Adaptive synaptogenesis implemented on a nanomagnetic platform

Faiyaz Elahi Mullick,[1,*] Supriyo Bandyopadhyay,[2,3,†] Rob Baxter,[4,‡] Tony J. Ragucci,[5] and Avik W. Ghosh[1,6,§]

[1] *Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, Virginia 22094, USA*
[2] *Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, Virginia 23284, USA*
[3] *Department of Physics, Virginia Commonwealth University, Richmond, Virginia 23284, USA*
[4] *Department of Neurosurgery, University of Virginia, Charlottesville, Virginia 22094, USA*
[5] *Teledyne Scientific and Imaging, Thousand Oaks, California 91360, USA*
[6] *Department of Physics, University of Virginia, Charlottesville, Virginia 22094, USA*

(Received 14 April 2025; revised 28 July 2025; accepted 18 November 2025; published 17 December 2025)

This paper is a contribution to the Physical Review Applied collection titled Physics-Inspired Computing.

The human brain functions very differently from artificial neural networks (ANNs) and possesses unique features that are absent in ANNs. An important one among them is "adaptive synaptogenesis," which modifies synaptic weights when needed to avoid *catastrophic forgetting* and to promote lifelong learning. In the model described here, a supervised form of adaptive synaptogenesis uses local error signals to modify synaptic weights and reduce errors. In the brain, supervisory signals may come from a variety of brain regions; in the model, supervisory signals are provided as class labels corresponding to each feature vector. In this work, we discuss various algorithmic aspects of adaptive synaptogenesis tailored to edge computing, demonstrate its function using simulations, and design nanomagnetic hardware accelerators for specific functions such as mean-firing-rate estimation. Our approach attempts to combine two disparate fields—neuroscience and spintronics—on a common hardware platform.

DOI: 10.1103/wdlf-34yy

I. INTRODUCTION

The need for brain-inspired computing far beyond conventional artificial neural networks (ANNs) is a pressing one. Conventional machine learning (ML) based on deep neural nets (DNNs) is ill suited for the rapid growth in edge intelligence owing to the considerable cost of wiring, exorbitant energy demands, and a limit on memory resources. In ANNs and DNNs, new sensory data are learned independent of past history, requiring added hardware and costly synaptic interconnects, with an exploding area footprint and energy budget. In edge environments requiring on-chip computing, such as autonomous robots exploring mines, battlefield vehicles navigating enemy terrain, underwater drones, or Mars rovers, robots must analyze rapidly varying situations with very limited memory resources and energy budgets, without relying on a cloud that is either unavailable or unreliable in the face of security breaches. In three-dimensional (3D) autonomous simultaneous localization and mapping (ASLAM) for, e.g., mobile robotics, the robot actions need to be calculated on the go offline, but this is traditionally done by deleting out-of-date data using a delayed nearest-neighbor data-association strategy [1–3]. Going beyond such a local adaptive mapping and navigation, one step at a time, becomes completely prohibitive over an expanded time horizon.

In a distributed neural net, the challenge with learning sequential data is called the *stability-plasticity dilemma*, the choice between integrating new knowledge versus remembering previously acquired knowledge. The conventional way to deal with this is to reduce overlap among the stored internal representations; e.g., by using sparse or interleaved learning. Connection networks must then absorb new inputs and adjust synaptic weights by *small increments*, thereby preventing sequential learning. Attempts to address this challenge have mostly been made in software; e.g., a Fahlman offset [4] in the derivative of the sigmoid function in back-propagation to avoid entrenchment in its flat parts. Intel's recent artificial intelligence (AI) robot [5] has relied on a learning phase in which prototype data are moved around but not erased in feature space, punishing the wrong category or rewarding

*Contact author: fm4fv@virginia.edu
†Contact author: sbandy@vcu.edu
‡Contact author: rbaxter37@gmail.com
§Contact author: ag7rq@virginia.edu

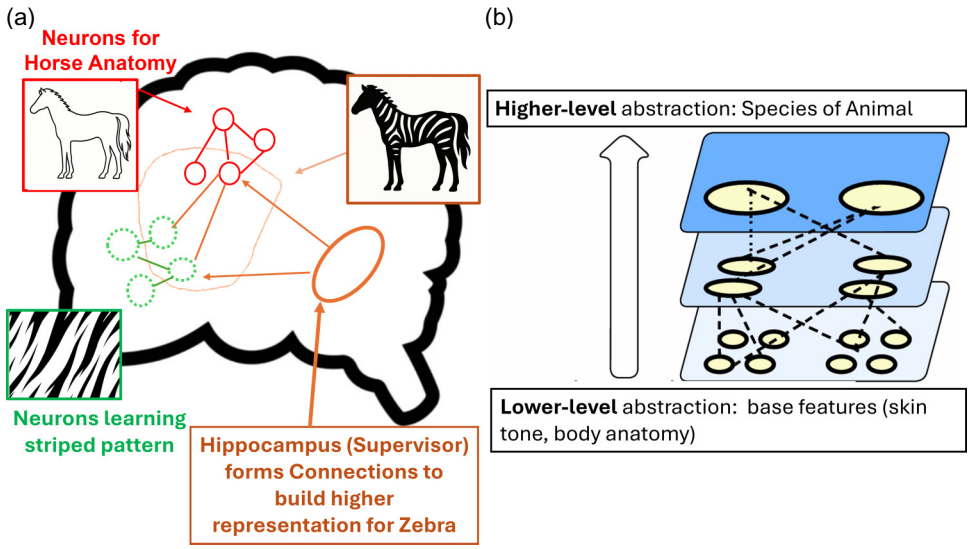

FIG. 1. (a) Lower-level representations of stripes and animal anatomy learnt by groups of neurons that will get wired together to (b) represent a higher-level abstraction such as an animal species of zebra.

the right category, and only allocating new resources if the error persists and an unknown category is thereby identified. Extending learning to a practical hardware environment, with acceptable size, weight, and power (SWaP) is still an unrealized target. In contrast, human cognitive abilities provide a remarkable example of how the brain encodes sequential learning very efficiently with a fixed memory bank, avoiding *catastrophic forgetfulness* (sequential data over-write) with incremental and contextual learning. The evolved trick is an over-riding hippocampus (HC) that acts as a memory support structure for a severely resource-constrained neocortex (NC), with dense local but very sparse, one in a million, nonlocal connectivity that mediates perceptions and decisions. In contrast to brain development, which proceeds from sensory, up through the cortical hierarchy (i.e., bottom-up), later learning proceeds in a top-down manner directed by the hippocampal system, a dense recurrently connected generative network with the ability to process and correlate long-term multidimensional sequences [6,7] to build a sophisticated "world model" for the organism. In addition to the hardware support structure of a dual memory, the brain also uses algorithmic techniques to encode episodic memory by hierarchically implementing depths of representation. Big ideas are built out of association between contexts, event histories, and multisensory microscopic details; for instance, reframing a "zebra" as a "striped horse" (Fig. 1).

The process of having a separate dense hippocampal network directing a sparser neocortical network allows us to avoid catastrophic forgetfulness and compose coherent long-term semantic memory (lifelong learning) from short, fragmented, conditionally independent, episodic events and their intermediate representations.

II. BRAIN-INSPIRED LEARNING MODELS

The goal of this section is to describe a mathematical model, inspired by neocortical (NC) computations, that encodes associative memories as new events are experienced. The model is based on adaptive synaptogenesis, which consists of synaptogenesis (the creation of synaptic connections), synaptic weight modification, and synaptic shedding (the removal of synaptic connections). In addition, during training, supervisory signals indicate the class (category) of each feature vector and are used to determine if the existing memory encodings are making an error. Errors tend to result in the formation of new encodings, whereas the absence of an error tends to strengthen existing encodings.

