Machine Learning Without a Processor: Emergent Learning in a Nonlinear Electronic Metamaterial

Sam Dillavou,* Benjamin D. Beyer, and Menachem Stern
Department of Physics and Astronomy, University of Pennsylvania, Philadelphia, PA, 19104, USA

Marc Z. Miskin
Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, 19104, USA

Andrea J. Liu and Douglas J. Durian
Department of Physics and Astronomy, University of Pennsylvania, Philadelphia, PA, 19104, USA and Center for Computational Biology, Flatiron Institute, Simons Foundation, New York, NY 10010, USA

(Dated: November 2, 2023)

Standard deep learning algorithms require differentiating large nonlinear networks, a process that is slow and power-hungry. Electronic *learning metamaterials* offer potentially fast, efficient, and fault-tolerant hardware for analog machine learning, but existing implementations are linear, severely limiting their capabilities. These systems differ significantly from artificial neural networks as well as the brain, so the feasibility and utility of incorporating nonlinear elements have not been explored. Here we introduce a *nonlinear* learning metamaterial – an analog electronic network made of self-adjusting nonlinear resistive elements based on transistors. We demonstrate that the system learns tasks unachievable in linear systems, including XOR and nonlinear regression, *without a computer*. We find our nonlinear learning metamaterial reduces modes of training error in order (mean, slope, curvature), similar to *spectral bias* in artificial neural networks. The circuitry is robust to damage, retrainable in seconds, and performs learned tasks in microseconds while dissipating only picojoules of energy across each transistor. This suggests enormous potential for fast, low-power computing in edge systems like sensors, robotic controllers, and medical devices, as well as manufacturability at scale for performing and studying emergent learning.

Keywords: Emergent Learning | Machine Learning | Soft Matter | Active Matter | Neuromorphic Computing

Arbitrarily complex functions can be produced by combining a sufficient number of even simple nonlinear operations [1]. Deep learning has exploited this fact, using digital computers to simulate ever larger artificial neural networks (ANNs) capable of elaborate and nuanced tasks in mechanical control, vision, and language [2–4]. Training and performing these tasks is typically accomplished using sequential digital logic to perform billions or more nonlinear calculations. However, this process is slow and power-hungry compared to neural networks in the human brain [5, 6], where 100 billion neurons simultaneously use nonlinear physical processes as their base level of computation [7, 8].

One aim of the field of *neuromorphic computing* [9] is to reduce the time and energy deficits of ANNs using hardware. A popular approach spreads computation across specialized devices and/or onto electrical or optical systems [10–18], reducing time and/or power required to perform a trained task. However, this class of hardware is typically still trained using gradient descent (as are all-digital ANNs), a process that requires both digital processing and near-perfect modeling of any physical computation [19]. In this regard the stacking of nonlinear functions is a disadvantage, as it can rapidly compound small errors in modeling. Another approach more explicitly imitates biological processes, *e.g.* spiking neural networks, using electronic circuits [6, 20]. These networks are powerful at performing specific tasks but general-purpose architectures remain a challenge, while memory requirements for learning hinder scalability [6].

A subfield of soft matter physics, *physical learning*, aims to train physical systems to exhibit desired responses to stimuli [21]. Unlike feed-forward ANNs, physical systems like mechanical or analog electrical networks can offer the advantage of being completely recurrent, with no imposed direction of information flow. However, in order for these systems to learn on their own, they must evolve during training by local rules [21]. In some cases, natural physical dynamics can be hijacked to produced desired responses [22], but understanding how complex responses can be trained remains an active problem [23–25].

Recently, the fusion of neuromorphic computing with physical learning has led to a 'learning metamaterial,' a system capable of performing supervised learning tasks adaptively without any external processing [26]. The system is a network of identical electronic components, namely variable resistors that self-adjust their resistances according to local rules, utilizing the Coupled Learning framework [27], closely related to Equilibrium Propaga-

* Correspondence email address: dillavou@sas.upenn.edu

tion [28] and Contrastive Hebbian Learning [29]. Like hybrid digital-analog hardware [19], the "forward computation" is performed by physical optimization, harnessing the principle of minimum power dissipation [13]; as voltage inputs are applied, natural physical dynamics lower the dissipated power, automatically "calculating" the voltage outputs. The metamaterial offers a distinct scalability advantage over hybrid digital-analog hardware because no companion simulation is required, so errors during training that arise from modeling the physical process do not compound with system size. Unfortunately, the variable resistor network suffers a fatal flaw – it is linear. Linear networks like the original perceptron [30] cannot perform simple nonlinear functions like XOR, regardless of how many elements they contain. *In silico* implementations of Equilibrium Propagation [28] have successfully trained nonlinear circuits [31] and feedforward nonlinear ANNs [32], and hybrid hardware-in-the-loop implementations are emerging [33]. However, the feasibility and functionality of a potentially scalable, nonlinear learning metamaterial has not been tested.

Here we report the laboratory implementation and study of a nonlinear learning metamaterial. The design may be microfabricated, establishing a paradigm for scalable learning that is distinct from that of biological brains. The system performs tasks using an electronic network of transistors wired as continuous and nonlinear variable resistors. During training the transistor gate voltages continuously evolve, encoding the learned state, until they are frozen by a supervisor. We detail the design of a single base element, which is repeated 32 times to create the autonomously trainable system. We demonstrate the system's ability to learn different nonlinear tasks, including regression and XOR, and show that it learns by fitting modes of the training data in order: mean, slope, curvature. Our system takes seconds to learn tasks, which it can then perform in microseconds while dissipating picojoules of power across each element.

SYSTEM DESCRIPTION AND DYNAMICS

We construct an electronic network using N-channel enhancement MOSFET transistors as the edges, as shown schematically in Fig. 1A. We operate these transistors in the Ohmic (passive) regime, such that their conductance K has dependence

$$K \propto G - V_{\text{th}} - \overline{V} \tag{1}$$

where G is the gate voltage of the transistor, $V_{\text{th}} \approx 0.7$ V is a fixed threshold voltage, and $\overline{V}$ is the average voltage at the two network nodes bookending the edge (V_1 and V_2 in Fig. 1A). Note that the dependence of conductance K on $\overline{V}$ is the source of nonlinearity in our system. As a result, edges with lower gate voltages are both less conductive and more nonlinear (see *Transistors as Nonlinear Resistors* in Appendix).

Applying ('input') voltages $\vec{I}$ to two or more nodes of the network creates a steady voltage state. Designating the measured voltage at another node as the 'output' O, we can think of this network as evaluating a nonlinear function

$$O = \mathcal{N}(\vec{I})_{\vec{G}} \tag{2}$$

where the form of $\mathcal{N}$ depends on network structure and the gate voltages $\vec{G}$, which serve as the *learning degrees of freedom* in our system. These values are analogous to *node weights* in machine learning: they evolve during training and encode the memory of that training. Eq. 2, *inference* in machine learning, is performed by physics, meaning that successful training results in a passive network whose structure 'solves' the task.

We now follow the Coupled Learning framework [27] to construct dynamical rules that collectively solve the inverse problem $L = \mathcal{N}(\vec{I})_{\vec{G}}$. In particular, the desired local update rule for an individual gate voltage G is

$$\dot{G} \propto \dot{K} \propto -\partial(P_C^{\text{tot}} - P_F^{\text{tot}})/\partial K = V_F^2 - V_C^2 \tag{3}$$

where K is the conductance of that edge, P^{tot} is the dissipated power for the entire system, V is the voltage drop across that edge (*i.e.* $V_1 - V_2$), and subscripts F and C denote two electrical states of the same system, 'free' and 'clamped.' To simultaneously access both states, we use a twin-network approach [26, 34, 35], as shown in Fig. 1B. We construct two copies of the same transistor network, and pair them by connecting the gates of commensurate edges to a single capacitor, as shown in Fig. 1B. This ensures our copies will always evolve identically. During training, we apply the same inputs $\vec{I}$ to both networks, *and* the label L to the 'clamped' network.

