Editors' Suggestion

Hardware-Aware *In Situ* Learning Based on Stochastic Magnetic Tunnel Junctions

Jan Kaiser,[1] William A. Borders,[2] Kerem Y. Camsari,[3,*] Shunsuke Fukami,[2,4,5,6,7,†] Hideo Ohno,[2,4,5,6,7] and Supriyo Datta[1]

[1] *Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47906, USA*

[2] *Laboratory for Nanoelectronics and Spintronics, Research Institute of Electrical Communication, Tohoku University, Sendai, Japan*

[3] *Department of Electrical and Computer Engineering, University of California Santa Barbara, Santa Barbara, California 93106, USA*

[4] *Center for Innovative Integrated Electronic Systems, Tohoku University, Sendai, Japan*

[5] *Center for Spintronics Research Network, Tohoku University, Sendai, Japan*

[6] *Center for Science and Innovation in Spintronics, Tohoku University, Sendai, Japan*

[7] *WPI-Advanced Institute for Materials Research, Tohoku University, Sendai, Japan*

(Received 19 July 2021; revised 1 November 2021; accepted 13 December 2021; published 13 January 2022)

One of the big challenges of current electronics is the design and implementation of hardware neural networks that perform fast and energy-efficient machine learning. Spintronics is a promising catalyst for this field with the capabilities of nanosecond operation and compatibility with existing microelectronics. Considering large-scale, viable neuromorphic systems however, variability of device properties is a serious concern. In this paper, we show an autonomously operating circuit that performs hardware-aware machine learning utilizing probabilistic neurons built with stochastic magnetic tunnel junctions. We show that *in situ* learning of weights and biases in a Boltzmann machine can counter device-to-device variations and learn the probability distribution of meaningful operations such as a full adder. This scalable autonomously operating learning circuit using spintronics-based neurons could be especially of interest for standalone artificial-intelligence devices capable of fast and efficient learning at the edge.

DOI: 10.1103/PhysRevApplied.17.014016

I. INTRODUCTION

Conventional computers use deterministic bits to operate and encode information. While this approach is effective for well-defined tasks like arithmetic operations, there are many difficult tasks like stochastic optimization, sampling, and probabilistic inference, which instead are readily addressed by utilizing stochasticity. A promising approach for solving these difficult tasks is using computers that are naturally probabilistic. In a well-known piece, Feynman [1] suggested that in the same way that the use of quantum computers is useful to simulate quantum phenomena, a probabilistic computer could be a natural solution to problems that are intrinsically probabilistic. Recently, utilizing spintronics technology, Borders *et al.* [2] demonstrated such an autonomously running probabilistic computer consisting of probabilistic bits (p-bits) with a stochastic magnetic tunnel junction (SMTJ), which can perform computationally hard tasks like integer factorization.

Machine learning is another field in which probabilistic computation and a large amount of random numbers could be highly beneficial. It holds promise for various tasks like image recognition, medical application, and autonomous driving [3–5]. For these applications, conventional von Neumann computers are inefficient and alternative computing architectures inspired by information processing in the human brain are of interest [6–10]. Boltzmann machines offer a promising route for hardware learning due to their local learning rule and tolerance to stochasticity [11–16]. Boltzmann machines are generative stochastic recurrent neural networks having a large application space ranging from optimization to generative machine learning [17–20]. This suggests that building a compact hardware implementation in the form of a probabilistic computer that resembles a Boltzmann machine could be highly beneficial in terms of energy consumption and training speed. While some hardware implementations have been presented for

*camsari@ece.ucsb.edu
†s-fukami@riec.tohoku.ac.jp

restricted Boltzmann machines (RBMs) [14,21,22], in this paper we focus on fully connected unrestricted Boltzmann machines. The usual challenge in learning unrestricted Boltzmann machines is that they are hard to train since the equilibrium samples of the network are harder to extract [19,23]. In this work we show a system that performs this sampling naturally and could hence make it possible to train unrestricted Boltzmann machines more efficiently using the natural physics of SMTJs.

A common concern for the development of neuromorphic systems based on emerging devices like SMTJs is the inevitable device variability [8,24]. This poses an obstacle to deploy these systems for real-world application on a large scale while preserving high reliability. Several approaches have been proposed to overcome these challenges on a device level for example by applying external magnetic fields [25], performing a calibration phase [2] or by postprocessing [26]. Another interesting approach to counter the effect of variability and realize high performance in neuromorphic systems is to perform training and inference on the same hardware system [27–29]. In this paper, we present a proof-of-concept demonstration of a probabilistic computer that can perform *in situ* learning allowing device-to-device variations to be countered naturally as part of its learning process. Here, device variability is addressed on a system's level. We show that devices with nonideal characteristics can be used to perform given tasks successfully without the necessity to individually calibrate each device. This is achieved by learning hardware-aware weights and biases. Such a natural variation tolerance could enable large-scaled implementations of MTJ-based probabilistic computers.

II. HARDWARE-AWARE LEARNING WITH MTJ-BASED *P*-BITS

The main building block of a probabilistic computer is the p-bit, analogous to a binary stochastic neuron (BSN) [30]. Its activation function can be described by [31]

$$m_i(t+\tau_N) = \mathrm{sgn}\{\tanh[I_i(t)] - r\}. \tag{1}$$

Here, m_i is the output of the p-bit and a bipolar random variable, τ_N is the time the p-bit takes to perform the activation operation, I_i is the dimensionless input to p-bit i, and r is a uniformly distributed random number between -1 and $+1$. Equation (1) can also be written in binary notation with a unit step function and a sigmoid function. To connect multiple p-bits, a synaptic function computes the input of every p-bit I_i by taking the weighted sum of all p-bit outputs m_i,

$$I_i(t+\tau_S) = \sum_j W_{i,j}\, m_j(t), \tag{2}$$

where τ_S is the synapse execution time and $W_{i,j}$ is the weight matrix that couples p-bit i and p-bit j. Here, the bias to p-bit i is subsumed into $W_{i,j}$. Given a particular weight matrix, every p-bit configuration has a defined probability given by the Boltzmann distribution where $P(m) \propto \exp\left[-\beta E(m)\right]$ with energy $E(m) = -\sum W_{i,j}\, m_i m_j$ and inverse temperature β. For training a Boltzmann machine, the goal is to find a weight matrix W that results in a Boltzmann distribution that fits closely to the given training vectors $\{v\}$. The distribution of training vectors is referred to as data distribution in this paper. To find a fitting weight matrix for a given data distribution, the weights are trained by performing gradient ascent of the log likelihood [32]. It is well known that the ideal Boltzmann-machine algorithm based on log-likelihood learning is generally intractable since learning time scales exponentially with the size of the system [19,33]. However, it has been shown that the approximate version of the Boltzmann learning rule like the contrastive divergence algorithm [11,12] can be used to perform approximate learning for large Boltzmann-machine systems. This algorithmic scaling motivates the use of domain-specific, efficient, and fast hardware accelerators like the p-bit building block that naturally represents the neuron function of the Boltzmann machine in order to accelerate the learning process [34]. To map the Boltzmann-machine learning algorithm to our hardware system, we use a continuous learning rule similar to the persistent contrastive divergence algorithm given by [35,36]

$$\frac{dW_{i,j}}{dt} = \frac{\langle v_i v_j \rangle - m_i m_j - \lambda W_{i,j}}{\tau_L}, \tag{3}$$

that can be implemented in hardware. Here, $\langle v_i v_j \rangle$ is the average correlation between two neurons in the data distribution where v_i is the training vector entry for p-bit i, $m_i m_j$ is the correlation of the p-bit outputs defined in Eq. (1) and τ_L is the learning time constant. Regularization parameterized by λ assures that weights do not become too large and helps the algorithm to converge to a solution [37]. This learning rule requires only the correlation between two p-bits $m_i m_j$ for updating weight $W_{i,j}$, which makes this learning algorithm attractive for hardware implementations. Equation (3) does not change when the system becomes larger. Another advantage of the presented hardware implementation of the Boltzmann machine is that the computational expensive part of getting the equilibrium samples of the correlation term $m_i m_j$ needed for learning is performed naturally.

