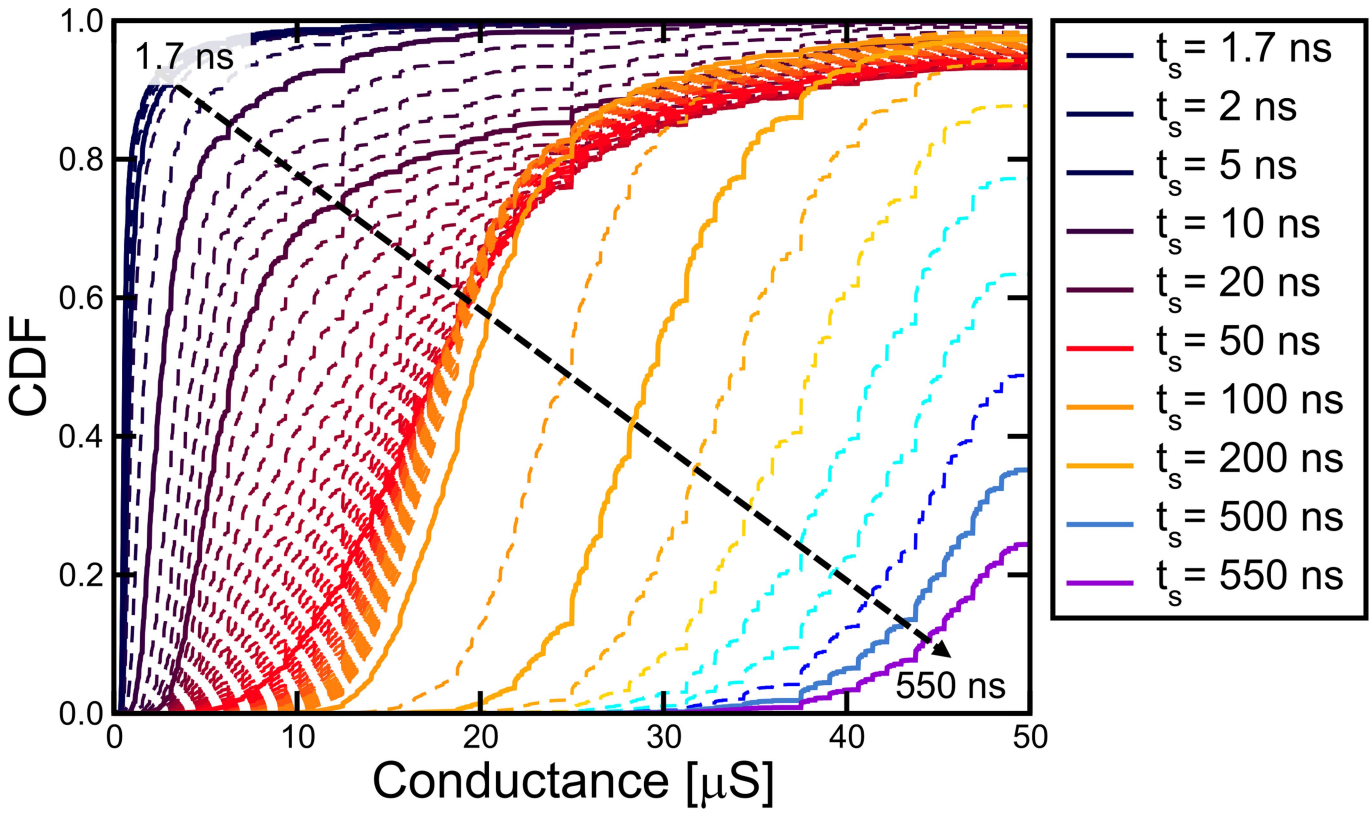
Extended Data Fig. 5 | The safety-margin concept. a, When the network classifies the output correctly (for example, the highest neuron output matches the highest ground truth), the safety margin is the positive difference between the correct neuron and the next-largest neuron. **b**, When the classification is incorrect, the safety margin is a negative number that indicates the gap by which the output neuron failed to be the highest neuron value. Preferably, we would like to calculate the safety margin for every image in each epoch, because safety margins change after each backpropagation. This is the choice made within our experiment; in a full-chip implementation of analogue-memory-based neural-network hardware accelerator with an effective minibatch size of 1, this would be fairly straightforward. Alternatively, either for minibatch-based training or for analogue hardware, we envision using a highly pipelined copy of the network designed for fast forward inference to compute safety margins using a recent copy of the network weights. These slightly 'stale' safety margins could then be used to implement example triage. **c**, Focus probability from 0% to 100% as a function of safety margin defined from −1 to 1. For all safety margins below some 'acceptable' threshold, the probability of choosing to perform backpropagation on this training example