We then add a small circuit to every edge pair that runs current to the joint gate capacitor and changes its voltage according to

$$\dot{G} = \frac{V_F^2 - V_C^2}{V_0 R_0 C_0} \tag{4}$$

where V_F and V_C are the voltage drops across the two transistors ('free' and 'clamped'), as shown in orange and purple respectively in Fig. 1B, and $R_0 = 100\ \Omega$, $C_0 = 22\ \mu$F, $V_0 = 0.33$ V are constants set by circuit components (see Twin Edge Circuitry in Appendix). We find it useful to extract a learning time constant $\tau_0 = R_0 C_0 (V_0 G_+ / V_+^2) = 18$ ms where $V_+ = 0.45$ V and $G_+ = 5$ V are the approximate ranges of node and gate voltages permitted in the system. Then the learning equation can be rewritten as

$$\frac{\dot{G}}{G_+} = \frac{V_F^2 - V_C^2}{\tau_0 V_+^2} \tag{5}$$

normalizing with node and gate voltage values. From Eq. 5 it is clear that evolution becomes significant on the scale of τ_0 and stops only when all free and clamped voltages are equal, meaning $O = L$ and the inverse problem is solved.

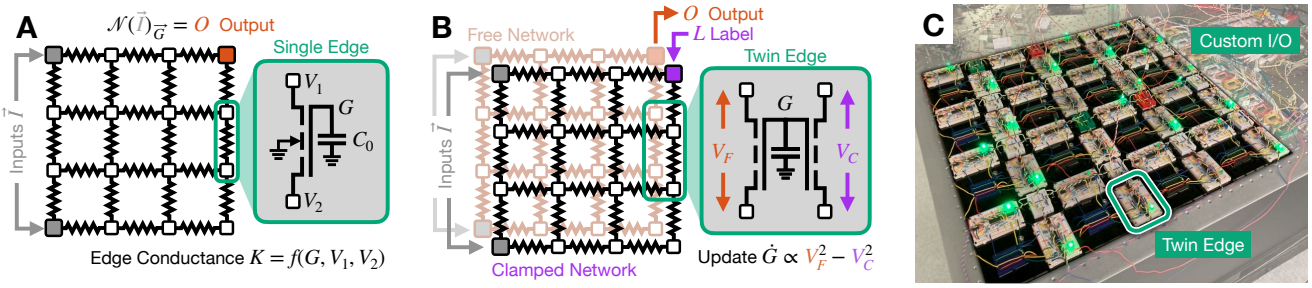

FIG. 1. **Description of the Learning Metamaterial (a) Physical response** We consider a physical system, an analog resistor network, as a function. Imposed voltages $\vec{I}$ (inputs) create a response throughout the network, and we choose a node to measure as output O. Our network is constructed using N-channel enhancement MOSFET transistors as variable nonlinear resistors. The conductance of each edge of the network thus depends on its connected node voltages V_1 and V_2, as well as the voltage on its gate G. $\vec{G}$ are the *learning degrees of freedom* that change during the course of training, and are each stored on local capacitors with capacitance C_0. **(b) Learning Response** We duplicate this network, and overlay the copies such that they do not interact directly, but the transistors on commensurate edges draw their gate voltages G from the same capacitor. This ensures two copies of the same electronic network. We designate one network as 'Free' and impose only inputs $\vec{I}$, and the other as 'Clamped' and impose inputs as well as the label (desired output) L. During training, circuitry on each twin edge continuously updates G by charging or discharging the local capacitor, depending on the local difference between the electronic states in the two networks. **(c) Image of the learning metamaterial.** The highlighted 'twin edge' is a transistor pair like the one shown schematically in (b), along with the circuitry to update G. This twin edge is repeated 32 times to create our system. Nodes in the network contain no circuitry and are simply connections between twin edges, and the only interaction a computer has with the network is through the custom input/output (I/O) hardware (described in *Supervisor and Measurement Circuitry* in Appendix, to impose boundary conditions ($\vec{I}$ and L), take measurements, and turn learning ($\dot{G}$) on and off. Note that the network has periodic connections, which are not pictured in the schematics in (a) and (b) for convenience.

We typically *nudge* the clamped output O^C towards the desired label L as instead of directly imposing it [26–28]. The clamped output O^C is then enforced as

$$O^C = \eta L + (1 - \eta)O \tag{6}$$

where $0 < \eta \leq 1$ controls the size of the nudge. We may also define the difference between two nodes as our output $O = O_+ - O_-$, as in [31], which allows inhibitory input-output relationships. This scheme is clamped using

$$O^C_\pm = O_\pm \pm \frac{\eta}{2}(L - O) \tag{7}$$

which recreates Eq. 6 when appropriately subtracted. Eq. 6 (or 7) is implemented with a feedback circuit, described in *Supervisor and Measurement Circuitry* in Appendix. As a result, O^C evolves continuously with O. We use $\eta = 1$ (directly clamping the label) for all regression tasks (Fig. 4 and 5), and $\eta = 0.5$ for Fig. 3.

Finally, we impose a global binary switch that toggles this evolution (Eq. 5) on and off. This is the only direct communication the supervisor has with the elements of the system. It allows us to freeze and measure the system and its functionality at any point, as well as to switch training data without unwanted effects. Freezing the system creates a passive, unchanging electrical network that can then be used for its trained purpose. For details about the circuitry used to implement learning and freezing, see *Twin Edge Circuitry* in Appendix. By contrast with the original digital linear circuitry [26], this new version is fully analog, operates without a clock, has nonlinear IV characteristics, and implements the exact Coupled Learning rule.

Once constructed, the metamaterial is a standalone system, and is trained purely by imposing boundary conditions in a two step process. First, a training example (datapoint) is chosen and applied to the network as boundary conditions - inputs to both networks and labels to clamped only - as shown in Fig. 1B. Next, learning is turned on for time t_h, and the gate voltages $\vec{G}$ evolve according to Eq. 5. This process is then repeated until training is complete, selecting a new datapoint each time. For all tasks in this work we use $t_h = 100$ μs.