We will introduce the mathematical algorithms in this section. In Sec. III, we will discuss the implications of these algorithms with a model problem, while in Sec. IV we will explore a potential hardware framework to realize this architecture on-chip using compact voltage-tunable magnetic neurons.

A. Neuron activations

External inputs to the model at each time step consist of an n-dimensional feature vector $X = \{x_1, x_2, x_3, \ldots, x_n\}$ and a K-dimensional supervision (class) vector $S = \{s_1, s_2, \ldots, s_K\}$. The features are analog values bounded by 0 and 1, i.e., $x_i \in [0, 1]$. The features are typically continuous values such as temperature readings from a factory sensor, pixel intensity in a two-dimensional (2D) image, or photodetector values. There are K classes and the class vector is a binary vector indicating the presence or absence of each class.

Coding neurons, indexed by j, have two types of synaptic weights: feature weights w_{ij} and class weights w_{jk}, where i and k index the features and classes, respectively. The coding neurons have two types of activations: feature activation, denoted by y_j^f, and class activation, denoted by y_j^c. The feature activation of neuron j is proportional to the cosine between the input feature vector $\mathbf{X}$ and the feature weight vector $\mathbf{W}_j$,

$$y_j^f = \mathbf{C}_j \frac{\mathbf{X}^T \mathbf{W}_j}{\|\mathbf{X}\| \|\mathbf{W}_j\|}, \tag{1}$$

where $\|\mathbf{X}\|$ denotes the L2 (Euclidean) norm of $\mathbf{X}$. Additionally, each neuron has associated with it a binary connection vector, denoted by $\mathbf{C}_j$, that indicates the features to which the neuron j is connected. The class activation y_j^c is related to the Hamming distance between the input (target) class vector and the class weight vector,

$$y_j^c = h(1 - \sum_k^K |s_k - w_{jk}|), \tag{2}$$

where $h(\chi) = 1$ if $\chi > 0$ and 0 otherwise. When $y_j^c = 1$, the class weight vector of neuron j matches the class label associated with the input feature vector; otherwise, $y_j^c = 0$. Note that $y_j^c = 0$ indicates a class mismatch between the input class vector and the class vector of the neuron.

B. Neuron outputs

The neuron output based only on feature activation is determined by

$$o_j = \mathrm{ReLU}(y_j^f, \theta_y), \tag{3}$$

where $\mathrm{ReLU}(y, \theta_y) = h(y - \theta_y)\, y$ is the activation function, with a threshold of θ_y. During training, both the feature activation and the class activations are used to determine the neuron output o_j^c,

$$o_j^c = \mathrm{ReLU}(y_j^f, \theta_y)\, y_j^c. \tag{4}$$

During testing, the class label is unknown, so only the feature activation is used to determine the output; i.e., the y_j^c term is not used (set equal to 1) during testing.

C. Winner selection

During training, the winning neuron that matches the input class vector is denoted by $j_{\max}^c$ and selected via competition as

$$j_{\max}^c = \arg\max_j \; o_j^c. \tag{5}$$

If there are multiple winners, one is selected at random. If all o_j^c are 0, then no neurons are producing an output.

During testing, the winning neuron that has the highest feature activation is denoted $j_{\max}$ and selected as

$$j_{\max} = \arg\max_j \; o_j. \tag{6}$$

When the class vector of the winning neuron does not match the class vector of the input (test) vector, then the network is making a prediction error.

D. Decision logic during training

During training, the outputs o_j^c and o_j and winning neurons $j_{\max}^c$ and $j_{\max}$ are determined. Then, there are three possible actions: add a neuron (neurogenesis), add connections (synaptogenesis), or modify the feature weights. The decision logic for these three conditions is as follows. If all o_j^c are 0, then a false negative error is being made, so add a neuron. If no more neurons are available (i.e., if the maximum allowed number of neurons is reached), then perform synaptogenesis. If $j_{\max} \neq j_{\max}^c$, then $j_{\max}$ is choosing the wrong neuron (which is a false positive error), so perform synaptogenesis. And, finally, if $j_{\max}^c = j_{\max}$, then no errors are being made, so perform synaptic modification to *move* the feature weights of the winning neuron closer to the input vector. This decision process is depicted in Fig. 2.

In general, the main objective of the decision logic is to minimize the prediction errors across all input vectors.

E. Neurogenesis

In biological systems, neurogenesis refers to the process of forming new neurons from neural stem cells. In our model, neurogenesis refers to adding a new neuron from a finite pool of unconnected neurons. When a new neuron is added, its feature weights are initialized to a fraction of the current input feature vector; thus, the initial feature weights are in the same direction as the current input feature vector. Also, the class weights of the new neuron are set equal to the input class vector **S**.

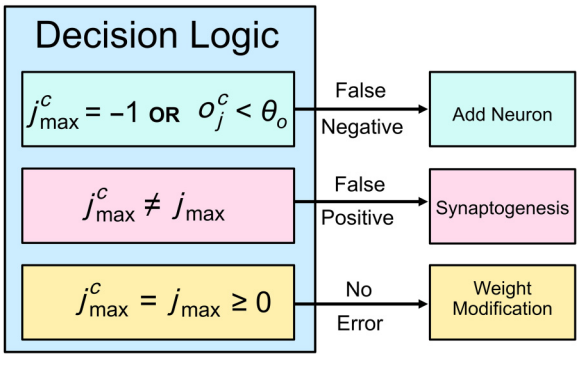

FIG. 2. The decision logic for three possible actions during training: neurogenesis, synaptogenesis, and synaptic weight modification. The overall objective is to minimize prediction errors.

F. Synaptogenesis

Whenever new information is presented to biological brains, they either modify existing neural connections or form new ones. In most ANNs, all neurons are preconnected and their weights are imposed during a training step. In the simulations presented here, there are initially no connected neurons.

In our model, synaptogenesis is only allowed to occur on winning neurons. Synaptogenesis determines if new connections can be added via the probability of forming a new connection on neuron $J = j^c_{\max}$ at input feature i,

$$p_{iJ} = \gamma (1 - c_{iJ})\, a_i\, o^c_J, \tag{7}$$

where $\gamma \in [0, 1]$ is the synapse formation rate and a_i is the avidity such that $a_i = 1$ if $x_i > A_v$ and 0 otherwise. (A_v exists so that noise does not invoke synaptogenesis.) If any $p_{iJ} > B$ where $B \sim U(0, 1)$ (a random number from a uniform distribution for each (i, J) pair), then c_{iJ} is set to 1. $\mathbf{C}_J$ is a binary vector of the connections on neuron J. The parameter γ controls the sparsity of synapse connections. Here, we set $\gamma = 1$ so that all nonexistent connections with $a_i = 1$ and $o^c_J > 0$ are formed with $p_{iJ} = (1 - c_{iJ})\, o^c_J$.

G. Synaptic weight modification

During training, the feature weights of the winning neuron $J = j^c_{\max}$ are modified using the following learning rule:

$$\Delta w_{iJ} = \epsilon\, (x_i - w_{iJ})\, c_{iJ}\, y^f_J\,, \tag{8}$$

where ϵ is the learning rate. The subtraction term w_{iJ} is an anti-Hebbian term that guarantees that the weights have an upper bound and decreases weights when $x_i < w_{iJ}$.