is 100%. As the safety margin increases above the acceptable threshold, the focus probability decreases linearly to a non-zero minimum focus probability, to ensure that some number of already well-learned images are also backpropagated despite their high safety margin. The mapping of safety margin to focus probability can be changed during training. In addition, reducing either the focus probability or the learning rate for examples with large negative safety margins (pink dotted line) avoids damage to overall generalization in pursuit of training examples that the network may never be able to successfully classify.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



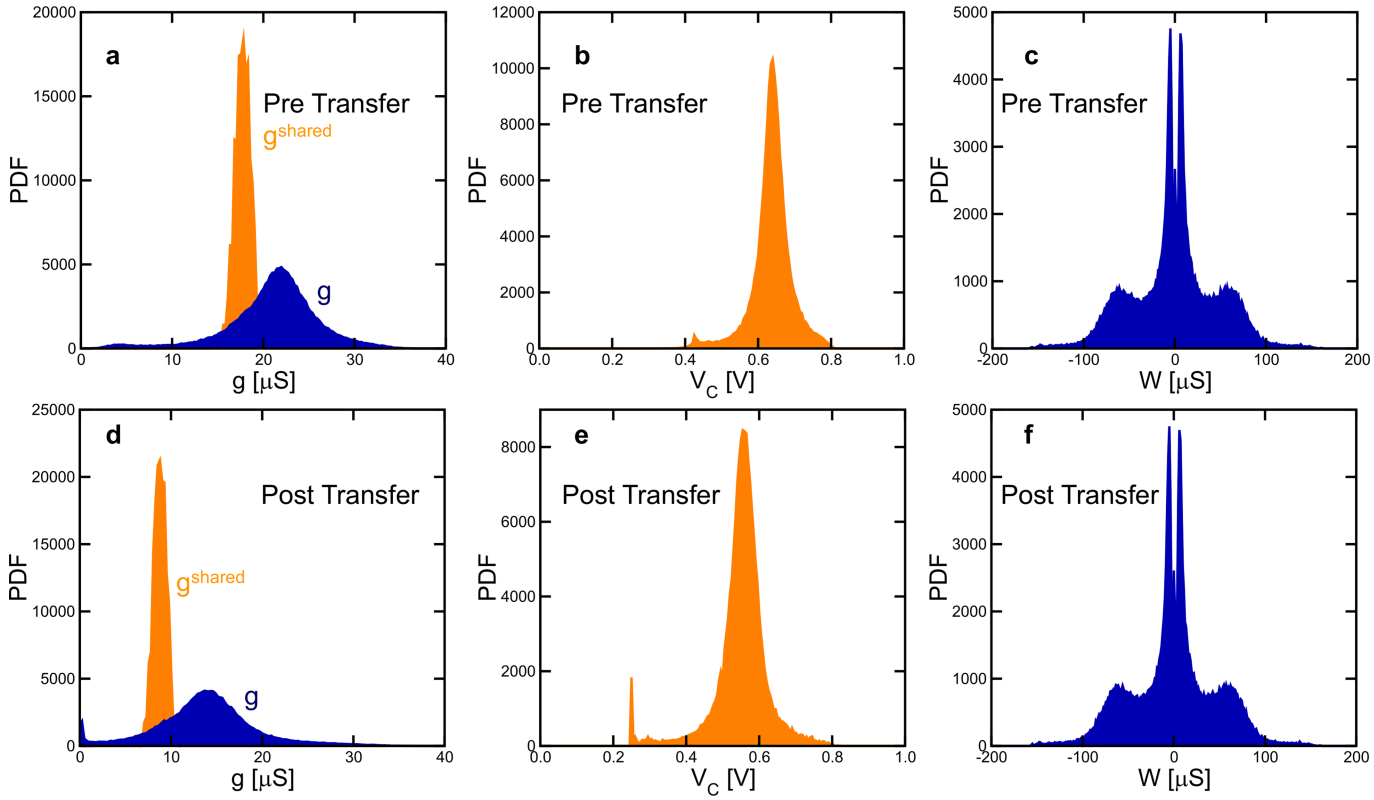
Extended Data Fig. 6 | Safety-margin evolution during training. During training (shown here for MNIST), the cumulative distribution of the safety margin shifts to the right, as training improves performance on the training examples. The intercept at a safety margin of zero represents the training error. Example triage can be thought of as the realization that the network does not need to train on all of the examples in the far right of this cumulative distribution, but should instead focus on those at small positive safety margins and below, with only a few training examples chosen from among those at high safety margins. The farther the safety margin distribution moves to the