


## Demonstration of Decentralized Physics-Driven Learning

Sam Dillavou<sup>1</sup>,\* Menachem Stern<sup>1</sup>, Andrea J. Liu<sup>1</sup>, and Douglas J. Durian<sup>1</sup>

*Department of Physics and Astronomy, University of Pennsylvania, 209 South 33rd Street, Philadelphia, Pennsylvania 19104, USA*

 (Received 11 January 2022; revised 7 March 2022; accepted 13 June 2022; published 18 July 2022)

In typical artificial neural networks, neurons adjust according to global calculations of a central processor, but in the brain, neurons and synapses self-adjust based on local information. Contrastive learning algorithms have recently been proposed to train physical systems, such as fluidic, mechanical, or electrical networks, to perform machine-learning tasks from local evolution rules. However, to date, such systems have only been implemented *in silico* due to the engineering challenge of creating elements that autonomously evolve based on their own response to two sets of global boundary conditions. Here, we introduce and implement a physics-driven contrastive learning scheme for a network of variable resistors, using circuitry to locally compare the response of *two* identical networks subjected to the two different sets of boundary conditions. Using this method, our system effectively trains itself, optimizing its resistance values without the use of a central processor or external information storage. Once the system is trained for a specified allostery, regression, or classification task, the task is subsequently performed rapidly and automatically by the physical imperative to minimize power dissipation in response to the given voltage inputs. We demonstrate that, unlike typical computers, such learning systems are robust to extreme damage (and thus manufacturing defects) due to their decentralized learning. Our twin-network approach is therefore readily scalable to extremely large or nonlinear networks, where its distributed nature will be an enormous advantage; a laboratory network of only 500 edges will already outpace its *in silico* counterpart.

DOI: [10.1103/PhysRevApplied.18.014040](https://doi.org/10.1103/PhysRevApplied.18.014040)

### I. INTRODUCTION

The confluence of ideas from neuroscience and machine learning has contributed immensely to our fundamental understanding of the nature of learning [1,2]. However, biological neural networks differ fundamentally from standard machine-learning algorithms in an important way [3,4]. A typical artificial neural network (ANN) requires a processing unit (e.g., a CPU) that trains the network by minimizing a global cost function [5], while repeatedly storing and retrieving information from a separate electronic memory. This von Neumann architecture is very successful but creates a severe computational bottleneck. In contrast, the brain and other biological networks [6,7] are more akin to extremely sophisticated and adaptive metamaterials: they are physical systems made of repeated locally responsive elements (e.g., neurons and synapses) that generate learning as a highly complex emergent property. This distribution and parallelization of computation and memory storage allows the human brain (approximately,  $10^{11}$  neurons and  $10^{14}$  synapses) to function at reasonable speeds despite signal propagation time scales millions of times slower than modern computational clock cycles. Furthermore, it allows the brain to recover from

massive damage [8] while consuming only modest power [9] compared to typical computers.

These advantages of the brain have spurred efforts to imitate its features [10–13]. Several of these have only been realized *in silico* [14–16] or in hybrid *in situ-in silico* form [17–19]. Actual laboratory realizations of “neuro-morphic” hardware that bypasses processors tend to mimic either standard machine-learning algorithms (e.g., back-propagation) [20–22] or phenomenological synaptic rules found in the brain (e.g., spike-timing-dependent plasticity) [23–27].

An alternative approach to learning without a processor is to exploit *physical processes* in tandem with simple *and* local rules [28]. Laboratory mechanical networks have been trained without any sort of processor to develop negative Poisson ratios using the process of “directed aging” [29,30], which exploits the natural physical tendency of a mechanical network to minimize elastic energy when a stress is applied. “Contrastive learning” [31] compares the response of the system to two different boundary conditions to adjust the degrees of freedom; this works more robustly than directed aging in laboratory mechanical networks [32] but has thus far required an external entity to enact these local rules. The “equilibrium-propagation” framework [15,33,34] can be viewed as combining the concept of directed aging with contrastive learning and

\*sam.dillavou@gmail.com

specifies simple local learning rules that, in principle, can be implemented in flow networks [15]. Equilibrium propagation nudges the network toward the desired target solution instead of imposing it directly; in the limit of infinitesimal nudges, the learning rule performs gradient descent on a loss function. A framework known as “coupled learning” [35] builds on equilibrium propagation, providing the foundation for our work. In both frameworks, although the learning rules are spatially local, they require simultaneous access to two distinct states of the same system. As a result, they are not *temporally* local and they require the use of memory when implemented *in silico*. This issue has thus far prevented them from being realized in the laboratory.

In this study, we report the laboratory realization of a physical learning machine composed of a pair of variable resistor networks. We resolve the highly restrictive and challenging requirement of contrastive learning in physical systems by using two identical twin networks to simultaneously measure responses of the “same” physical system to two different sets of boundary conditions. When we expose the system to training data, the physical imperative to minimize energy dissipation carries out the forward calculation to “compute” the outputs within nanoseconds, while local rules that adjust the resistances of the edges take the place of backpropagation, obviating the need for a processor or memory storage. We demonstrate that such a network can learn to perform and switch among a variety of tasks, including allostery, regression, and classification. Finally, we show that because the learning is fully distributed and each edge learns individually, the network functionality is highly robust to network changes and damage, making it readily scalable.

## II. APPROACH

In previous work, simulated and laboratory mechanical networks, and simulated flow networks, have been trained to perform desired tasks by adjusting their internal degrees of freedom [15,29,30,32,33,35–43]. This has been accomplished either by minimizing a global cost function [36–40] or by using local rules aided by an external processor [15,29,30,32,33,35,41–43]. Here, we consider a *self-adjusting* electronic network comprised of nodes connected by variable resistors, the values of which we call the “learning degrees of freedom.” When voltages  $\vec{V}^I$  are applied at input nodes, the voltages at designated output nodes  $\vec{V}^O$  are physically determined as functions of the input voltages and the resistance values  $\vec{R}$  of the network edges, as the system minimizes the total energy dissipation. The coupled-learning [35] framework for supervised learning specifies local evolution rules for how each resistance should evolve to produce desired output voltages. In doing so, the system exploits the physical processes that

govern the network to perform computation and implements contrastive learning as a spatially local rule, in a similar manner to equilibrium propagation [33].