This learning rule ensures that the feature weights of the neuron will be associated with its class weights. Also, the feature weights of the neuron will tend to represent the mean of all input feature vectors within θ_y of the feature weights of the neuron and having the class vector of the neuron. While there is no global (nonlocal) objective function that is minimized, under mild conditions the weights can be shown to converge to the dominant eigenvector of the correlation matrix of the input feature vectors that belong to the class vector of the neuron and that are within θ_y of its feature weights.

H. Mean firing rates

Monitoring the mean firing rate of each connected neuron is a good way of monitoring network performance. If the input feature vectors are evenly distributed across the classes, then one expects at least one neuron from the input (target) class to be firing in response to each input after sufficient training has occurred. Determining the neurons with the highest mean firing rates for each class allows the examination of the feature vectors of the neuron that are most representative or presented most often for that class. Many other insights can be gained from analyzing the mean firing rates.

The estimation of the mean firing rate can be expressed as

$$\bar{o}_j\,(t_n) = (1 - \alpha)\bar{o}_j\,(t_n - 1) + \alpha o_j\,(t_n - 1), \tag{9}$$

where the current average firing rate $\bar{y}_j\,(t)$ is a running average calculated using the previous average $\bar{y}_j\,(t_n - 1)$ and a "portion" of the current instantaneous (no bar) output $y_j\,(t_n - 1)$ at each time step (t_n). The variable α controls how much of the current output is used to update this running average. Here, o_j can be set to o^c_j during training if desired.

III. MODEL SIMULATIONS

In the simulations presented here, the network starts out with no connected neurons. As input feature and class vectors are presented, new connected neurons are formed. The threshold θ_y is a key parameter that controls how closely the input feature vector must match the winning feature-weight vector of the neuron. The class vectors of the input and the winning neuron are always required to match.

A. Supervised synaptogenesis

We first demonstrate, using synthetic data sets, how our model can be used for classification while reusing synaptic weights. Our synthetic data are generated as one-dimensional orthogonal vectors, as shown in Figs. 3(a) and 3(b). Each class, such as "ship," "cat," etc., is a unique vector orthogonal to all other classes. However, within each class itself, the data vectors should not be identical to one another and, instead, should have some similarities to one another (data overlap). One could imagine this as images of different-sized cats. We achieve this by first creating a 'pool' of values indicating the extent of overlap we want (such as 10% to 90%). We then randomly sample from this pool and, depending on the selected value, generate a vector that has an overlap with the original vector to that extent. We can see in Fig. 3(c) that generated vectors only overlap and lose orthogonality in their assigned classes and vectors across classes retain orthogonality. We set the simulation parameters to the ones in Table I and ran experiments using the following directions:

(a) We generate variants with partial overlaps in each class, ensuring that there are no interclass overlaps.

(b) We generate a total of 50 variants per class (overlap range 5%–40%).

(c) We train a model with no variants and then one variant per class, up to 50 variants per class. This means that we have a total of 100 different models.

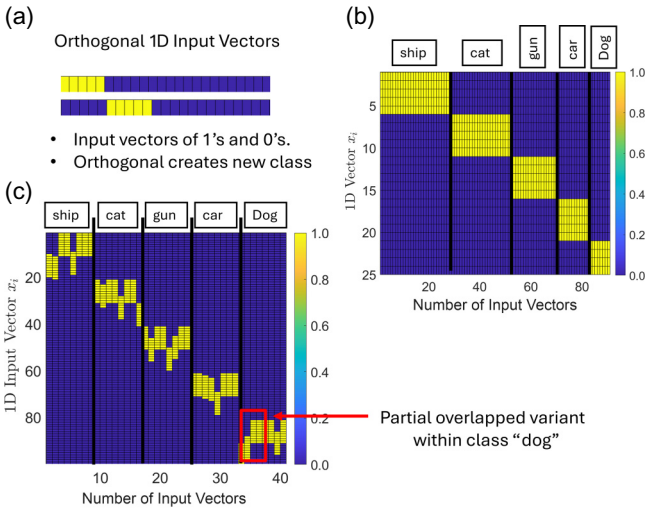

FIG. 3. (a) Orthogonal vectors belonging to two unique classes: 1's and 0's are used to simplify the approach: 1D, one-dimensional. (b) The final data set for five classes. (c) Overlap generates variations within each class. (d) The supervised model.

(d) We repeat training 50 times with different random seeds (for probabilistic synaptogenesis) for a total of 2500 models.

(e) We analyze the neuron count plots and test the accuracy (80:20 split).

(f) We compare the accuracy to k-nearest neighbors for benchmarking.

In Fig. 4, we show the results of our runs. We see an initial rise in neuron count with increase in variation. However, this tapers off since the model no longer needs new neurons to reach maximum training accuracy. This shows that the shared features of the data set have been learnt by the model and that the model is able to reuse the same neurons. Accuracy is on par with k-nearest neighbors (kNNs) for benchmarking reasons. We use the SCIKIT-LEARN software packages for kNNs [8].

TABLE I. The supervised model parameters.

Name	Value
Synapse formation rate γ	1
Learning rate ϵ	0.05
Initial weight W_o	1
Shed threshold W_{shed}	0.01
Epochs	50
Average firing rate ρ	0.1
Inhibition factor A	1
Number of trials	50
Input neurons x_i	1
Hidden neurons y_j	50
Output neurons y_k	5
Output activation threshold θ_y	0.02

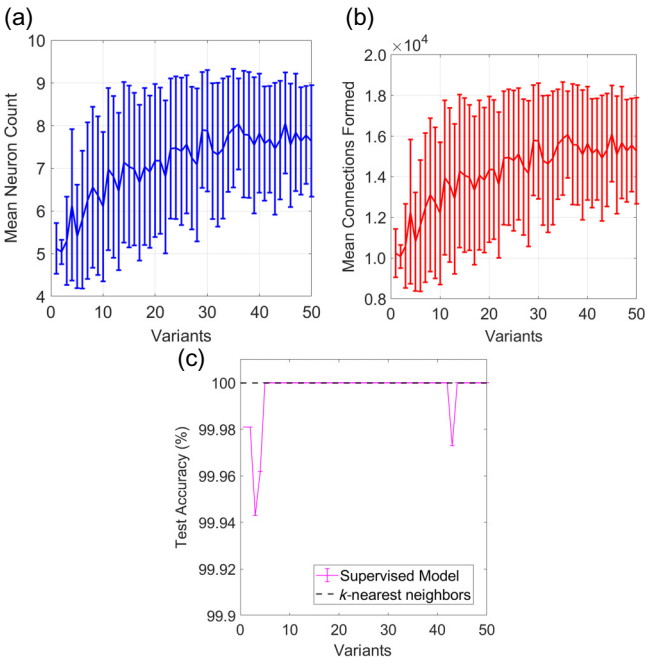

FIG. 4. The average number of allocated (a) neurons and (b) synapse connections employed during training of the supervised model using data sets with overlapping (shared features) and (c) their corresponding test-set accuracy benchmarked against k-nearest neighbors. We see neuron allocation "plateaus" as the shared features between input vectors allow reuse of neurons.

B. Benchmark on real data sets

Our simulations on synthetic data sets enable us to understand and explain the model. However, we also compare the performance on a few common benchmarks on real data sets to demonstrate the usefulness of our model. For this, we run our model as well as kNNs on both the Modified National Institute of Standards and Testing (MNIST) [9] and Fashion-MNIST (F-MNIST) [10] data sets. Since we wish for our model to be used for edge applications, we run two separate benchmarks for each model. In one case, we use the full training set of 60 000 samples to emulate a mainframe server cluster scenario with ample access to resources and in another, we run on a resource-constrained scenario, as we would see in edge applications, of 1000 samples per class, leading to 10 000 total training samples. We refer to our model as supervised adaptive synaptogenesis (SAS). In Table III, we show an extensive comparison of the test accuracy versus the total number of parameters for each model. For each data set, we run kNNs for $k = 3, 5, 7, 9$ and chose the model with the best accuracy. For our SAS models, we use the parameters shown in II. In each case, we run three different models based on unique seeds and for the 10 k runs, we use a unique seed to randomly sample data points from the full training sets. We find a mean accuracy and parameter count in each case. On the full MNIST, our model loses out on roughly 2%

TABLE II. The MNIST and F-MNIST SAS parameters.