Equations (1), (2), (3) are implemented in hardware to build a probabilistic circuit that performs learning. The dimensionless quantities of Eqs. (1) and (2) are converted to the physical quantities shown in Fig. 1 as follows: $m_i = 2 \times V_{\mathrm{out},i}/V_{\mathrm{DD}} - 1$ with p-bit output voltage $V_{\mathrm{out},i}$ and supply voltage V_{DD}, and $I_i = V_{\mathrm{in},i}/V_0$ with p-bit input voltage

$V_{\text{in},i}$ and p-bit reference voltage V_0, which is defined by the response of the p-bit [38]. Equation (3) can be written into circuit parameters using RC elements [36]

$$C\frac{dV_{i,j}}{dt} = \frac{V_{v;i,j} - V_{m;i,j} - V_{i,j}}{R}, \tag{4}$$

where $V_{i,j}$ is the voltage across capacitor C, R is the series resistance, $V_{v;i,j} \mathrel{\widehat{=}} \langle v_i v_j \rangle$ is the voltage representing the average correlation of two neurons in the data distribution and $V_{m;i,j} \mathrel{\widehat{=}} m_i m_j$ is the voltage representing the correlation of p-bit outputs [39]. Equations (3) and (4) can be converted into each other by setting $W_{i,j} = A_v V_{i,j}/V_0$, $\lambda = V_0/(A_v V_{\text{DD}}/2)$, and $\tau_L = \lambda RC$, where A_v is a voltage gain factor between the voltage across the capacitor and the used weight value for the weighted sum in Eq. (2). While for memory usage, nonvolatile storage of a capacitor can be detrimental, the discharging of the capacitor is used here as weight decay or regularization in the learning process that ensures that the learning converges. The voltage gain is used to adjust the regularization parameter λ for the update rule, Eq. (3). High λ produces smaller weight values during learning. More information about the learning rule is presented within the Supplemental Material [40]. Note that while we choose a RC network in this proof-of-concept experiment to conveniently represent analog voltages as weights, the synaptic functionality in our system could also be implemented out of memristor crossbar arrays [27,41,42] to support *in situ* learning by mapping the weight update rule [Eq. (3)] to an equation of changing conductance $G_{i,j}$ instead of changing voltage $V_{i,j}$. The use of memristor crossbars would have the main advantage that the weight storage becomes nonvolatile.

Figure 1(a) shows the block diagram of the learning circuit. The neurons [Eq. (1)] are implemented with a SMTJ in series to a transistor and a resistor R_S. The random number in Eq. (1) is generated by the SMTJ, which fluctuates between two resistance values R_P and R_{AP}, which represents the parallel and antiparallel configuration of the fixed and free layer of the MTJ. While the fixed layer is a normal ferromagnet (FM), the free layer is designed to be a low-barrier magnet (LBM), where the magnetic orientation changes due to thermal noise resulting in resistance fluctuations of the MTJ. The drain voltage gets thresholded by using a comparator [2,38] where the reference voltage is chosen to be $V_{\text{ref}} = V_{\text{DD}} - I_{50/50}\left(R_P + R_{\text{AP}}/2\right)$ with $I_{50/50}$ being the bias current where the stochastic MTJ stays in the parallel and antiparallel 50% of the time. The synapse [Eq. (2)] is implemented by using a microcontroller in conjunction with a digital-to-analog converter (DAC) where the p-bit output voltages $\{V_{\text{out}}\}$ and capacitor voltages $\{V_C\}$ with $V_{i,j} = V_{v;i,j} - V_{C;i,j}$ are taken as an input. To compute the correlation of p-bit outputs $m_i m_j$ a XNOR gate is needed between the p-bit and the learning block [Eq. (3)], where the weights are updated using a RC array. Figure 1(b) shows the printed circuit board (PCB) with the five p-bits and the RC array with 15 RC elements used in the experiment. In Sec. V E, more details about the experimental implementation are presented.

TABLE I. Truth table of full adder. A and B are inputs, C_{in} is the carry in, S the sum, and C_{out} the carry out. In the Boltzmann-machine context, all visible units are equivalent so that inputs and outputs can be written as v'_{1-5}. The bipolar training vectors v_i of Eq. (3) can be calculated from the truth table by converting them from binary to bipolar $v_i = 2v'_i - 1$, where $[v'_1, v'_2, v'_3, v'_4, v'_5] = [A, B, C_{\text{in}}, S, C_{\text{out}}]$ for the data distribution. $P_{\text{ideal}}(v)$ is the ideal data probability distribution where every line has a probability of $p = 1/8 = 0.125$.

A v'_1	B v'_2	C_{in} v'_3	S v'_4	C_{out} v'_5	$P_{\text{ideal}}(v)$
0	0	0	0	0	0.125
0	0	1	1	0	0.125
0	1	0	1	0	0.125
0	1	1	0	1	0.125
1	0	0	1	0	0.125
1	0	1	0	1	0.125
1	1	0	0	1	0.125
1	1	1	1	1	0.125

III. VARIATION-TOLERANT LEARNING OF A FULL ADDER

We demonstrate the learning of the hardware circuit using the data distribution of a full adder (FA). In general, for a fully visible Boltzmann machine with N p-bits, $(N+1)N/2$ weights and biases have to be learned. A FA has three inputs and two outputs resulting in $N = 5$ p-bits. To connect these p-bits, ten weights and five biases have to be learned [in total 15 RC elements as shown in Fig. 1 (b)]. For the FA, the binary inputs $[ABC_{\text{in}}]$ get added and the outputs are given by the sum S and the carry out C_{out}, as shown in Table I. This corresponds to a data distribution that is given by 8 out of the 32 (2^N) possible configurations. Because of the probabilistic nature of this circuit, input and outputs are treated equally, which allows, for example, invertible full-adder operation [31,38] and distinguishes our probabilistic circuit from conventional logic gates that can operate only in one direction. While we choose the FA truth table as data distribution, any probability distribution could be chosen to be represented by our probabilistic circuit. In Sec. V F, the data distribution in the form of the truth table of the FA and the mapping from the truth table to analog voltages $V_{v;i,j}$ is explained in more detail. For the FA, the learning is performed for a total of 3000 s. In the Supplemental Material [40], learning examples for an AND, OR, and XOR gate with less p-bits are shown.

