

# Binary-Stochasticity-Enabled Highly Efficient Neuromorphic Deep Learning Achieves Better-than-Software Accuracy

Yang Li, Wei Wang,\* Ming Wang, Chunmeng Dou, Zhengyu Ma, Huihui Zhou, Peng Zhang, Nicola Lepri, Xumeng Zhang, Qing Luo, Xiaoxin Xu, Guanhua Yang, Feng Zhang, Ling Li, Daniele Ielmini, and Ming Liu

In this work, the requirement of using high-precision (HP) signals is lifted and the circuits for implementing deep learning algorithms in memristor-based hardware are simplified. The use of HP signals is required by the backpropagation learning algorithm since the gradient descent learning rule relies on the chain product of partial derivatives. However, it is both challenging and biologically implausible to implement such an HP algorithm in noisy and analog memristor-based hardware systems. Herein, it is demonstrated that the requirement for HP signals handling is not necessary and more efficient deep learning can be achieved when using a binary stochastic learning algorithm. The new algorithm proposed in this work modifies elementary neural network operations, which improves energy efficiency by two orders of magnitude compared to traditional memristor-based hardware and three orders of magnitude compared to complementary metal-oxide-semiconductor-based hardware. It also provides better accuracy in pattern recognition tasks than the HP learning algorithm benchmarks.

## 1. Introduction

By adopting the in-memory computing paradigm, a neuromorphic system using analog memristors<sup>[1]</sup> as synaptic weights can parallelly and efficiently accelerate the massive high-precision (HP) multiply-and-accumulate (MAC) operations<sup>[2]</sup> in deep learning. The system can parallelly accomplish the MAC operations of each neural network layer in the forward or backward propagations in one step by using Kirchhoff's current law and Ohm's law,<sup>[1,3]</sup> where voltages represent the input signals, the conductance in a crossbar array of memristors store the weights between the input nodes and output nodes, and currents represent the output signals. However, to complete this task, one needs to accurately tune the input voltages and

accurately sense the output currents, which require HP digital-to-analog converters and analog-to-digital converters (ADCs), respectively. This results in high power consumption and a large circuit footprint, thus counterbalancing the analog and parallel in-memory computation gain.<sup>[4–7]</sup> In addition, while the system can execute the gradient calculation and weight update in each neural network layer in parallel,<sup>[8,9]</sup> it is challenging for the system to tune the conductance of the memristor synapses precisely and gradually due to device variations. Thus, efficient online, in-memory learning in a stand-alone non-von-Neumann architecture becomes a prohibitive task. Moreover, in a biological system, voltage spikes, i.e., all-or-none action potentials,<sup>[10,11]</sup> transmit neural signals, which mathematically translate the signals into binary formats and make the use of HP signals biologically implausible. Finally, the memristor-based hardware systems use analog and noisy memristor artificial synapses, which are inherently subject to stochastic fluctuations, to represent the synaptic weights.<sup>[12,13]</sup> Current hardware systems have not fully utilized this stochasticity to further simplify the hardware design.


In contemporary deep learning theory, the HP handling of the signals and errors is an inherent requirement since the gradient descent learning rule relies on the product of a partial derivative chain, i.e., the chain rule of calculus.<sup>[14,15]</sup> The computing systems need HP numbers to accurately represent the continuous changes of the signals and errors, as well as HP synaptic weights to accurately accumulate the descending gradient. Specialized

Y. Li, W. Wang, Z. Ma, H. Zhou, P. Zhang  
Peng Cheng Laboratory  
Shenzhen 518000, China  
E-mail: wangwei@pcl.ac.cn

M. Wang, X. Zhang, M. Liu  
Frontier Institute of Chip and System  
State Key Laboratory of Integrated Chips and Systems  
Zhangjiang Fudan International Innovation Center  
Fudan University  
Shanghai 200433, China

C. Dou, Q. Luo, X. Xu, G. Yang, F. Zhang, L. Li  
Institute of Microelectronics  
Chinese Academy of Sciences  
Beijing 100029, China

N. Lepri, D. Ielmini  
Dipartimento di Elettronica  
Informazione e Bioingegneria  
Politecnico di Milano  
20133 Milano, Italy

 The ORCID identification number(s) for the author(s) of this article can be found under <https://doi.org/10.1002/aisy.202300399>.

© 2023 The Authors. Advanced Intelligent Systems published by Wiley-VCH GmbH. This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.1002/aisy.202300399

algorithms such as neural network quantization<sup>[16]</sup> and binarization<sup>[17]</sup> mainly aim at reducing the computational cost in the inference stage. In the learning stage, it is essential to estimate partial derivatives and HP descriptions of error signals.<sup>[18,19]</sup> Similarly, we need surrogate representations of derivatives and errors with sufficient accuracy<sup>[20]</sup> to train a deep spiking neural network.

In this article, we go beyond the HP requirement of deep learning and propose a binary stochasticity (BS)-based hardware-friendly approach. First, we stochastically binarize both the forwarding signals and the derivatives of the activation function in each layer. Instead of viewing the stochastic binarization as a non-differentiable function and attempting to estimate its derivative,<sup>[18]</sup> we view the stochastic binarization of the signal as equivalent to its floating-point representation. Second, the algorithm only backpropagates the sign of the errors and ignores any error magnitude information, thus enabling the highly efficient hardware implementation of the error backpropagation. Finally, we propose a periodical-carry stepwise weight update method, supporting the in-memory deep learning using noisy and fluctuating memristor synapses. When implementing this algorithm in a crossbar array of analog memristors, with the help of external or intrinsic noise, highly simplified peripheral/neuronal circuits can be employed to accomplish the stochastic binarization operations. The hardware does not need to calculate the activation function or its derivative, nor need to explicitly and accurately sense the crossbar array outputs (i.e., the electrical currents). Thus, the neuronal circuits eliminate the complex and expensive analog-to-digital and digital-to-analog converters. The stepwise weight update method inherently enables the quantization of the weight during the training, which guarantees a large tolerance to the noisy and nonlinear synaptic plasticity of analog memristors. We systematically investigate the effect of these algorithms in fully connected and convolutional deep neural networks for modified National Institute of Standard Technology (MNIST) and Canadian Institute For Advanced Research (CIFAR)-10 datasets. When implementing the proposed BS algorithms, an analog memristor-based neuromorphic system improves the energy efficiency of deep learning tasks by two orders of magnitudes compared to the traditional memristive neuromorphic algorithms. Compared to traditional HP algorithms using metal-oxide-semiconductor (CMOS) technology in the graphical processing units (GPU) of von-Neumann architecture, using the proposed algorithms, an analog memristor-based neuromorphic system improves energy efficiency by more than three orders of magnitudes. The highly efficient neuromorphic deep learning system unexpectedly achieves better-than-software accuracy.

## 2. Results

### 2.1. Hardware-Friendly and Biologically Plausible Algorithms for Deep Learning

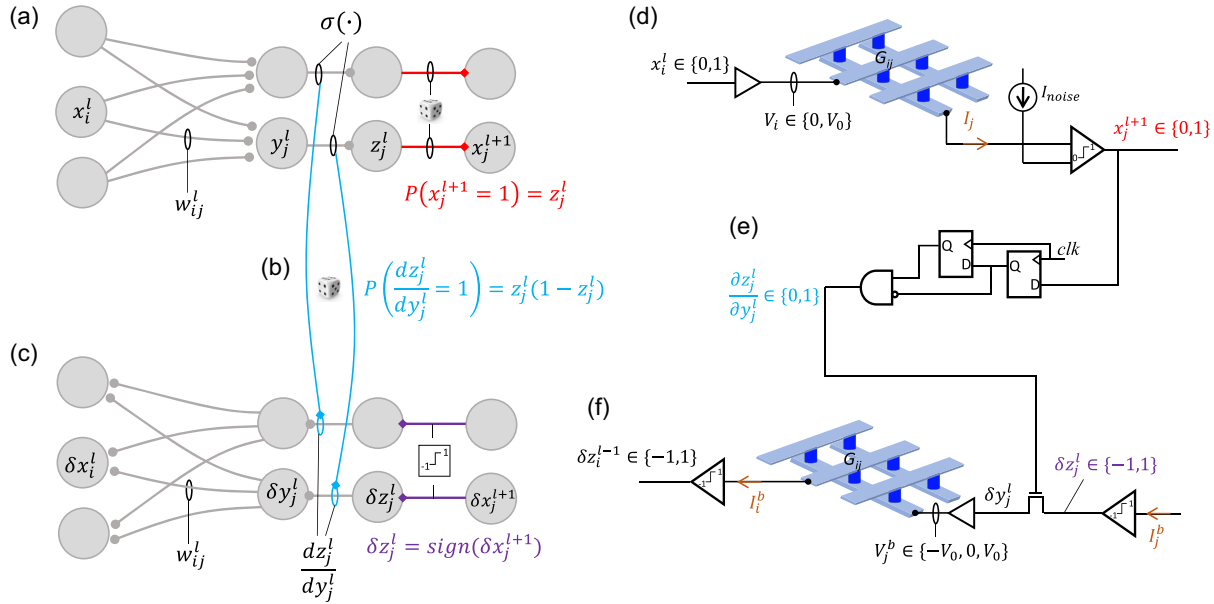
Deep neural networks compute via multiple layers of neurons interconnected by tunable weights. Signals and errors propagate through the layers in the forward and backward directions,

respectively, while the learning is achieved by updating the weights of each layer via the gradient descent rule. Our neural networks rely on three key concepts, namely 1) stochastic binarization of the forward propagating signals (Figure 1a); 2) stochastic binarization of the activation derivatives (Figure 1b); and 3) signed binarization of the backpropagating errors (Figure 1c).

#### 2.1.1. Stochastic Binarization of the Forwarding Signals

In the forward pass, signals of a training sample are transmitted layer by layer to obtain a tentative output in the last layer. Within a typical layer  $l$ , the state of a neuron  $j$  is a nonlinear function of the weighted summation ( $y_j^l$ , i.e., the membrane potential) of input signals ( $x_i^l$ ) from the previous layer, denoted by  $z_j^l = \sigma(y_j^l) = \sigma(\sum_i x_i^l w_{ij}^l)$ , where  $w_{ij}^l$  is the transmitting weight (synapse) from the  $i$ th input signal to the neuron  $j$ , and  $\sigma(\cdot)$  is the activation function (Figure 1a). Instead of directly transmitting the  $l$ th layer's output as the  $(l+1)$ th layer's input, that is  $x_j^{l+1} = z_j^l$ , we activate the intra-layer transmitting signal by a stochastic binarization process (i.e., the Bernoulli process), where the transmitting signal is activated (state "1") with a probability of  $z_j^l$  and is deactivated (state "0") otherwise, that is  $P(x_j^{l+1} = 1) = z_j^l$  (Figure 1a). Note that the input signals ( $x_i^l$ ) should have been in binary states since the previous layer follows the same stochastic binarization rule.

The layer-wise forward propagation can be mapped to a crossbar array of memristors, where the input signals are denoted by the voltages ( $V_i$ ) applied on the top parallel electrodes, the transmitting weights correspond to the conductance ( $G_{ij}$ ) of the memristors in the intersections of top and bottom parallel electrodes, and the currents ( $I_j$ ) from the bottom electrodes represent the membrane potential (Figure 1d).<sup>[3,21,22]</sup> The crossbar array inherently performs weighted summation  $I_j = \sum_i V_i G_{ij}$ , where the multiplications and summations are governed by Ohm's law and Kirchhoff's current law, respectively. In the conventional feedforward propagation, the input voltages ( $V_i$ ) are accurately tuned according to the HP input signals. The output currents  $I_j$  should be sensed with sufficient accuracy to allow the processing of the membrane potential by the activation function. In our algorithm, instead, the input voltage is binarized with only two values, namely 0 V and a fixed read voltage  $V_0$ . The stochastically binarized output signals ( $x_j^{l+1}$ ) are directly obtained by comparing the output currents  $I_j$  of the crossbar array with a noise current signal  $I_{\text{noise}}$  (see Figure 1d, Experimental Section, and Figure S1a and S2a–c, Supporting Information).<sup>[23,24]</sup> It should be noted that the stochastic binarization of  $I_j$  is consistent with the biological neuron in the human brain, where activation occurs when the membrane potential reaches a noisy threshold. The activation function provides the probability of a neuron to be activated,<sup>[25]</sup> and thus can be viewed as the cumulative density function (CDF) of the stochastic threshold of a McCulloch–Pitts neuron<sup>[24,26]</sup> (Experimental Section and Figure S2c, Supporting Information). In this sense, the activation function should always be a monotonically nondecreasing function bounded between 0 and 1.