In supervised learning, training examples determine the inputs  $\vec{V}^I$  as well as the *desired* output responses  $\vec{V}^D$  for each example. These desired output voltages can be achieved by adjusting the resistances of all the edges,  $\vec{R}$  (the *learning degrees of freedom*). During training, the  $\vec{R}$  are adjusted based on a comparison of two distinct electrical states imposed on the same network. In the *free* state, the network attempts the desired task: input voltages  $\vec{V}^I$  are applied and the network produces output voltages  $\vec{V}_F^O$ . In the *clamped* state, the same inputs  $\vec{V}^I$  are applied but voltages are also applied at the output nodes; those voltages are *clamped* at values  $\vec{V}_C^O$  closer to the desired values than  $\vec{V}_F^O$ :

$$\vec{V}_C^O = \eta \vec{V}^D + (1 - \eta) \vec{V}_F^O, \quad (1)$$

where  $0 < \eta \leq 1$  is the amplitude of the nudge toward the desired state.

When input voltage values  $\vec{V}^I$  are applied to the network, physical laws dictate current flow through the network, adjusting all other node voltages—which we call the “physical degrees of freedom.” These voltages naturally find a configuration that minimizes total energy dissipation (the “physical cost function”). Therefore, clamping the outputs nodes away from their “free” state toward the goal both requires additional power and creates a lower-error electrical state. Small adjustments to the learning degrees of freedom,  $\vec{R}$ , that lower the energy dissipation of the (higher-power) clamped state  $P^C$  relative to the (lower-power) free state  $P^F$  will create a new free-state equilibrium with output voltages that lie between the old free and clamped states. The determination of whether to increase or decrease each resistance requires only spatially local information, namely which state (free or clamped) has a higher energy dissipation (voltage drop) across the edge in question, allowing edges to update their own resistance. Similar to equilibrium propagation, this algorithm approximates global gradient descent in the limit  $\eta \ll 1$  [35], allowing a system to *train itself* by repeating this update process. However, this algorithm is not *temporally* local, in that it requires simultaneous access to the response for two distinct sets of boundary conditions which, by definition, cannot be imposed simultaneously. It is this requirement that makes contrastive learning in physical systems so challenging to realize.

Here, we resolve this conundrum by building two identical electrical networks to run the free and clamped states. We use digital variable resistors (see Appendix C) on each edge, which have 128 possible discrete resistance values.

The original (continuous) coupled-learning update rule is,

$$\Delta R_i = \frac{\gamma}{R_i^2} ([\Delta V_i^C]^2 - [\Delta V_i^F]^2), \quad (2)$$

where  $\gamma$  is a learning rate and  $\Delta V_i^C$  and  $\Delta V_i^F$  are the voltage drops in edge  $i$  of the clamped and free states, respectively. In our discrete-resistor networks, the two networks adjust their (identical) resistances according to an approximation of the original rule,

$$\Delta R_i^C = \Delta R_i^F = \begin{cases} +\delta R & \text{if } |\Delta V_i^C| > |\Delta V_i^F|, \\ -\delta R & \text{otherwise.} \end{cases} \quad (3)$$

equivalent to taking the sign of Eq. (2) multiplied by  $\gamma = \delta R$ . This now Boolean operation is carried out by integrated circuits housed on each edge of the network; the entire system is illustrated in Fig. 1(a) (for the details regarding the implementation of this rule, see Appendix C). Because the learning process is decentralized, our system functions without a central processor and training the network to perform a task is straightforward. The procedure is detailed in Fig. 1(b): apply the desired input voltages to the free and clamped networks, as well as clamped output voltages  $\vec{V}_C^O$  to the clamped network. Edge updates are triggered by a global clock and no further instructions to the edges are required, as each edge is responsible for its own evolution.

To demonstrate the operation of our learning elements, we train a two-edge network [Fig. 2(a)] as a voltage divider: we ask the network to produce a single desired voltage  $V^D$  at its output (middle) node, while the input nodes (top and bottom) are held at 5 V and 0 V, respectively. To train, the following algorithm is repeated every clock cycle:

- (1) Update the clamped state output-node voltage, per Eq. (1).
- (2) Every edge updates its own resistance, per Eq. (3).

In machine-learning language, the ‘‘supervisor’’ tells the network the right answer through the clamped boundary condition. The network itself decides *how* to achieve this answer, as it receives no external instructions about which edges to push up or down in resistance. That is, shown the right answer, the network *trains itself* to produce it. In this simple example, this distinction may seem trivial, but as we increase the size of the network, the job of the supervisor does not grow in complexity; it is always given by Eq. (1). This is in stark contrast to ANNs, where the number of gradient calculations grows rapidly with network size.

As previously described, edges modify their resistance to bias the electrical state of the system away from the free state and toward the clamped state. This results in the free-state output voltage(s) ‘‘following’’ the clamped

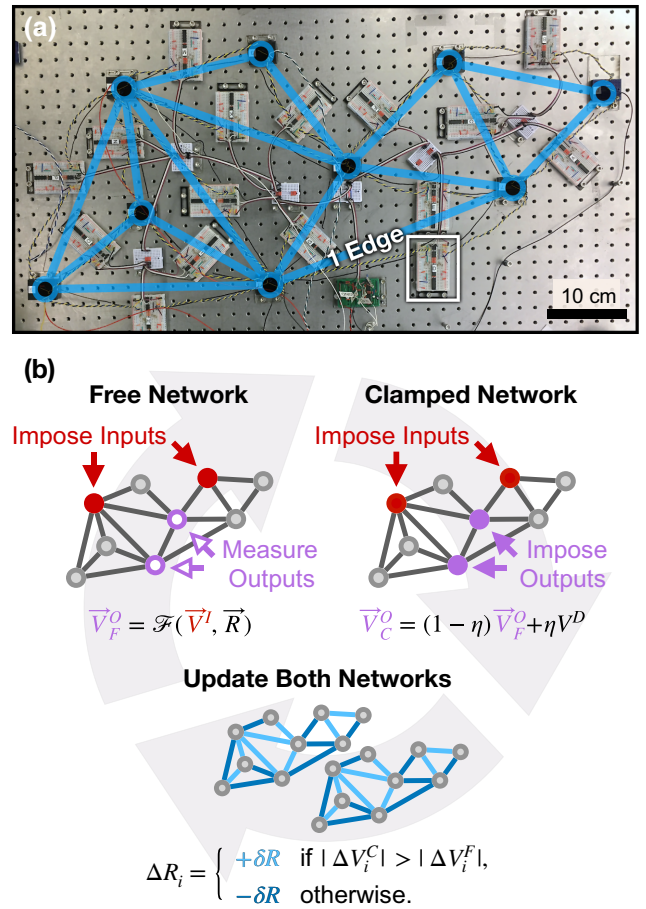


FIG. 1. A physics-driven learning machine (a). An image of the 16-edge circuitry, with the network structure overlaid in blue. Each breadboard, like the one highlighted in white, houses commensurate edges in the free and the clamped network (for the circuitry details, see Appendix C). (b) The procedure for training the learning machine. A supervisor (i) imposes voltages to the inputs (red) in the free network and (ii) to the inputs and outputs (purple) in the clamped network. The network (iii) updates its own resistances and  $\vec{V}_F^O$  is ‘‘calculated’’ by physical laws.