Variable	Value (unit)	Description
W_{shed}	0.001	Weight-shedding threshold
W_{set}	0.1 × input	Initial weight for new synapses
ϵ	0.005	Learning rate for weight modification
γ	1	Synapse formation rate
θ_y	0.866025	ReLU threshold
A_v	0.05	Avidity threshold for input detection

accuracy with a 12.5 times reduction in parameter count. on the 10k run, we start seeing some interesting behavior. Both models lose out on accuracy; however, our model performs better than kNNs. We lose this advantage when the models are run on F-MNIST. In this case, the accuracy of our model is less in each case but the parameter count is much smaller than for kNNs. Overall, we see that our model is competitive versus kNNs while being smaller in size, making it a viable candidate for edge applications.

IV. HARDWARE IMPLEMENTATION OF ADAPTIVE SYNAPTOGENESIS

We now discuss a set of hardware accelerators for adaptive synaptogenesis mainly comprising *nanomagnetic* devices for their superior energy efficiency and nonvolatility.

The main equations for synapse formation, weight modification, and firing-rate monitoring in adaptive synaptogenesis are Eqs. (7), (8), and (9), respectively. It is clear from this set of equations that a hardware platform for adaptive synaptogenesis will require seven major components: (1) a device to measure the firing rate of a neuron, (2) a comparator for comparing the measured firing rate with a threshold, (3) an analog multiplier to multiply the analog (voltage or current) outputs of two neurons, (4) an analog subtractor to subtract one neuronal output from another, (5) analog (preferably nonvolatile) weights, (6) a probability generator the probability distribution of which can be tuned as desired, and (7) a neuron, such as a McCullough-Pitts-type neuron or perceptron.

TABLE III. The mean accuracy and parameter count of SAS versus kNNs.

Data set	Model	Accuracy (%)	Parameter count, M
MNIST (60k)	kNNs	98.52	47.00
	k=3	±0.00	±0.00
	SAS	96.68	3.74
		±0.08	±0.01
MNIST (10k)	kNNs	91.54	7.84
	k=3	±0.19	±0.00
	SAS	95.45	0.93
		±0.06	±0.003
F-MNIST (60k)	kNNs	85.33	47.00
	k=5	±0.00	±0.00
	SAS	80.96	2.98
		±0.29	±0.01
F-MNIST (10k)	kNNs	81.62	7.84
	k=7	±0.30	±0.00
	SAS	77.08	0.68
		±0.90	±0.001

We will implement most of the hardware with magnetics, as opposed to electronics, primarily because magnetic devices are usually (not always) more *energy efficient* than their electronic counterparts and they are *nonvolatile*. Both attributes are very desirable for edge computing and/or computing in resource-constrained environments (deep space, underground or underwater, etc.) where energy sources are scarce and all data must be stored *in situ* in nonvolatile elements because the cloud is either unavailable (deep space) or unreliable (enemy territory). The best-designed hardware will comprise mostly magnetic elements with a smattering of electronic elements for gain, interfacing, high speed, and error resilience, if needed.

Below, we describe the design of the seven essential elements.

A. The neuron

We will implement a McCullough-Pitts neuron or perceptron with a straintronic spin neuron (SSN) [11]. This model not only has excellent energy efficiency and a fast firing speed (subnanosecond) but also keeps in line with our firing-rate-based synaptogenesis model. It relies on magnetization switching with electrically generated mechanical strain (hence the name "straintronic").

The device is shown in Fig. 5(a) and consists of a "*skewed* straintronic magnetic tunnel junction" (ss-MTJ), which has elliptical magnetic hard and soft layers the major axes of which are *not* collinear (hence called "skewed"). The soft layer is in elastic contact with a piezoelectric thin film. The resistors $r_1 \cdots r_m$ encode synaptic weights. Because of any residual dipole coupling between the hard and the soft layer, the ss-MTJ is in the high-resistance state in the absence of inputs. When the weighted sum of the input voltages $V_{in1}, V_{in2} \cdots V_{in\ m}$ exceeds a certain threshold, the strain generated in the piezoelectric (and transferred to the soft layer) flips its magnetization and abruptly switches the resistance of the ss-MTJ from the high value R_H to the low value R_L, thereby switching the output voltage V_{out} from 0 to $I\,(R_L - R_H)$, where I is a bias current driven through the ss-MTJ. The working of this neuron has been explained in Ref. [11] and hence is omitted here. The output voltage V_{out} is given by [11]

$$V_{\text{out}} = f\left(\sum_i w_i V_{in\ i}\right), \tag{10}$$

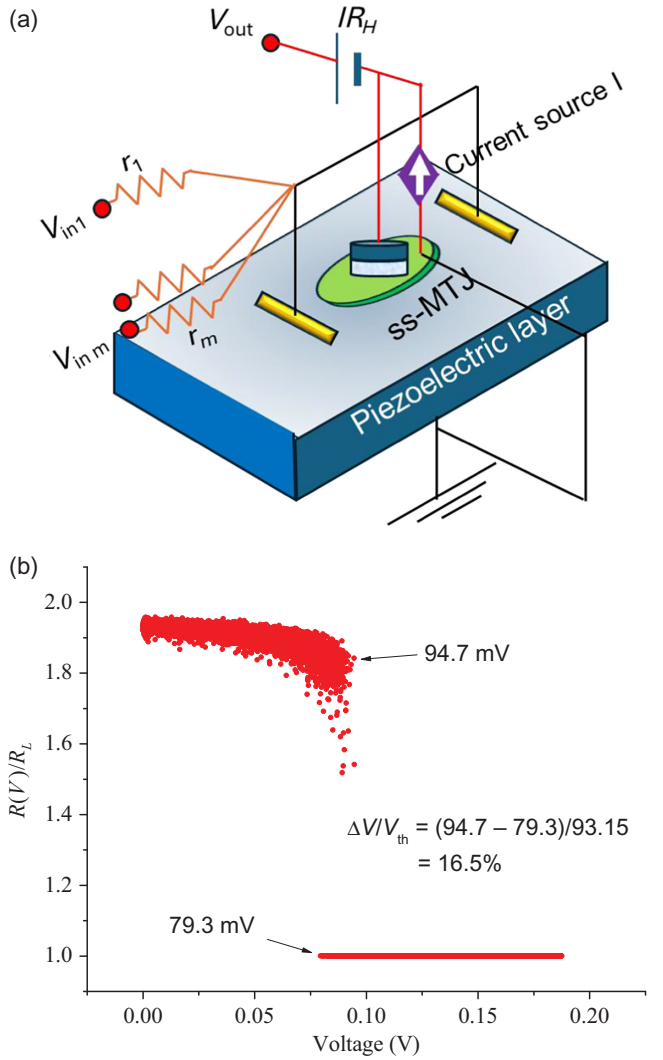

FIG. 5. (a) A schematic of a straintronic spin neuron. (b) The simulated firing characteristics of the straintronic spin neuron at room temperature computed with stochastic Landau-Lifshitz-Gilbert simulations of the magnetodynamics of the soft layer of the "*skewed* straintronic magnetic tunnel junction" (ss-MTJ) (in contact with the piezoelectric) in the presence of voltage-generated stress. The ratio of the ss-MTJ resistance to the low-resistance value is plotted as a function of the strain-generating voltage appearing across the piezoelectric owing to the weighted sum of the input voltages. The dispersion in the high-resistance state of the neuron is due to thermal fluctuations in the magnetization. Since the simulation is terminated immediately upon completion of firing, no fluctuations in the transfer characteristic are visible in the low-resistance state. The soft layer is assumed to be made of Terfenol-D and has an in-plane energy barrier of 73 kT. The piezoelectric is assumed to be a 100-nm-thick layer of PZT. Reproduced from Ref. [11] with the permission of the Institute of Physics.