**Figure 1.** Binary-stochastic (BS) learning algorithm and its hardware implementation. a) Stochastic binarization of the signal forwarding. In a typical layer  $l$ , the neurons are stochastically binarized/activated to be transmitted to the post-layer, i.e.,  $P(x_j^{l+1} = 1) = z_j^l = \sigma(y_j^l) = \sigma(\sum_i x_i^l w_{ij}^l)$ . The input to the weight matrix is binary valued, i.e.,  $x_i^l \in \{0, 1\}$ , since the neurons of the pre-layer have also been stochastically activated. b) Stochastic binarization of the activation derivative. The derivative of the activation function is stochastically sampled for the uses in error backpropagation, i.e.,  $P\left(\frac{dz_j^l}{dy_j^l} = 1\right) = z_j^l(1 - z_j^l)$ ,  $\frac{dz_j^l}{dy_j^l} \in \{0, 1\}$ . c) Error sign backpropagation. Only the signs of the errors from the post-layer are taken to be backpropagated, i.e.,  $\delta z_j^l = \text{sign}(\delta x_j^{l+1})$ ,  $\delta z_j^l \in \{-1, 1\}$ . The backward input  $\delta y_j^l$  to the weight matrix will be ternarily valued since  $\delta y_j^l = \delta z_j^l \frac{dz_j^l}{dy_j^l}$ . d) The implementation of the stochastically binarized signal forwarding in a crossbar array of memristors. The binary inputs to the array are converted to read voltages by level shifters, and the stochastically binarized outputs are obtained by comparing the output currents of the crossbar array with noise currents, that is, a stochastic activation process. e) The stochastic binarization of the activation derivative by individually sampling the stochastically activated forwarding signal twice and processing the sampled signals through a simple logic gate. f) The implementation of the error sign backpropagation utilizing the same array of memristors. The sign of the error from the post-layer is obtained by a comparator and its product with the binarized activation derivative is performed by a transistor. Level shifters are used to apply the ternary valued  $\delta y_j^l$  back into the crossbar array.

### 2.1.2. Stochastic Binarization of the Activation Derivatives

The activation function's derivative  $\frac{dz_j^l}{dy_j^l} = \sigma'(y_j^l)$ , which is required to complete the backward propagation, is stochastically binarized in our algorithm via a Bernoulli process. In particular, for a logistic activation function  $z_j^l = \frac{1}{1 + \exp(-y_j^l)}$ , its derivative has a simple form of  $\frac{dz_j^l}{dy_j^l} = z_j^l(1 - z_j^l)$ . Instead of directly using the activation derivative, we binarize it into "1" with a probability of  $z_j^l(1 - z_j^l)$  and "0" otherwise (Figure 1b).

To map this binarized backpropagation in hardware (Figure 1e), we use two consecutive flip-flops to independently sample the stochastically binarized transmitting signal ( $x_j^{l+1}$ ) and process the sampled binary results of the flip-flops by a logic gate. The derivative  $\frac{dz_j^l}{dy_j^l}$  takes the value of "1" only when the first flip-flop is "1" and the second flip-flop is "0". The probability of the derivative  $\frac{dz_j^l}{dy_j^l}$  being "1" is the product of the probability of the first flip-flop being "1" ( $z_j^l$ ) and the probability of the second

flip-flop being "0" ( $1 - z_j^l$ ), which meets the algorithm's requirement exactly (see Experimental Section and Figure S1b and S2d, Supporting Information).

### 2.1.3. Signed Binarization of the Backpropagating Errors

In the backward pass, the errors between the tentative output and target output in the last layer backpropagate all the way back to the first layer. As shown in Figure 1c, in our algorithm, only the signs of the post-layer errors  $\delta x_j^{l+1}$  are transmitted to the neurons of the current layer, i.e.,  $\delta z_j^l = \text{sign}(\delta x_j^{l+1})$  where  $\delta z_j^l$  is equal to 1 when  $\delta x_j^{l+1}$  is nonnegative, otherwise it is equal to  $-1$ . According to the chain rule of partial derivatives, within the layer  $l$ , the errors of membrane potentials  $\delta y_j^l$  are the product of the neuron errors  $\delta z_j^l$  and the activation function's derivative  $\frac{dz_j^l}{dy_j^l}$ , i.e.,  $\delta y_j^l = \delta z_j^l \frac{dz_j^l}{dy_j^l}$ , and the errors of input signals  $\delta x_i^l$  are the weighted summation of the errors of membrane potential, i.e.,  $\delta x_i^l = \sum_j \delta y_j^l w_{ij}^l$  (Figure 1c). Note that the errors of

membrane potentials  $\delta y_j^l$  are ternary valued (“−1”, “0”, or “1”).  $\delta x_j^{l+1}$  stands for  $\frac{\partial L}{\partial x_j^{l+1}}$  for simplicity, where  $L$  is the cross-entropy loss between the actual output and the target output of a training sample. The same nomenclature applies to other variables, such as  $\delta y_j^l$ ,  $\delta z_j^l$ , and  $\delta x_j^l$ .

When implemented in hardware, the voltages ( $V_j^b$ ) representing the errors of the membrane potential ( $\delta y_j^l$ ) are applied to the bottom electrodes, while the currents  $I_i^b$  are collected at the top electrodes. In the traditional algorithm where the errors from the post-layer are directly transmitted to the current layer, i.e.,  $\delta z_j^l = \delta x_j^{l+1}$ , and the derivatives are in the original form, i.e.,  $\frac{dz_j^l}{dy_j^l} = z_j^l(1 - z_j^l)$ , the errors of membrane potential  $\delta y_j^l$  should be calculated in the digital or analog domain with sufficient accuracy. Thanks to the binarized activation derivatives and the signed errors, our algorithm results in a strong simplification of the peripheral circuits for the error backpropagation (Figure 1f and S1c, Supporting Information). The sign operation can be performed by comparing the output current ( $I_j^b$ ) of the crossbar array with a zero current, to yield a negative or positive output voltage representing the signed error  $\delta z_j^l$ . A single transistor can carry out the multiplication between the signed errors and the binary derivatives. The ternary errors of membrane potentials are then represented in the states of the negative voltage (“−1”), the high impedance (“0”), and the positive voltage (“1”). The ternary output is then converted to a voltage  $V_j^b$  with amplitudes  $-V_0$ , 0, and  $V_0$  to be applied to the bottom electrodes of the crossbar array.

Weight updates are performed after the completion of the forwarding and backpropagation passes for a batch of training samples, according to the gradient descent rule  $w_{ij}^l \leftarrow w_{ij}^l - \eta \langle \delta w_{ij}^l \rangle_{\text{batch}}$ , where  $\eta$  is the learning rate,  $\delta w_{ij}^l = \frac{\partial L}{\partial w_{ij}^l} = x_i^l \frac{\partial z_j^l}{\partial y_j^l} \delta z_j^l$  is the partial derivative of the loss function  $L$  to the weight and  $\langle \delta w_{ij}^l \rangle_{\text{batch}}$  is the average  $\delta w_{ij}^l$  over a training batch.

## 2.2. BS Improves the Learning Performance

We trained a fully connected three-layered neural network for the classification of the handwritten digits from the MNIST<sup>[15]</sup> dataset (Figure 2a and Experimental Section). The same network was trained with either HP or BS approaches. In HP learning, all signals, derivatives, and errors were represented with 32-bit floating point (FP32) numbers, whereas in BS learning the forwarding signals (0 or 1), activations derivatives (0 or 1), and backpropagating errors (−1 or 1) were all mapped with binary states according to the algorithms in Figure 1a–c.

More details about the HP and BS learning algorithm are given in Table S1, Supporting Information. For HP learning, the weight update rule can be written as

$$w^l \leftarrow w^l - \eta \underbrace{x^l}_{FP} \underbrace{\frac{\partial z^l}{\partial y^l}}_{FP} \underbrace{\delta z^l}_{FP} \quad (1)$$

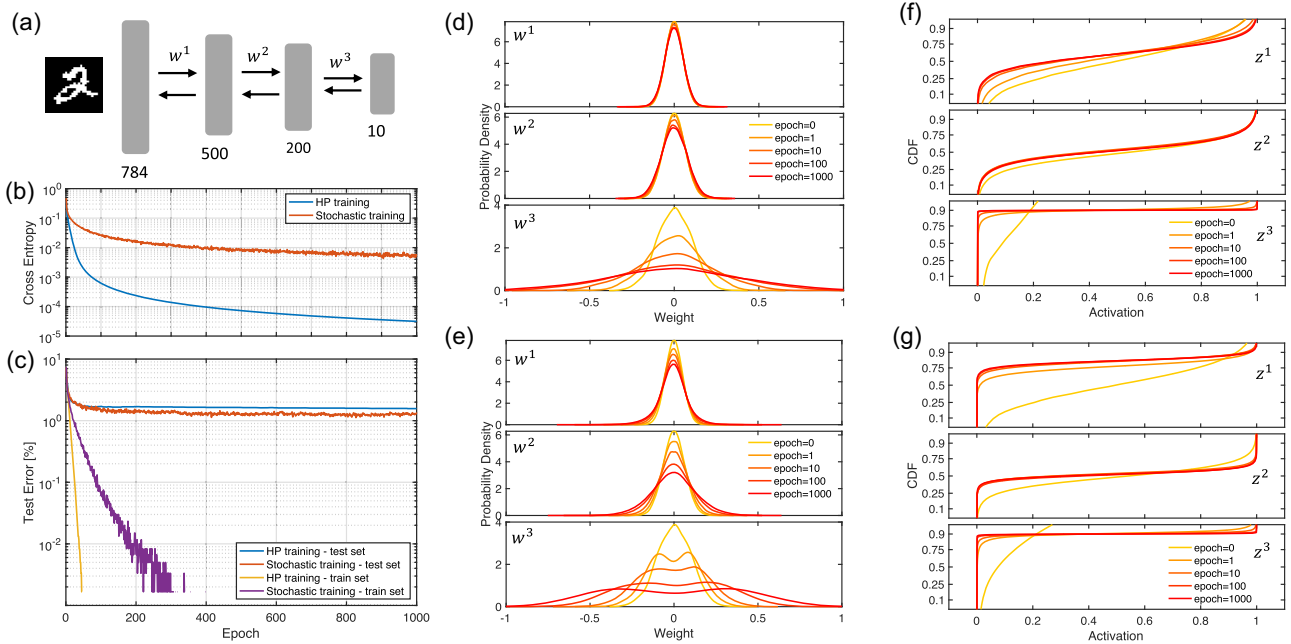
whereas, in BS learning, the weight update follows

$$w^l \leftarrow w^l - \eta \left\langle \underbrace{x^l}_{\{0,1\}} \underbrace{\frac{\partial z^l}{\partial y^l}}_{\{0,1\}} \underbrace{\delta z^l}_{\{-1,1\}} \right\rangle_{\text{batch}} \quad (2)$$

where the subscripts are disregarded for legibility. According to the law of total expectation, the two weight update rules are equivalent: in fact, the three terms represented in floating point precision in Equation (1) are expectations of the three binarized terms in Equation (2). Both BS learning and HP learning show good convergence tendency as the training epoch number increases (Figure 2b). Although the cross entropies between the final output and the target output in the BS learning are higher than that in the HP learning, they both monotonically decrease as a function of the training epoch, thus showing a good learning convergence (Figure 2b). BS learning algorithm achieves better performance compared to HP learning algorithm since the network trained by stochastic binarization algorithm shows lower recognition error compared to the one trained by HP algorithm on the test set (Figure 2c). We use inference errors instead of inference accuracies to show the inference result, since the errors are usually small and comparing errors directly is more obvious for observation. The HP learning shows an obvious overtraining effect than the BS learning since the test error on the handwritten digits from the training set quickly diminishes to 0. However, the test error on unseen handwritten digits from the test set is lower for BS learning, thus highlighting the higher generalization capability of the BS approach (Figure 2c).

The BS learning can adjust the synapse weights in earlier layers more efficiently (Figure 2d,e and S3a,d,e, Supporting Information). This can be explained by the signed operations normalizing the backpropagating errors thus preventing the error vanishing issue for the earlier layers. In HP learning, the activations of neurons in each layer  $z^l$  tend to segregate near 0 or 1 as the learning proceeds (Figure 2f). This tendency is more obvious in BS learning, which highlights that the stochastic binarization well preserves the information of forwarding signals (Figure 2f,g and S3b,c, Supporting Information).