right, the more of an acceleration factor that example triage can provide. Example triage can be considered a form of curriculum learning[44] based on the safety margin, as a highly accurate analogue measure of the current degree of certainty of the neural network. However, a substantial difference is that curriculum learning focuses on the beginning of training, with the philosophy of starting with easy examples and moving to difficult training examples. By contrast, example triage becomes effective only once the network shows some degree of performance on the training set, and is then designed to skip over easy examples in favour of difficult training examples.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



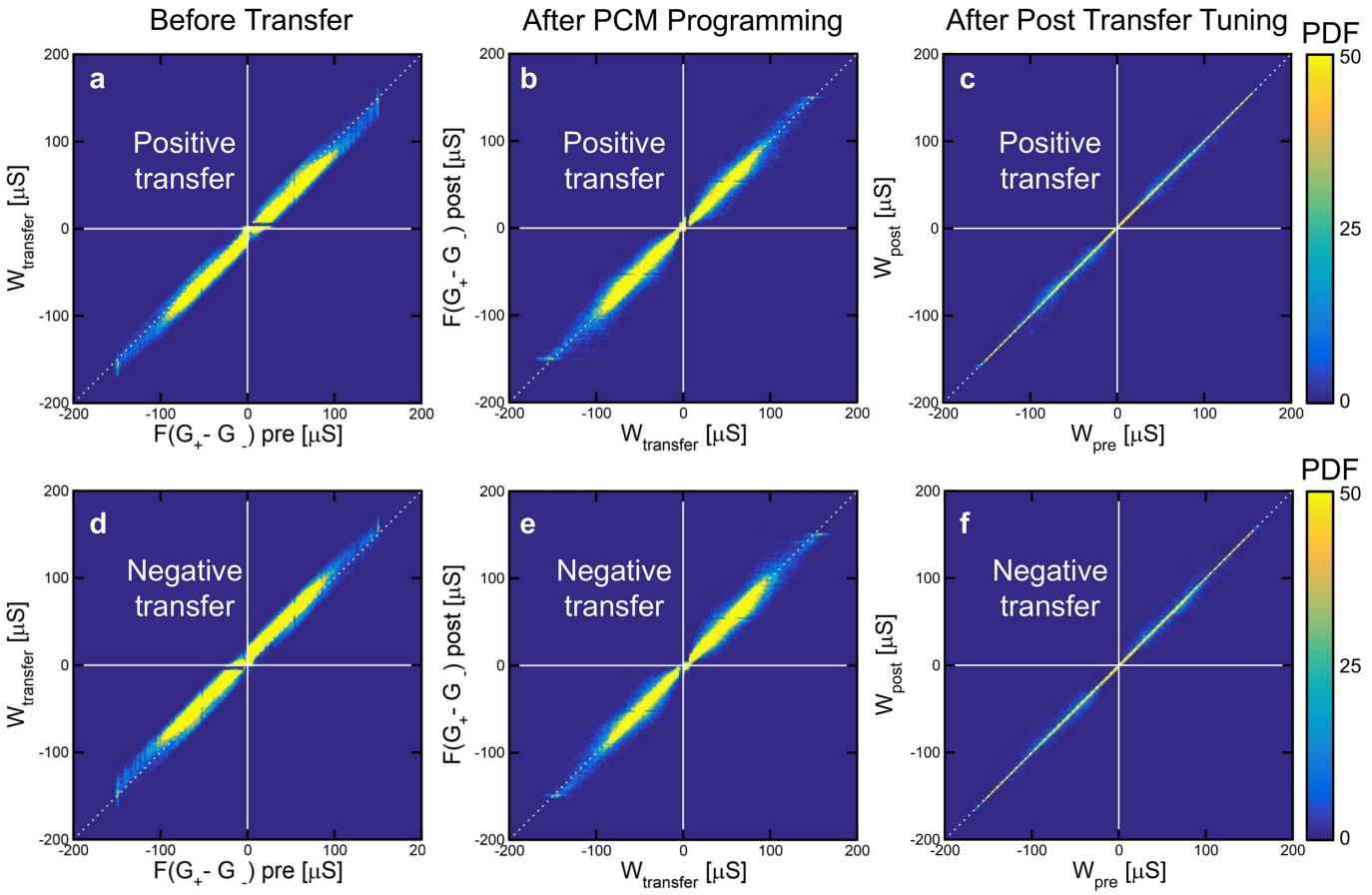
Extended Data Fig. 7 | Experimental PCM programming distributions. The measured cumulative distribution function of the conductances of 512 × 1,024 devices programmed from full reset state with eight-step set transition rampdown pulse sequences ranging from 1.7 ns to 550 ns in step-size (for example, from 13.6 ns to 4.4 μs in total duration) is shown. Even though the degree of control is worse for high conductances (above 20 μS), to the extent that the monotonicity of the mapping from duration to conductance is disrupted, the vast majority of conductances are programmed to conductances below 20 μS (see Fig. 4 and Extended Data Fig. 9).