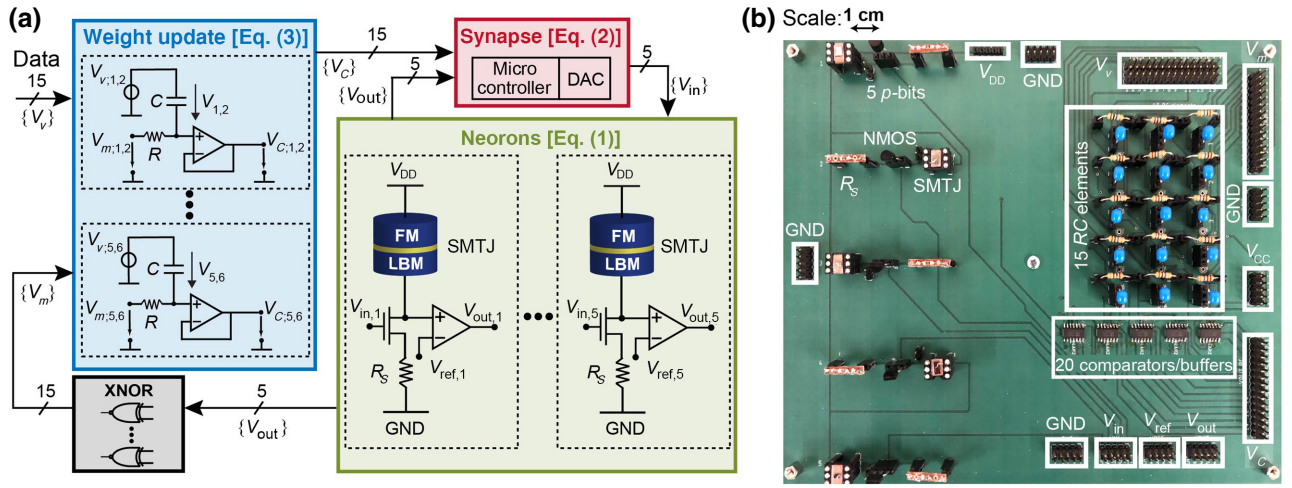

FIG. 1. Probabilistic learning circuit. (a) Block diagram of the learning circuit with p-bit output voltages $\{V_{\text{out}}\}$, p-bit input voltages $\{V_{\text{in}}\}$, weight voltages $\{V_{i,j}\}$, capacitor voltages $\{V_C\}$, p-bit correlation voltages $\{V_m\}$, and data distribution correlation voltages $\{V_v\}$. (b) A photograph of the PCB with the five p-bits (each consisting of a SMTJ, a NMOS transistor, and a source resistor R_S) and 15 RC elements and 20 operational amplifiers (five used as a comparator and 15 as a buffer). The p-bits are interconnected with the RC array as shown in (a).

A. Full-adder learning with emulated ideal MTJ

Figure 2(a) shows the normalized, time-averaged p-bit response of every p-bit using the ideal SMTJ implementation when the input voltage V_{in} is swept. These SMTJs are emulated in hardware with two resistances that are randomly selected by a multiplexer (MUX) to obtain nearly ideal p-bit response characteristics (see Sec. V B for more details). Due to variations in the circuit, every curve is slightly shifted from the ideal 50/50 point at $V_{\text{in}} = 1.95$ V. Even though we are using the MUX model here, it has been shown by Borders *et al.* [2] that near ideal p-bit responses can be obtained with real SMTJs. In previous hardware p-circuit implementations, lateral shifts of the p-bit response had to be eliminated by adjusting synaptic biases to calibrate the experiment [2,43]. By contrast in this demonstration, since the biases are learned during operation, no calibration phase is necessary. This is a significant advantage since learning can account for transistor and SMTJ variations between p-bits. After obtaining the response of all p-bits, the learning experiment is performed (see Sec. V E for more detail about the experimental procedure).

The goal of the learning process is that the p-bits fluctuate according to a set data distribution. Since at every point in time the p-bits can just be in one bipolar state, to monitor the training progress, the distribution of the p-bits $P_{\text{exp}}(t)$ is collected as a histogram of the p-bit output states $([m_1, m_2, m_3, m_4, m_5] + 1)/2 = [A, B, C_{\text{in}}, S, C_{\text{out}}]$ over a fixed time window of 60 s, normalized to 1, and compared to the ideal distribution of a full adder given by the eight lines of the truth table (see Table I). The experimental distribution at $t = 0$, $P_{\text{exp}}(t = 0)$ is shown in Fig. 2(b). At the start of learning the weights and biases are small and the distribution is close to a uniform random distribution. However, due to slight mismatches in the p-bit response of every individual p-bit [Fig. 2(a)] some peaks are more prominent than others. The distribution at the end of learning $P_{\text{exp}}(t = 3000\text{ s})$ is shown in Fig. 2(c), where the highest peaks correspond to the correct distribution for the FA, demonstrating the circuit's ability to learn the given data distribution. To compare two probability distributions quantitatively the Kullback-Leibler divergence (KL divergence) defined by $\text{KL}[P_{\text{ideal}}||P_{\text{exp}}(t)] = \sum_{\mathbf{m}} P_{\text{ideal}}(\mathbf{m}) \log[P_{\text{ideal}}(\mathbf{m})/P_{\text{exp}}(\mathbf{m}, t)]$ is commonly used [44]. Figure 2(d) shows the learning performance measured by the KL divergence versus time t. The difference between the ideal data distribution and the experimental distribution decreases significantly in the first 500 s of learning. At the end of learning, the KL divergence reaches a value of around 0.2. We note that as long as the learned peaks are about equal, the KL divergence can be reduced further by increasing all weight values equally, i.e., decreasing the temperature of the Boltzmann machine. In Fig. 3, the ten weight voltages across the capacitors $V_{i,j} = V_{v;i,j} - V_{C;i,j}$ extracted from the circuit are shown. The weights are measured throughout the whole learning process. The blue lines show the weight voltages for the ideal MTJ. After around 500 s the weights saturate and do not change anymore. In the Supplemental Material [40], the weights values are compared to the weight matrix commonly used for the FA in logic applications [45,46].

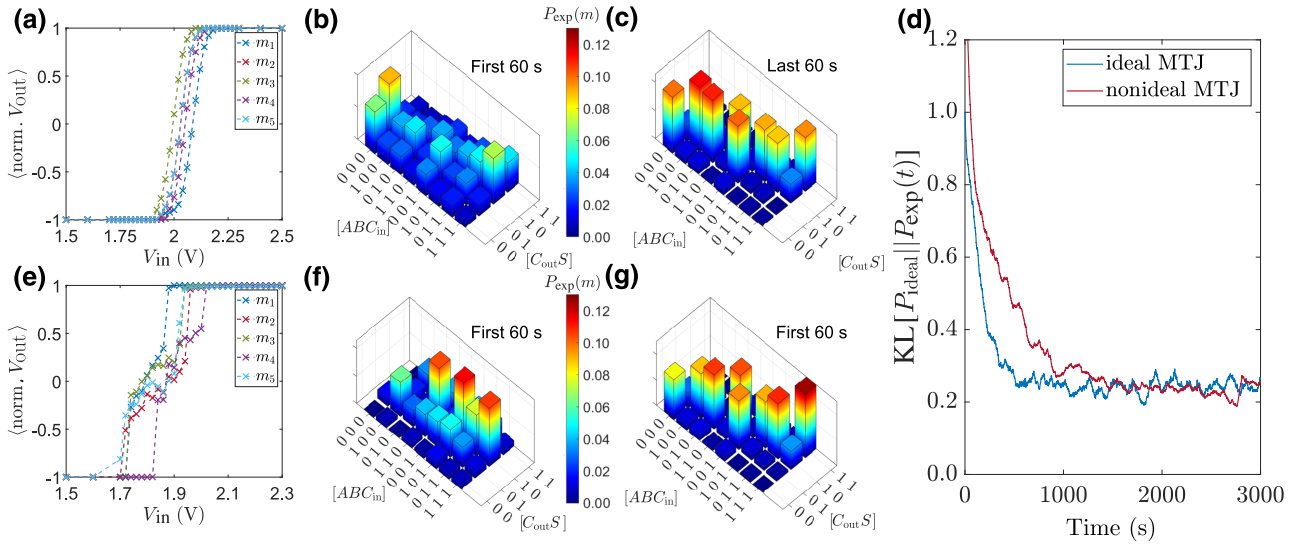

FIG. 2. Full-adder learning. (a) Average response of emulated ideal MTJ p-bits for the five p-bits used in the FA with average normalized output voltage $\langle \text{norm. } V_{\text{out},i}\rangle = 2 \times \langle V_{\text{out},i}\rangle / V_{\text{DD}} - 1 = \langle m_i \rangle$. Every point is averaged over 15 s. (b) Experimental distribution of emulated ideal MTJ circuit $P_{\text{exp}}(m)$ with p-bit output states $([m_1, m_2, m_3, m_4, m_5] + 1)/2 = [A, B, C_{\text{in}}, S, C_{\text{out}}]$ where $m_i = 2 \times V_{\text{out},i}/V_{\text{DD}} - 1$ collected as a histogram over for the first 60 s of learning. (c) Experimental distribution of emulated ideal MTJ circuit collected as a histogram over the last 60 s of learning. (d) KL divergence between ideal and experimental distribution $\text{KL}[P_{\text{ideal}}||P_{\text{exp}}(t)]$ versus time of ideal and nonideal MTJ system. The experimental distribution is obtained over 60 s of learning. (e) Average response of nonideal MTJ p-bits for the 5 p-bits used in the FA with average normalized output voltage $\langle \text{norm. } V_{\text{out},i}\rangle = 2 \times \langle V_{\text{out},i}\rangle / V_{\text{DD}} - 1 = \langle m_i \rangle$. Every point is averaged over 15 s. (f) Experimental distribution of nonideal MTJ circuit $P_{\text{exp}}(m)$ collected as a histogram over the first 60 s of learning. (g) Experimental distribution of nonideal MTJ circuit $P_{\text{exp}}(m)$ collected as a histogram over the last 60 s of learning.