To check the individual effect of the BS algorithms, we permutationally combine them with the HP ones and tested the learning performance of these partially BS-trained networks (Figure S4, Supporting Information). The results show that the BS in signal forwarding can prevent overtraining and improve the test accuracy of the network on unseen data (Figure S4a–c, Supporting Information). The signed binarization of backpropagation errors is instead responsible for effectively propagating the errors to earlier layers of the network (Figure S4d–k, Supporting Information). The stochastic binarization of the activation function derivative does not show a significant effect on the learning process. We also tested BS learning algorithms using various activation functions and derivative functions (Figure S5, Supporting Information). The results show that as long as the activation has a sigmoidal shape and the derivative function has a bell shape, the BS learning algorithm converges to high accuracy (Figure S5, Supporting Information, and Discussion).



**Figure 2.** High-precision (HP) learning versus BS learning. a) A three-layered fully connected neural network for testing the learning algorithm on learning the handwritten digits from the modified National Institute of Standard Technology (MNIST) dataset. b) The cross entropy of the output layer as a function of the learning epoch. c) The test error as a function of training epoch for HP learning and the proposed BS learning. The BS learning shows a less overfitting effect on the train set and better recognition performance on unseen handwritten digits from the test set. d,e) The evolution of the  $k$ -s density of weight distributions during d) HP learning and e) BS learning. In BS learning, more weights in earlier layers have been updated. f,g) The evolution of the cumulative density function (CDF) of the activations during f) HP learning and g) stochastic learning. In BS learning, the bipartite of activations to the regions near 0 and 1 are more obvious.

### 2.3. BS Improves Inference Performance

To assess the impact of stochastic binarization on inference, we compared three inference methods, namely 1) one-time inference using HP forwarding signals (HP inference, **Figure 3a**), 2) one-time inference using deterministic binarized forwarding signals (binary inference, **Figure 3b**), and 3) majority voting of repeated inferences using BS forwarding signals (stochastic inference, **Figure 3c**). In the stochastic inference method, results of repeated inference using BS forwarding signals are obtained and a final recognition decision is made by voting. Only the unseen data are used to test the inference performance.

In the first neural network which is trained by HP learning (**Figure 3d**), the HP inference has the lowest inference error. The binary inference has a higher inference error but largely simplifies neural operations and hardware circuits. The stochastic inference has the highest one-time inference error, although the inference error can be dramatically decreased by repeating the stochastic inference and taking the majority vote as the final recognition result. Overall, the accuracy of the stochastic inference can be much lower than the binarized inference and approximate the one of HP inference.

In the second neural network which is trained by BS learning (**Figure 3e**), all three inference methods have lower inference errors compared to each method in the first neural network, respectively. Surprisingly, the HP inference test error is no

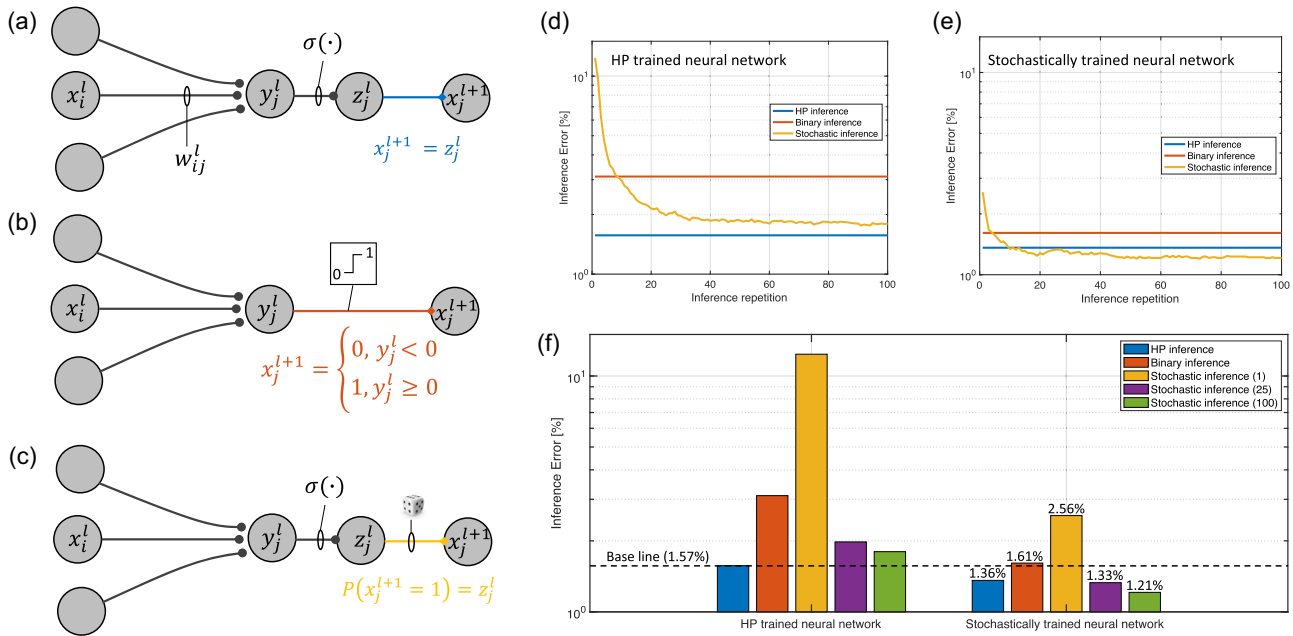
longer the asymptotic line for the stochastic majority vote inference method: the stochastic inference achieves lower inference error than HP inference after 15 times votes (**Figure 3e**).

The HP-learned neural networks naturally show lower inference accuracies when they are used for stochastic binary inference algorithm, because propagated information suffers accuracy loss. Here, we compare the inference results for neural networks trained by both HP and stochastic binarization algorithms only for a comprehensive comparison. Compared to the error of the HP inference in the first neural network, which is about 1.57%, the stochastic inference in the second neural network has the lowest inference error (1.21%) (**Figure 3f**). A better performance is achieved in multiple learning schemes that permute the information representation methods as shown in **Figure S6**, Supporting Information.

### 2.4. BS Is Efficient in Deep Convolutional Neural Networks

To study the efficiency of BS learning in convolutional neural networks, we considered the network shown in **Figure 4a**, consisting of two convolutional layers, two max-pooling layers, and one fully connected layer, for learning and recognizing the handwritten digits from the MNIST dataset.

The BS learning performance, which is shown as the test error on the test set during the training, is better than the one of the HP learning (**Figure 4b**), similar to the fully connected neural network on the same data set. For the neural network trained



**Figure 3.** Inference methods and inference accuracies. a) HP inference with the input of the post-layer being the activation of the pre-layer. b) Binary inference with deterministic binary activation function. c) Stochastic inference with stochastic binarization within each layer and a majority vote in the output layer. d) Comparison of the inference errors for different inference methods of HP-learned neural network. The HP inference is the asymptotic line of the stochastic inference with increasing repetitions. e) Comparison of the inference errors for different inference methods of BS-learned neural network. Better inference performance than HP inference is obtained for stochastic inference after 10 repetitions. f) Summary of the inference errors comparing HP-learned neural network and binary stochastically learned neural network. The performance (1.57%) in traditional algorithms, i.e., HP learning and HP inference, is taken as the baseline. The BS learning reduces the recognition error by 0.21%, and the stochastic inference reduces the recognition error by 0.15% after 100 repetitions.

with an HP learning algorithm, the HP inference shows the best performance and is taken as the baseline (Figure 4c). For the neural network trained with a BS learning algorithm, the inference performance in all cases has been improved dramatically and the performance of the stochastic inference exceeds the HP learning after 10 times repetition of the inferences (Figure 4d). Figure 4e shows the summary of the various inference results for the neural network trained with the two learning algorithms. Overtraining is completely avoided, and more salient features are learned in the convolutional kernels when using BS learning (Figure S7, Supporting Information).

Figure 4f shows a deeper convolutional neural network in visual geometry group (VGG) style<sup>[27]</sup> consisting of six convolutional layers, three max-pooling layers, and three fully connected layers, for learning and recognizing the images from the CIFAR-10 dataset.<sup>[28]</sup> The HP learning shows better learning performance than BS learning (Figure 4g). For the neural network trained with an HP learning algorithm, while the HP inference shows the best performance (the baseline), the binary inference and the stochastic inference are no longer suitable (Figure 4h). For the neural network trained with a BS learning algorithm, while the HP inference and the binary inference show similar accuracies, the stochastic inference quickly exceeds them after several repetitions (Figure 4i). Another interesting result is that the binary inference shows even though slightly but better inference performance than the HP inference. This result shows that HP inference is not always the best solution. Figure 4j shows the

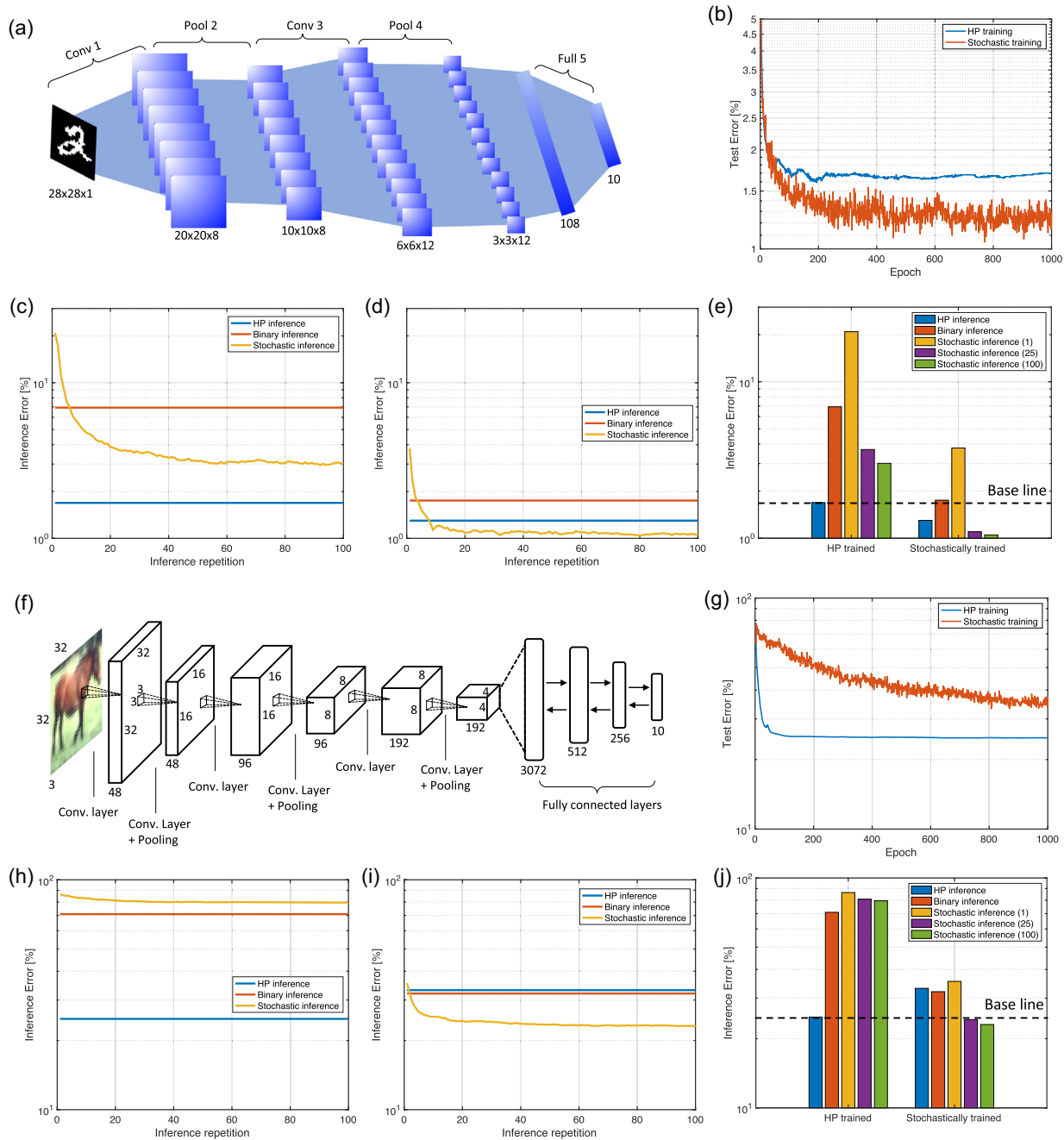
summary of the various inference results for the neural network trained with the two learning algorithms.

Note that, for a fair comparison, no other learning performance enhancement techniques, such as dropout, batch normalization, or data preprocessing methods, are employed in the deep neural network for both learning algorithms.