state voltages, which in turn move progressively toward the desired voltage [Fig. 2(b)]. In our voltage divider, the desired voltage is changed every 100 training steps. At the start, all edges are initialized at the center of their resistance ranges (approximately 50 k $\Omega$ ). Two phases in each training are evident. At first, the clamped and free networks are quite different and the two edges evolve in opposite directions until the desired voltage is achieved [Fig. 2(c)]. Once the network has reduced the error sufficiently, noise dominates the signal to the comparators, resulting in occasional incorrect evaluations when comparing voltages differing by less than 0.01 V [as shown in Appendix C; see also Fig. 7(e)]. These occasional errors create an error floor but also allow the network to explore the phase space of valid solutions; the ratio of the two resistance values [blue line in Fig. 2(c)] remains nearly

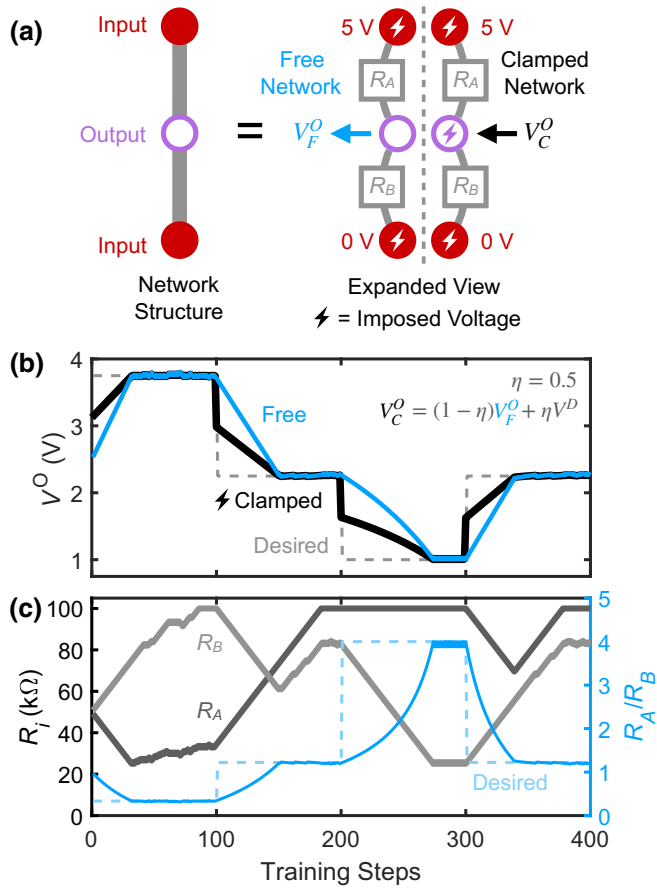


FIG. 2. A self-training voltage divider. (a) A diagram of the network structure, as depicted in later figures (left) and expanded (right) to show both free and clamped networks. A voltage is imposed on the input nodes (red) in both networks and on the output nodes (purple) only in the clamped network. The resistance of each edge is identical in both networks. (b) The output-node voltage  $V^O$  versus training steps for both the free (blue) and clamped (black) networks. The desired voltage  $V^D$  is shown as a gray dashed line. Note that the clamped state effectively guides the free state toward the desired voltage, which is changed every 100 steps, from 3.75 V, to 2.25 V, to 1 V, and finally to 2.25 V. (c) The resistance values of the two edges in the network (grays) and their ratio (blue) as a function of training steps. The light blue dashed line represents the ratio that will produce the desired network output.

constant while both resistance values drift. This stochasticity may be useful for more complex networks and tasks; similar exploration of the available solutions space can promote generalization in both biological [44] and artificial networks [45].

### III. RESULTS

We now demonstrate the success of our system by training a 16-edge network [Fig. 1(a)] to perform three

types of tasks inspired by biology (allostery), mathematics (regression), and computer science (classification). We then demonstrate its flexibility and robustness.

Allostery is a common feature of proteins [37], in which an input signal, namely strain applied to a local region of the protein by binding a regulatory molecule, gives rise to a desired strain or conformational change elsewhere in the protein, enabling or preventing binding of a substrate molecule. In a related problem of “flow allostery” [39,47,48], a pressure drop in one region of a flow network, (e.g., across input arteries in the brain vascular network) gives rise to desired pressure drops elsewhere in the brain at designated output locations that can be quite distant from the input arteries, allowing the vascular system to deliver enhanced blood flow and therefore more oxygen to active parts of the brain. In the context of electrical networks, allostery corresponds to producing specified output voltages in response to given input voltages. This functionality can be useful for tasks such as allocating power to various connected devices.

We choose a three-input three-output allosteric task as an example [Fig. 3(a) inset]. Using a nudge of  $\eta = 0.5$ , the network successfully learns to deliver 3 V at all output nodes, in response to three simultaneous input node voltages of 5, 1, and 0 V. The mean-squared error for this task drops during the learning process by over 4 orders of magnitude [Fig. 3(a)]. We note that in the theoretical treatment,  $\eta \ll 1$  is assumed;  $\eta \sim 1$  will in effect be taking a finite-difference gradient with a large step size and thus will substantially degrade the accuracy [35]. However, in a physical system, noise (order 0.01 V) will dominate the learning process if  $\eta$  is too small. Thus the success of the network at finite  $\eta$  is a nontrivial demonstration of its feasibility in real systems.