Our system is a pair of self-adjusting twin networks, but it can also be viewed as a metamaterial, composed of many copies of a single element, as shown in Fig. 1C. This element, the MOSFET transistor pair, self-adjusts according to its electrical state, and this local evolution across all 32 twin edges produces supervised learning as an emergent property. An example of this process for a simple (single datapoint) task is shown in Fig. 2. Input boundary conditions $\vec{I}$ are applied (0 and 0.43 V), generating equilibrium voltages across both networks, including output O (Fig. 2A). A label $L = 0.13$ V is imposed on the output node of the clamped network, and evolution is unfrozen, allowing edges to evolve under Eq. 5. The direction of evolution of each edge in this example is shown on the network schematic in Fig. 2B. The system evolves continually until learning is frozen or, in this case, the network finds a state where the networks align, and the system stops evolving. As the voltage equilibration timescale is much shorter than the learning timescale

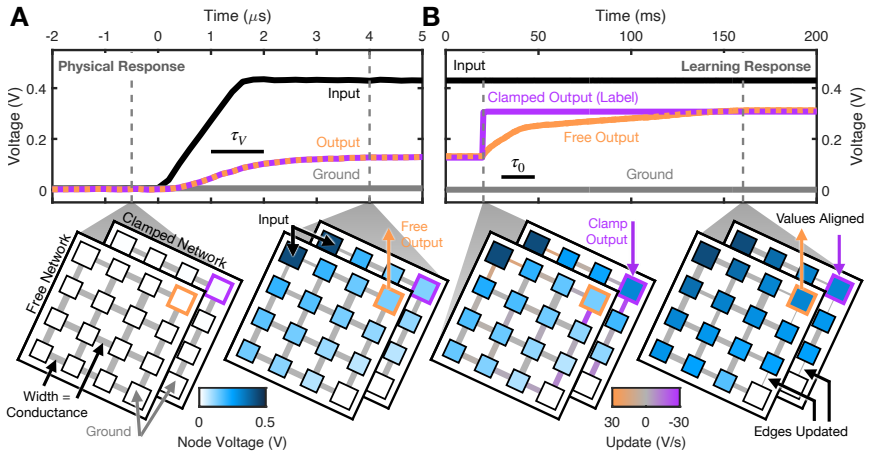

FIG. 2. **Emergent Supervised Learning** (a) Physical Response. When inputs are applied, nodes in the system find electronic equilibrium on a timescale of $\tau_V \sim 1$ μs. In this way, the network 'calculates' the output (orange) naturally from the inputs. Note that the voltage change does affect the conductance of some edges (Eq. 1), but not the learning degrees of freedom ($\vec{G}$) which are frozen. (b) Learning Response. Enforcing an output value in the clamped network and unfreezing the evolution of the gate voltages $\vec{G}$ will allow the system to continuously change these learning degrees of freedom. They will evolve with timescale $\tau_0 = 18$ ms until frozen, or until the system reaches a state where the two networks have identical voltages, that is, the labels are naturally generated from the inputs, and the task has been learned.

	$I_1 = 0$ V	$I_1 = I_0$
$I_2 = I_0$	$L = L_0$	$L = 0$ V
$I_2 = 0$ V	$L = 0$ V	$L = L_0$

TABLE I. XOR Training Data / Truth Table. Here $I_0 = 0.45$ V, and $L_0 = -0.087$ V

$\tau_V \approx 1\mu s \ll \tau_0 = 18$ ms, this process works continuously, with voltages rapidly equilibrating when $\vec{G}$ are changed. In this and subsequent examples we report training time as the time spent unfrozen, ignoring time to freeze the system and to switch datapoints and equilibrate, if necessary. Including these times would approximately double training times, and this accounting is further described in the discussion section.

RESULTS

As the first demonstration with the above circuitry, we train our system to perform the canonical nonlinearly-separable function XOR. We choose the input schema shown in Fig. 3A, with constant inputs $I_- = 0.11$ V and $I_+ = 0.33$ V, two variable inputs I_1 and I_2, and a differential output scheme $O = O_+ - O_-$. Our four-datapoint training set is described in table I, with $L_0 = -0.087$ V. We train our system for ten seconds, randomly selecting training points at each step. Initially, the network evolves rapidly, focusing on direct connections between sources and output(s), including the boxed region in Fig. 3A between I_+ and O_+. Learning slows over time until the network reaches its final state around five seconds, shown in Fig. 3A, right panel.

During training, the network goes through a variety of output truth tables before arriving at the desired checkerboard pattern of XOR, as shown in Fig. 3B. To characterize the learning dynamics that produces this nonlinear result, we produce an orthonormal basis, $\hat{n}_{jk}$, where j and k represent the dependence on I_1 and I_2 respectively. These vectors have four elements, one for each training datapoint, and thus only four vectors [$\hat{n}_{00}$, $\hat{n}_{01}$, $\hat{n}_{10}$, and $\hat{n}_{11}$] are required to make a complete basis. Each $\hat{n}_{jk}$ is summarized graphically in Fig. 3C and detailed in *Orthonormal Basis Construction*.

To characterize the evolution of the network, we calculate $|\mathcal{E}_{jk}(t)| = |\hat{n}_{jk} \cdot \vec{E}(t)|$ over time, that is, projecting the error onto this basis. Note that $\vec{E}(t) = \vec{O}(t) - \vec{L}$, also with vectors of length four. In the first stage of training (until approximately 0.15 s), the network primarily corrects the mean $\mathcal{E}_{00}$. Slight increases to the first order error $\mathcal{E}_{01}$ and $\mathcal{E}_{10}$ are made during this period, but are later corrected. The lone nonlinear mode $\mathcal{E}_{11}$ is the last to adjust, accompanying the final steep drop in error, and the desired checkerboard output table in Figs. 4B, right side. Note that adjustments to this nonlinear mode emerged last, despite its larger contribution than the linear modes $\mathcal{E}_{01}$ and $\mathcal{E}_{10}$.

A visualization of the network's evolution over training is available as supplementary video 1.

Next, as a more complex task, we let our network train for nonlinear regression to the eight datapoints shown as black circles in Fig. 4A. We impose constant voltages on two nodes of the network $I_- = 0$ V and $I_+ = 0.45$ V, and choose single nodes for our input I and output O. At time zero, the initial fit is a line. During training, we cycle through datapoints in order (a random ordering gives identical results). In the first 4 s of training

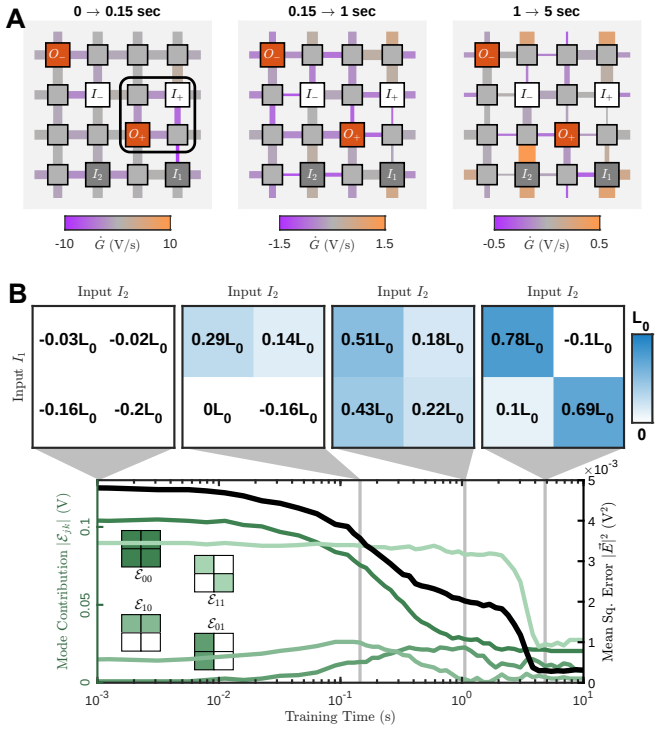

FIG. 3. **Learning XOR** a) Schematic of the network showing input and output node locations and average rate of change in G (edge color) for three time spans during training. Edge widths correspond to average conductance. The boxed region in the first panel highlights changes associated with altering the average output (reducing $|\mathcal{E}_{00}|$). b) Network output O plotted as a function of inputs I_1 and I_2 in a truth-table format for four points during training. Color corresponds to output value. c) Mean squared error (black) and error contributions broken down by mode $|\mathcal{E}_{jk}|$ (green) over time. Error modes are depicted graphically next to lines. Time points indicated in (b) are denoted by vertical gray bars.