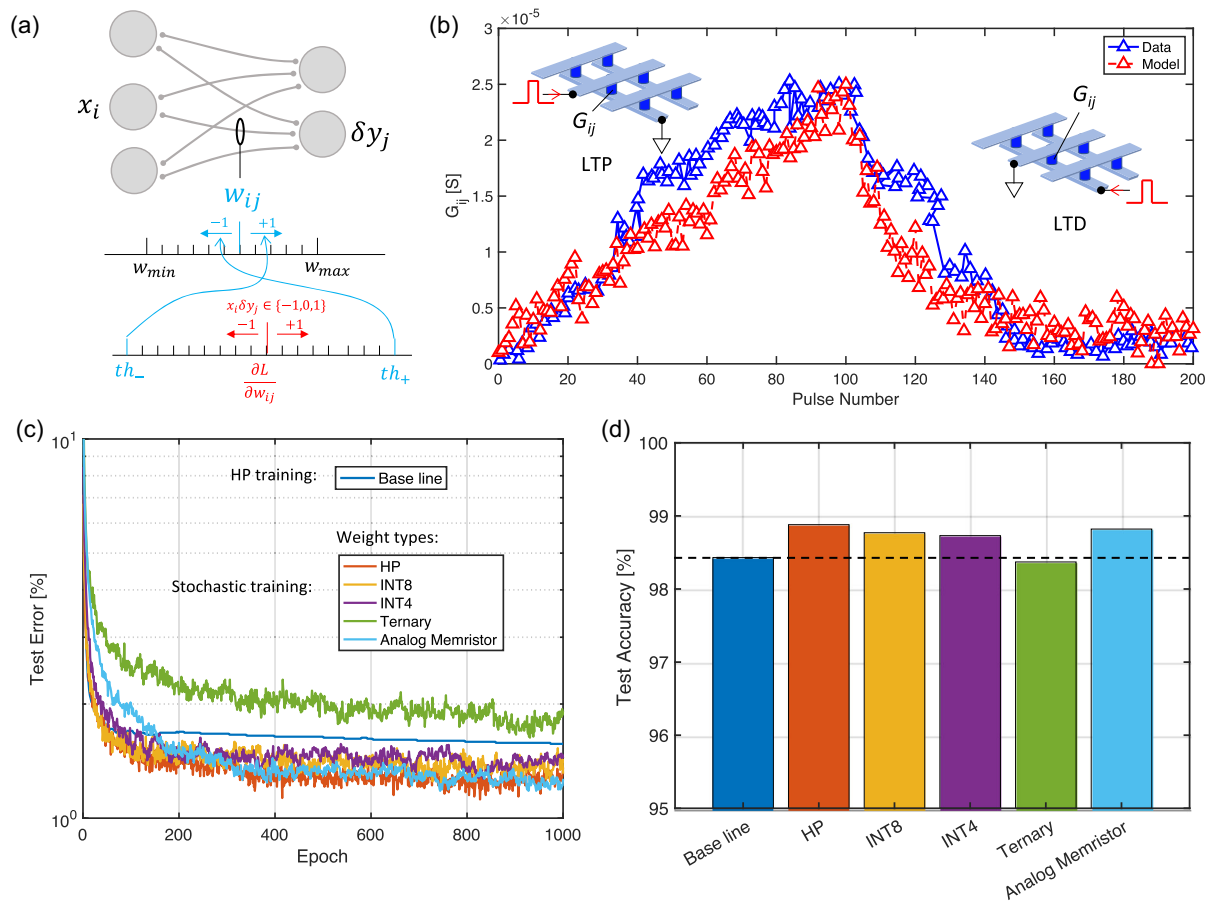
### 2.5. Quantized Weights and Analog Weights Using Memristors

Thanks to binarization operations in signal forwarding, activation derivative, and error backpropagation, the partial derivative of the loss function  $L$  of a single training sample to the weight,  $\partial w_{i,j}^l$ , that is, the gradient of the loss to the weight,  $\partial L / \partial w_{i,j}^l$ , has a ternary value. However, to stabilize the learning procedure, the gradient is averaged over a batch of training samples, according to Equation (2). Thus, the weight needs to be updated or tuned with sufficient precision. A stepwise update of the weight is generally beneficial since it is compatible with quantized weights (e.g., integers) or noisy and analog weights (e.g., analog memristors).

To achieve this goal, we used a periodical carry method<sup>[29,30]</sup> to update the weight, as illustrated in Figure 5a. The gradient of the loss to the weight,  $\partial L / \partial w_{i,j}^l$ , is accumulated in a digital counter. The weight is updated with a fixed step when and only when the accumulated gradient reaches a positive threshold ( $th_+$ ) or a negative threshold ( $th_-$ ). For instance, the weight can be represented



**Figure 4.** The BS learning and stochastic inference for convolutional neural networks and deeper neural networks. a) A five-layer convolutional deep neural network for the MNIST dataset. b) Comparison of the MNIST dataset HP inference error rate of each training epoch in two neural networks. Two networks are trained using an HP learning algorithm and a BS learning algorithm, respectively. c) Comparison of the MNIST dataset inference error rate dependency on inference repetition times using different inference methods. The five-layer convolutional neural network is trained using an HP learning algorithm. d) Comparison of the MNIST dataset inference error rate dependency on inference repetition using different inference methods. The network is trained using a BS learning algorithm. e) Comparison of the test error rates using five different inference methods. The networks are trained using HP learning and BS learning, respectively. f) A convolutional deep neural network for the CIFAR-10 dataset. g) Comparison of the CIFAR-10 dataset HP inference error rate of each training epoch in two neural networks. Two networks are trained using an HP learning algorithm and a BS learning algorithm, respectively. h) Comparison of the CIFAR-10 dataset inference error rate dependency on inference repetition times using different inference methods. The five-layer convolutional neural network is trained using an HP learning algorithm. i) Comparison of the CIFAR-10 dataset inference error rate dependency on inference repetition times using different inference methods. The neural network is trained using a BS learning algorithm. j) Comparison of the test error rates using five different inference methods. The networks are trained using HP learning and BS learning, respectively.



**Figure 5.** Weight quantization and analog weight using memristors. a) The integer-styled periodical carry method for training the stochastic neural network in quantized weights represented by integers and analog weights using memristors. b) Typical long-term potentiation (left) and long-term depression behaviors (right) of memristor devices under identical potentiation and depression pulses, respectively. The data are retrieved from the SiGe epitaxial memory<sup>[32]</sup> and a model is developed to capture the nonlinearities and fluctuations of the weight updates. c) The learning curves of the stochastic training using various types of weights compared to the baseline HP training. d) Summary of the inference accuracy of the stochastically trained using various types of weights compared to the baseline HP-trained neural network. Better-than-baseline learning performances are obtained for the quantized weights in 8-bit signed integer (INT8) and 4-bit signed integer (INT4) as well as for the noisy weights in analog memristors.

by an 8-bit signed integer (INT8, taking a value from  $-128$  to  $127$ ). When the accumulated gradient reaches the positive threshold, the corresponding weight subtracts 1, and *vice versa*. The accumulated gradient is cleared whenever such a weight update event happens. The learning rate is thus defined by the thresholds and a scaling factor between the integer weight and the effective weight (Experimental Section).

Using the analog memristor as synaptic weight, the weight update can be largely simplified compared to the conventional iterative write and verify which is usually employed to tune the conductance of the memristor in sufficient accuracy.<sup>[7,31]</sup> In our software-based *in situ* deep learning experiments, the long-term potentiation (LTP) and long-term depression (LTD) behaviors of a real memristor under identical pulses<sup>[32]</sup> were used to verify the deep learning performance. The memristor device, like the biological synapses, has high fluctuations and nonlinear plasticity under identical potentiation pulses (Figure 5b, left side) and identical depression pulses (Figure 5b, right side). The features of the memristor devices

are captured by a device model. The model considers the on/off ratio, the nonlinearities, the asymmetry between potentiation and depression, and cycle-to-cycle write variations. More details of the device model can be found in Experimental Section. Using the periodical carry method,<sup>[33,34]</sup> when the accumulated gradient reaches the positive threshold, a depression pulse is applied to the corresponding memristor in the crossbar array, and *vice versa*. The conductance of the memristor is updated as is, or in other words, blindly, regardless of the nonlinearity and fluctuation of the weight changes: we do not read the initial and the updated conductance to verify the correctness of the amplitude and direction of the weight changes.

Figure 5c shows the learning performance of a fully connected neural network (same as in Figure 2a) for BS learning using weights of FP32 numbers (without the periodical carry), INT8, 4-bit signed integers (INT4), ternary values ( $-1, 0, 1$ ), and analog memristors, compared with the HP learning (the baseline). More details about the neural network setups and the training results are reported in Experimental Section and Figure S8, Supporting

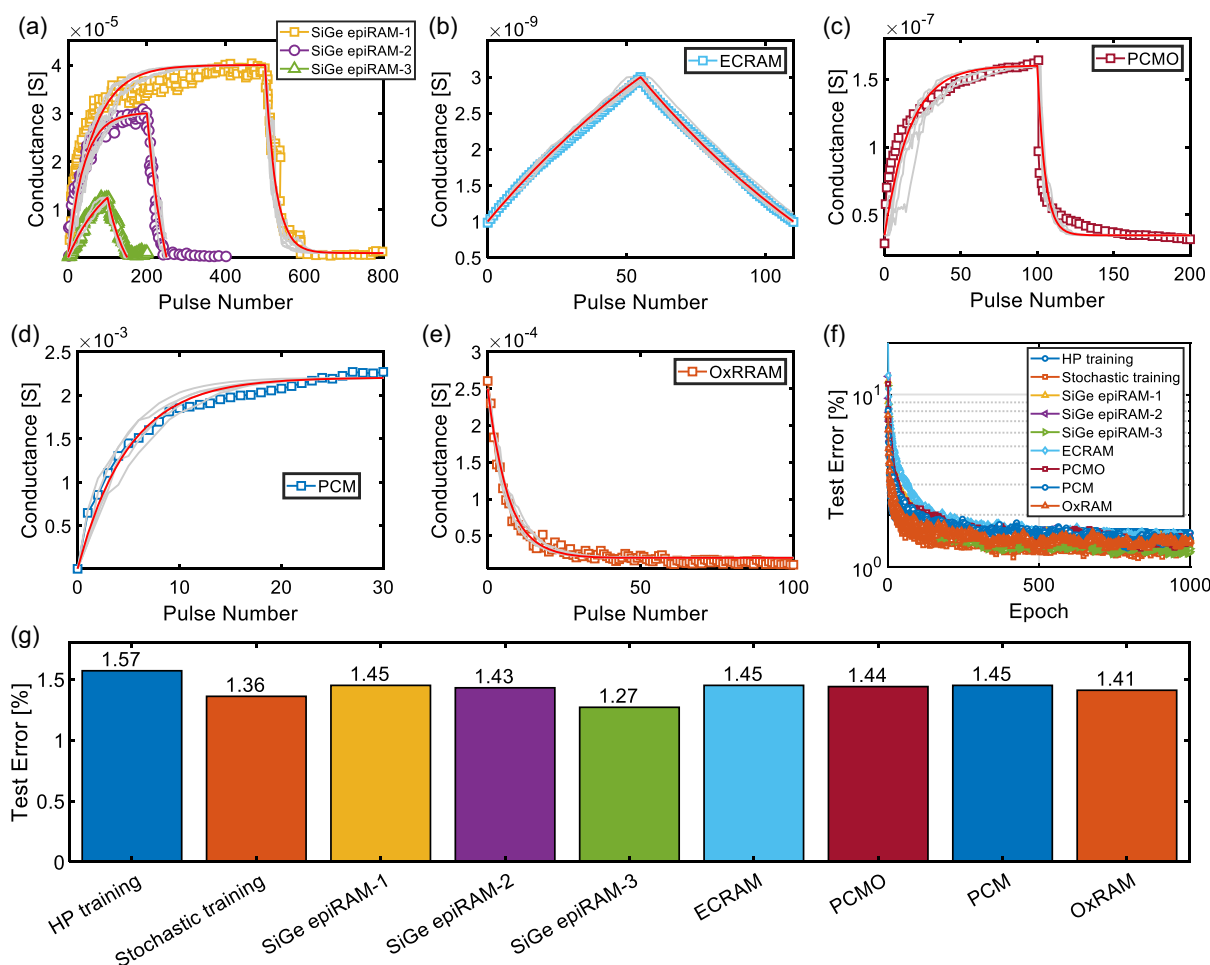


Information. The analog memristor-based neural network shows an inference accuracy (98.82%) higher than the baseline (98.43%) and slightly lower than the binary stochastically learned neural network based on floating-point weights (98.88%), as summarized in Figure 5d. It should be emphasized that the neural network is trained in situ in artificial synapses of analog memristors which have noisy plasticity just like their biological counterparts.