B. Full-adder learning with nonideal MTJ

To examine the effects of variability, we investigate the learning experiment implemented with fabricated SMTJs (see Sec. V A for more details regarding the fabrication). Figure 2(e) shows the V_{out} versus V_{in} characteristics for the five MTJ-based p-bits averaged over 15 s. At the transition point between the stochastic and the deterministic region of the response curve, the slope of the response is sharper compared to the center of the curve, which shows a gradual increase. The combination of these two characteristics leads to a nonideal p-bit response that deviates from the ideal response described by Eq. (1). The reason for the distorted shape of the p-bit response is due to the fact that the MTJs show stochastic behavior for a large window of current flow in the order of > 10 μA. The change of the current flow in the MTJ-transistor branch due to change voltage at the gate of the transistor is not large enough to pin the MTJ to R_P or R_{AP} state. This leads to the distorted shape of the p-bit response in Fig. 2(e). For the best MTJ characteristics, the stochastic range for current flow should be in the order of around 5 μA in the design used here.

Figures 2(f) and 2(g) show the histogram of P_{exp} during the first and last 60 s of learning. At the end of learning the eight desired peaks are the largest, showing that even though the learning algorithm is based on an ideal p-bit response derived from the Boltzmann distribution, the circuit can still learn the desired functionality. Despite the noted nonidealities, the KL divergence saturates to a level comparable between ideal and nonideal MTJ, as shown in Fig. 2(d). This can be explained by the fact that *in situ* learning has the capabilities to counter device-to-device variations by adjusting weights and biases to fit the system (see Supplemental Material [40] for more details on the learned bias voltages).

In Fig. 3, the red lines show the weight voltages of the nonideal MTJ over the duration of the learning process. It can be clearly seen that the weights differ significantly between the ideal and nonideal p-bit implementation while achieving similar performance in the KL divergence, leading to the conclusion that feedback in the system between data and p-bit outputs is able to learn around variations, a crucial ingredient to achieve a high level of performance under device variability. In the Supplemental Material [40] a system simulation on the MNIST dataset [47] is presented to show that the variation tolerance exists when the proposed circuit is scaled up.

The fact that the circuit can learn around variations can be useful not just for classical machine learning tasks like classification or unsupervised learning but also for tasks that have been demonstrated on probabilistic computers like optimization [2,48], inference [49,50], or invertible logic [25,31]. Instead of externally setting the coupling

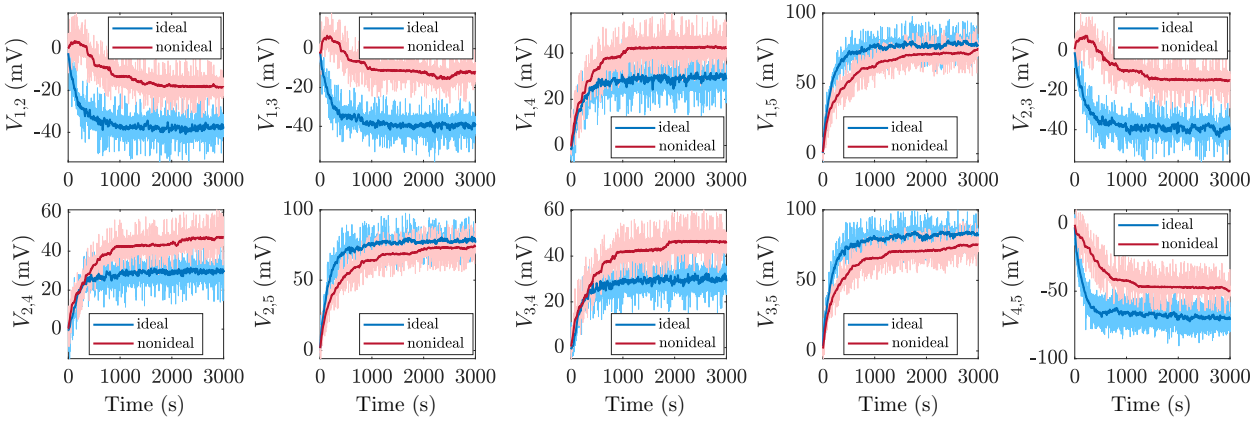

FIG. 3. Weight voltages during FA learning. The ten weight voltages are shown during the 3000 s of learning. Blue lines are the weights learned with the ideal MTJ circuit; red lines show the weights for the nonideal MTJ circuit. The solid lines in the middle are the moving average of the actual weights taken over a window of 10 s.

between p-bits, an additional learning task could improve the performance of the p-circuit by assuring that the coupling between the p-bits is adjusted to the exact hardware p-bit response. In addition, the proposed hardware can be used to represent many different distinct probability distributions by adjusting the coupling between p-bits accordingly. For the particular combination of MTJ and transistor, voltage change at the input can change the output of the p-bit on a transistor response time scale. Because the transistor response can be faster than the implemented synapse, for this particular experiment each p-bit is updated sequentially through the microcontroller instead of autonomously to preserve functionality (see Ref. [51] for more details).

C. Weight extraction

In the previous sections, we compare the distribution of the output configurations of the hardware p-bits averaged over 60 s with the ideal distribution by taking the Kullback-Leibler divergence. In this section we compare how the weights extracted as voltages across the capacitors in the circuit would perform on an ideal platform, i.e., to the Boltzmann distribution where $P(m) \propto \exp\left[-\beta E(m)\right]$ and β is the inverse temperature of the system. The temperature in a Boltzmann machine is a constant factor that all weights and biases are multiplied with and represents how strongly coupled the p-bits are with each other. The comparison has particular relevance since the nonideal effects during learning should have an effect on the weights compared to the weights that would be learned on an ideal machine. Figure 4 shows the Boltzmann distribution with the weights of Fig. 3. The conversion factor between the voltages V across the capacitors and dimensionless weights W of the Boltzmann distribution represented by the temperature factor β is chosen in a way that the relative difference between the peaks of the distribution can be seen clearly. To reduce the effect of noise, the weight values are averaged over the last 10 s of learning. For the example of the FA, it is known from the truth table that an ideal system has no bias. Hence, we do not use the extracted bias but set it to 0 for the Boltzmann distribution. In Fig. 4(a) it can be clearly seen that compared to Fig. 2(c) the learned distribution differs more from the ideal distribution since the peaks are not as uniform. The peaks for configuration $[ABC_{\text{in}}] = 000$, $[C_{\text{out}}S] = 00$ and $[ABC_{\text{in}}] = 111$, $[C_{\text{out}}S] = 11$ are not as prominent as the other six peaks that have been learned. This discrepancy becomes even more visible in Fig. 4(b) compared to Fig. 2(g) where the weights used in the Boltzmann distribution are learned using a less ideal response of the p-bits. Here, only peaks $[ABC_{\text{in}}] = 000$, $[C_{\text{out}}S] = 00$ and $[ABC_{\text{in}}] = 111$, $[C_{\text{out}}S] = 11$ are prominent. This shows that the learned weights fit to the activation of the hardware p-bits but not for the ideal Boltzmann distribution. Hence, we can conclude that the probabilistic computer adapted to the nonideal p-bit response during the *in situ* learning process.