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



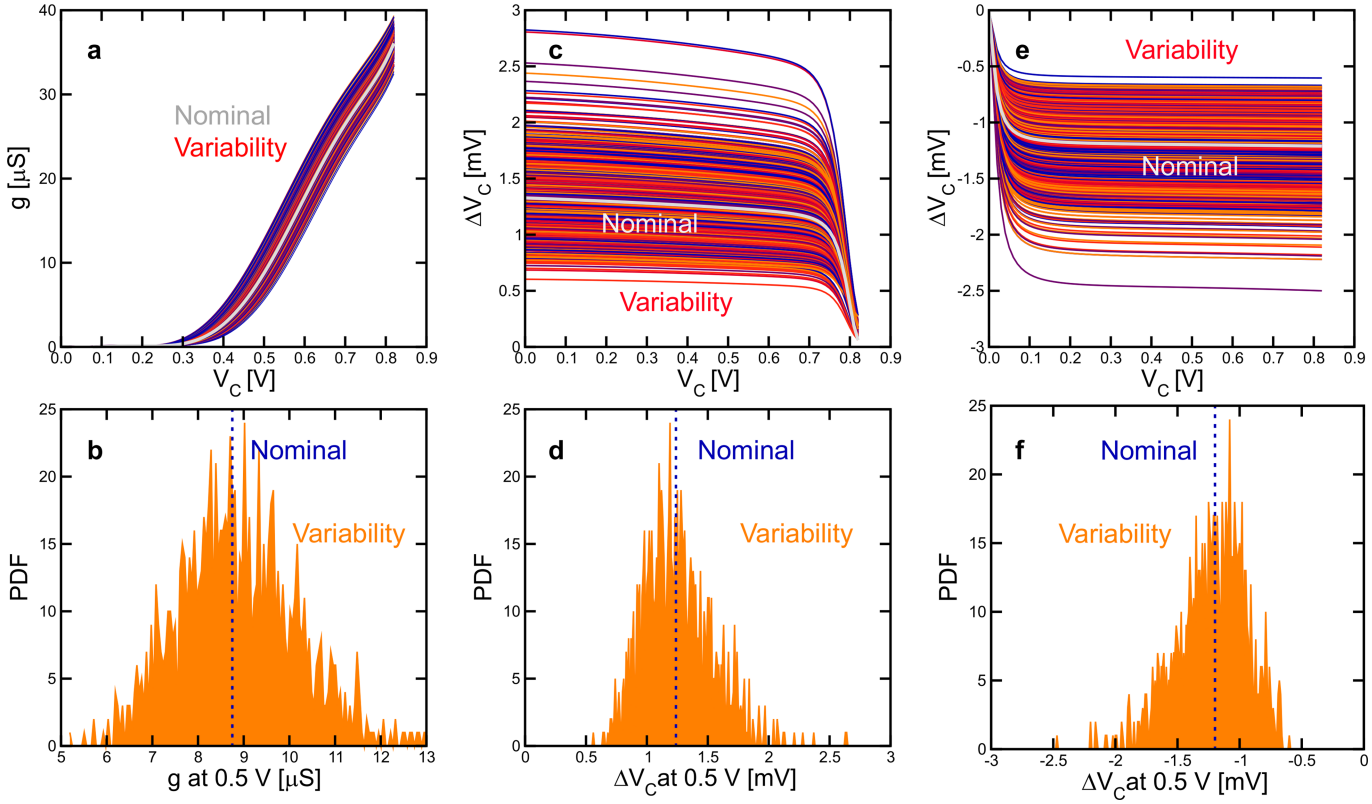
Extended Data Fig. 8 | Analysis of weight transfer from lower- to higher-significance conductance pairs. a–**c**, Distributions obtained before and after the last transfer in the MNIST experiment: g and g^{shared} distributions before transfer (**a**), the voltage on the capacitor of g (**b**) and the distribution of weights (**c**). g^{shared} devices are implemented as an average of the read current from three 3T1C devices for every 128 dedicated g devices to help to reduce variability. Just before transfer, the voltages on both g and g^{shared} are programmed to 0.5 V after their contribution to the weight has been extracted. **d**–**f**, Just after the PCM transfer, the polarity of g is inverted; the dedicated g devices are then tuned to correct the transfer error during PCM programming operation. This leads to a broad distribution of voltages on these capacitors, centred at lower voltages than just before transfer (**e**). During the long transfer interval, charge leakage in all capacitors (through both NFETs and the PFET) causes voltages to increase towards about 0.8 V. During post-transfer tuning, the lowest voltage available to the charge subtraction circuitry is increased so that no 3T1C device can be programmed below 0.25 V (cut-off visible in **e**). Because all 3T1C conductances below that capacitor voltage are effectively zero (see Extended Data Fig. 10a), if any device were allowed to return to the weight-update operations with such an extremely low capacitor voltage, the network would be forced to fire many positive weight updates before it could effectively change that weight. Although g and g^{shared} show different shapes, the weight distribution is nearly the same as before transfer. The last transfer is shown not because it is the easiest but because it is the most important. The network has very little ability to recover from mistakes made during these last few transfers. However, data extracted for any of the other transfers throughout training would be almost indistinguishable from those shown here for the last transfer operation.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



Extended Data Fig. 9 | Effect of PCM imperfections on weight transfer. Correlation maps obtained from the last two transfers in the MNIST experiment illustrate a typical transfer operation. The target weight W_{transfer} that we attempt to write into the PCM devices is not exactly the overall weight W, but instead $W_{\text{transfer}} = W - \text{offset} - [g(V = 0.5\,\text{V}) - g^{\text{shared}}(V = 0.5\,\text{V})]$. The final two terms are the residual difference between the conductances of the g and g^{shared} devices even when initialized to the same voltage, which allows the PCM devices to compensate partially for CMOS variability during transfer. The offset, equal to 2 μS, is added because g devices are not equally good at compensating positive and negative conductance errors. At the initialization voltage of 0.5 V, device conductance is relatively small (see