Regression is a more difficult test because the desired output voltages are not constants but, rather, functions of the input voltages. We ask the network to solve two equations for two unknowns, choosing the two equations

$$V_1^D = 0.15V_1^I + 0.20V_2^I \quad \text{and} \quad V_2^D = 0.25V_1^I + 0.10V_2^I. \quad (4)$$

We generate a data set of 420 randomly chosen input pair values between 1 and 5 V and calculate the desired voltage for each input pair using the above equations. We set an additional input node at 0 V to remove the freedom for a global shift in voltage, resulting in three input and two output nodes [Fig. 3(b), inset]. We divide the data into a training set (400 elements) and a test set (20 elements). Every clock cycle, the network is shown a new example from the training set and it updates its resistance values accordingly. Between these examples, the network is given the entire test set one by one and its free-state outputs are recorded as an indication of the performance of the network. Given these conditions and  $\eta = 0.2$ , our learning



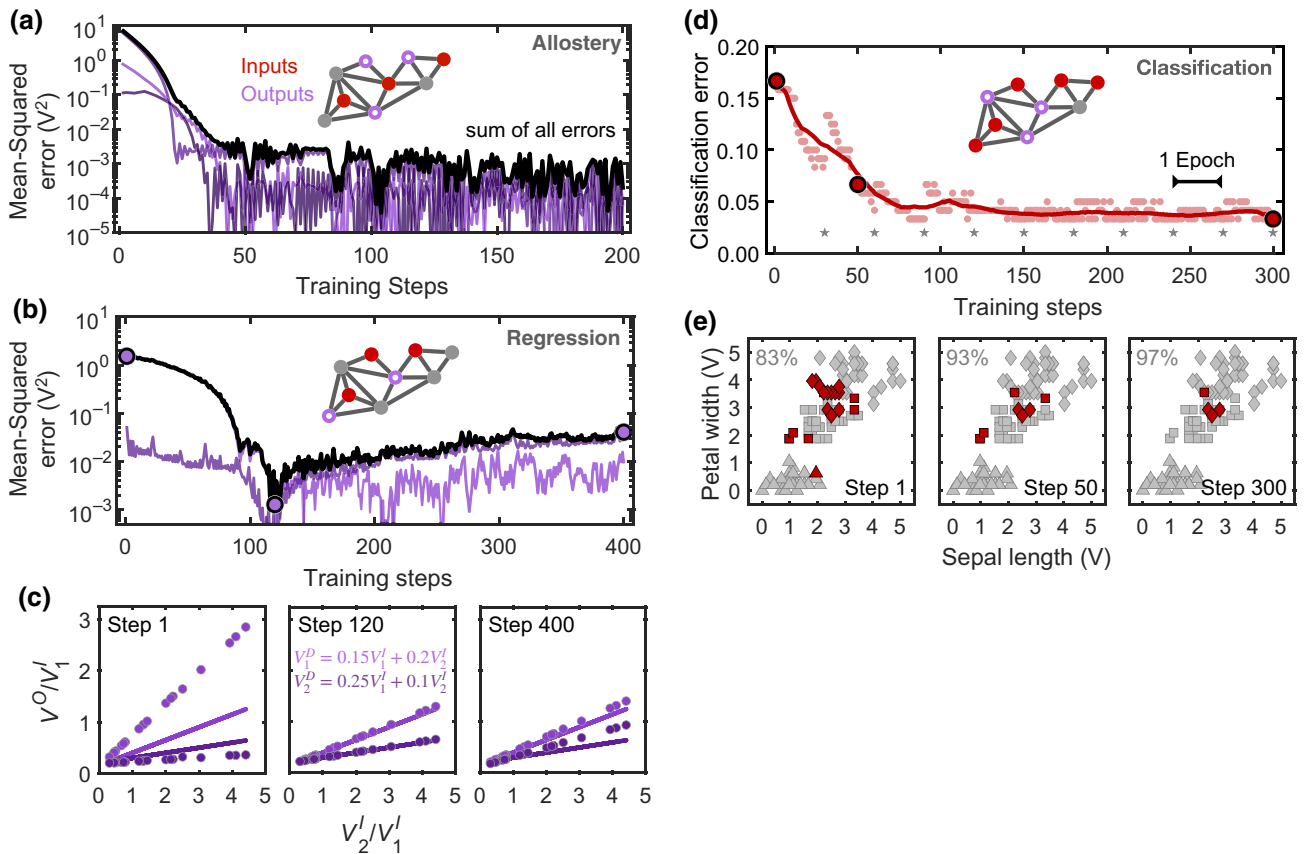


FIG. 3. One physical system performs many tasks. (a) The mean-squared error for each of three outputs and their sum (black) versus training steps for an example allosteric task. (b) The mean-squared error for each of two outputs and their sum (black) for a two-parameter regression task for each output node. The large purple circles indicate the training steps shown in (c). (c) Snapshots of the values for both outputs at three steps during training for the regression task in (b). The lines indicate the desired output values. Regression involves two parameters and thus both axes are scaled by  $V_1^1$  to project the results into two dimensions. (d) The test-set classification error for the iris benchmark data set [46] versus training steps (faded symbols). Smoothing of the data with a window of 30 training steps (solid line) highlights that the final plateau accuracy is above 95%. The large red circles indicate the training steps shown in (e). The desired voltage for each class is remeasured every epoch, indicated by the gray stars (for details, see Appendix A). (e) Snapshots of the classification success of the test set projected into the two-dimensional space of two of the four inputs (sepal length and petal width, rescaled to 0–5 V). The species of iris is denoted by the marker shape. The gray shapes are correctly classified, while the red shapes are incorrectly classified.

machine reduces the mean-squared error for the entire test set by over 2 orders of magnitude [Fig. 3(b)], producing an accurate result despite its small size [Fig. 3(c)]. Note that during training, the network finds an extremely good fit to the data around step 120 but cannot maintain it due to some combination of noise, sampling error from sequential training, and small bias in the internal logic circuitry of the edges. The observed rise in test error before the final plateau is a common feature in machine learning [49].