We further simulated the learning performance of the newly proposed algorithm in a fully connected neural network (same as in Figure 2a) using different types of memristive devices as synapses. We captured different device behaviors using our device model, as shown in Figure 6a–e, including three types of the SiGe epitaxial random access memory (ePRAM) devices,<sup>[32]</sup> electro-metalization random access memory (ECRAM) device,<sup>[35]</sup> Pr<sub>1-x</sub>Ca<sub>x</sub>MnO<sub>3</sub> (PCMO) device,<sup>[36]</sup> phase change memory (PCM) device,<sup>[37]</sup> and oxide resistive random access memory (OxRRAM) device.<sup>[38]</sup> The result of SiGe ePRAM device 3 is the same to the analog memristor in Figure 5c. Different learning performances are compared in Figure 6f,g. The learning performance for HP learning method using weights of FP32

synapses (HP training), stochastic binarization learning method using weights of FP32 synapses (stochastic training), and stochastic binarization learning method using different simulated devices as synapses (SiGe ePRAM-1 to OxRRAM) during the 1000 training epochs are shown in Figure 6f. The results all show decreased test errors as the training epoch numbers increase. This indicates good convergence tendency for all cases. Figure 6g shows the test error comparisons after 1000 training epochs for each case. It can be seen that all learning performance for stochastic binarization training method show lower test error as compared to the HP training method regardless of the synapses used. The result verifies the effectiveness of the proposed learning algorithm using different devices as synapses.

The primary goal of the simulations using different memristor devices as the artificial synapses is to provide a simulation platform to estimate the effects of various nonideal factors of memristor devices. The device model supports analysis considering other concerns, such as the line resistance and the sneak current issues. We do not go into details to discuss the line resistance and the sneak path current issues since they are not the major concerns here. This is because the line resistance is not a major



**Figure 6.** Test error comparisons using different synaptic devices. The experimental and simulated behaviors of different memristive devices: a) three types of SiGe ePRAM,<sup>[32]</sup> b) ECRAM device,<sup>[35]</sup> c) PCMO device,<sup>[36]</sup> d) PCM device,<sup>[36]</sup> and e) OxRRAM device,<sup>[38]</sup> f) test errors comparison for each training case during the 1000 training epochs; g) test error comparison for each training case after the 1000 training epochs.

concern for usual crossbar array size ( $128 \times 128$ ) and memristor conductance ranges discussed above ( $10^{-3}$ – $10^{-9}$  S),<sup>[39]</sup> and the negative effect introduced by the line resistance could also be mitigated by online training. As for the sneak path current issue, it is only an issue when RRAM devices need to be accessed individually. In the vector–matrix-multiplication situation using the devices shown in Figure 6, the sneak path current is not an issue<sup>[22,40]</sup> since all device in the crossbar array have definite voltages on both terminals.

Reduction of the computation demand and the energy cost can be achieved in multiple aspects according to the specific application scenarios of BS learning. For instance, if implemented in CMOS technology<sup>[41]</sup> and using the weights represented by FP32 numbers, the BS learning algorithm reduces the energy consumption of the elementary multiply-and-accumulate (MAC) operation from 4.6 to 0.9 pJ (Table S2, Supporting Information). By quantizing the weight, the energy consumption could be reduced further to 0.03 pJ, 15 fJ, and 5.6 fJ for weights implemented in INT8, INT4, and ternary values, respectively (Experimental Section). INT4 weights allow reducing the energy consumption by 307 times without sacrificing the learning performance, while ternary-valued weights reduce the energy consumption by 821 times with only a slight degradation of learning accuracy. When implementing the BS learning in a memristor-based neuromorphic system, the energy consumption for a single MAC is reduced from 0.18 pJ to 1.8 fJ (100 times reduction) since the input is in a 1-bit binary state and the production does not need ADCs.<sup>[7]</sup> Reduction of the footprint of the circuits in integrated chips by 57.8 times could also be projected (Experimental Section). Note that the neural network implementation in CMOS technology needs to retrieve the weight data from memory frequently, whereas the weight data are stored in the memristor devices in the memristor-based neuromorphic system. Thus, conservatively, more than three orders of magnitudes of energy reduction can be obtained using the proposed BS learning algorithm combined with the memristive neuromorphic technology.

### 3. Discussion

Introducing stochasticity to a neural network has been proven to be beneficial in several aspects, for instance, escaping from local minima in the Boltzmann machine<sup>[42]</sup> and preventing the overfitting effect in the dropout technique.<sup>[43]</sup> Stochasticity has a similar role in this work, resulting in better learning performance. The better performance by the majority vote of repeated stochastic inference than the HP inference (Figure 3e, and 4d,i), requires deeper insight. One might expect that the HP inference should be the asymptotic line of the repeated stochastic inference, in the sense that a real number between 0 and 1 in sufficient precision contains all the information of a neuron state. The sampled binary-valued numbers, which take the HP number as their probability, can restore the full information of the HP number only after sufficient times of repetition. However, the neural network experiments show that a group of sampled binary-valued numbers after only a few repetitions could already convey more information than the real numbers. This may also explain why the recognition accuracy improved after introducing the randomness into the system.

Binarization or, more generally, quantization of the forwarding signals has been extensively investigated to reduce the computational loads for inference in edge applications.<sup>[16]</sup> We compared the inference accuracy of our work with other resistive random access memory (RRAM)-based binary fully connected and convolutional neural networks in Table S3 and S4 (Supporting Information),<sup>[44–46]</sup> respectively. However, these works generally focus on the inference process without the online training and come at a cost of decreased inference accuracy. Neural networks trained with binarized or quantized activation, i.e., quantization-aware training, need to estimate or surrogate the derivative of the non-differentiable activation functions.<sup>[19,20]</sup> Here, we use the stochastically binarized state of the neurons as another representation of the activation value, with no need to estimate any derivative. In other words, the “straight-through estimator”<sup>[18]</sup> should be a straightforward solution to the non-differential activation function issue, providing that the binarization is stochastic and the activation function is sigmoid and bounded between 0 and 1. The correctness of the neural network learning is guaranteed by the law of total expectation according to which Equation (1) and (2) should be equivalent.

Since the stochastic binarization operation is applied to each layer, the proposed learning algorithm is a universal scheme and can be easily applied to different neural networks, such as fully connected neural networks and convolutional neural networks illustrated in this work. This algorithm works well in neural networks smaller than 10 layers, but its performance begins to show accuracy decay in neural networks deeper than 10 layers.

### 4. Conclusion

In summary, we have shown that deep learning algorithms can be reformulated to be more biologically plausible and hardware friendly for neuromorphic implementation. We stochastically binarized the forwarding signals and the activation function derivatives. Only the signs of the errors are backpropagated in the backward pass. This algorithm largely simplifies the neural network operations and results in higher deep learning accuracy. Additionally, the stochastic binarization in the forwarding pass also results in better inference accuracy. Mathematically, we prove the correctness of the learning algorithm by the law of total expectation, avoiding the derivative estimation of the non-differential activation functions. We also provide a new view that the activation function should be understood as the probability of the neurons being activated, and the stochastically activated neurons in binary states are more informative than the activation function in real numbers. Finally, a periodical carry strategy is employed to quantize the weight during the learning and adapt the deep learning algorithm to be tolerant to the fluctuation and noisy synapses based on analog memristors. The energy efficiency for deep learning tasks is improved by more than three orders of magnitudes, in addition to better deep learning accuracy.

### 5. Experimental Section

*Crossbar Array of Memristors for Signed Weight Matrix:* Assuming that a typical fully connected layer (labeled as  $l$ ) had  $n$  neurons that processed forwarding information from  $m$  neurons of the previous layer (layer  $l - 1$ ),

in memristor-based hardware implementation, the vector–matrix multiplications in both forward pass and backward pass could be implemented by a crossbar array of memristors, which had  $m + 1$  horizontal bars/wires in the top and  $n + 1$  vertical bars/wires in the bottom. In each intersection of the horizontal bars and vertical bars except the intersection of the  $(m + 1)$ th horizontal bar and the  $(n + 1)$ th vertical bar, there was one memristor (or programmable resistor). The conductances of the memristors in the intersections of the 1-to- $m$  horizontal bars and the 1-to- $n$  vertical bars were denoted as  $G_{ij}$ , where  $i = 1, \dots, m$  and  $j = 1, \dots, n$ . The memristors in the intersections of the 1-to- $m$  horizontal bars and the  $(n + 1)$ th vertical bar and the intersections of the  $(m + 1)$ th horizontal bar and the 1-to- $n$  vertical bars are called reference memristors, which have fixed conductance denoted as  $G_{ref}$  (Figure S1a,c, Supporting Information). The reference memristors were suppressed as shown in the illustration of Figure 1d,f and the discussion in the main text for legibility. They were needed to form differential pairs since the synaptic weights were signed numbers while the device conductance is always positively valued. The synapse weights  $w_{ij}^l$  are represented by the conductance difference between regular memristors and reference memristors  $G_{ij} - G_{ref} = G_0 w_{ij}^l$ , where  $G_0$  is a scaling factor. Assuming that the memristor had the maximal conductance and minimal conductance of  $G_{max}$  and  $G_{min}$ , respectively, the reference conductance was set to the middle point of the memristor conductance, i.e.,  $G_{ref} = \frac{G_{max} + G_{min}}{2}$ . To represent the weight values in the range between  $w_{min}$  and  $w_{max}$ , the scaling factor  $G_0$  is defined as  $\frac{G_{max} - G_{min}}{w_{max} - w_{min}}$ .  $w_{min}$  and  $w_{max}$  are empirical parameters and take the default values of  $-1$  and  $1$ , respectively.

**Forward Weighted Summation by the Memristor Crossbar.** If the previous layer had already binarized the forwarding signals, that is, the input information  $x_i^l$  in this layer was in binary states, the weight summation for the membrane potential  $v_j^l = \sum_i^m x_i^l w_{ij}^l$  could be easily implemented. The binary states, “1” or “0”, are represented by logic voltage levels with an amplitude of  $V_{DD}$  or 0 V, respectively, where  $V_{DD}$  is the supply voltage of the digital circuits. Since  $V_{DD}$  was too large to be directly applied to the memristor array without changing the conductance of memristors, level shifters were used to convert  $V_{DD}$  to a read voltage with the amplitude of  $V_0$ . Thus, the voltages  $V_i = x_i^l V_0$  ( $i = 1, \dots, m$ ) were applied to the  $i$ th top bar of the memristor array (Figure 1d and S1a, Supporting Information). According to Ohm’s law and Kirchhoff’s current law, if the  $j$ th bottom bar was grounded, it had the current output of  $I_j = \sum_i^m V_i G_{ij}$  ( $j = 1, \dots, n$ ), and the  $(n + 1)$ th bottom bar had the output current of  $I_{ref} = \sum_i^m V_i G_{ref}$ . The weighted summation was completed in the sense that  $v_j^l = \frac{I_j - I_{ref}}{V_0 G_0}$ . However, the currents  $I_j$  and  $I_{ref}$  will not be measured, nor do we need to explicitly calculate the membrane potential  $v_j^l$ .

**Stochastic Binarization of the Forwarding Signals in Hardware Circuit.** The stochastic binarization of the forwarding signals, i.e., the Bernoulli sampling process,  $P(x_j^{l+1} = 1) = z_j^l = \frac{1}{1 + \exp(-v_j^l)}$  could be directly implemented in a dedicatedly designed hardware circuit (Figure 1d and S1a, Supporting Information). First, a noise current signal  $I_{noise}$  was added into the output current of the bottom bars of the memristor array  $I_j$ . The combined current  $I_j + I_{noise}$  and the reference current  $I_{ref}$  were then converted into voltage signals of  $V_j = -R_t(I_j + I_{noise})$  and  $V_{ref} = -R_t I_{ref}$ , respectively, through trans-impedance amplifiers with  $R_t$  being the feedback resistance. The trans-impedance amplifiers also pulled the bottom bars into a virtually grounded state. The voltage signals of the trans-impedance amplifiers were fed to a comparator that outputs logic 1 voltage level (i.e.,  $V_{DD}$ ) when  $V_j < V_{ref}$  and logic 0 voltage level (i.e., 0V) otherwise. Figure S2a,b, Supporting Information, shows the typical behaviors of such a circuit, where the noise current was obtained by amplifying the thermal noise of resistors. The comparator’s output voltage  $V_{outj}^l$  was sampled by a flip-flop and the sampled value was taken as the input signal  $x_j^{l+1}$  for the next layer.