Extended Data Fig. 10a), providing less dynamic range to move to smaller conductances and to correct PCM devices programmed to weights that are too positive. The initial 0.5 V was chosen carefully, to accommodate substantial 'decay' towards 0.8 V, providing much more dynamic range for increasing 3T1C conductance. A positive offset value strongly favours negative errors, allowing us to exploit the capability for g values to increase. When W_{transfer} is positive but smaller than the offset we reset both PCM devices and use g to correct the residual error. **a**, Correlation between the weight portion encoded in PCMs before transfer, such as $F(G^+ - G^-)$, with W_{transfer}. Here we expect a difference because the neural-network training has changed the weights—we now need to checkpoint these weight changes from volatile storage on the 3T1C devices into non-volatile storage on the PCM devices. **b**, Correlation between the desired W_{transfer} conductance differences and the actual $F(G^+ - G^-)$ values obtained after PCM programming operation. With perfect devices and no offset, this should be a diagonal line along $y = x$. The variability we see is caused partly by PCM programming error (unintended), partly by the intentional offset and partly by CMOS initialization mismatch (where we are intentionally aiming for a 'wrong' PCM conductance difference to help to compensate for our flawed CMOS devices). **c**, Correlation between the weights before (W_{pre}) and after (W_{post}) transfer, after post-transfer tuning of g to compensate for programming errors in **b**. The goal of the transfer operation is to obtain $W_{\text{post}} = W_{\text{pre}}$, which would correspond to all points falling on the diagonal $y = x$. The effect of post-transfer tuning is clear by comparing the variability in **b** to the near-ideal behaviour in **c**. **d**–**f**, As in **a**–**c**, but for negative polarity transfer. Because the polarity of g is inverted, the offset is negative, and so the large dynamic range can be used to increase g to compensate for positive errors in PCM weight.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.



Extended Data Fig. 10 | SPICE modelling of CMOS variability. **a**–**f**, Monte Carlo circuit simulations of parameter variability in 3T1C cells: measured conductance versus instantaneous voltage on the capacitor V_C (**a**); PDF of the measured conductance at $V_C = 0.5$ V (**b**); change in voltage versus the instantaneous voltage for up pulses (**c**); PDF of change in up voltage at $V_C = 0.5$ V (**d**); change in voltage versus the instantaneous voltage for down pulses (**e**); and PDF of change in down voltage at $V_C = 0.5$ V (**f**). Each graph shows data from 1,000 trials. Bold lines in **a**, **c** and **e** and dotted lines in **b**, **d** and **f** show the nominal transistor response. **a**, **b**, Variability in the read transistor whose gate is tied to the capacitor; **c**–**f**, variability due to variation in threshold voltage in the PMOS pull-up/NMOS pull-down FETs.

© 2018 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.