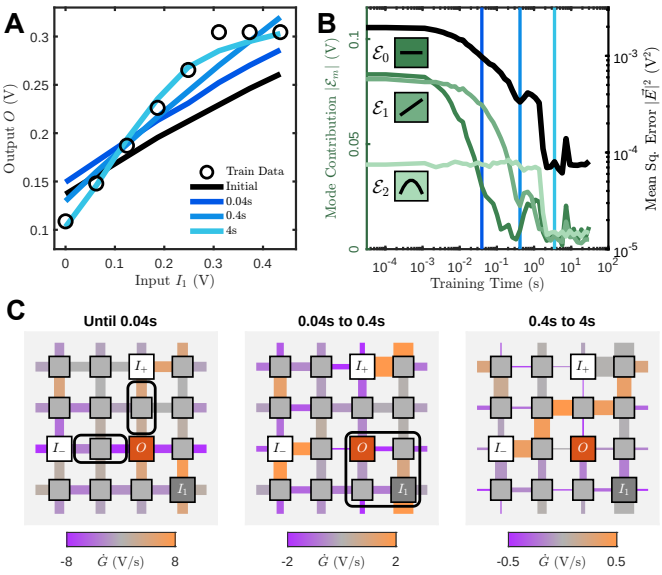

FIG. 4. **Nonlinear Regression** a) Initial output (black line) and outputs at four points throughout training (blue lines) are shown versus variable input I_1 value. Training labels L (black circles) are overlaid. b) Mean squared error of all datapoints (black) and error contributions broken down by mode $|\mathcal{E}_m|$ (green) over time. Time points indicated in (a) are shown as vertical blue bars. c) Schematic of the network showing input and output node locations and average rate of change in G (color) for the three time spans during training defined by the blue lines in (a) and (b). Edge widths of the network correspond to average conductance of the edges. The boxed regions in the first two panels indicate interpretable changes discussed in the main text.

the network solves the task, as shown by sequential improvement of the the blue output curves in Fig. 4A. This progress follow an interesting pattern: first the mean is adjusted by shifting the line, then the slope is adjusted, and finally the line is bent. That is, the system learns the modes of the data in order just like for the XOR task.

To tease this out quantitatively we again construct an orthonormal basis $\hat{n}_m$, this time with eight elements, like the training data. We focus only on the first three vectors, $\hat{n}_0$, $\hat{n}_1$, and $\hat{n}_2$, as we find higher modes negligible for this task. The construction of these vectors is detailed in Appendix under *Orthonormal Basis Construction*, but their shapes can be simply summarized as flat, linear, and parabolic when their elements are plotted corresponding input values of the training data (*e.g.* $(\hat{n}_1)_i = A(I_1)_i + B$). As shown in Fig. 4B, error in this basis $|\mathcal{E}_m|$ decreases in mode order: constant $\mathcal{E}_0$, then linear $\mathcal{E}_1$, then parabolic $\mathcal{E}_2$. These changes can be visually confirmed by noting the output shapes in Fig. 4A, whose times are plotted as vertical blue bars in Fig. 4B. After these modes decrease, the mean squared error $|\vec{E}(t)|^2$ plateaus.

We gain further insight by directly inspecting the behavior of individual edges of the system during training. In the first 0.04 s of training time, the network primarily decreases the output mean ($\mathcal{E}_0$), accomplished by strengthening the output's connection to I_+ while weakening its connection to I_-, as highlighted in the first panel of Fig. 4C. From 0.04 to 0.4 s of training time, the network reduces the linear error contribution ($\mathcal{E}_1$), by increasing the slope of its output. This is accomplished not by simply increasing the connection between I_1 and the output (highlighted in Fig. 4C middle panel), but rather by increasing this connection *relative* to the connections to the other inputs (I_+ and I_-), which both decrease during this period. As mentioned previously, edges become more nonlinear as their gate voltage, and thus conductance, shrinks (Eq 1). Therefore, by decreasing all input-output connections, the network positions positioned itself to curve the output as a function of the variable input I_1, which it does during the third highlighted section (Fig. 4C right panel).

For this and all tasks shown in this work, the network is overparameterized, and therefore has access to many directions in $\vec{G}$ space that do not affect the output. In

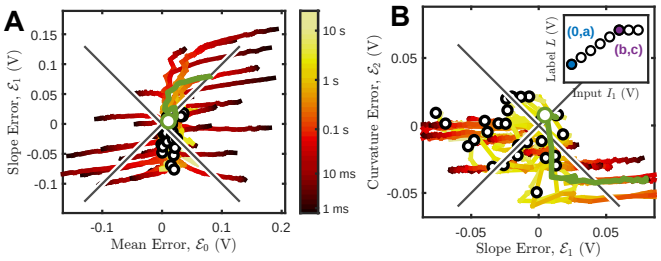

FIG. 5. **Error Mode Evolution for Many Tasks** a) Signed error contributions from the lowest two modes $\mathcal{E}_0$ and $\mathcal{E}_1$ for 29 unique regression tasks during training. Input values and network setup are identical to Fig. 4. Color indicates training time, and traces end at hollow black circles. Gray diagonal lines represent $|\mathcal{E}_0| = |\mathcal{E}_1|$. Green line corresponds to the experiment shown in Fig. 4. b) Same as (a) but for linear $\mathcal{E}_1$ and curvature $\mathcal{E}_2$ modes. Inset: training data for each of these 29 tasks is 8 datapoints in a piece-wise linear form like the one shown here. Parameters a, b, c are varied (see text) to produce a range of initial error modes.

practice, this means that the system is pushed along these dimensions by *any* imperfection in the update circuitry, no matter how small. We find this bias to vary between edges and for individual edges between tasks, belying a voltage-state dependence. At long times (> 1 s), the result is often many extreme gate voltage values (high and low) as shown in Fig. 4C, and difficulty interpreting specific changes to the network. While the network can and does find solutions in spite of these imperfections, bias sets an inevitable error floor. It is important to note however, that because each edge evolves according to its own local update rules, these errors do not compound. That is, unlike differentiating an entire simulation of a physical system, they do not require tighter and tighter precision as the system scale increases.

We find this sequential mode reduction picture to be consistent across tasks. To demonstrate this, we compare the error mode contributions to each other over time for 29 different nonlinear regression tasks, as shown in Fig 5A ($\mathcal{E}_0$ vs $\mathcal{E}_1$) and 5B ($\mathcal{E}_1$ vs $\mathcal{E}_2$), with the same initialization of the circuitry. Each task is piece-wise linear with 8 datapoints and the same node setup and initial conditions as Fig. 4A (the data from Fig. 4A is highlighted in green). The tasks all share the common features of a positive linear slope from $(0, a)$ to (b, c), and a flat line afterwards, as detailed in 5B inset. The tasks included each have a unique parameter combination, within the ranges of a = [0.07, 0.24] V, c = [0.13, 0.30] V, and with b taking one of two values 0.12 V or 0.31 V. These parameters were chosen to vary the initial mode error contributions, while keeping $\mathcal{E}_0 < 0.2$ V to allow entire trajectories to be shown on a single plot. Note that $c > a$ for all tasks to ensure a positive slope.

When nearly any zero mode $\mathcal{E}_0$ is present, its magnitude is decreased first, as shown by the shape of the trajectories in Fig. 5A. The linear error mode $\mathcal{E}_1$ is then shrunk, followed by the curvature contribution $\mathcal{E}_2$, as shown in Fig. 5B. In this single-output configuration, the network is limited in its functionality (no inhibitory connections using O_-), and thus tasks the network can accomplish have small $\mathcal{E}_2$ contributions. Note that nonzero final error modes are at least in part caused by the aforementioned biases in each edge's learning rule evaluation.