Data classification is an even more stringent test of the network. We use a benchmark data set of three species of iris flowers [46]. The network is tasked with classifying these flowers based on four measurements: the petal and sepal length and width. We withhold 120 of the 150 flowers as a test set and train on 30 flowers, ten from each species. We designate five input nodes (one for each

measurement plus one fixed ground) and three output nodes [Fig. 3(d), inset]. Between training steps, the entire test set of 120 flowers is run through the network and a flower is considered correctly classified if its three outputs are closest ( $L_2$  norm) to the desired outputs of the correct species. We implement a custom output scheme in which the desired outputs for a given species are recalculated every epoch by averaging outputs of that species. This provides protection against training toward infeasible outputs and robustness to initial conditions (for the full task specification and training details, see Appendix A). Using this algorithm with  $\eta = 0.1$ , the network is able to classify the iris data set with over 95% accuracy [Fig. 3(d)]. For comparison, a linear classifier trained using logistic regression on these data achieves a test accuracy of 98%. The small discrepancy likely stems from the bias of the update rule

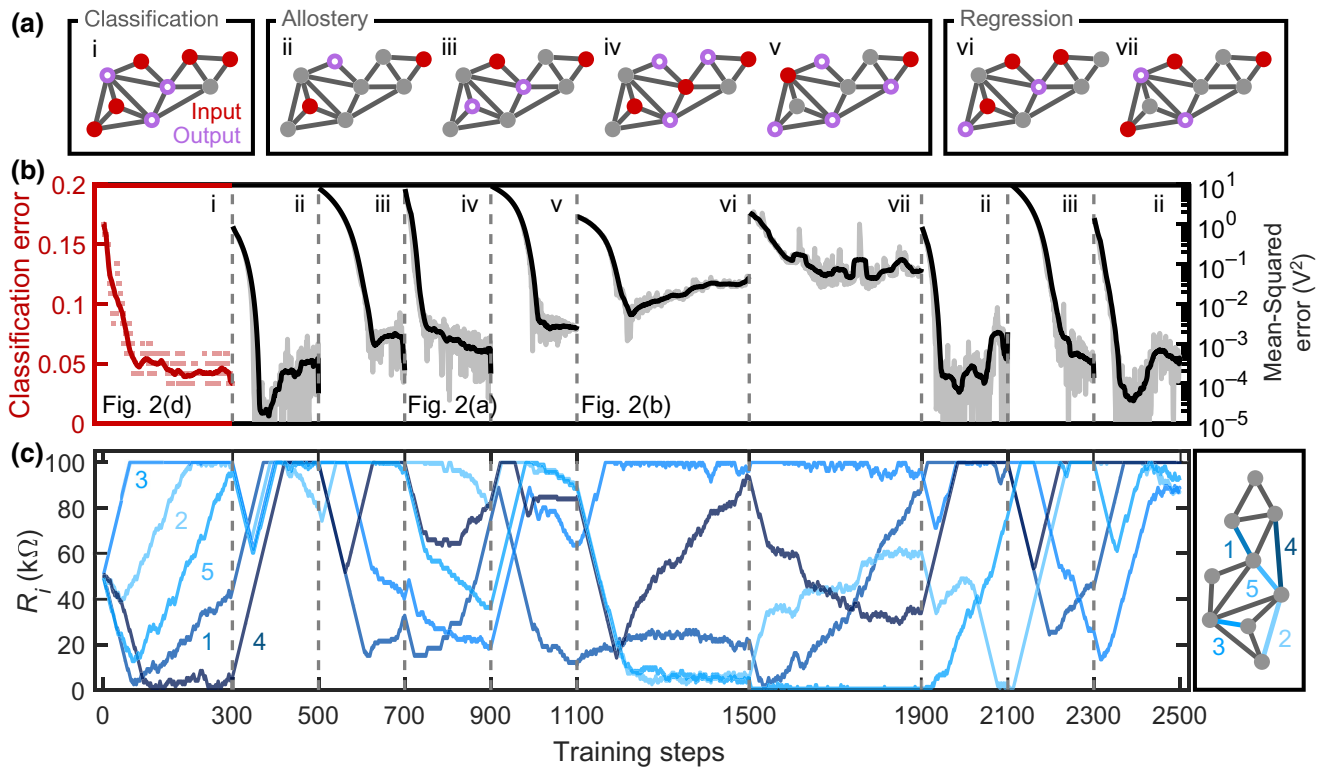


FIG. 4. The learning machine is flexible and retrainable. (a) The network structure, with the input (solid red) and output (purple outlined) nodes indicated for seven distinct tasks (further details for each task are given in Appendix A). (b) The classification error (task i) and mean-squared error (tasks ii–vii) versus training steps. The data are smoothed over a window of 30 training steps, with raw data shown faded in the background. The network performs tasks i–vii in order, then tasks ii, iii, and ii again. (c) Left, the resistance values of five numbered edges over the entire training process; right, the network structure with these five numbered edges highlighted.

(regularization), as well as the restrictions to resistance values, namely that they are chosen from a set of positive discrete values, which limits adjustments to the hypersurfaces separating the classes of data. The two-dimensional projection of the four-dimensional input data [Fig. 3(e)] shows that incorrectly classified flowers lie along overlapping edges of class clusters.

We now highlight some features of the system. The first is the ability to learn new tasks. Unlike simulated networks, a physical learning machine must be physically manufactured. Therefore, a given network is far more useful if it can switch from one task to another on demand. For our system, there is no imposed direction of information travel as in a feed-forward neural network, so any node can be used as an input node, output node, or hidden node. We demonstrate this flexibility by training our network to perform seven distinct tasks in succession, using different input-output configurations [Fig. 4(a)]. In this sequence, our 16-edge network performs one classification task (i), four allosteric tasks with numbers of output nodes ranging from 1 to 4 V (ii–v), and two two-parameter linear regression tasks (vi–vii). The network successfully learns each task in turn, as indicated by the reductions in the mean-squared error [Fig. 4(b)]. The edges are not reset

between tasks but simply find new values as the network adjusts to its new task and training examples [Fig. 4(c)]. Because of this ability to retrain using any input-output combination, a network does not need to be designed specifically to perform certain tasks—it can be trained on *any* task that can be framed in terms of input and output voltages. This flexibility stems, in part, from the ability of the system to “solve” a problem in multiple ways. In this sequence of tasks, our 16-edge network performs task ii, an allosteric task with one output, three different times. Each time, the solution involves different values of edge resistances  $\bar{R}$  and, furthermore, explores this space of approximately equally valid solutions that lie within the noise floor [Fig. 4(c)]. We purposefully bias this drift of resistor values to increase on average (see Appendix C), which pushes the network to avoid high-power solutions that may strain or damage hardware or waste energy. By linearly biasing our update rule, given in Eq. (C2), we effectively implement “lasso” regularization in our network [50], which is known to promote test-set generalization for the price of small increases in the error floor. The network quickly erases memory of previous tasks, as is typical in linear networks [42,51], as seen by the similar initial error in performing task ii each time [Fig. 4(b)].