where f is our perceptron function and

$$w_i = \frac{r_1 \parallel r_2 \parallel \cdots \parallel r_m}{r_1 \parallel r_2 \parallel \cdots \parallel r_m + r_i}. \tag{11}$$

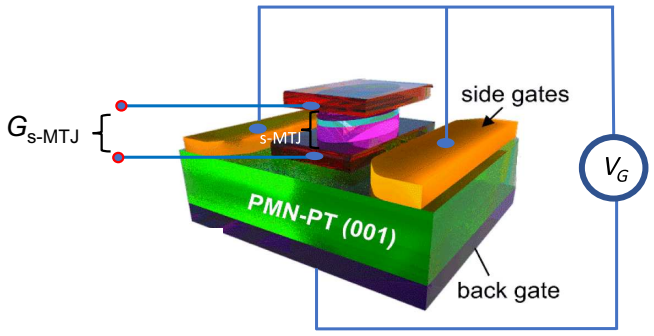

FIG. 6. A straintronic magnetic tunnel junction (s-MTJ). Reproduced from Ref. [12] with the permission of the American Institute of Physics. The gate voltage V_G is applied across the piezoelectric Lead Magnesium Niobate-Lead Titanate (PMN-PT) between the (shorted) side gates and the back gate to switch the s-MTJ resistance. This device exhibited a room-temperature tunneling magnetoresistance ratio (TMR) of slightly larger than 100%.

The firing sequence of the neuron is shown schematically in Fig. 5(b).

We can update the weights by updating the resistance r_i. Weights can be implemented with memristors, the resistance of which can be changed or updated with a (gate) voltage, but a better option is a simple straintronic magnetic tunnel junction (s-MTJ) [12], the only difference of which compared to a generic MTJ is that the magnetization of the soft layer is rotated with electrically generated strain rather than the more common spin-transfer-torque, spin-orbit torque, or voltage-controlled magnetic anisotropy. The structure of an s-MTJ is shown in Fig. 6. The reason why the s-MTJ is superior is that its transfer characteristic has a linear region in which the conductance is *linearly proportional* to the gate voltage [13,14] (for the computed transfer characteristic of an s-MTJ, see Fig. 8). Reference [13] has provided analytical proof that such a linear region must exist and that its width is proportional to the antiparallel resistance of the s-MTJ. This linearity is very desirable and not easily available in any other device such as a memristor. The linearity allows more accurate control of the conductance, where equal increments or decrements of the gate voltage will result in equal increase (long-term potentiation) or decrease (long-term depression) of the conductance, improving accuracy in classification tasks [15,16]. It may also benefit adaptive synpatogenesis.

B. Nonvolatile weights

Unlike memristor-based weights, weights implemented with s-MTJs may appear to be *volatile*, since the gate voltage should be kept on to maintain the conductance or resistance of the s-MTJ at a particular value. However, it must be noted that all that the gate voltage does is produce a strain in the soft layer of the s-MTJ. If the produced strain survives after the gate voltage is removed, then the

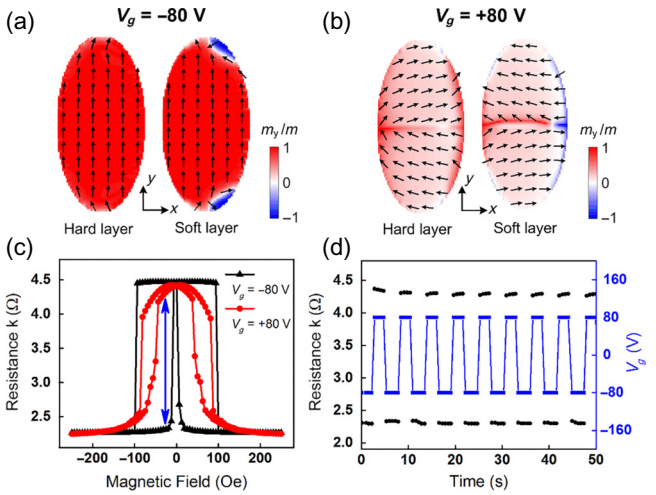

FIG. 7. The magnetization configuration of the hard and soft Co-Fe-B layers of the s-MTJ shown above after application of a gate voltage of (a) −80 V and (b) +80 V . The dimension of the s-MTJ is 3μm × 6 μm. (c) Magnetoresistance loops for gate voltages of −80 V and +80 V. The blue arrow indicates the switchable high- and low-resistance states. (d) Toggling of the MTJ between high- and low-resistance states with application of ±80 V gate-voltage pulses. The blue line represents the gate voltage and the black represents the MTJ resistance state. The ratio of high-to-low resistance is slightly larger than 2:1. A small bias magnetic field of 30 Oe is applied along the $+y$ axis to overcome the dipole interaction between the two magnetic layers. Reproduced from Ref. [12] with the permission of the American Institute of Physics.

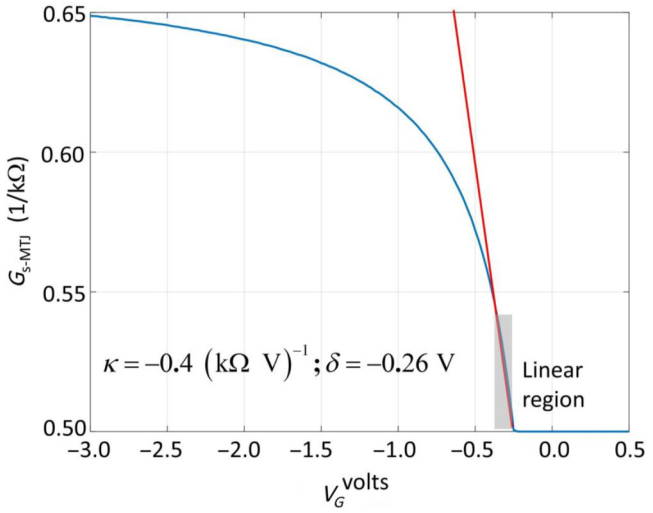

FIG. 8. The transfer characteristic of an s-MTJ computed with the use of stochastic Landau-Lifshitz-Gilbert simulation of the magnetodynamics of the soft layer in the presence of gate-voltage-generated strain at room temperature. Reproduced from Ref. [13] with the permission of the Institute of Electrical and Electronics Engineers. The parameters for the soft layer are as follows: major axis = 800 nm, minor axis = 700 nm, thickness = 2.2 nm, saturation magnetization $M_s = 8.5 \times 10^5$ A/m, dipole coupling field $H_d = 1000$ Oe, Gilbert-damping constant = 0.1, saturation magnetostriction $\lambda_s = 600$ ppm (parts per million), Young's modulus $Y = 120$ GPa, piezoelectric coefficient $d_{33} = 1.5 \times 10^{-9}$ C/N, piezoelectric layer thickness $t = 1$ μm. The value of the antiparallel resistance R_{AP} is assumed to be 2 kΩ and the value of the parallel resistance R_P is 1 kΩ.

gate voltage need not be kept on and the weight would be nonvolatile. This obviously requires *nonvolatile strain*. This is demonstrated in the plots of Fig. 7. Nonvolatile strain induced by an electrical voltage in a piezoelectric has been reported by a number of authors [17–20]. At this time, however, the robustness of nonvolatile strain is questionable, i.e., whether it survives thermal recycling, and the retention time is undetermined, but the fact that remanent strain is found after withdrawal of the voltage that induces strain in a piezoelectric is a very encouraging development, which portends *nonvolatile weights* when they are implemented with s-MTJs.