The results presented in this section suggest that learning and inference must be performed on the same hardware to operate reliably. In contrast, initially training on this nonideal machine, then transferring the weight values to an ideal system to complete convergence and perform the programmed task could allow for a hardware-based speedup of the typically time-consuming weight training step. This is similar in spirit to using pretrained weights in a neural network [52,53]. While this can be a disadvantage, the advantages of using the efficient and compact learning circuit that can be used for training and inference should outweigh the problems of transferability between platforms.

In this section, we show that device-to-device variations can be countered by performing hardware-aware *in situ* learning by comparing the learning performance of two systems, one system with ideal p-bit responses and

the other with nonideal p-bit responses that differ significantly compared to Eq. (1). We show that the overall performance is the same for both systems after the training is finished while the learned weights (Fig. 3) are different. However, we also show that if the weights are extracted from the learning circuit and used to calculate the Boltzmann distribution, the obtained distribution differs substantially from the desired data distribution [Fig. 4(b)]. These observations show clearly that the circuit can learn around device-to-device variations.

IV. DISCUSSION

In this paper, we present a proof-of-concept demonstration of an autonomously operating fully connected Boltzmann machine using MTJ-based p-bits. Furthermore, we show how device-to-device variations can be countered by performing hardware-aware *in situ* learning. In the following paragraphs, we compare the presented probabilistic computer with other platforms like conventional CMOS architectures.

On the device level, the closest digital CMOS alternative to the MTJ-based p-bit is a linear feedback shift register (LFSR), without considering the analog tunability of the p-bit. A detailed comparison between p-bit versus LFSR has been performed by Borders *et al.* [2]. The compact MTJ-based p-bit uses around $10\times$ less energy per random bit and has about $300\times$ less area than a 32-bit LFSR. Besides these advantages, a standard LFSR is not tunable like the hardware p-bit and relies on pseudo randomness. The p-bit based on an SMTJ relies on thermal noise and is, hence, a true random number generator. This can be significant for applications for which the quality of the randomness is relevant.

On the system level, the p-bits in combination with the synapse [Eqs. (1) and (2)] are utilized to collect samples of the distribution given by the current weights to update the weights according to the correct gradient. Collecting statistics by sampling drives the learning process since every sample is directly utilized to update the weight voltages [Eq. (3)]. Thus, the numbers of samples per unit time are significant for the speed of the learning process. The MTJ fluctuation time of the p-bit τ_N is a significant time scale for the generation of samples since it describes how fast Eq. (1) can be computed in hardware. The learning time constant τ_L has to be larger than the MTJ fluctuation time τ_N to collect enough statistics to ensure convergence of the learning process. To ensure that every p-bit input is correctly calculated based on the state of the other p-bits, it is necessary that the synapse time τ_S is smaller than τ_N. In this experiment, since the synapse time defined by the microcontroller is in the order of 100 μs to 1 ms, τ_N is in the order of 10–100 ms, which results in slow training in the order of 10^3 s. However, it has to be noted that the time scales of the circuit can be reduced significantly in an integrated version of the proposed circuit where the synapse based on crossbar architectures can operate with GHz speeds with execution times down to 10 ps [51,54,55] and the fluctuation time of SMTJs can be in the order of 100 ps [56–58]. This would allow a substantial decrease of τ_L and an increase of the learning speed by up to 9 orders of magnitude. Regarding energy consumption of the synapse block, the efficient p-bit building block presented here can be combined with any synapse option that provides the most power efficiency. For full inference operation, the RC array used here to represent weights as voltages requires a constant memory refresh similar to mainstream dynamic random-access memory (DRAM). To save energy during the learning process, the presented p-bit building block could be combined with nonvolatile synapse implementations like memristive crossbar arrays [13,14,55]. The learned weights could also be extracted from the RC array and stored in a nonvolatile memory array after the learning process.

The overall power consumption can be estimated using numbers from the literature. The MTJ-based p-bit consumes about 20 μW [57]. In a memristive crossbar, each memristor consumes about 1 μW and operational amplifiers around 3 μW [51,55,59]. The XNOR operation consumes 10 μW. For the overall circuit with five p-bits, 15 XNOR gates and memristors, and five operational amplifiers would take approximately 300 μW. This is the projected power consumption of a fully connected Boltzmann-machine hardware shown in this work. For specified applications where less weight connections between neurons are needed (for example, restricted Boltzmann machines in digital computers), the number of components can be reduced, which results in improved power consumption. In this regard, the estimated power consumption in our work can also be significantly reduced by employing a higher-level approach.

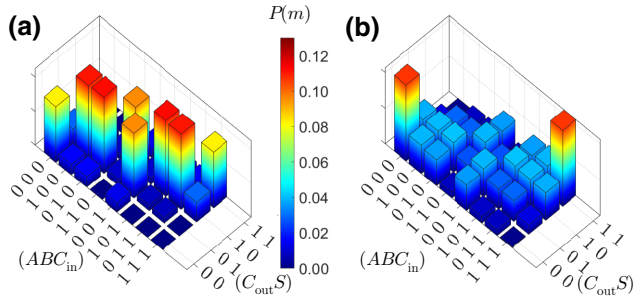

FIG. 4. Boltzmann distribution obtained from learned weights. (a) Boltzmann distribution $P(m) = 1/Z \exp(-\beta E)$ with energy $E = -\sum W_{i,j} m_i m_j$ computed by using the learned weights $W_{i,j}$ of the FA with the emulated ideal SMTJ p-bit circuit where the bipolar p-bit states $([m_1, m_2, m_3, m_4, m_5] + 1)/2 = [A, B, C_{\text{in}}, S, C_{\text{out}}]$. (b) Boltzmann distribution $P(m)$ computed by using the learned weights of the FA with the nonideal SMTJ p-bit circuit. Biases are set to 0.

Another significant advantage of the probabilistic circuit is that due to the compactness and area savings of the *p*-bit, when scaling up, many more *p*-bits can be put on a chip compared to CMOS alternatives like LFSRs. In addition, the *p*-bit hardware implementation does not rely on any clocking in order to function and is hence autonomously operating. This has the advantage that many autonomously operating *p*-bits can function in parallel leading to an overall acceleration of the operation. In this context, it has to be noted that the information of the current state of a *p*-bit has to be propagated to all other *p*-bits that are connected to it on a time scale τ_S that is much shorter than the neuron time τ_N for the probabilistic circuit to function properly. When the *p*-bit fluctuation time varies between different *p*-bit it has to be assured that the fastest *p*-bit with fluctuation time $\tau_{N,f}$ fluctuates slower than τ_S. Depending on the sparsity of the weight matrix and the ratio of τ_S to τ_N, the number of parallel operating *p*-bits has to be adjusted to ensure fidelity of the operation [51]. In a recent paper by Sutton *et al.* [51] a FPGA design was implemented that emulates a probabilistic circuit where the MTJ-based *p*-bit is envisioned as a drop-in replacement. In this complete system-level hardware realization of a *p*-computer that can perform only inference not learning, a drastic reduction in area footprint of the compact *p*-bit design compared to digital implementations is confirmed. This shows that an integrated version of the proposed learning circuit based on the *p*-computer architecture could be very beneficial.

While we address that device-to-device variations of the shape and shift of the *p*-bit response can be accounted for by hardware-aware learning, it is worthwhile to note that rate variation of the stochastic MTJ between *p*-bits cannot be reduced by this approach. The system will in the worst case learn as fast as the fluctuation rate of the slowest *p*-bit $\tau_{N,s}$, which can slow down the overall operation. However, in the case of *p*-bits with stochastic MTJs where the thermal barrier of the magnet in the free layer is in the order of k_BT, the fluctuation rate does not go exponentially with the size of the magnet making the system less susceptible to rate variations [56,57,60,61]. It has to be noted that a way to reduce rate variation in probabilistic circuits based on stable MTJs that are biased using voltages and magnetic fields was presented by Lv *et al.* [25].