If  $I_{noise}$  follows normal distribution  $I_{noise} \sim N(\mu, \sigma^2)$ , we have

$$\begin{aligned} P(x_j^{l+1} = 1) &= P(V_j < V_{ref}) = P(I_j + I_{noise} > I_{ref}) \\ &= P(I_{noise} > I_{ref} - I_j) = 1 - \int_{-\infty}^{I_{ref} - I_j} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) dx \\ &= \frac{1}{2} \left[ 1 + \operatorname{erf}\left(\frac{(I_j - I_{ref}) + \mu}{\sigma\sqrt{2}}\right) \right] = \frac{1}{2} \left[ 1 + \operatorname{erf}\left(\frac{v_j^l V_0 G_0 + \mu}{\sigma\sqrt{2}}\right) \right]. \end{aligned} \quad (3)$$

Note that Equation (3) is also a sigmoid function. For the given read voltage  $V_0$  and scaling factor  $G_0$ , Equation (3) can closely approximate the desired logistic function of  $z_j^l = \frac{1}{1 + \exp(-v_j^l)}$  if the expectation and the standard deviation of the current noise, i.e.,  $\mu$  and  $\sigma$ , could be appropriately chosen.

For typical values  $V_0 = 0.1$  V and  $G_0 = 10^{-6}$  S, the circuit behaviors with different noise current levels were simulated. The simulation showed that when  $\mu = 0$   $\mu$ A and  $\sigma = 0.175$   $\mu$ A, the circuit behavior captured the logistic function well (Figure S2c, Supporting Information). Similar stochastically activated neuronal behaviors have been recently reported exploiting various types of noise sources.<sup>[23,47,48]</sup> When  $\mu = 0$   $\mu$ A, changing of noise levels was equivalently scaling the membrane potential  $v_j^l$  in the logistic function with a prefactor  $a$ , that is,  $\frac{1}{1 + \exp(-av_j^l)}$ . Since the prefactor would not affect the overall learning performance (Supporting Information Figure S5a–e), a coarse control of the noise level would be sufficient.

Also, note that Equation (3) is the CDF of the normally distributed current noise  $I_{noise}$ , to the current difference  $I_j - I_{ref}$ . We have confirmed that the proposed neural network algorithms work well as long as the activation function is of the sigmoid type (Figure S5, Supporting Information). Thus, we were not constrained to approximate the logistic function. In other words, we were not limited to using the noise source that followed a normal distribution, and any type of noise can be used for the stochastic binarization, thanks to the simple fact that the CDF is always of the sigmoid type.

**Stochastic Binarization of the Derivatives in Hardware Circuit:** After the logistic function  $z_j^l = \frac{1}{1 + \exp(-v_j^l)}$  was stochastically binarized, the stochastic

binarization of its derivative  $P\left(\frac{\partial z_j^l}{\partial v_j^l} = 1\right) = z_j^l(1 - z_j^l)$  could be easily implemented. As shown in Figure 1e and S1b, Supporting Information, we used two flip-flops to conduct two independent Bernoulli sampling processes on the comparator’s output  $V_{outj}^l$  and the sampled values (logic signal A and B) were processed by a logic gate. The logic gate was composed of a NOT gate and an AND gate. The NOT gate reversed the second sampled value ( $\bar{B}$ ), while the AND gate output the logical conjunction of the first sampled value A and the reversed second sampled value  $\bar{B}$ , that is,  $A \cap \bar{B}$ . The logic gate output 1 only when the first sampled value was 1 and the second sampled value was 0. Thus, assuming that  $I_{noise}$  follows normal distribution  $I_{noise} \sim N(\mu, \sigma^2)$ , we have

$$\begin{aligned} P\left(\frac{\partial z_j^l}{\partial v_j^l} = 1\right) &= P(A = 1)[1 - P(B = 1)] = P(V_j < V_{ref})[1 - P(V_j < V_{ref})] \\ &= P(I_{noise} > I_{ref} - I_j)P(I_{noise} < I_{ref} - I_j) \\ &= \frac{1}{2} \left[ 1 + \operatorname{erf}\left(\frac{v_j^l V_0 G_0 + \mu}{\sigma\sqrt{2}}\right) \right] * \frac{1}{2} \left[ 1 - \operatorname{erf}\left(\frac{v_j^l V_0 G_0 + \mu}{\sigma\sqrt{2}}\right) \right] \\ &\approx \frac{1}{1 + \exp(-v_j^l)} \left[ 1 - \frac{1}{1 + \exp(-v_j^l)} \right] = z_j^l(1 - z_j^l). \end{aligned} \quad (4)$$

Figure S2d, Supporting Information, shows the comparison between the desired derivative probability and circuit behaviors of several different noise levels. When  $\mu = 0 \mu\text{A}$  and  $\sigma = 0.175 \mu\text{A}$ , the memristor array hardware's behavior could well resemble the desired activation's derivative.

The derivation of Equation (4) relied on two assumptions: 1) the similarity between the logistic function and the CDF of the normal distribution; 2) the simple expression of the derivative of the logistic function  $z_j^l(1 - z_j^l)$  where the argument of the logistic function,  $y_j^l$ , was not explicitly needed. However, these two assumptions should not be the priorities for us to use the designed hardware circuit for the stochastic binarization of the activation function's derivative. As we have confirmed shown in Figure S5f-k, Supporting Information, the BS learning algorithm would work nicely as long as the activation was of sigmoid type (has an "S" shape) and the "function derivative" was bell shaped. The "function derivative" did not need to be the exact derivative of the activation function. For any type of noise source, if the membrane potential  $y_j^l \propto I_j - I_{\text{ref}}$  was small enough, the  $P(A = 1) = P(I_{\text{noise}} > I_{\text{ref}} - I_j)$  approximated 0, and so did  $P\left(\frac{\partial z_j^l}{\partial y_j^l} = 1\right)$ . While, if the membrane potential was large enough, the  $1 - P(B = 1) = P(I_{\text{noise}} < I_{\text{ref}} - I_j)$  approximated 0, and so did  $P\left(\frac{\partial z_j^l}{\partial y_j^l} = 1\right)$ . When the membrane potential was near the expectation of the noise sources, that is,  $I_{\text{noise}} \approx I_{\text{ref}} - I_j$ , both  $P(A = 1)$  and  $1 - P(B = 1)$  approximated 0.5, and  $P\left(\frac{\partial z_j^l}{\partial y_j^l} = 1\right)$  took the maximal value of 0.25. Thus, a "bell" shape of the "function derivative" was well produced.

Equation (4) has the maximal value of 0.25. It could be linearly scaled to have the maximum values of 0.5 and 1 (Figure S5b,c, Supporting Information) by repeating the circuit behavior of Equation (4) for two times and four times, respectively, and taking the logical disjunction of the outputs of these repetitions as the stochastic binarization of the derivative. The probability of the logical disjunction was the union of the individual implementation of Equation (4).

Note that the two logic levels  $A$  and  $B$  should be sampled independently, which is now realized by sampling the comparator's output  $V_{\text{out}_j}$  in two clock cycles. Otherwise, the joint probability in Equation (4) was not valid. It should also be reminded that these two sampling processes should both be independent of the sampling of the forwarding signals in Equation (3). Otherwise, the learning rule denoted by Equation (2) would not succeed since the equivalence to the Equation (1) by the law of total expectation was invalid. Thus, the forwarding pass needed at least three clock cycles to obtain the stochastic binarizations of both the forwarding signals and the derivatives (top right inset in Figure S1, Supporting Information).

**Multiple Methods of Realizing the BS in Hardware:** The stochastic binarization for the forwarding signals and activation derivatives could be achieved in multiple ways, in addition to our proposal (Figure S2, Supporting Information). For instance, the intrinsic noise in the crossbar array of memristors,<sup>[23]</sup> and the stochastic nature of the switching process of a diffusive memristor<sup>[47]</sup> or a magnetic tunnel junction<sup>[48]</sup> could also be exploited to obtain the stochastic binary output in the neuron.

**Limitation of Previous Utilizations of the Hardware Stochasticity:** The BS that could be provided by the intrinsic noise or stochastic nature made them ready to be employed in Hopfield-type neural networks, such as finding the global minima in constraint satisfaction problems<sup>[23,48-50]</sup> or learning through contrastive divergence in restricted Boltzmann machines.<sup>[51,52]</sup> However, for deep learning with the error backpropagation and gradient descent rule, the jigsaw puzzle of the in situ learning independent of the von-Neumann architecture within a neuromorphic system was not completed. For instance, a neural sampling machine with stochastic synapses of ferroelectric field effect transistors was recently reported for the learning and inference of a fully connected neural network.<sup>[53]</sup> While the forwarding signals were stochastically binarized, the error backpropagation and the gradient of the weights were calculated in a traditional HP scheme.<sup>[54]</sup> Thus, only the inference and the forward pass in the learning stage were accelerated. The error backpropagation still needed complex

peripheral circuits, and the weight update operation needed to be precisely controlled. We completed the jigsaw puzzle by introducing the stochastic binarization of the activation derivatives and the sign of backpropagating errors. They helped to accelerate the error backpropagation and weight update in the hardware implementation of deep learning.

**Backward Weighted Summation by the Memristor Crossbar:** The same memristor crossbar was used for the backward weighted summation of the backpropagating errors  $\delta x_i^l = \sum_j \delta y_j^l w_{ij}^l$ . The ternary valued errors of member potentials  $\delta y_j^l$  (taking values of "-1", "0", and "1") are represented by signals of  $-V_{\text{DD}}$ , high impedance, and  $V_{\text{DD}}$ , respectively. Level shifters were used to convert the voltage signals  $-V_{\text{DD}}$  and  $V_{\text{DD}}$  to read voltages with the amplitude of  $-V_0$  and  $V_0$ , respectively, and to convert the high-impedance signal to 0V. Thus, the voltages  $V_j^p = V_0 \delta y_j^l$  ( $j = 1, \dots, n$ ) were applied to the  $j$ th bottom bar of the memristor array (Figure 1f and S1c, Supporting Information). According to Ohm's law and Kirchhoff's current law, if the  $i$ th top bar was grounded, it had the current output of  $I_i^p = \sum_j V_j^p G_{ij}$  ( $i = 1, \dots, m$ ), and the  $(m + 1)$ th top bar had the output current of  $I_{\text{ref}} = \sum_j V_j^p G_{\text{ref}}$ . The backward weighted summation was completed in the sense that  $\delta x_i^l = \frac{I_i^p - I_{\text{ref}}}{V_0 G_0}$ . Similar to the case in the forward-weighted summation, the currents  $I_i^p$  and  $I_{\text{ref}}$  will not be measured, nor do we need to explicitly calculate the errors  $\delta x_i^l$ .

**Sign Operation of the Backpropagating Errors in the Hardware Circuit:** The sign operation of the backpropagating error on a neuron  $\delta z_j^l = \text{sign}(\delta x_j^{l+1})$  was conducted by a current comparator who compared the currents  $I_i^p$  with  $I_{\text{ref}}$  (Figure 1f and S1c, Supporting Information). The current comparator output the positive voltage  $V_{\text{DD}}$  to represent the signed error 1 when  $I_i^p \geq I_{\text{ref}}$  and output the negative voltage  $-V_{\text{DD}}$  to represent the signed error -1 otherwise (Figure 1f). In Figure S1c, Supporting Information, the current operator was implemented by first converting the currents to voltages through trans-impedance amplifiers and then comparing the voltages through a voltage comparator. The trans-impedance amplifiers also pulled the top bars into a virtually grounded state.

The multiplication between the error on a neuron and the activation function's derivative  $\delta y_j^l = \delta z_j^l \frac{dz_j^l}{dy_j^l}$  could be implemented by a single transistor (Figure 1f and S1c, Supporting Information), since the error on a neuron  $\delta z_j^l$  was binary valued ( $-V_{\text{DD}}$  and  $V_{\text{DD}}$  for -1 and 1, respectively), and the derivative  $\frac{dz_j^l}{dy_j^l}$  was binary valued (0 and  $V_{\text{DD}}$  for 0 and 1, respectively). The former was applied to the source/drain of a transistor, while the latter was applied to the gate of the transistor. The product was then presented in the drain/source terminal of the transistor, being  $-V_{\text{DD}}$ , high impedance, and  $V_{\text{DD}}$  for -1, 0, and 1, respectively.