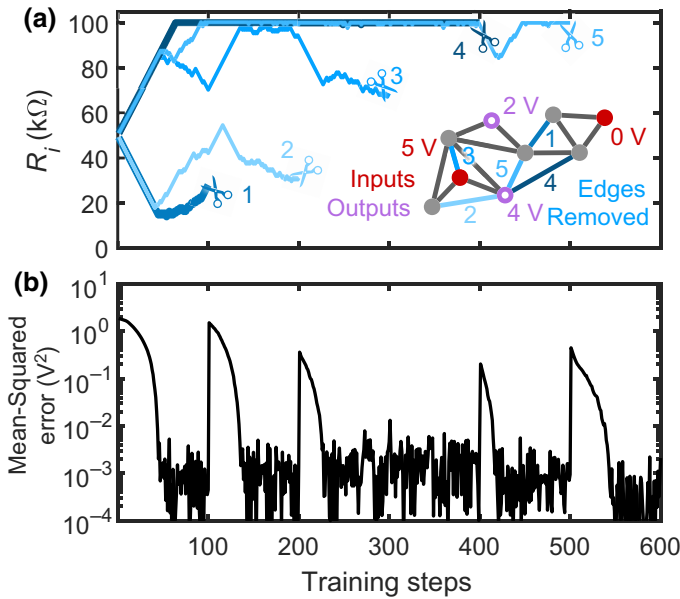


FIG. 5. The learning machine is robust to damage. (a) Edge resistance versus training steps for an allosteric task, as edges are cut. The inset shows the network structure with edges numbered by order of removal. The voltage values indicate the task being performed. The input nodes are solid red and the output nodes are outlined in purple. (b) The mean-squared error for this two-output allosteric task versus training steps. Note the spikes every 100 training steps, when an edge is removed, followed by recovery.

The “capacity” of these networks (e.g., the maximum number of trainable output nodes as a function of the number of nodes and edges in the system) and their ability to retain memory of previous tasks are subjects for future work.

A second useful feature of our network as a learning system is its robustness to damage. Physical systems used to implement simulated neural networks, such as CPUs, are often quite fragile. The breakage or removal of part of a computer usually disables it completely. In contrast, biological systems can often function despite massive damage; given the right conditions, a plucked flower not only survives but it can generate an entirely new plant. While our system cannot grow new edges, it can easily recover its desired function after substantial damage. To demonstrate this feature, we train our network to perform a two-output allosteric task [Fig. 5(a), inset]. We track the resistance values of five edges [Fig. 5(a)], removing one every 100 training steps. During training, our 16-edge network reduces the mean-squared error of the outputs by several orders of magnitude from its initial value [Fig. 5(b)]. The removal of an edge can produce an immediate spike in error as the currents adjust to the new network structure. However, the network recovers each time by finding a novel solution to the task *even after nearly one third of*

*the network structure is destroyed*. Because the network is homogeneous, no edge is special and no single part of the network is essential to its proper functioning. We note that in this demonstration, we prune edges empirically found to be important; that is, we choose edges the removal of which produces a spike in error. In fact, a substantial fraction of edges produce only a modest change in error when pruned, as is the case for the third pruned edge. Even in this linear system, memories can be robust enough against damage that retraining often is not even needed.

#### IV. DISCUSSION AND CONCLUSIONS

We have built a flexible robust physics-driven learning circuit that learns complex tasks by adjusting its internal elements without top-down instruction from a human or computer. Even with only 16 edges, it is capable of a variety of tasks unspecified in its design, namely classification, regression, and allosteric functionality.

Four key concepts underlie the system. First, the edge resistances are “learning” degrees of freedom, distinct from the node voltages, which are “physical” degrees of freedom. Physics constantly adjusts the physical degrees of freedom to minimize energy dissipation, while the learning degrees of freedom are only adjusted during training. Then they are frozen, preserving the ability to perform the learned task. Second, there are more than enough learning degrees of freedom even in our small system to satisfy all the constraints applied in the task examples. This is why the system is able to satisfy all the tasks [39] and why it is robust to substantial damage. Third, our approach specifies a local rule for adjusting the learning degrees of freedom that approximates minimization of the cost function [15,35]. The cost functions themselves are different for different learning tasks but the form of the learning rule, i.e., the adjustment procedure of each edge, remains the same for any task. This is why the system can learn new tasks. Fourth, the implementation of two identical networks resolves a nontrivial constraint for contrastive learning, wherein two states of a single system, corresponding to different boundary conditions, must be compared. Our implementation does not require any additional on-board memory storage, help from a CPU, or the use of temporal signals to enact. As such, it is massively scalable and robust to operate. It is also therefore robust to manufacturing errors such as nonfunctioning edges in both networks. This robustness could be further enhanced by increasing the connectivity of the network, allowing for more edges to be removed before nodes are disconnected from one another. The robustness of our system to additional types of malfunction, such as discrepancies between the two networks, is a subject for future work. There is still much to be understood about even our modest 16-edge system but the simplicity of its local rules and its

basis in well-understood physical laws suggest the possibility of understanding exactly what and how it learns [47,48,52]. Certainly, theoretical understanding seems less difficult to attain for the physical learning machine than for many other neuromorphic realizations, not to mention the brain itself.

Although the abilities of our current prototype are modest compared to artificial neural networks, the successful realization of a physical learning machine opens up numerous paths for future work. Our system allows us to explore modes of learning that are not practical in an artificial neural networks but are present in biological systems, such as updating learning degrees of freedom in a desynchronous manner [53] or before the physical degrees of freedom equilibrate [54]. From a practical perspective, potentiometers with more (or continuous) states, as well as logarithmic or pseudologarithmic spacing of the resistance values, will greatly improve the network flexibility and reduce the error floor [35]. Diodes or other nonlinear circuit elements will allow the system to perform currently prohibited operations such as mimicking an XOR gate [15,55]. Importantly, we can improve both the network size and speed while reducing the size of the components. Our largest network has only 16 edges, each on its own breadboard, and takes up several square feet. Our voltage application and measurement hardware limits the network to steps at 3–5 Hz but the network itself is capable of operating multiple orders of magnitude faster. Furthermore, due to its Boolean logic and simultaneous comparison of two networks, as opposed to the use of memory or temporal signals, and its robustness to damage and thus manufacturing defects, the system is massively scalable. We estimate that the system can easily be scaled up in the number of edges and in the frequency of training steps by at least 6 orders of magnitude using readily available circuit-fabrication methods [56]. Such a circuit would have a footprint 5 orders of magnitude smaller than our prototype (for back-of-envelope calculations of these numbers, see Appendix B).