DISCUSSION

To our knowledge, our learning metamaterial is the only physical implementation of a standalone system that can adaptively perform nonlinear supervised learning without a processor. Unlike previous implementations [26, 34, 35], this circuit solves nonlinear tasks, varies its learning degrees of freedom continuously, and implements the full (not approximate) Coupled Learning rule. Our system learns by tackling modes of the data in order, much as if it is peeling layers of an onion, similar to learning with ANNs [36]. In the infinitesimal $\eta \ll 1$ limit, Coupled Learning aligns well [27] with gradient descent, the algorithm used in much of deep learning. However, it is interesting that the system learns in mode order even for $\eta = 0.5$ or 1.

Our circuitry learns tasks adaptively in seconds, and performs them power-efficiently. Training times reported in this work are integrated time spent learning (unfrozen). An approximately equal amount of time is spent applying voltages, but this can be easily reduced to a negligible factor with modestly better control hardware (*e.g.* 1 MHz versus the current ≈100 kHz, see *Supervisor and Measurement Circuitry* in Appendix). Future versions may also realize large speed gains by shortening adjustable timescales of the system ($t_h = 100$ μs, $\tau_0 = 18$ ms). Even without these changes, our 32-component metamaterial is 10× faster than our Python simulation of nonlinear Coupled Learning. When performing the trained tasks in this work, our system dissipates approximately 10-20 pJ across each edge (See *Power Dissipation* in Appendix). This is a calculation from measured voltage and conductance values and therefore a useful lower bound. Strikingly, this bound is already competitive with some of the most efficient digital supercomputers performing inference on feed-forward ANNs, at approximately 15 pJ per parameter. Even at 32 edges, the system's speed, power efficiency, and damage tolerance make it potentially useful for on-edge computing applications like smart sensors and medical devices.

Our circuitry's design has several features that make scalability a realistic possibility. First, because physics performs the forward computation (Eq. 2), and updates (evolution) are calculated by each element in parallel (Eq. 5), the duration of a training step is independent of system size, provided $\tau_V \ll t_h$. By contrast, for ANNs, including neuromorphic implementations [10–12, 14, 16–18] and hybrid analog-digital pairings [19, 25], memory requirements and training step duration scale with system size. Second, our design uses standard circuitry

components amenable to microfabrication, allowing large system sizes ($\sim 10^7$ edges in ~ 10 mm^2 with well-established 180 nm CMOS technology [26]) and reduction of equilibration timescale τ_V. Third, unlike centralized computing devices, our system would not require individually-addressable components, and supervising the system does not get more complex with scale. Finally, it is robust to damage [26], so it is also fault tolerant. By contrast, manufacturing error is a major impediment to scaling up top-down processor-based chips for other neuromorphic hardware realizations [6].

Because of its scalability, low power, and speed, future versions of nonlinear learning metamaterials may compete with artificial neural networks for some applications. However, many interesting issues need to be addressed to reach that stage. For example, while the nonlinear ReLU activation function is known to suffice for ANNs, it remains unclear what kind of nonlinearity in the IV characteristics of the resistive edges is needed in general, much less what kind is optimal, for accomplishing nonlinear tasks with Coupled Learning. Moreover, for these fully recurrent networks, the optimal network topology for a broad range of supervised tasks remains to be determined; the highly-connected layered structure so important for backpropagation in feed-forward networks may not be needed, but the connectivity of the square lattice used in our system is also likely too low.

From a fundamental point of view, our learning metamaterial provides a unique opportunity for studying emergent learning. In comparison to biological systems including the brain, our system relies on simpler, well-understood dynamics, is precisely trainable, and uses simple modular components. Our current system has already been used to characterize a competition between energy use and precision [37], a biologically-relevant trade-off, used for example by the neocortex during starvation in mice [38]. Learning metamaterials may also be more explainable than ANNs. Their properties are sculpted by the interaction of the training process with physical constraints (*e.g.* power minimization), leaving physical signatures of learning [39, 40] that give valuable microscopic insight into emergent learning.

While the collective behavior of many-body systems differs fundamentally from the behavior of one body, the behavior of *very many* bodies is generally similar to that of *many* bodies. This is what makes simulation of relatively small systems useful for understanding condensed matter. Brains are a form of matter where the behavior of *very many* is extremely different from that of *many*. This can be seen simply by comparing the cognitive abilities of *C. elegans* (302 neurons) to those of humans (10^{11}). *Very many* differs from *many* because the number of learning degrees of freedom (*e.g.* synaptic strengths) increases with the number of bodies (neurons). Our nonlinear learning metamaterial shares this property: the number of learning degrees of freedom (gate voltages) scales with the number of bodies (network edges). The speed and scalability of learning metamaterials imply that we may someday be able to study emergent physical learning on scales from *many* to *very many*, a feat that is impossible *in silico*.

ACKNOWLEDGMENTS

S.D. would like to thank Kieran Murphy for helpful discussions. This research was supported by the National Science Foundation via the UPenn MRSEC/DMR-1720530 and MRSEC/DMR-DMR-2309043 (S.D. and D.J.D.), and DMR-2005749 (A.J.L.), the Simons Foundation # 327939 (A.J.L.), and the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering award DE-SC0020963 (M.S.). S.D. acknowledges support from the University of Pennsylvania School of Arts and Sciences Data Driven Discovery Initiative. M.Z.M. acknowledges support from the Packard Foundation. D.J.D. and A.J.L. thank CCB at the Flatiron Institute, as well as the Isaac Newton Institute for Mathematical Sciences under the program "New Statistical Physics in Living Matter" (EPSRC grant EP/R014601/1), for support and hospitality while a portion of this research was carried out.

[1] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, Neural Networks **2**, 359 (1989).

[2] H. Nguyen and H. La, Review of Deep Reinforcement Learning for Robot Manipulation, in *2019 Third IEEE International Conference on Robotic Computing (IRC)* (2019) pp. 590–595.

[3] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (2022), arxiv:2207.02696 [cs].

[4] J. A. Baktash and M. Dawodi, Gpt-4: A Review on Advancements and Opportunities in Natural Language Processing (2023), arxiv:2305.03195 [cs].

[5] V. Balasubramanian, Brain power, Proceedings of the National Academy of Sciences **118**, e2107022118 (2021).

[6] D. V. Christensen, R. Dittmann, B. Linares-Barranco, A. Sebastian, M. L. Gallo, A. Redaelli, S. Slesazeck, T. Mikolajick, S. Spiga, S. Menzel, I. Valov, G. Milano, C. Ricciardi, S.-J. Liang, F. Miao, M. Lanza, T. J. Quill, S. T. Keene, A. Salleo, J. Grollier, D. Marković, A. Mizrahi, P. Yao, J. J. Yang, G. Indiveri, J. P. Strachan, S. Datta, E. Vianello, A. Valentian, J. Feldmann, X. Li, W. H. P. Pernice, H. Bhaskaran, S. Furber, E. Neftci, F. Scherr, W. Maass, S. Ramaswamy, J. Tapson, P. Panda, Y. Kim, G. Tanaka, S. Thorpe, C. Bartolozzi, T. A. Cleland, C. Posch, S. Liu, G. Panuccio, M. Mahmud, A. N. Mazumder, M. Hosseini, T. Mohs-

enin, E. Donati, S. Tolu, R. Galeazzi, M. E. Christensen, S. Holm, D. Ielmini, and N. Pryds, 2022 roadmap on neuromorphic computing and engineering, Neuromorphic Computing and Engineering **2**, 022501 (2022).