**The Input to the First Layer:** The first layer of a deep neural network, or the input layer, received information directly from the training samples during the learning. To make the first layer compatible with the BS learning algorithm and the memristive hardware implementation scheme, the input information from the training samples should also be stochastically binarized. The input information was first normalized to the range of 0 to 1 and then binarized to either 0 or 1 through the Bernoulli process.

For instance, in the task of learning the handwritten digits from the MNIST dataset, the raw data are represented in INT8: the gray scale of each pixel of the digit image is represented by an integer from 0 to 255 with 0 being fully black and 255 fully white (Figure 2a). These integers were scaled to values between 0.0 and 1.0 by dividing the integer with the largest number 255. The scaled pixel values were used as the probability of the corresponding input nodes taking value 1 instead of 0. In the control experiment of HP learning, the scaled pixel values were directly used as the values of input nodes.

**The Activation of the Output Layer:** The output layer of the neural networks should also be specially cared for. In the demonstrations of the classification tasks (Figure 2 and 4), the SoftMax activation function and cross-entropy loss were used for the output layer. The activation of the

output layer was given by  $z_j = \frac{\exp(\gamma_j)}{\sum_k \exp(\gamma_k)}$ , where  $\gamma_j$  is the membrane potential of the neurons in the output layer and  $j = 1, \dots, 10$  for the classification tasks in this work. The cross-entropy loss given by  $L = -\sum_j t_j \log(z_j)$ , where  $t_j$  ( $j = 1, \dots, 10$ ) is the target membrane potential of the neuron given by the corresponding label of the training sample and encoded in a one-hot format. The strategy of deep learning was to reduce cross-entropy loss by adjusting all the weights in the neural network.

Traditionally, the errors of the membrane potential of the neurons in the output layer ( $\delta\gamma_j$ ), that is, the blames of the resulting cross-entropy loss that could be assigned to the membrane potentials, were given by the gradient of the cross-entropy loss to the membrane potential of each neuron, reading

$$\delta\gamma_j = \frac{\partial L}{\partial \gamma_j} = \sum_k \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial \gamma_j} = z_j - t_j \quad (5)$$

The errors of the neurons  $\delta\gamma_j$  were then backpropagated in the deep neural network. Note that the value of the cross-entropy loss  $L$  was not needed for calculating  $\delta\gamma_j$ .

In BS learning, the activation of the output layer  $z_j$  acted as the probability of the output neuron to be activated, that is  $P(z_j^B = 1) = z_j$ , where  $z_j^B$  is the binary state of the output neuron. The error of the membrane potential  $\delta\gamma_j$  for backpropagating was replaced by

$$\delta\gamma_j = z_j^B - t_j \quad (6)$$

Since the target outputs  $t_j$  were also binary valued in one-hot format, the errors  $\delta\gamma_j$  were ternary valued ( $-1, 0, \text{ or } 1$ ), the same as the errors in previous layers.

The calculation of the SoftMax activation and the Bernoulli sampling process was currently implemented in the software. However, they could also be implemented by hardware circuits with the help of noise sources without explicitly obtaining the exact value of the membrane potential  $\gamma_j$  and calculating the SoftMax activation function. To achieve this, the winner-take-all mechanism with inhibitory connections among neurons should be employed, similar to the case in biological systems.<sup>[55]</sup>

**Fully Connected Neural Network:** The neural network shown in Figure 2a consisted of three fully connected layers with the size of  $784 \times 500$ ,  $500 \times 200$ , and  $200 \times 10$ . The 784 input nodes corresponded to 784 ( $28 \times 28$ ) pixels of one MNIST training sample and the 10 output nodes corresponded to the 10 classes of digits.

In Figure S5a–e, Supporting Information, we tested the effect of the shape parameter  $a$  of the logistic activation function  $z_j^l = \frac{1}{1 + \exp(-a\gamma_j^l)}$  in various signal/error/derivative handling methods. The forwarding signals (F) could be in HP or stochastic binary (S) format; the backpropagation errors (E) could be in HP or signed (S) format; the derivative (D) could be in HP or stochastic binary (S) format, as shown in Figure S5e, Supporting Information. In stochastic binarization, the “probability” that is higher than 1 was truncated (e.g., in Figure S5d, Supporting Information). The cases labeled by “F: HP, E: HP, D: HP” corresponded to the HP learning in the main text, whereas the cases labeled by “F:S, E:S, D:S” corresponded to the BS learning. The highest learning performance (>99% recognition accuracy) was achieved when the forwarding signals and the derivatives were binarized and the backpropagating errors were in HP, i.e., “F: S, E: HP, D: S” with the shape parameter  $a$  being 8. However, this case was not fully compatible with the designed memristive hardware circuit.

We also tested the effect of other types of activation functions in Figure S5f–k, Supporting Information. Note that, for the rectifying linear unit activation function (Figure S5f, Supporting Information), the learning algorithms with HP forwarding signals worked well, whereas the learning algorithms with BS forwarding signals were disruptive. As shown in Figure S5i,j, Supporting Information, the “derivative functions” that were used in the learning were faked (they are not the derivative function of the

activation functions). In these two cases, the learning algorithms still worked well.

We used the logistic activation function with the shape parameter  $a$  being 4 (Figure S5c, Supporting Information) for the learning results shown in Figure 2a,3, S3, and S4, Supporting Information.

All the learnings used a batch size of 100 and a fixed learning rate  $\eta = 0.1$ . By default, we trained the neural network for 1000 epochs. No other learning performance enhancement techniques, such as dropout, batch normalization, and data preprocessing, were used.

**Deep Convolutional Neural Network:** The learning algorithms for the fully connected layers could be directly transferred to the convolutional layers with minor changes. The weighted summation in the fully connected layer between the 1D input vector and the 2D synaptic matrix became the convolution between 2D/3D feature maps and convolutional kernels. For the convolutional layer without being followed by a pooling layer, the transmission of the forwarding signals, the derivatives, and the backpropagating errors were the same as in the fully connected layers. For the convolutional layer followed by a pooling layer, it should be decided where the activation function and the Bernoulli sampling should be performed. In this work, we used the max-pooling layer, with the following strategy: 1) the activation function was performed on the membrane potentials of the neurons in the convolutional layers; 2) the max-pooling was performed on the activation; and 3) the Bernoulli samplings of both the forwarding signals and the derivatives were performed on the neurons in the pooling layers.

In Figure 4a, for learning and recognizing the handwritten digits in the MNIST dataset, the deep convolutional neural network consisted of two convolutional layers, two max-pooling layers, and one fully connected layer. The first convolutional layer had eight filters that were using kernel sizes of  $9 \times 9$ . The second convolutional layer had 12 filters that were using kernel sizes of  $5 \times 5$ . Both convolutional layers were followed by nonoverlapping max-pooling layers with pooling sizes of  $2 \times 2$ . The fully connected layer used a  $108 \times 10$  synaptic weight matrix. We used the logistic activation function with the shape parameter  $a$  being 4 for regular layers, except the output layer which used the SoftMax activation function. The neural network was trained using a batch size of 100 and a fixed learning rate of 0.1.

In Figure 4f, for learning and recognizing images in the CIFAR-10 dataset, the input was of dimensions of  $32 \times 32 \times 3$ , with 3 being the red, green, and blue channels of the colored images. The deep convolutional neural network uses a VGG style, consisting of six convolutional layers, three max-pooling layers, and three fully connected layers. All convolutional layers had the same kernel size of  $3 \times 3$  with padding on the edges. All max-pooling layers used the nonoverlapping pooling windows with the same size of  $2 \times 2$ . There was one pooling layer following two convolutional layers. The sizes of the feature maps and the channels for each layer are given in Figure 4f. The neural network was trained using a batch size of 100 and a fixed learning rate of 0.01.

**Modeling of the Synaptic Behaviors of the Memristors:** An empirical model capturing the synaptic behavior of LTP and LTD under identical pulses was used to simulate the synaptic plasticity of the analog memristors with non-idealities.<sup>[52]</sup> This model considered the on/off ratio, the nonlinearities ( $\alpha_p$  and  $\alpha_d$ ), the asymmetry between potentiation and depression, and the write variations. The median conductance changes (without cycle-to-cycle write variations) for a memristor device with conductance  $G_{ij}$  under potentiation pulses and depression pulses can be written as

$$\overline{\Delta G_{\text{pot}}} = \left[ \frac{G_{\text{max}} - G_{\text{min}}}{1 - e^{-\alpha_p}} - (G_{ij} - G_{\text{min}}) \right] (1 - e^{-\alpha_p/N_p}) \quad (7)$$

and

$$\overline{\Delta G_{\text{dep}}} = - \left[ \frac{G_{\text{max}} - G_{\text{min}}}{1 - e^{-\alpha_d}} - (G_{\text{max}} - G_{ij}) \right] (1 - e^{-\alpha_d/N_d}) \quad (8)$$

respectively. Here,  $N_p$  and  $N_d$  are the numbers of pulses needed to fully potentiate and fully depress the memristor devices, respectively, and  $\alpha_p$  and  $\alpha_d$  are the nonlinearities of weight updates in the potentiation and depression phases, respectively.

The cycle-to-cycle write variations are modeled by adding a Gaussian distribution to the conductance change with its standard deviation proportional to the median conductance change from Equation (7) or (8)

$$\Delta G \sim \mathcal{N}[\overline{\Delta G}, (\gamma \overline{\Delta G})^2] \quad (9)$$

where  $\overline{\Delta G} = \overline{\Delta G}_{\text{pot}}$  for potentiation pulses,  $\overline{\Delta G} = \overline{\Delta G}_{\text{dep}}$  for depression pulses, and  $\gamma$  is a parameter controlling the cycle-to-cycle variations. For the synaptic behavior in Figure 5b, the values of the parameters were  $G_{\text{max}} = 25 \mu\text{S}$ ,  $G_{\text{min}} = 0.1 \mu\text{S}$ ,  $N_p = N_d = 100$ ,  $\alpha_p = 1$ ,  $\alpha_d = 2$ , and  $\gamma = 2$ . Note that, due to the write variation, the sign of the actual conductance change  $\Delta G$  had a large chance to be in the opposite direction of the desired change.

**Estimation of the Energy Consumption and on-Chip Footprint:** The energy consumption was estimated in terms of the elementary MAC operations in neural networks (Table S2, Supporting Information).

Implemented in CMOS technology, i.e., using the central processing units, GPUs, or other dedicated application-specific integrated chips, the MAC operations for traditional HP learning were conducted typically using FP32 numbers. Each MAC operation consisted of the multiplication between two FP32 numbers and the addition of the production to another FP32 number. In a 45 nm CMOS technology node and at 0.9 V supply voltage, this MAC operation<sup>[41]</sup> consumed a power of  $3.7 \text{ pJ} + 0.9 \text{ pJ} = 4.6 \text{ pJ}$ . In the stochastic binary learning algorithm, the multiplication became the production of a Boolean number (0 or 1) and an FP32 number, which did not need to be conducted explicitly. If the Boolean number was 1, the MAC operation only required the addition of the FP32 number on another FP32 number. If the Boolean number was 0, the addition was not needed. Conservatively, we assumed each MAC operation needed one addition between two FP32 numbers, thus the consumed power being reduced to 0.9 pJ. When the weight was quantized to integers, the MAC operation degraded to addition between two integers. For INT8 weights, each MAC operation consumed at most 0.03 pJ. It was convenient to assume that the energy consumption of the addition of two integers was proportional to the bit-width of the integers. Thus, the MAC operations in INT4 weights and ternary valued weights ( $\approx 1.5$  bits) were estimated to consume power of 15 and 5.6 fJ, respectively.

The crossbar array of the memristors performed all MAC operations in one vector–matrix multiplication parallelly. For instance, the crossbar array with a typical size of 128-by-128 performed  $128 \times 128 = 16384$  MAC operations, parallelly. A macrocore of such an array<sup>[7]</sup> that processed 1-bit input and sensed the output currents of the array with analog-to-digital converters consumed a power of 371.89 pJ. To implement the HP learning algorithm, sufficient input accuracy (e.g., 8-bit) was needed for acceptable degradation of the learning performance. There were also shift and adder circuits in the macrocore to shift and accumulate the bit-wised MAC results. Thus, each effective MAC operation consumed a power of  $\frac{371.89 \text{ pJ} \times 8}{16384} = 0.18 \text{ pJ}$ . Using our stochastic binary learning algorithm, the ADCs and shift and adder were not needed, thus parallel MAC operations consumed only 29.23 pJ. Additionally, only 1-bit input was needed. Thus, each MAC operation consumed a power of  $\frac{29.23 \text{ pJ}}{16384} = 1.8 \text{ fJ}$ . The expense induced by trans-impedance amplifiers, the comparators, and the flip-flops in the proposed hardware implementation (Figure S1, Supporting Information) was not counted in the comparison. They were performing the calculation of the activation function which was done in the digital domain in the benchmark work.<sup>[7]</sup> The energy efficiency for the MAC operation was approximated to be  $556 \text{ TOP}_5^{-1} \text{ W}^{-1}$ .