In computational neural networks, the computation time increases rapidly with the number of edges. An exciting feature of our system is that the addition of edges to the network does not increase computation time per training step, since all edges perform their own adjustments completely in parallel. This feature arises because outputs are not computed but are physical responses to stimuli and because the job of imposing the clamping voltages does not increase in complexity as the network grows. The speed of learning depends on the physical size of the system and its inherent (tiny) capacitance, which together determine the time scale on which the voltages reach equilibrium (of the order of nanoseconds in our system). In the current prototype, this is far faster than our clock-cycle time and thus does not affect training times. Furthermore, due to its nonspecific structure, flexibility, and ability to withstand to damage, the scaling of our system is robust to imperfections and

defects that invariably seep in when the number of components increase. It is possible that this ready scalability of physical learning machines may one day allow them to compete with computational neural networks. Already, with a modest increase of  $\times 100$  in network size with no speed change, our prototype would outperform a simulation implementation as in Ref. [35] due to the inherent bottleneck in the simulation of relying on a processor and memory. Furthermore, as seen in simulations of flow networks [35,39,40], we expect the accuracy and computational capacity of our design to increase with the number of edges and nodes in the system.

We can anticipate many potential uses for our system even in a realization closer to its current modest form. Our system is energy-efficient, drawing of the order of 10 mW to “calculate” outputs, with a rough upper bound of power = voltage<sup>2</sup>/resistance  $\sim (5V)^2/1k\Omega = 25$  mW. The efficiency may be improved substantially by decreasing the voltage range of the network and/or increasing the resistance of the edges by a constant factor. Because it draws little power and does not require separate memory storage, our system may be preferable to a CPU- or GPU-simulated neural network when energy or space are at a premium. Furthermore, power consumption is not concentrated [as in a CPU or a graphics processing unit (GPU)] but distributed evenly across the learning machine, allowing future versions to massively increase speeds without overheating. Because its function is not encoded in its design, our system may be appropriate for tasks that require on-demand flexibility; for example, as a sensor that detects deviations from an as-yet unspecified background signal. Furthermore, because of this *in situ* training, such applications would avoid the simulation-reality gap, training on real-world data. Coupled learning may also be generalizable to encompass other learning paradigms such as unsupervised [57] or reinforcement [58] learning, where the cost function or reward, respectively, could be encoded in the clamped state. Finally, because it is robust to damage, physics-driven learning may be useful for scenarios where a system is exposed to danger.

Our system is robust to damage because it is composed of many repeated identical elements that update themselves in response to stimuli. It is therefore a kind of “learning material” or metamaterial in the sense that it is a many-element system with learning as an emergent collective property that is not inherent in the arrangement of its elements, nor in the selection of input or output locations. If constructed appropriately, physics-based learning networks should be easily modifiable after construction; just as the removal of arbitrarily chosen edges does not destroy functionality, additional edges do not require precise placement to be useful. It is not outlandish to imagine a future adaptive realization of similar learning circuits that would have no need for any *a priori* design



in order to learn and could be augmented or divided like clay.

### ACKNOWLEDGMENTS

We thank James MacArthur for advice regarding circuit design and Marc Miskin for instructive discussions, especially regarding scalability. This work was supported by the National Science Foundation via the University of Pennsylvania Materials Research Science & Engineering Center (MRSEC) Grants No. DMR-1720530 (S.D. and D.J.D.) and No. DMR-2005749 (A.J.L.) and by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering Award No. DE-SC0020963 (M.S.). A U.S. Patent Application (No. 17/750,072) has been filed for the design of the physics-driven learning circuit.

### APPENDIX A: TASK DETAILS

The tasks listed in Fig. 4 in the main text are detailed in Fig. 6. For allosteric tasks, the input or desired output voltages are listed as single values. For regression tasks, the training and test-set inputs are selected using a uniform random distribution between 1 and 5 V and the output desired voltages are functions of these inputs, as listed. For the classification task, each input (e.g., all petal widths) is rescaled to span 0–5 V. A typical classification output scheme in an artificial neural network (ANN) would designate one output node for each class and train toward producing a high value (e.g., 5 V) at the node of the correct class and 0 V at all other output nodes. However, this output basis is not feasible because our network is linear. We instead choose an output basis as follows. At the start of every epoch (every 30 training steps), we measure the output response of the network to the *average* input values from each species of flower in the training set. In a linear

network, this is identical to calculating the average output values from all elements in the training set, as done in previous work [43]. During the ensuing epoch, the desired output voltage for each flower is this average response for the appropriate species. These desired voltages evolve as the network trains but eventually settle at consistent values. Because these output averages depend solely on training data, they may be useful in the future for determining when to stop training a learning network. Furthermore, this averaging method improves the initial accuracy beyond the expected 33%, since it picks target values with a minimal distance to the network response for a given species.

### APPENDIX B: SCALING THE ELECTRONICS

Our prototype is not built with speed or scale as a priority and, as a result, leaves much room for improvement in these regards. Our system takes up several square feet and operates at about 3–5 Hz, limited by the data-acquisition and voltage-setting hardware. Analog networks utilizing variable weights and comparators (without utilizing “physics as computation”) have been accomplished with under 100 transistors per edge equivalent of our prototype (often referred to as synapses) [24]. State-of-the-art CMOS fabrication can yield roughly  $300 \times 10^6$  transistors per  $\text{mm}^2$ , operating on nanosecond time scales or faster [56]. Using these estimates, a  $10^7$ -edge physical learning network could be implemented with a footprint less than  $10 \text{ mm}^2$ . Such a system would represent a  $10^6$  increase in edge count, a  $10^5$  decrease in footprint, and a  $10^8$  increase in speed from our prototype.