[7] B. Sengupta and M. B. Stemmler, Power Consumption During Neuronal Computation, Proceedings of the IEEE **102**, 738 (2014).

[8] H. Jaeger, B. Noheda, and W. G. van der Wiel, Toward a formal theory for computing machines made out of whatever physics offers, Nature Communications **14**, 4911 (2023).

[9] C. Mead, Neuromorphic electronic systems, Proceedings of the IEEE **78**, 1629 (1990).

[10] S. Kim, C. Du, P. Sheridan, W. Ma, S. Choi, and W. D. Lu, Experimental Demonstration of a Second-Order Memristor and Its Ability to Biorealistically Implement Synaptic Plasticity, Nano Letters **15**, 2203 (2015).

[11] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication, in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (2016) pp. 1–6.

[12] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, Fully hardware-implemented memristor convolutional neural network, Nature **577**, 641 (2020).

[13] S. K. Vadlamani, T. P. Xiao, and E. Yablonovitch, Physics successfully implements Lagrange multiplier optimization, Proceedings of the National Academy of Sciences **117**, 26639 (2020).

[14] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, Survey of Machine Learning Accelerators, in *2020 IEEE High Performance Extreme Computing Conference (HPEC)* (2020) pp. 1–12, arxiv:2009.00993 [cs].

[15] S. Dutta, A. Khanna, A. S. Assoa, H. Paik, D. G. Schlom, Z. Toroczkai, A. Raychowdhury, and S. Datta, An Ising Hamiltonian solver based on coupled stochastic phase-transition nano-oscillators, Nature Electronics **4**, 502 (2021).

[16] R. Wang, T. Shi, X. Zhang, J. Wei, J. Lu, J. Zhu, Z. Wu, Q. Liu, and M. Liu, Implementing in-situ self-organizing maps with memristor crossbar arrays for data mining and optimization, Nature Communications **13**, 2289 (2022).

[17] J. Laydevant, D. Markovic, and J. Grollier, Training an Ising Machine with Equilibrium Propagation (2023), arxiv:2305.18321 [quant-ph].

[18] T. Mei and C. Q. Chen, In-memory mechanical computing, Nature Communications **14**, 5204 (2023).

[19] L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. T. Schachter, Z. Hu, and P. L. McMahon, Deep physical neural networks trained with backpropagation, Nature **601**, 549 (2022).

[20] X. Wang, X. Lin, and X. Dang, Supervised learning in spiking neural networks: A review of algorithms and evaluations, Neural Networks **125**, 258 (2020).

[21] M. Stern and A. Murugan, Learning Without Neurons in Physical Systems — Annual Review of Condensed Matter Physics (2023).

[22] N. Pashine, D. Hexner, A. J. Liu, and S. R. Nagel, Directed aging, memory, and nature's greed, Science Advances **5**, 4215 (2019).

[23] N. Pashine, Local rules for fabricating allosteric networks, Physical Review Materials **5**, 065607 (2021).

[24] C. Kaspar, B. J. Ravoo, W. G. van der Wiel, S. V. Wegner, and W. H. P. Pernice, The rise of intelligent matter, Nature **594**, 345 (2021).

[25] J. B. Hopkins, R. H. Lee, and P. Sainaghi, Using binary-stiffness beams within mechanical neural-network metamaterials to learn, Smart Materials and Structures **32**, 035015 (2023).

[26] S. Dillavou, M. Stern, A. J. Liu, and D. J. Durian, Demonstration of Decentralized Physics-Driven Learning, Physical Review Applied **18**, 014040 (2022).

[27] M. Stern, D. Hexner, J. W. Rocks, and A. J. Liu, Supervised Learning in Physical Networks: From Machine Learning to Learning Machines, Physical Review X **11**, 021045 (2021).

[28] B. Scellier and Y. Bengio, Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation, Frontiers in Computational Neuroscience **11**, 10.3389/fncom.2017.00024 (2017).

[29] J. R. Movellan, Contrastive Hebbian Learning in the Continuous Hopfield Model, in *Connectionist Models*, edited by D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (Morgan Kaufmann, 1991) pp. 10–17.

[30] F. Rosenblatt, *The perceptron - A perceiving and recognizing automaton*, Tech. Rep. 85-460-1 (Cornell Aeronautical Laboratory, Ithaca, New York, 1957).

[31] J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, Training End-to-End Analog Neural Networks with Equilibrium Propagation, arXiv:2006.01981 [cs] (2020), arxiv:2006.01981 [cs].

[32] B. Scellier, M. Ernoult, J. Kendall, and S. Kumar, Energy-Based Learning Algorithms: A Comparative Study, in *ICML Workshop on Localized Learning (LLW)* (2023).

[33] M. Drouhin, C. Turck, K.-E. Harabi, A. Renaudineau, T. Bersani-Veroni, E. Vianello, J.-M. Portal, J. Grollier, and D. Querlioz, Nanoelectronic implementation of Equilibrium Propagation, in *Emerging Topics in Artificial Intelligence (ETAI) 2023*, Vol. PC12655 (SPIE, 2023) p. PC1265507.

[34] J. F. Wycoff, S. Dillavou, M. Stern, A. J. Liu, and D. J. Durian, Desynchronous learning in a physics-driven learning network, The Journal of Chemical Physics **156**, 144903 (2022).

[35] M. Stern, S. Dillavou, M. Z. Miskin, D. J. Durian, and A. J. Liu, Physical learning beyond the quasistatic limit, Physical Review Research **4**, L022037 (2022).

[36] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, On the Spectral Bias of Neural Networks, in *Proceedings of the 36th International Conference on Machine Learning* (PMLR, 2019) pp. 5301–5310.

[37] M. Stern, S. Dillavou, D. Jayaraman, D. J. Durian, and A. J. Liu, Physical learning of power-efficient solutions (2023), arxiv:2310.10437 [cond-mat].

[38] Z. Padamsey, D. Katsanevaki, N. Dupuy, and N. L. Rochefort, Neocortex saves energy by reducing coding precision during food scarcity, Neuron **110**, 280 (2022).

[39] J. W. Rocks, A. J. Liu, and E. Katifori, Revealing structure-function relationships in functional flow networks via persistent homology, Physical Review Research **2**, 033234 (2020).

[40] M. Stern, A. J. Liu, and V. Balasubramanian, The Physical Effects of Learning (2023), arxiv:2306.12928 [cond-

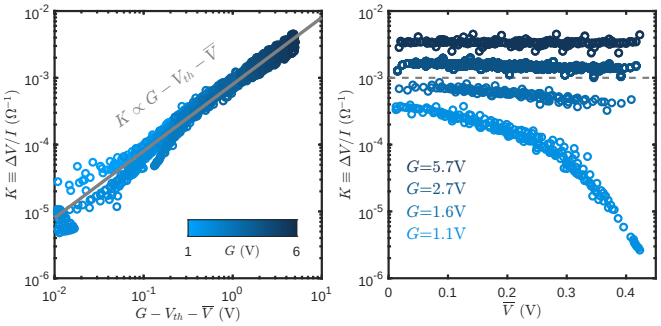

FIG. 6. **Transistors as Nonlinear Resistors** Left: Transistor source-drain conductance K as a function of gate voltage G, threshold voltage $V_{\text{th}} \approx 0.7$ V, and average of source and drain voltages $\overline{V}$. Color is gate voltage, and the gray line is a guide for the eye of constant proportionality. Right: Transistor source-drain conductance as a function of average of source and drain voltages. Color represents gate voltage, with the same scale as left side. The dashed line represents a 1 kΩ linear resistor, which has the same conductance regardless of the voltages it experiences.

mat].
[41] https://www.top500.org/lists/green500/2023/06/.