We note that the fluctuation rate will also be affected by the temperature of the probabilistic circuit. When increasing the temperature, the fluctuation rate of the *p*-bits will increase exponentially. However, the temperature variation will not affect the average *p*-bit response of the MTJ. For proper operation it has to be assured that the synapse time τ_S is shorter than the fluctuation time $\tau_{N,f}$ of the fastest fluctuating *p*-bit. As overall design criteria for the autonomous circuit the following conditions have to be met: $\tau_S \ll \tau_{N,f}$ and $\tau_{N,s} \ll \tau_L$.

In conclusion, we show a proof-of-concept demonstration of a fully connected probabilistic computer built with MTJ-based *p*-bits that can perform learning. We present multiple learning examples for up to five *p*-bits and 15 learning parameters. The learning is robust and can operate even with strong device-to-device variations due to hardware-aware learning. This shows that when scaled up and with faster fluctuating building blocks, probabilistic computers could accelerate computation while reducing energy cost for a wide variety of tasks in the machine-learning field such as generative learning or sampling, as well as for tasks that could benefit from variation tolerance like optimization or invertible logic.

V. MATERIALS AND METHODS

A. MTJ fabrication and characterization

The MTJs used in this work are fabricated with a stack structure as follows, from the substrate side: Ta(5)/ Pt(5)/ $[Co(0.4)/Pt(0.4)]_6$/ Co(0.4)/ Ru(0.4)/ $[Co(0.4)/Pt(0.4)]_2$/ Co(0.4)/ Ta(0.2)/ $(Co_{0.25}Fe_{0.75})_{75}B_{25}(1)$/ MgO/ $(Co_{0.25}Fe_{0.75})_{75}B_{25}(1.7)$/ Ta(5)/ Ru(5)/ Ta(50). The numbers in parentheses are the nominal thicknesses in nanometers. All films are deposited on a thermally oxidized silicon substrate by dc and rf magnetron sputtering at room temperature. The stacks are then processed into circular MTJs with nominal junction size of 20–25 nm in diameter by electron-beam lithography and argon ion milling. The samples are annealed at 300 °C in vacuum for an hour. MTJs are then cut out from wafers and bonded with wires to IC sockets to be placed in the *p*-bit circuit board. To determine nonideal MTJs with suitable characteristics, the MTJ resistance is measured by sweeping the current from negative to positive values, and the time-averaged and high-frequency signals are read across a voltmeter and oscilloscope, respectively. We measure an approximate tunnel magnetoresistance ratio of 65% fluctuating between an average $R_P = 18$ kΩ and $R_{AP} = 30$ kΩ. The current at which the resistance switches by half is determined to be $I_{50/50}$, which is the bias current at which the MTJs will spend equal time in the P and AP states. The $I_{50/50}$ used in this work ranges from 3 to 5 μA. We measure the average fluctuation time τ_N by performing retention time measurements when the MTJ is in either the high (AP) or the low (P) state using voltage readings from the oscilloscope. To ensure reliable collection of data, the oscilloscope sampling rate is set 10 times faster than the fastest recorded fluctuation time of the MTJ. The retention times used in this work range from 1 to 100 ms.

B. Hardware implementation of the *p*-bit

Equation (1) is implemented with the SMTJ-based *p*-bit proposed by Camsari *et al.* [38] and experimentally demonstrated by Borders *et al.* [2]. The *p*-bit implementation in this paper follows Ref. [2] and is built with a SMTJ in series to a transistor (2N7000,T0-92-3 package) and a

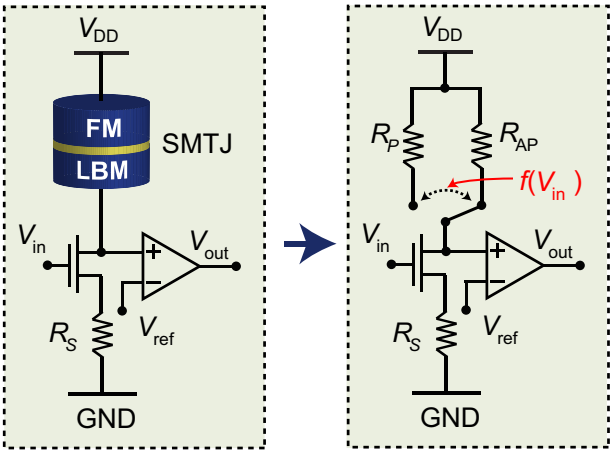

FIG. 5. Multiplexer emulation. The SMTJ-based p-bit on the left is modeled by a multiplexer that switches randomly between R_P and R_{AP} but as a function of V_{in} so that the right statistics are preserved [43].

source resistor R_S. The supply voltage of the MTJ transistor branch is set to $V_{DD} = 200$ mV whereas the remaining circuit operates at $V_{DD} = 5$ V. The source resistance R_S is chosen so that $I_{50/50}$ is flowing through the circuit when $V_{in} = 1.95$ V. The transistor is biased in the subthreshold region. The voltage at the drain of the transistor is then thresholded using a comparator (AD8694, 16-SOIC package) with a bandwidth of 10 MHz. The reference voltage is chosen to be $V_{ref} = V_{DD} - I_{50/50}(R_P + R_{AP}/2)$. We use a comparator to add another node where we can fine tune V_{ref}. However, in an integrated circuit the transistor should be chosen so that $V_{ref} = V_{DD}/2$ so that the comparator can be replaced by a simple inverter as simulated in Refs. [36,38,57]. The overall p-bit is then just built with one MTJ and three transistors. For the experiment with ideal MTJs, the SMTJ is emulated by a MUX model that includes all major characteristics of a real SMTJ and has been developed by Pervaiz *et al.* [43] as illustrated in Fig. 5. The SMTJ is emulated by providing a noise signal to the MUX where the statistics of the noise depend on V_{in} and are generated using a microcontroller that switches between a resistor R_P and R_{AP} representing the two resistive states of the SMTJ. Here, the resistors values are chosen to be $R_P = 11$ kΩ and $R_{AP} = 22$ kΩ. The advantage of this approach is that the MTJ parameters like stochastic range and resistance can be easily manipulated in this model. For the MUX, a MAX 394 quad analog multiplexer is used.

C. Implementation of the synapse

The synapse is implemented with an Arduino MEGA microcontroller and an eight-channel PMOD DA4 Digital-Analog-Converter. The digital output voltages of the p-bits $\{V_{out}\}$ are fed into the microcontroller together with the analog weight voltages $\{V_C\}$ of the learning circuit. The internal analog-digital converter (ADC) of the microcontroller is used for sensing the weight voltages. Equation (2) is then computed and the analog input voltages $\{V_{in}\}$ are wired back to the neurons by utilizing the DAC. To reduce the synapse time in every iteration of the synapse operation, only one of the 15 analog voltages are read out and updated. This does not affect the circuit performance since the capacitor voltages V_C are changing slowly. The synapse operation time τ_S is < 1 ms which is shorter than the MTJ fluctuation time. The condition $\tau_S \ll \tau_N$ has to be satisfied to ensure fidelity of the autonomous operation of the p-circuit.

D. Implementation of weight updating

For proper operation it is important that the learning time constant τ_L is much larger than the neuron time τ_N. To achieve this, a high RC constant is chosen with a 1 MΩ resistor and a 10 μF capacitor. Since this circuit has a high resistance in series to the capacitor, to ensure that the reading of the weight voltage does not discharge the capacitor, a buffer stage is used between the capacitor and the synapse. The buffer is implemented with an operational amplifier (AD8694, 16-SOIC package).

For the FA experiment, the voltage gain factor A_v of Eq. (4) is chosen to be 3 which turned out to be a reasonable value for achieving a good degree of regularization while achieving high peaks in the learned distribution. The voltage gain operation is performed with the microcontroller. Additional details regarding Eq. (4) can be found in Ref. [36].