C. A device to measure the firing rate of a neuron

To measure the firing rate of the neuron, we employ a domain-wall synapse (DWS) [21,22], shown in Fig. 9. It consists of a perpendicular Magnetic Tunnel Junction (p-MTJ) where the ferromagnetic hard and soft layers have perpendicular magnetic anisotropy. The p-MTJ is fabricated on a heavy-metal (HM) layer or a topological insulator (TI). The output voltage pulses generated by a firing neuron are converted to current pulses and injected into the HM layer or TI to create successive pulses of spin-orbit torque that move the domain wall in the soft layer of the p-MTJ by discrete amounts Δx_n.

The conductance of the p-MTJ is given by [21]

$$G_{\text{p-MTJ}} = \frac{x}{L} G_{\text{P}} + \frac{W}{L} G_{\text{DW}} + \left(1 - \frac{x+W}{L}\right) G_{\text{AP}}, \quad (12)$$

where G_{P} is the parallel conductance and G_{AP} is the antiparallel conductance of the p-MTJ, G_{DW} is the conductance associated with the domain wall, L is the length of the soft layer, and $x = \sum_i^m \Delta x_i$, the total domain-wall displacement after m pulses.

In order to measure the firing rate, we build the circuit of Fig. 10, where the DWS is placed in series with a resistor $R1$ and connected to a power-supply voltage V_s through an NMOS transistor. The gate voltage of the NMOS is turned on for a time duration T and we assume that the SSN fires m times during that period. Because of slight dipole coupling between the hard and the soft layers of the p-MTJ, it will initially be in the antiparallel state. After time T, the

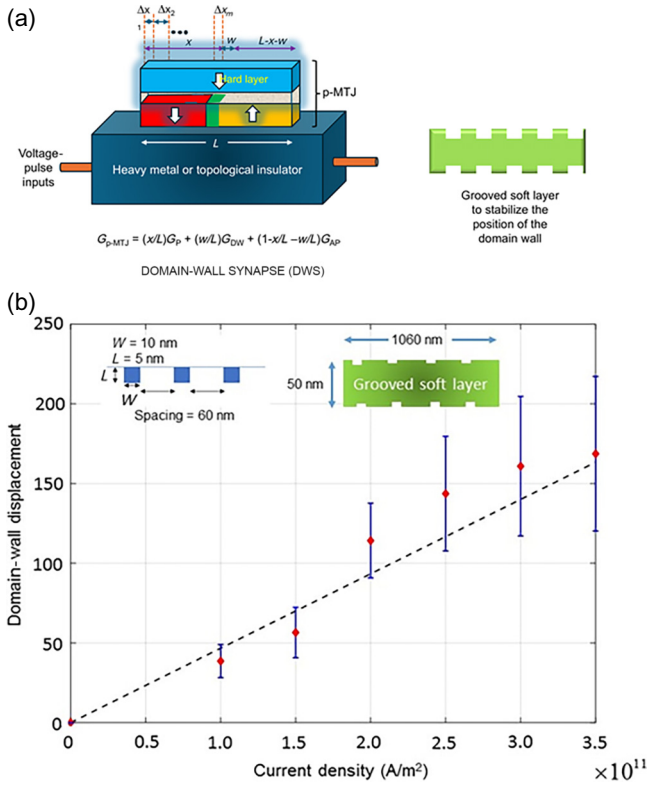

FIG. 9. (a) A domain-wall synapse consisting of a perpendicular Magnetic Tunnel Junction (p-MTJ) fabricated on a heavy-metal (HM) or a topological insulator (TI) layer. The domain-wall position in the soft layer of the p-MTJ can be translated with current pulses injected into the HM or TI and can stabilized at a location in the presence of thermal noise by using a periodically grooved soft layer. (b) The mean domain-wall displacement versus the current density in a grooved Terfenol-D soft layer obtained from micromagnetic simulation (MuMax3) in the presence of thermal noise. The current is injected into the underlying HM layer. The mean is obtained by averaging over the results of 1000 simulations. The inset shows the groove dimensions and spacing. The error bars represent the standard deviations in the domain-wall displacement. This figure is not to scale. The point near the origin corresponds to the minimum current density of 1.6×10^{10} A/m^2 and the domain-wall displacement at this current density is approximately 5 nm. Assuming a current pulse width of 0.5 ns, the maximum energy dissipation for a multiply-and-accumulate (MAC) operation is estimated to be 530 aJ in Ref. [13]. Reproduced from Ref. [13] with the permission of the Institute of Electrical and Electronics Engineers.

conductance of the p-MTJ will be

$$\begin{aligned} G_{\text{p-MTJ}} &= \frac{\sum_i^m \Delta x_i}{L} G_{\text{P}} + \frac{W}{L} G_{\text{DW}} + \left(1 - \frac{\sum_i^m \Delta x_i + W}{L}\right) G_{\text{AP}} \\ &= m \frac{\Delta x}{L} G_{\text{P}} + \frac{W}{L} G_{\text{DW}} + \left(1 - \frac{\Delta x + W}{L}\right) G_{\text{AP}} \\ &= m \frac{\Delta x}{L} (G_{\text{P}} - G_{\text{AP}}) + \frac{W}{L} G_{\text{DW}} + \left(1 - \frac{W}{L}\right) G_{\text{AP}}, \end{aligned} \tag{13}$$

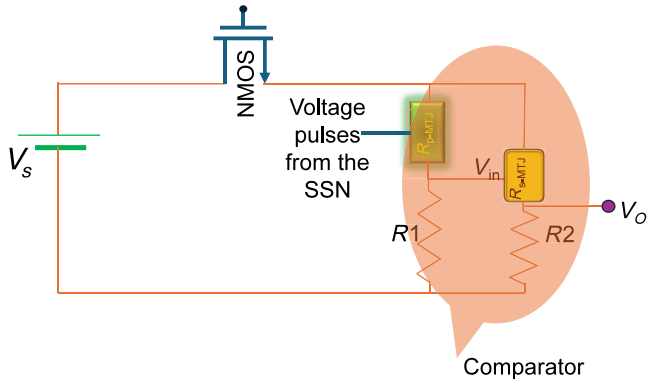

FIG. 10. The circuit to measure the firing rate and compare it with a threshold.

where Δx is the average displacement of the domain wall after a pulse. It can be determined experimentally with a magnetic force microscope after the soft layer has been subjected to a number of pulses. Therefore,

$$\begin{aligned} m = &\frac{L}{\Delta x} \frac{1}{(G_p - G_{\text{AP}})} \\ &\times \left[G_{\text{p-MTJ}} - \frac{W}{L} G_{\text{DW}} - G_{\text{AP}} \left(1 - \frac{W}{L}\right) \right]. \end{aligned} \tag{14}$$

The quantities W, L, G_{P}, G_{AP}, G_{DW}, and Δx are known. Hence by measuring $G_{\text{p-MTJ}}$, one can deduce the value of m. The firing rate is simply m/T. Thus, one can measure the firing rate.