Appendix A: Transistors as Nonlinear Resistors

In the Ohmic (passive) regime, the voltage drop between the source and drain ΔV on our N-channel enhancement MOSFETs (ALD1106PBL) has proportional dependence $\Delta V \propto (G - V_{\text{th}} - \overline{V})I$, where G is the gate voltage, I is the current between source and drain, $V_{\text{th}} \approx 0.7$ V is the threshold voltage inherent to the transistors, and $\overline{V}$ is the average voltage of the source and drain, or equivalently V_1 and V_2 in Fig. 1A, or V^+ and V^- in Fig. 6, left side. Because our MOSFETs' bodies are tied to ground, there is no distinction between source and drain. We define the conductance across the source-drain $K \equiv \Delta V/I$, voltage drop divided by current.

Transistor conductances are measured by imposing a gate voltage, as well as a voltage drop across the transistor source-drain in series with a known linear resistor, ranging from $R = 100\ \Omega$ to 100 MΩ. Current across the transistor is calculated as the voltage drop across the resistor divided by its resistance. Results of these tests demonstrate a clear proportionality, as shown in Fig. 6, left. They further demonstrate nonlinearity, which comes in the form of $\overline{V}$ dependence, shown in Fig. 6, right. In our system, nonlinearity is coupled with conductance, and thus the network must push relevant edges into a low-conductance regime to solve nonlinear tasks.

Appendix B: Twin Edge Circuitry

The circuitry for a twin edge (the fundamental unit of our system) is described and shown schematically and in an image in Fig. 7. The learning timescale $\tau_0 \approx 18$ ms is a function of circuit components and of the voltage ranges allowed for nodes $V_+ = 0.45$ V and gates $G_+ = 5$ V,

$$\tau_0 \equiv \frac{G_+}{V_+^2} V_0 R_0 C_0 \approx 18 \text{ ms} \tag{B1}$$

where $R_0 = 100\ \Omega$, $C_0 = 22\ \mu F$, and V_0 is set by the gain in the node measurement amplification, a factor of 2 for averaging, and the analog multiplier (see Fig. 7):

$$V_0 = \left(\frac{10\text{ k}\Omega}{10\text{ k}\Omega + 100\text{ k}\Omega} \times 2\right)^2 \times V_{\otimes} \approx 0.33 \text{ V} \tag{B2}$$

Where $V_{\otimes} = 10$ is a property of the analog multiplier.

Appendix C: Power Dissipation

For the XOR task in Fig. 3, the final equilibrium state power dissipation, calculated as $K\Delta V^2$ across each edge, is 151 μW. For the 29 tasks in Fig. 5, the value is 345±72 μW. These equate to approximately 5 or 10 μW per edge, respectively. Note that we have not included other factors such as the power required to run the voltage application hardware, or the capacitance of breadboards and wires, but this is still a useful lower bound. Voltages in our circuit equilibrate in approximately 4 μS or less, certainly slowed by our voltage application hardware (see Fig. 2). As dissipation is low at the start of equilibration, we estimate energy cost by assuming 2 μS of equilibrium-level power dissipation. This leads to 10 or 20 pJ of dissipation per edge for these tasks, respectively.

As a reference point, we calculate the power-efficiency of running a standard feed-forward neural network on the most efficient supercomputer on the Green500 list[41]. Using the quoted value ∼65 GigaFLOPS/Watt = 65 GigaFLOP/Joule we find 15 pJ per FLOP. Feed-forward neural networks perform inference using of order 1 FLOP/parameter, resulting in approximately 15 pJ per parameter. Note that digital supercomputers use at least 32 bits for each parameter, far greater than our measurement hardware's capability of 8. However, because G values are adjusted based on continuous analog feedback, they are certainly more precise than we can measure with our current setup; we sometimes see improvements in final output without measurable changes in G values for just this reason. 32 bit precision is certainly beyond this system's reach, however, as that would require accuracy down to single nano-Volts.

Appendix D: Supervisor and Measurement Circuitry

The role of supervisor for our system is quite simple, as the learning metamaterial itself is doing all of the updating. The only supervisor responsibilities are to apply input voltages I to both networks, enforce outputs O^C in the clamped network, and unfreeze learning for

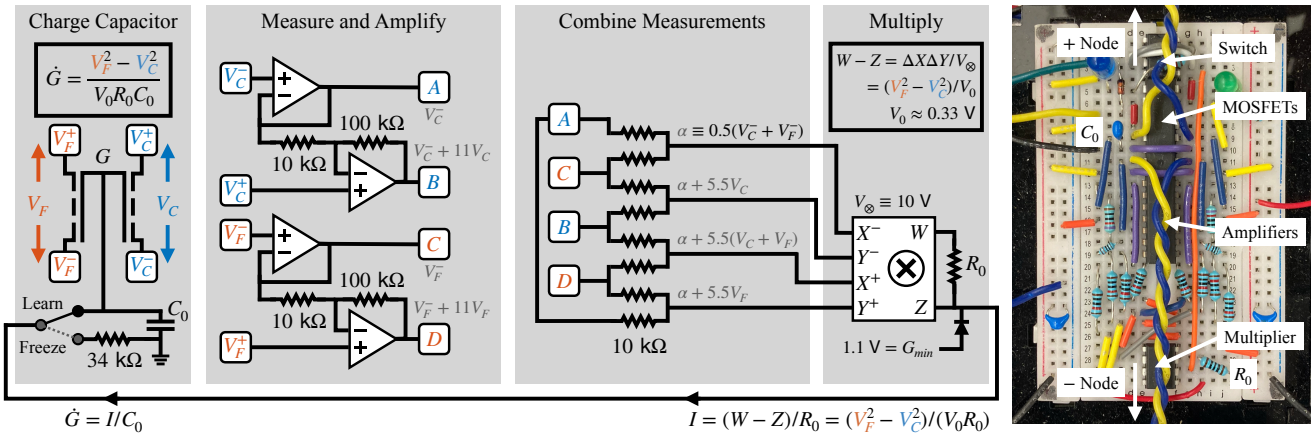

FIG. 7. **Twin Edge Circuitry** Schematic and image of the circuitry that implements the learning rule, which is the entirety of the operational elements of each twin edge (informational LEDs and power supplies are excluded). Starting from the left, The twin edge is comprised of two N-channel enhancement MOSFETS (ALD1106PBL) with body tied to ground (not shown). The output of the learning circuitry is a current which is fed into a charging capacitor $C_0 = 22\ \mu$F, that holds gate voltage G. A switch (MAX322CPA+) can freeze the learning by instead shunting this current to ground. To start the calculation process, the voltages at the adjacent nodes in both networks are measured (one is arbitarily designated + and the other −). The free and clamped measurements are combined and amplified using operational amplifiers (TLV274IN), then averaged in pairs and multiplied together using an analog multiplier (AD633ANZ). Note that $V_\otimes \equiv 10$ V is a property of the multiplier circuitry, and $V_0 \approx 0.33$ V is an amalgamation of factors from all stages of the circuitry (see Eq. B2). The multiplier output is wired to produce a current $(W - Z)/R_0$, completing the loop. $R_0 = 100\ \Omega$. Note that gate voltages are prevented from decreasing indefinitely using $G_{\min} = 1.1$ V and a diode (1N4148). The image on the right shows a physical edge, with each integrated circuit, charging capacitor, charging resistor, and adjacent nodes labeled.