The footprint of the implementation circuit would also be greatly reduced. To implement the HP learning algorithm, the parallel  $128 \times 128 = 16384$  MAC operations needed an on-chip area of  $63801.92 \mu\text{m}^2$ . Assuming that each input bit took 50 ns, the area efficiency for 8-bit input was calculated as  $\frac{16384}{8 \times 50 \text{ ns} \times 63801.92 \mu\text{m}^2} = 641.99 \text{ GOP}_5^{-1} \text{ mm}^{-2}$ . Excluding the ADCs and shift and adder, the chip area reduced to  $8824.3 \mu\text{m}^2$ . The effective area efficiency was then  $\frac{16384}{50 \text{ ns} \times 8824.3 \mu\text{m}^2} = 37.13 \text{ TOP}_5^{-1} \text{ mm}^{-2}$ , reduced by 57.8 times.

**Periodical Carry to Accumulate the Ternary Gradient:** The essential of the periodical carry was to accumulate the gradient of the loss function to the weight in a separated memory cell and update the synaptic weight periodically and stepwise. It was efficient in compensating for the nonlinearity and fluctuation of weight changes.<sup>[29,33,34,56]</sup> Traditionally, the calculation and accumulation of the gradient should be conducted with sufficient precision, for instance, in dedicated capacitors<sup>[56]</sup> or HP digital circuits.<sup>[34]</sup> In this work, the binarized three factors resulted in a ternary gradient (valued as  $-1, 0, \text{ or } 1$ ), which leads to twofold benefits. First, the calculation of the gradient for an array of weights, a vector–vector outer product, could be performed parallelly.<sup>[52]</sup> Second, the gradient was accumulated in a unit step.

**Limitation on Neural Network Depth:** BS learning was, however, limited by the depth of the neural network. Learning of a neural network with more than 10 layers was difficult. This was because a long chain of stochastic binarization over initially random weight matrices lost meaningful information. This issue could be partially compensated by pretraining the neural network in HP format or could increase the batch size, according to our experiences with software simulation. It should be further investigated for extending the proposed algorithms to deeper neural networks for state-of-the-art artificial intelligence applications. However, it should be noted that the human-brain visual ventral pathway mainly consists of several areas, including the retina, lateral geniculate nucleus (LGN), V1, and V4, which correspond to the artificial neural network layers in this work.<sup>[57]</sup> Within this depth limit, the BS learning algorithm worked well.

## Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

## Acknowledgements

The authors acknowledge the support from Basic and Frontier Research Project of the Peng Cheng Laboratory, major key project of the Peng Cheng Laboratory (grant no. PCL2023AS2-3), major key project of the Peng Cheng Laboratory (grant no. PCL2023AS3-1), National Natural Science Foundation of China (NSFC) (grant no. 62206141), National Key R&D Program of China (grant no. 2021YFB3601200), and National Nature Science Foundation of China (grant no. 62104042).

## Conflict of Interest

The authors declare no conflict of interest.

## Data Availability Statement

The data that support the findings of this study are available in the supplementary material of this article. The source codes that support the findings of this study are available at <https://github.com/leonlee2023/binary-stochasticity>.

## Keywords

backpropagation, deep learning, neuromorphic computing, signal binarization, stochastic sampling

Received: July 10, 2023  
Revised: September 4, 2023  
Published online:

- [1] M. A. Zidan, J. P. Strachan, W. D. Lu, *Nat. Electron.* **2018**, *1*, 22.
- [2] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, J. P. Strachan, *Adv. Mater.* **2018**, *30*, 1705914.
- [3] D. Ielmini, H.-S. P. Wong, *Nat. Electron.* **2018**, *1*, 333.
- [4] Y. LeCun, Y. Bengio, G. Hinton, *Nature* **2015**, *521*, 436.
- [5] K. He, X. Zhang, S. Ren, J. Sun, in *2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Piscataway, NJ **2016**, pp. 770–778.
- [6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, B. Kingsbury, *IEEE Signal Process. Mag.* **2012**, *29*, 82.
- [7] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, H. Qian, *Nature* **2020**, *577*, 641.
- [8] S. Agarwal, S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James, M. J. Marinella, in *2016 Inter. Joint Conf. on Neural Networks (IJCNN)*, IEEE, Piscataway, NJ **2016**, pp. 929–938.
- [9] T. Gokmen, Y. Vlasov, *Front. Neurosci.* **2016**, *10*, 333.
- [10] B. P. Bean, *Nat. Rev. Neurosci.* **2007**, *8*, 451.
- [11] M. Häusser, *Nat. Neurosci.* **2000**, *3*, 1165.
- [12] D. E. Feldman, *Neuron* **2012**, *75*, 556.
- [13] G. Bi, M. Poo, *J. Neurosci.* **1998**, *18*, 10464.
- [14] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *Nature* **1986**, *323*, 533.
- [15] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, *Proc. IEEE* **1998**, *86*, 2278.
- [16] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, T. Blankevoort, (Preprint) arXiv:2106.08295, **2021**.
- [17] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, (Preprint) arXiv:1602.02830, **2016**.
- [18] Y. Bengio, N. Léonard, A. Courville, (Preprint) arXiv:1308.3432, **2013**.
- [19] L. Deng, P. Jiao, J. Pei, Z. Wu, G. Li, *Neural Networks* **2018**, *100*, 49.
- [20] E. O. Neftci, H. Mostafa, F. Zenke, *IEEE Signal Process. Mag.* **2019**, *36*, 51.
- [21] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, *Nat. Commun.* **2018**, *9*, 2385.
- [22] Q. Xia, J. J. Yang, *Nat. Mater.* **2019**, *18*, 309.
- [23] M. R. Mahmoodi, M. Prezioso, D. B. Strukov, *Nat. Commun.* **2019**, *10*, 5113.
- [24] R. J. Williams, *Mach. Learn.* **1992**, *8*, 229.
- [25] J. J. Hopfield, *Proc. Natl. Acad. Sci.* **1984**, *81*, 3088.
- [26] W. S. McCulloch, W. Pitts, *Bull. Math. Biophys.* **1943**, *5*, 115.
- [27] K. Simonyan, A. Zisserman, in *3rd Int. Conf. Learn. Represent. ICLR 2015*, San Diego, CA May 7–9, **2015**, <https://dblp.org/db/conf/iclr/iclr2015.html>.
- [28] A. Krizhevsky, Learning multiple layers of features from tiny images, **2009**, <http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf>.
- [29] T. Gokmen, W. Haensch, *Front. Neurosci.* **2020**, *14*, 103.
- [30] W. Wang, L. Danial, Y. Li, E. Herbelin, E. Pikhay, Y. Roizin, B. Hoffer, Z. Wang, S. Kvatinsky, *Nat. Electron.* **2022**, *5*, 870.
- [31] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H.-S. P. Wong, G. Cauwenberghs, *Nature* **2022**, *608*, 504.
- [32] S. Choi, S. H. Tan, Z. Li, Y. Kim, C. Choi, P. Y. Chen, H. Yeon, S. Yu, J. Kim, *Nat. Mater.* **2018**, *17*, 335.
- [33] S. Agarwal, R. B. J. Gedrim, A. H. Hsia, D. R. Hughart, E. J. Fuller, A. A. Talin, C. D. James, S. J. Plimpton, M. J. Marinella, in *Digest of Technical Papers - Symp. on VLSI Technology*, IEEE, Piscataway, NJ **2017**, pp. T174–T175.
- [34] S. R. Nandakumar, M. Le Gallo, C. Piveteau, V. Joshi, G. Mariani, I. Boybat, G. Karunaratne, R. Khaddam-Aljameh, U. Egger, A. Petropoulos, T. Antonakopoulos, B. Rajendran, A. Sebastian, E. Eleftheriou, *Front. Neurosci.* **2020**, *14*, 406.
- [35] J. Tang, D. Bishop, S. Kim, M. Copel, T. Gokmen, T. Todorov, S. Shin, K. T. Lee, P. Solomon, K. Chan, W. Haensch, J. Rozen, in *Technical Digest - Inter. Electron Devices Meeting, IEDM*, IEEE, Piscataway, NJ **2018**, pp. 13.1.1–13.1.4.
- [36] J.-W. Jang, S. Park, G. W. Burr, H. Hwang, Y.-H. Jeong, *IEEE Electron Device Lett.* **2015**, *36*, 457.
- [37] M. Suri, O. Bichler, D. Querlioz, B. Traoré, O. Cueto, L. Perniola, V. Sousa, D. Vuillaume, C. Gamrat, B. Desalvo, *J. Appl. Phys.* **2012**, *112*, 4749411.
- [38] P. Huang, D. Zhu, S. Chen, Z. Zhou, Z. Chen, B. Gao, L. Liu, X. Liu, J. Kang, *IEEE Trans. Electron Devices* **2017**, *64*, 614.
- [39] T. Cao, C. Liu, Y. Gao, W. L. Goh, *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2022**, *12*, 436.
- [40] L. Shi, G. Zheng, B. Tian, B. Dkhil, C. Duan, *Nanoscale Adv.* **2020**, *2*, 1811.
- [41] M. Horowitz, in *2014 IEEE Inter. Solid-State Circuits Conf. (ISSCC)*, IEEE, Piscataway, NJ **2014**, pp. 10–14.
- [42] S. E. Fahlman, G. E. Hinton, T. J. Sejnowski, in *AAAI-83 Conf.*, **1983**, pp. 109–113.
- [43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, *J. Mach. Learn. Res.* **2014**, *15*, 1929.
- [44] M. Zahedi, T. Shahroodi, S. Wong, S. Hamdioui, (Preprint) arXiv:2211.06261, **2022**.
- [45] S. Yin, X. Sun, S. Yu, J. S. Seo, *IEEE Trans. Electron Devices* **2020**, *67*, 4185.
- [46] S. Yu, Z. Li, P. Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, H. Qian, in *Technical Digest - Inter. Electron Devices Meeting, IEDM*, IEEE, Piscataway, NJ **2017**, pp. 16.2.1–16.2.4.
- [47] K. S. Woo, J. Kim, J. Han, W. Kim, Y. H. Jang, C. S. Hwang, *Nat. Commun.* **2022**, *13*, 5762.
- [48] W. A. Borders, A. Z. Pervaiz, S. Fukami, K. Y. Camsari, H. Ohno, S. Datta, *Nature* **2019**, *573*, 390.
- [49] F. Cai, S. Kumar, T. Van Vaerenbergh, X. Sheng, R. Liu, C. Li, Z. Liu, M. Foltin, S. Yu, Q. Xia, J. J. Yang, R. Beausoleil, W. D. Lu, J. P. Strachan, *Nat. Electron.* **2020**, *3*, 409.
- [50] Z. Jonke, S. Habenschuss, W. Maass, *Front. Neurosci.* **2016**, *10*, 118.
- [51] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, G. Cauwenberghs, *Front. Neurosci.* **2016**, *10*, 241.
- [52] W. Wang, B. Hoffer, T. Greenberg-Toledo, Y. Li, M. Zou, E. Herbelin, R. Ronen, X. Xu, Y. Zhao, J. Yang, S. Kvatinsky, *Adv. Intell. Syst.* **2022**, *4*, 2100249.
- [53] S. Dutta, G. Detorakis, A. Khanna, B. Grisafe, E. Neftci, S. Datta, *Nat. Commun.* **2022**, *13*, 2571.
- [54] G. Detorakis, S. Dutta, A. Khanna, M. Jerry, S. Datta, E. Neftci, in *33rd Inter. Conf. on Neural Information Processing Systems*, Vancouver, BC, Canada **2019**, pp. 3291–3302, <https://dblp.org/db/conf/nips/nips2019.html>.
- [55] B. Nessler, M. Pfeiffer, L. Buesing, W. Maass, *PLoS Comput. Biol.* **2013**, *9*, e1003037.
- [56] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, D. Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, Y. Jaoudi, G. W. Burr, *Nature* **2018**, *558*, 60.
- [57] D. G. Amaral, P. L. Strick, in *Principles of Neural Science* (Ed: E. R. Kandel), McGraw Hill, New York **2013**, pp. 338–341.