For learning the correlations $m_i m_j$, represented by voltage $V_{m;i,j}$, are crucial. To obtain the current correlations between neuron m_i and m_j their product has to be computed. This is done here by using another microcontroller. Since the output m is bipolar ($m \in \{-1, 1\}$) only negative or positive correlation is possible. Voltage $V_{m;i,j}$ is limited by the output voltages of the DAC, which has a range from 0 to 2.5 V. $V_{m;i,j}$ can hence be calculated by solving $V_{m;i,j} = (m_i m_j + 1)/2 \times 2.5$ V. Voltage $V_{m;i,j}$ is fed back to the corresponding RC element by utilizing another DAC. The described operation is the same as computing the XNOR operation between two binary variables. Hence, the operation is straight forward and the programmability of the microcontroller not essential for operation of the circuit.

E. Experimental procedure

Before the start of training the capacitor is fully discharged so that $V_{i,j}(t = 0) = 0$ V corresponding to $V_{C;i,j}(t = 0) = V_{v;i,j}$. At $t = 0$ the training starts and voltages $\{V_C\}$ and the p-bit output voltages $\{V_{out}\}$ are measured at sampling frequency f_S. The training is run for $T = 3000$ s.

The data is collected with an *NI USB-6351 X SERIES DAQ* that has analog inputs for the 15 weights and biases and digital inputs for the five p-bit outputs. The software *Labview* is utilized to record data with a sampling frequency of $f_S = 1$ kHz.

In this paper we train the bias due to mismatch of p-bit responses together with the bias needed to learn the data distribution. In principle, these can be separated to obtain a better bias value that can be used on other platforms. However, this separation of calibration and learning is only possible for the bias of every p-bit and not for the weights connecting them since the calibration cannot be performed with ideal p-bit responses with the hardware system.

F. Mapping of the truth table to node voltages for learning

For a fully visible Boltzmann machine with N neurons, $(N+1)N/2$ weights and biases have to be learned. The goal for learning is that the fully trained network has the same distribution as the data distribution. For a FA, the data distribution is given by the truth table shown in Table I. The data distribution can be described by a matrix in which the number of columns is equal to the number of neurons N and the number of rows is equal to the number of training examples d. For the biases, another neuron unit with value 1 is added so that there are $(N+1)$ columns. For the example of a FA, $N = 5$ and $d = 8$ for eight lines in the truth table. The matrix V_{FA} is then a 6×8 matrix where all 0s of the truth table are converted to -1s since we are using the bipolar representation:

$$V_{\mathrm{FA}} = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{5}$$

The density matrix is then calculated by computing $D = V^T V/d$, which is a 6×6 matrix for the FA:

$$D_{\mathrm{FA}} = \frac{V_{\mathrm{FA}}^T V_{\mathrm{FA}}}{d} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 & -0.5 & 0 \\ 0.5 & 0.5 & 0.5 & -0.5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6}$$

with $d = 8$. The values in the last column of the density matrix correspond to the average value of every neuron in the data distribution and are used to learn the biases. Only the terms above the diagonal of D are needed and converted to voltages $V_{v;i,j}$ in the circuit. Since the DAC operates with positive voltages in the range of 0 to 2.5 V, $V_{v;i,j} = (D_{i,j} + 1)/2 \times 2.5$ V.

ACKNOWLEDGMENTS

J.K. thanks A.Z. Pervaiz for helpful discussions. This work is supported in part by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA and in part by JST-CREST JPMJCR19K3, JSPS Kakenhi 19J12206, and Cooperative Research Projects of RIEC. K.Y.C gratefully acknowledges support from Center for Science of Information (CSoI), an NSF Science and Technology Center, under Grant No. CCF-0939370.

[1] Richard P. Feynman, Simulating physics with computers, Int. J. Theor. Phys **21**, 467 (1982).
[2] William A. Borders, Ahmed Z. Pervaiz, Shunsuke Fukami, Kerem Y. Camsari, Hideo Ohno, and Supriyo Datta, Integer factorization using stochastic magnetic tunnel junctions, Nature **573**, 390 (2019).
[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, Deep learning, Nature **521**, 436 (2015).
[4] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean, A guide to deep learning in healthcare, Nat. Med. **25**, 24 (2019).
[5] Juergen Schmidhuber, Deep learning in neural networks: An overview, Neural. Netw. **61**, 85 (2015).
[6] Big data needs a hardware revolution, Nature **554**, 145–146 (2018).
[7] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang, in *2017 IEEE Custom Integrated Circuits Conference (CICC)* (2017), p. 1.
[8] J. Grollier, D. Querlioz, K. Y. Camsari, K. Everschor-Sitte, S. Fukami, and M. D. Stiles, Neuromorphic spintronics, Nat. Electron. **3**, 360 (2020).
[9] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha, A million spiking-neuron integrated circuit with a scalable communication network and interface, Science **345**, 668 (2014).
[10] Mike Davies *et al.*, Loihi: A neuromorphic manycore processor with on-chip learning, IEEE Micro **38**, 82 (2018).
[11] Geoffrey E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Comput. **14**, 1771 (2002).
[12] Miguel A. Carreira-Perpinan and Geoffrey E. Hinton, in *Aistats*, (Citeseer, 2005),Vol. 10, p. 33.

[13] Maxence Ernoult, Julie Grollier, and Damien Querlioz, Using memristors for robust local learning of hardware restricted Boltzmann machines, Sci. Rep. **9**, 1851 (2019).

[14] Mahdi Nazm Bojnordi and Engin Ipek, in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2016), p. 1.

[15] Shamma Nasrin, Justine L. Drobitch, Supriyo Bandyopadhyay, and Amit Ranjan Trivedi, Low power restricted Boltzmann machine using mixed-mode magneto-tunneling junctions, IEEE Electron Device Lett. **40**, 345 (2019).

[16] Nicolas Le Roux and Yoshua Bengio, Representational power of restricted Boltzmann machines and deep belief networks, Neural Comput. **20**, 1631 (2008).

[17] Emile H. L. Aarts and Jan H. M. Korst, in *PARLE Parallel Architectures and Languages Europe*, edited by G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, J. W. Bakker, A. J. Nijman, and P. C. Treleaven (Springer Berlin Heidelberg, Berlin, Heidelberg, 1987), Vol. 258, p. 34.

[18] Thomas R. Osborn, in *International Neural Network Conference: July 9–13, 1990 Palais Des Congres – Paris – France* (Springer Netherlands, Dordrecht, 1990), p. 785.

[19] Ruslan Salakhutdinov and Geoffrey Hinton, in *Artificial Intelligence and Statistics* (2009), p. 448.

[20] Nitish Srivastava and Russ R Salakhutdinov, in *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., 2012), p. 2222.

[21] Sukru Burc Eryilmaz, Emre Neftci, Siddharth Joshi, SangBum Kim, Matthew BrightSky, Hsiang-Lan Lung, Chung Lam, Gert Cauwenberghs, and Hon-Sum Philip Wong, Training a probabilistic graphical model with resistive switching electronic synapses, IEEE Trans. Electron Devices **63**, 5004 (2016).

[22] Chang-Hung Tsai, Wan-Ju Yu, Wing Hung Wong, and Chen-Yi Lee, A 41.3/26.7 pJ per neuron weight RBM processor supporting on-chip Learning/Inference for IoT applications, IEEE. J. Solid-State. Circuits **52**, 2601 (2017).

[23] Ruslan Salakhutdinov, Learning and evaluating Boltzmann machines, Utml Tr **2**, 21 (2008).

[24] Raffaele De Rose, Marco Lanuzza, Felice Crupi, Giulio Siracusano, Riccardo Tomasello, Giovanni Finocchio, and Mario Carpentieri, Variability-aware analysis of hybrid MTJ/CMOS circuits by a micromagnetic-based simulation framework, IEEE Trans. Nanotechnol. **16**, 160 (2017).