$t_h = 100\mu$s. Then to repeat this cycle with each datapoint in turn, until a pre-specified number of steps. While not required for operation, we also measure the internal state of the network during the training process. At prespecified test steps, the network is frozen, and input sets (datapoints) are applied to the network one at a time, and their outputs O measured. Optionally, the node voltages in both networks and the gate voltages G are also measured.

Because we needed rapid application and measurement of many voltage values, we constructed custom digital-to-analog (DAC) and analog-to-digital (ADC) circuitry, and control the entire process using an Arduino Due. While faster than off-the-shelf solutions, the clock speed of the Arduino (84 MHz) and the speed with which we can apply voltages (≈ 100 kHz) are speed-limiting factors in our system, and both are far from state of the art.

Using custom code on both ends, experiments are designed and parameters (e.g. nudge amplitude η, train step time t_h, total training steps, etc), are chosen in MATLAB, fed to the Arduino via the serial USB port. Once data has passed to the Arduino, it runs an experiment, occasionally passing data back to MATLAB through the serial port, but receiving no further instructions. All applied and measured voltages are 8-bit (0-255), meaning our least significant bit (LSB) for applying voltages to the network is $V_+/256 \approx 1.8$ mV. Inputs I and Labels L are passed with a serial digital signal to a shift register (SN74HC595N), then to a digital-to-analog converter (DAC) (TLC7226CN). The DAC converts signals into a 5V range, which are then buffered with an amplifier (TLV274IN) wired to have 1/11 fractional gain, giving us our voltage range for the circuit ($V_+ \approx 5V/11$). Our hardware has 8 individually addressable analog voltages like the one just described. These 8 can be applied in parallel (but are fed in sequentially), which sets the restriction input nodes + output nodes ≤ 8. In this work no more than 6 total were used. Inputs are applied directly to the nodes of the network with wires changed by hand for each experiment.

Clamping outputs requires a weighted average of the free network output, and the desired label (Eq. 5). For a single output node (non-differential) scheme, a measurement wire is connected to the output node in the free network, measuring O. This is fed through a unity gain buffer (TLV274IN) onto one end of a 10 kΩ digital potentiometer (AD5220 10k), which is a digitally-adjustable voltage divider. The label L from the DAC is enforced at the other end. The voltage in the middle of this voltage divider is now between O and L, creating the nudge. It is fed through another unity gain buffer (TLV274IN) and onto the output node in the clamped network. Prior to an experiment, the Arduino digitally sets η using the potentiometer, and this feedback circuit automatically and continuously enforces Eq. 5. Using a differential output scheme has one additional step. First, a label voltage for each individual node ($L_- \to O_-^C$ and $L_+ \to O_+^C$) is produced using multiple amplifiers (TLV274IN) wired as summing amplifiers. These labels are $L_\pm = (O_+ + O_-)/2 \pm L/2$, equivalent to Eq. 6 with

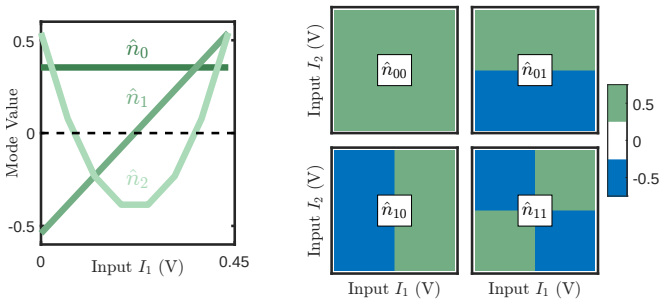

FIG. 8. **Orthonormal Modes** Left: Visualization of the modes for one variable input with eight equally-spaced input values between 0 V and 0.45 V (Fig. 4 and 5). Right: Visualization for two variable inputs, each with value 0 V or 0.45 V (Fig. 3). Construction is described in the text.

$\eta = 1$. Each label $L_\pm$ is then into its own potentiometer circuit described above, with the appropriate free network output $O_\pm$ on the other end, and the result is buffered and enforced at the clamped network output, reproducing the nudge with the desired η parameter Eq. 6.

Note that all of the clamping above is done continuously through feedback during the experiment; the Arduino only supplies the labels L at each step. Freezing and unfreezing the network is accomplished via a single binary signal that reaches every edge, and is generated using a digital switch (MAX322CPA+). This set of operations up to this point is sufficient to train the system - output measurement by the Arduino is not even required.

To measure the outputs and the state of the system, we use unity gain buffers (TLV274IN) attached to every node in both networks (16+16) and every gate capacitor (32). These 64 values are fed into four quadruple 4-to-1 demultiplexers (MAX396CPI+) ($4^3 = 64$), then further demultiplexed 2-to-1 using switches (MAX322CPA+). The Arduino can thus select a single node and a single gate capacitor to measure using a 5-bit signal ($2^5 = 32$). The selected node value is buffered and amplified 11x (TLV274IN), whereas the selected gate capacitor is buffered and de-amplified by 2x (TLV274IN). These signals are converted to 8-bit digital using an ADC (TLC0820ACN) and fed into a shift register (CD74HCT597E), which outputs a serial digital signal read by the Arduino. The least significant bit (LSB) measuring the nodes is identical to the voltage application LSB ($V_+/256 \approx 1.8$ mV), and the LSB for gate voltage G measurement is $10V/256 \approx 40$ mV. Measurements are repeated and averages taken to lower this resolution, but the bound remains close to these values.

Appendix E: Orthonormal Basis Construction

The orthonormal basis $\hat{n}_m$ used to analyze how our system learns in Fig. 4 and 5 is constructed in order as follows. We take $\vec{I}_1$ to be the vector of all values applied at the variable input node in the training set, which for these two figures is 8 evenly spaced values ranging from 0 to $V_+ = 0.45$ V. Let $\vec{I}_1^k$ be equal to the same vector with each element raised to the k^{th} power. Then, for each mode m in order,

$$\vec{n}_m = \vec{I}_1^m - \left(\sum_{k=0}^{m-1} (\vec{I}_1^k \cdot \hat{n}_k) \hat{n}_k \right) \tag{E1}$$

$$\hat{n}_m = \vec{n}_m / |\vec{n}_m| \tag{E2}$$

Which results in an orthonormal basis. These basis vectors are shown plotted against $\vec{I}_1$ in Fig. 8 in green.

For the two-input case of XOR, we define $\vec{I}_{12}^{jk}$ the vector whose i^{th} element is the i^{th} element of $\vec{I}_1^j$ and of $\vec{I}_2^k$ multiplied together. We then have a similar construction,

$$\vec{n}_{jk} = \vec{I}_{12}^{jk} - \left(\sum_{J=0}^{J-1} \sum_{K=0}^{K-1} (\vec{I}_{12}^{JK} \cdot \hat{n}_{JK}) \hat{n}_{JK} \right) \tag{E3}$$

$$\hat{n}_{jk} = \vec{n}_{jk} / |\vec{n}_{jk}| \tag{E4}$$

Where any parallel components to lower modes are subtracted away. The modes used in Fig. 3 are plotted as heatmaps in Fig. 8. Because only four datapoints are used in this case, four modes (00, 01, 10, 11) form a complete basis.