D. Determining if the firing rate exceeds a threshold—the comparator

From Eq. (13), we see that the p-MTJ conductance is linearly proportional to m and hence to the firing rate m/T. Let the p-MTJ conductance corresponding to a threshold firing rate be $G^T_{\text{p-MTJ}}$ and let its reciprocal be the resistance $R^T_{\text{p-MTJ}}$. In the voltage-divider circuit of Fig. 10, the voltage V_{in} is $R1/(R1 + R_{\text{p-MTJ}})$. At the threshold value,

$$V^T_{\text{in}} = \frac{R1}{R1 + R^T_{\text{p-MTJ}}} V_s. \tag{15}$$

In Fig. 10, the s-MTJ is a large-cross-section straintronic MTJ, the resistance of which switches somewhat abruptly from high to low when the voltage applied to its gate exceeds a threshold value [12]. The threshold depends on the energy barrier within the soft layer of the s-MTJ and hence can be engineered by engineering the shape of the s-MTJ (the ellipticity of the soft layer). We design the s-MTJ such that the threshold voltage for switching is equal to V^T_{in}. Then, if the firing rate of the SSN exceeds the threshold, the s-MTJ resistance will switch from high to low. Since $V_O = R2/(R2 + R_{\text{s-MTJ}})$, a high value of V_O will imply

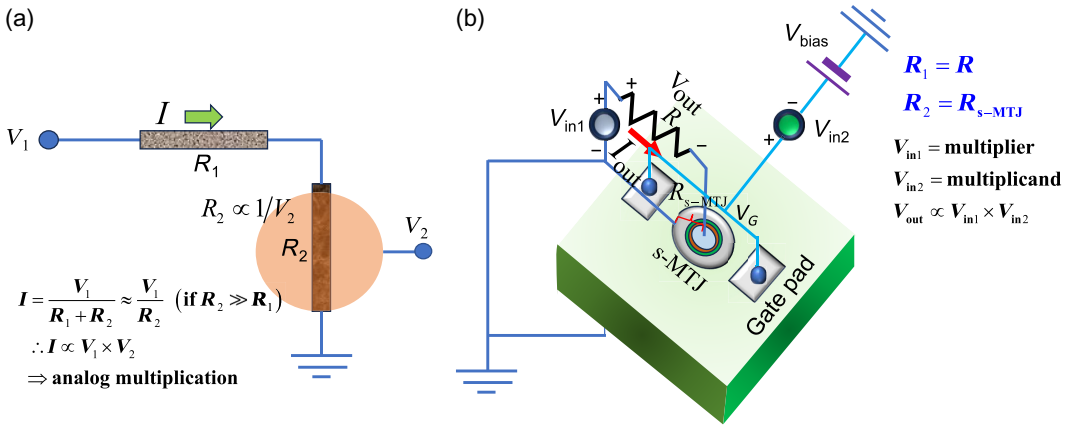

FIG. 11. (a) A voltage divider with a gate-voltage-tunable resistor R_2 (resistance inversely proportional to the gate voltage, such as an s-MTJ biased in the linear region of the transfer characteristic) can implement an *analog multiplier*. The multiplier and multiplicand are encoded in the input voltage and the gate voltage. The product is encoded in the current flowing through the circuit. (b) An actual implementation of an analog multiplier with an s-MTJ biased in the linear region of the transfer characteristic and some bias sources. Reproduced from Ref. [23] with the permission of the Institute of Electrical and Electronics Engineers.

that the threshold firing rate has been exceeded and a low value will indicate that the threshold has not been reached. Thus, by monitoring V_O, we can infer if the firing rate has exceeded a threshold. This is the basis of the comparator.

E. Analog multiplier

An analog multiplier can be realized with a simple voltage-divider network consisting of two resistors, the resistance of one of which can be tuned with a gate voltage such that the resistance is inversely proportional to the gate voltage. The implementation is shown in Fig. 11. The voltage-dependent resistor R_2 is obviously implemented with an s-MTJ biased in the linear region of the transfer characteristic shown in Fig. 8, where the conductance is proportional to the applied gate voltage and hence the resistance is inversely proportional to the gate voltage. An exact implementation of the analog multiplier can be found in Ref. [13,23] and hence is not repeated here.

F. Analog subtractor

An analog subtractor can also be realized with two voltage-divider networks, where once again we use s-MTJs to implement resistors the resistances of which are inversely proportional to the gate voltage. We show only the circuit representations in Fig. 12, since the device implementation follows trivially from there.

G. A probability generator randomly generating binary bits 0 and 1 with tunable distribution of the probabilities of either bit

This device can be realized with a simple binary stochastic neuron (BSN) implemented with a low-barrier nanomagnet. The low-barrier nanomagnet forms the soft layer of an MTJ, the resistance of which encodes the magnetization orientation of the soft layer. The resistance is given by

$$R_{\mathrm{MTJ}} = R_{\mathrm{P}} + \frac{R_{\mathrm{AP}} - R_{\mathrm{P}}}{2}[1 - \cos\theta], \tag{16}$$

where $R_{\mathrm{P(AP)}}$ is the parallel (antiparallel) resistance of the MTJ and θ is the angle between the magnetizations of the hard and the soft layer. If the magnetization of the soft layer subtends an acute angle with that of the hard layer, then the resistance is low and is interpreted as bit 0, whereas if the angle is obtuse, the resistance is high and is interpreted as bit 1. The probability of obtaining the bit 0 can be tuned by injecting a spin-polarized current into the soft layer with varying spin polarization.

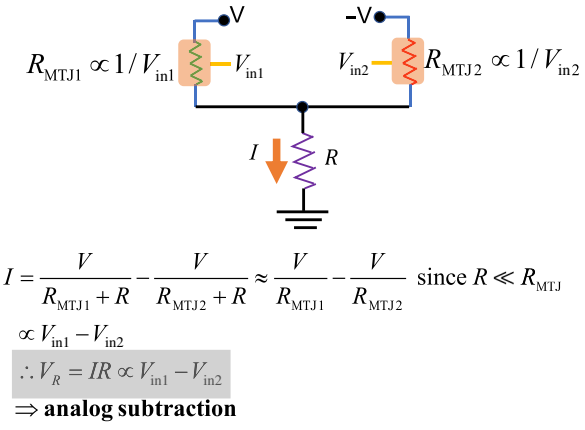

FIG. 12. An *analog subtractor* implemented with two s-MTJs acting as resistors, the resistances of which are inversely proportional to the gate voltages applied to them.

Let us say that the unit vector along the magnetization of the hard layer is $\hat{m}$ and that along the spin polarization is $\hat{s}$. If the scalar product $\hat{m} \cdot \hat{s}$ is positive, we will call the spin-polarized current positive; otherwise, we will consider it as negative. The probability of the measured bit being 0 depends on the sign and magnitude of the spin-polarized current

V. CONCLUSIONS

Adaptive synaptogenesis has been discussed as a neuromorphic learning approach that mitigates the challenge of catastrophic forgetfulness inherent to conventional ANN retraining. Rather than over-writing all past memory in a single disruptive event, adaptive synaptogenesis enables constructive learning over time and efficient long-term memory retention. Adaptively and continually adding and culling synaptic connections in an artificial neural network retains long-term learning gleaned through longitudinal experience. Conventional global retraining approaches are inherently incapable of such comprehensive subjective experiential learning that encompasses even temporally distant or sparse events. The adaptive-synaptogenesis approach is particularly relevant for edge-sensing applications, in which data connectivity, bandwidth limitations, or computing resource constraints make complete periodic retraining impractical or impossible.