[25] Yang Lv, Robert P. Bloom, and Jian-Ping Wang, Experimental demonstration of probabilistic spin logic by magnetic tunnel junctions, IEEE Magn. Lett. **10**, 1 (2019).

[26] Yuanzhuo Qu, Bruce F. Cockburn, Zhe Huang, Hao Cai, Yue Zhang, Weisheng Zhao, and Jie Han, Variation-resilient true random number generators based on multiple STT-MTJs, IEEE Trans. Nanotechnol. **17**, 1270 (2018).

[27] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, Wenhao Song, John Paul Strachan, Mark Barnell, Qing Wu, R. Stanley Williams, J. Joshua Yang, and Qiangfei Xia, Efficient and self-adaptive in-situ learning in multilayer memristor neural networks, Nat. Commun. **9**, 2385 (2018).

[28] Thomas Dalgaty, Niccolo Castellani, Clément Turck, Kamel-Eddine Harabi, Damien Querlioz, and Elisa Vianello, In situ learning using intrinsic memristor variability via markov chain monte carlo sampling, Nat. Electron. **4**, 151 (2021).

[29] Brian Kiraly, Elze J. Knol, Werner M. J. van Weerdenburg, Hilbert J. Kappen, and Alexander A. Khajetoorians, An atomic Boltzmann machine capable of self-adaption, Nat. Nanotechnol. **16**, 414 (2021).

[30] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski, A learning algorithm for Boltzmann machines, Cogn. Sci. **9**, 147 (1985).

[31] Kerem Yunus Camsari, Rafatul Faria, Brian M. Sutton, and Supriyo Datta, Stochastic p -Bits for Invertible Logic, Phys. Rev. X **7**, 031014 (2017).

[32] Daphne Koller and Nir Friedman, *Probabilistic Graphical Models: Principles and Techniques* (MIT Press, Cambridge, Massachusetts London, 2009).

[33] Vinod Nair and Geoffrey E Hinton, Implicit mixtures of restricted Boltzmann machines, Adv. Neural Inf. Proc. Syst., 1145 (2009).

[34] Kathleen E. Hamilton, Catherine D. Schuman, Steven R. Young, Ryan S. Bennink, Neena Imam, and Travis S. Humble, Accelerating scientific computing in the post-moore's Era, ACM Trans. Parallel Comput. **7**, 6:1 (2020).

[35] Tijmen Tieleman, in *Proceedings of the 25th International Conference on Machine Learning - ICML '08* (ACM Press, Helsinki, Finland, 2008), p. 1064.

[36] Jan Kaiser, Rafatul Faria, Kerem Y. Camsari, and Supriyo Datta, Probabilistic circuits for autonomous learning: A simulation study, Front. Comput. Neurosci. **14**, 14 (2020).

[37] Andrew Y. Ng, in *Twenty-First International Conference on Machine Learning - ICML '04* (ACM Press, Banff, Alberta, Canada, 2004), p. 78.

[38] K. Y. Camsari, S. Salahuddin, and S. Datta, Implementing p-bits with embedded MTJ, IEEE Electron Device Lett. **38**, 1767 (2017).

[39] The exact mapping of the correlation voltages $V_{v;i,j}$ and $V_{m;i,j}$ is discussed in the methods section.

[40] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevApplied.17.014016 for more information regarding the learning rule and learning examples for AND, OR, and XOR gates.

[41] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M. Shelby, Irem Boybat, Carmelo di Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan C. P. Farinha, Benjamin Killeen, Christina Cheng, Yassine Jaoudi, and Geoffrey W. Burr, Equivalent-accuracy accelerated neural-network training using analogue memory, Nature **558**, 60 (2018).

[42] M. R. Mahmoodi, M. Prezioso, and D. B. Strukov, Versatile stochastic dot product circuits based on nonvolatile memories for high performance neurocomputing and neurooptimization, Nat. Commun. **10**, 5113 (2019).

[43] Ahmed Zeeshan Pervaiz, Supriyo Datta, and Kerem Y. Camsari, in *2019 IEEE BiCMOS and Compound Semiconductor Integrated Circuits and Technology Symposium (BCICTS)* (2019), p. 1.

[44] S. Kullback and R. A. Leibler, On information and sufficiency, Ann. Math. Stat. **22**, 79 (1951).
[45] O. Hassan, K. Y. Camsari, and S. Datta, Voltage-driven building block for hardware belief networks, IEEE Design Test **36**, 15 (2019).
[46] Ahmed Zeeshan Pervaiz, Brian M. Sutton, Lakshmi Anirudh Ghantasala, and Kerem Y. Camsari, Weighted p-bits for FPGA implementation of probabilistic circuits, IEEE Trans. Neural Netw. Learn. Syst. **30**, 1920 (2019).
[47] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges, The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/ (2010).
[48] Brian Sutton, Kerem Yunus Camsari, Behtash Behin-Aein, and Supriyo Datta, Intrinsic optimization using stochastic nanomagnets, Sci. Rep. **7**, 44370 (2017).
[49] Rafatul Faria, Jan Kaiser, Kerem Y. Camsari, and Supriyo Datta, Hardware design for autonomous Bayesian networks, Front. Comput. Neurosci. **15**, 584797 (2021).
[50] Rafatul Faria, Kerem Y. Camsari, and Supriyo Datta, Implementing Bayesian networks with embedded stochastic MRAM, AIP Adv. **8**, 045101 (2018).
[51] Brian Sutton, Rafatul Faria, Lakshmi A. Ghantasala, Risi Jaiswal, Kerem Y. Camsari, and Supriyo Datta, Autonomous probabilistic coprocessing with petaflips per second, IEEE Access **8**, 157238 (2020).
[52] Geoffrey E. Hinton and Russ R. Salakhutdinov, in *Advances in Neural Information Processing Systems* (2012), p. 2447.
[53] Kaiming He, and Ross Girshick, and Piotr Dollar, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (IEEE, Seoul, Korea (South), 2019), p. 4917.
[54] Peng Gu, Boxun Li, Tianqi Tang, Shimeng Yu, Yu Cao, Yu Wang, and Huazhong Yang, in *The 20th Asia and South Pacific Design Automation Conference* (2015), p. 106.
[55] Fuxi Cai, Suhas Kumar, Thomas Van Vaerenbergh, Xia Sheng, Rui Liu, Can Li, Zhan Liu, Martin Foltin, Shimeng Yu, Qiangfei Xia, J. Joshua Yang, Raymond Beausoleil, Wei D. Lu, and John Paul Strachan, 'Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks, Nat. Electron. **3**, 409 (2020).
[56] J. Kaiser, A. Rustagi, K. Y. Camsari, J. Z. Sun, S. Datta, and P. Upadhyaya, Subnanosecond Fluctuations in Low-Barrier Nanomagnets, Phys. Rev. Appl. **12**, 054056 (2019).
[57] O. Hassan, R. Faria, K. Y. Camsari, J. Z. Sun, and S. Datta, Low-barrier magnet design for efficient hardware binary stochastic neurons, IEEE Magn. Lett. **10**, 1 (2019).
[58] M. Pufall, W. Rippard, Shehzaad Kaka, S. Russek, T. Silva, Jordan Katine, and Matt Carey, Large-angle, gigahertz-rate random telegraph switching induced by spin-momentum transfer, Phys. Rev. B **69**, 214409 (2004).
[59] Boxun Li, Yi Shan, Miao Hu, Yu Wang, Yiran Chen, and Huazhong Yang, in *International Symposium on Low Power Electronics and Design (ISLPED)* (IEEE, 2013), p. 242.
[60] William Fuller Brown, Thermal fluctuations of a single-domain particle, Phys. Rev. **130**, 1677 (1963).
[61] William T. Coffey and Yuri P. Kalmykov, Thermal fluctuations of magnetic nanoparticles: Fifty years after brown, J. Appl. Phys. **112**, 121301 (2012).