Simulation models have demonstrated the functional viability of this construct, both in supervised and unsupervised modes of learning. Model inputs correlated with classes have been automatically encoded through the creation of new synaptic connections and unused or spurious connections have been shed for computational efficiency. The learning accuracy for adaptive-synaptogenesis modes has been comparable to that of kNNs in a series of benchmark tests. We have compared the SAS performance to kNNs because it is a common benchmark and because it is directly comparable to SAS with a high θ_y. Other models, such as Kohonen's self-organizing maps (SOMs) and Boltzmann machines, also exist, where the former uses a neighborhood function and the latter uses an explicit global-objective function with an annealing schedule. Our model does not use either of these methods, instead using a simple loss function minimizing errors based on the neuron with maximum excitation using lateral inhibition. This learning rule causes, under mild conditions, the neuron weights to approach the dominant eigenvalue of the correlation matrix of the input feature vectors belonging to the input class within a specific angle from the weights of the neuron.

Beyond the theoretical construct and modeling of this architecture, a hardware implementation has been proposed using power-efficient nanomagnetic devices, capable of variable output that can be changed within a nanosecond. These straintronic spin neurons use linearly gate-regulated channel conductance to encode weights in memory. While nonvolatility of the strain state has been reported, more work is required to understand the environmental limits. Furthermore, several logic devices have been described as functional elements of an ANN with adaptive-synaptogenesis capability, including a firing-rate comparator, multiplier, and subtractor.

In future work, the adaptive-synaptogenesis model will be implemented using s-MTJ devices and characterized for comparison to the simulation results reported herein. Subsequently, the hardware will be used for real-time learning applications, such as embedded classification techniques in sensing applications and fluid neuromuscular control in robotic applications.

ACKNOWLEDGMENTS

We are indebted to our late colleague Professor William Levy of the University of Virginia for many illuminating discussions pertaining to neuroscience. This project was funded in part by the U.S. National Science Foundation IUCRC Multifunctional Integrated System Technology Center, the U.S. Air Force Office of Scientific Research under Grant No. FA865123CA023, and by the Virginia Innovation Partnership Corporation under Grant No. CCF23-0114-HE.

DATA AVAILABILITY

The data that support the findings of this paper are not publicly available. The data are available from the authors upon reasonable request.

[1] H. J. S. Feder, J. J. Leonard, and C. M. Smith, Adaptive mobile robot navigation and mapping, Int. J. Rob. Res. **18**, 650 (1999).

[2] Y. Mei, Y.-H. Lu, Y. Hu, and C. Lee, Deployment of mobile robots with energy and timing constraints, IEEE Trans. Robot. **22**, 507 (2006).

[3] I. Lluvia, E. Lazkano, and A. Ansuategi, Active mapping and robot exploration: A survey, Sensors **21**, 2445 (2021).

[4] M. Mermillod, A. Bugaiska, and P. Bonin, The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects, Front. Psychol. **4**, 504 (2013).

[5] E. Hajizada, P. Berggold, M. Iacono, A. Glover, and Y. Sandamirskaya, in *International Conference on Neuromorphic Systems 2022*, ICONS '22 (ACM, New York, 2022).

[6] W. B. Levy, A sequence predicting CA3 is a flexible associator that learns and uses context to solve hippocampal-like tasks, Hippocampus **6**, 579 (1996).

[7] D. A. August and W. B. Levy, Temporal sequence compression by an integrate-and-fire model of hippocampal area CA3, J. Comput. Neurosci. **6**, 71 (1999).
[8] F. Pedregosa *et al.*, J. Mach. Learn. Res. **12**, 2825 (2011).
[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE **86**, 2278 (1998).
[10] H. Xiao, K. Rasul, and R. Vollgraf, Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms, arXiv:1708.07747 [cs.LG] (2017).
[11] A. K. Biswas, J. Atulasimha, and S. Bandyopadhyay, The straintronic spin neuron, Nanotechnology **26**, 285201 (2015).
[12] Z. Y. Zhao, M. Jamali, N. D'Souza, D. Zhang, S. Bandyopadhyay, J. Atulasimha, and J. P. Wang, Giant voltage manipulation of MgO-based magnetic tunnel junctions via localized anisotropic strain: A potential pathway to ultra-energy-efficient memory technology, Appl. Phys. Lett. **109**, 092403 (2016).
[13] R. Rahman and S. Bandyopadhyay, A non-volatile all-spin non-binary matrix multiplier: An efficient hardware accelerator for machine learning, IEEE Trans. Elec. Dev. **69**, 7120 (2022).
[14] S. Bandyopadhyay, Straintronic magnetic tunnel junctions for analog computation: A perspective, J. Phys. D: Appl. Phys **58**, 152001 (2025).
[15] R. S. Yadav, P. Gupta, A. Holla, K. I. A. Khan, P. K. Muduli, and D. Bhowmik, Demonstration of synaptic behavior in a heavy metal ferromagnetic metal oxide heterostructure based spintronic device for on-chip learning in crossbar-array-based neural networks, ACS Appl. Electron. Mater. **5**, 484 (2023).
[16] V. B. Desai, D. Kaushik, J. Sharda, and D. Bhowmik, On-chip learning of a domain-wall-synapse-crossbar-array-based convolutional neural network, Neuromorph. Comput. Eng. **2**, 024006 (2022).
[17] A. Chen, Y. Wen, B. Fang, Y. Zhao, Q. Zhang, Y. Chang, P. Li, H. Wu, H. Huang, Y. Lu, *et al.*, Giant nonvolatile manipulation of magnetoresistance in magnetic tunnel junctions by electric fields via magnetoelectric coupling, Nat. Commun. **10**, 243 (2019).
[18] L. Yang, Y. Zhao, S. Zhang, P. Li, Y. Gao, Y. Yang, H. Hu, P. Miao, Y. Liu, A. Chen, *et al.*, Bipolar loop-like nonvolatile strain in the (001)-oriented $Pb(Mg_{1/3}Nb_{2/3})O_3$-$PbTiO_3$ single crystals, Sci. Rep. **4**, 4591 (2014).
[19] T. Wu, P. Zhao, M. Bao, A. Bur, J. L. Hockel, K. Wong, K. P. Mohanchandra, C. S. Lynch, and G. P. Carman, Domain engineered switchable strain states in ferroelectric (011) $[Pb(Mg_{1/3}Nb_{2/3})O_3]_{1-x}$-$[PbTiO_3]_x$ (PMN-PT, x=0.32) single crystals, J. Appl Phys. **109**, 124101 (2011).
[20] T. Wu, A. Bur, K. Wong, P. Zhao, C. S. Lynch, P. Khalili Amiri, K. L. Wang, and G. P. Carman, Electrical control of reversible and permanent magnetization reorientation for magnetoelectric memory devices, Appl. Phys. Lett. **98**, 262504 (2011).
[21] A. Sengupta, Y. Shim, and K. Roy, Proposal for all-spin artificial neural network: Emulating neural and synaptic functionalities through domain wall motion in ferromagnets, IEEE Trans. Biomed. Circ. Syst. **10**, 1152 (2016).
[22] S. Liu, I. P. Xiao, C. Cui, J. A. C. Incorvia, C. H. Bennett, and M. J. Marinella, A domain wall magnetic tunnel junction artificial synapse with notched geometry for accurate and efficient training of deep neural networks, Appl. Phys. Lett. **118**, 202405 (2021).
[23] S. Bandyopadhyay, Magnetic straintronics for ultra-energy-efficient unconventional computing: A review, IEEE Trans. Magn. **60**, 4100110 (2024).