### APPENDIX C: CIRCUITRY

Our electrical network uses variable resistors as edges (AD5220 digital potentiometers wired as rheostats). These “digipots” are not continuously adjustable as assumed by the original coupled-learning rule [35] but, instead, have 128 resistance values evenly spaced by  $\delta R = 100 \text{ K}\Omega/128 \sim 781\Omega$ . We therefore restrict the evolution of each edge to discrete steps  $\pm\delta R$  in either direction. The coupled-learning rule then simplifies to

$$\Delta R_i = \begin{cases} +\delta R & \text{if } |\Delta V_i^C| > |\Delta V_i^F|, \\ -\delta R & \text{otherwise.} \end{cases} \quad (\text{C1})$$

Other learning rules that only depend on the signs of the gradient of cost functions have been shown to be successful [59]. This new rule is also easier to implement digitally, as it only requires a Boolean comparison of voltage drops instead of a difference in energy dissipation. However, Eq. (C1) still requires access to both the free and the clamped electrical state. To this end, we construct two identical networks for comparison, one running the free state and the other running the clamped state. The corresponding

| Task | Node:          | 1                  | 2            | 3           | 4                  | 5                  | 6                  | 7   | 8           | 9     |
|------|----------------|--------------------|--------------|-------------|--------------------|--------------------|--------------------|-----|-------------|-------|
| i    | Classification | $O_1$              | Petal Length | Petal Width | $O_2$              | Sepal Length       | $O_3$              |     | Sepal Width | 2.5 V |
| ii   | Allosteric     |                    |              | 4 V         |                    |                    |                    |     | 5 V         | 0 V   |
| iii  | Allosteric     |                    |              | 4 V         | 1 V                |                    |                    |     | 2 V         | 0 V   |
| iv   | Allosteric     | 3 V                | 3 V          | 3 V         | 5 V                |                    |                    |     | 1 V         | 0 V   |
| v    | Allosteric     | 3 V                |              | 2 V         |                    | 4 V                | 5 V                | 1 V |             | 0 V   |
| vi   | Regression     |                    | 0 V          | $I_1$       | $0.25I_1 + 0.1I_2$ | $0.15I_1 + 0.2I_2$ |                    |     | $I_2$       |       |
| vii  | Regression     | $0.15I_1 + 0.1I_2$ |              | $I_1$       |                    | 0 V                | $0.2I_1 + 0.25I_2$ |     |             | $I_2$ |

FIG. 6. The task details. The voltage specification by node number for each task detailed in Fig. 4. The red cells are input nodes and the purple-outlined cells are output nodes. The node numbers correspond to the network structure as shown in the inset.

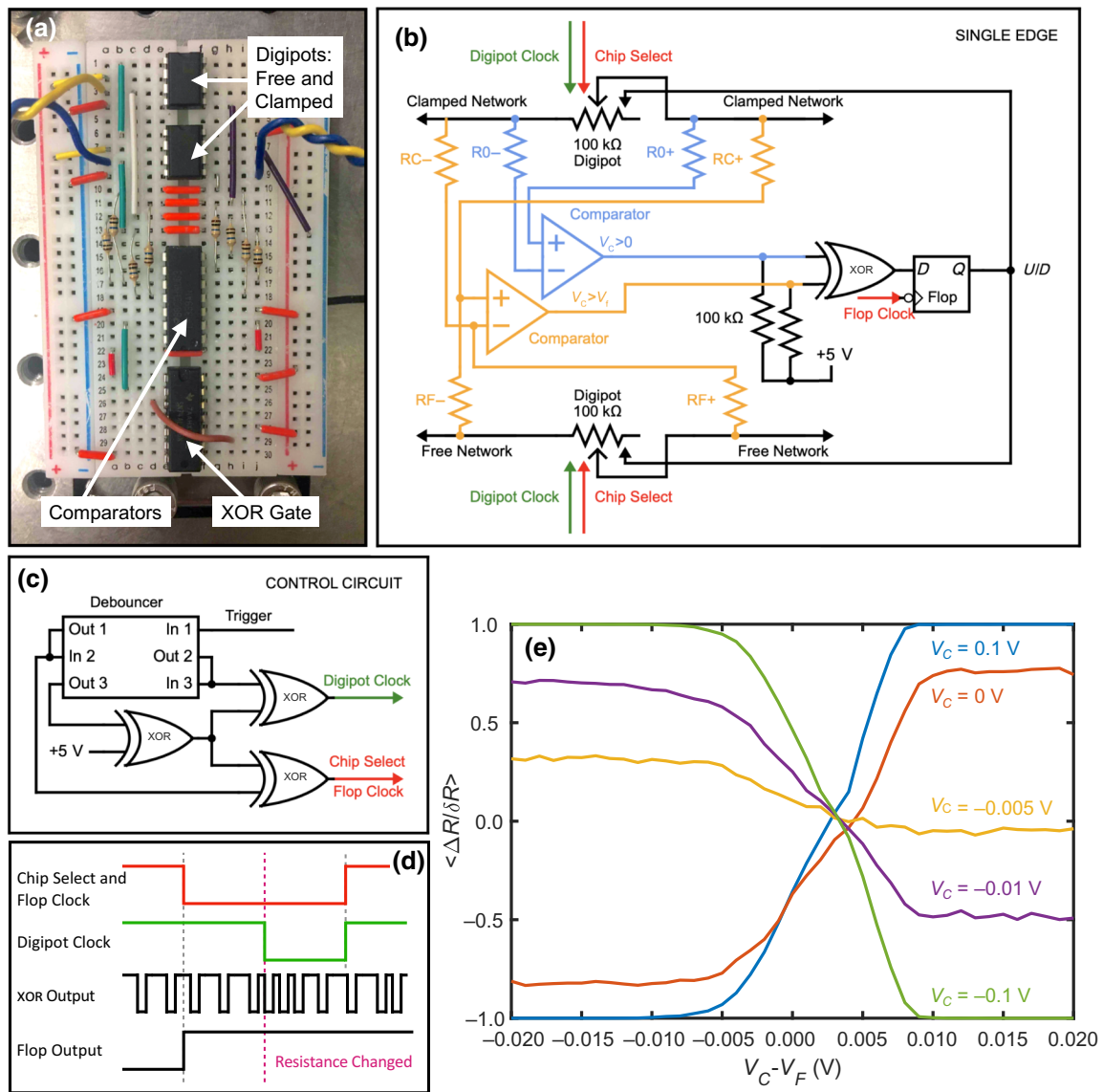


FIG. 7. A single edge of the network. (a) An image of an edge, as constructed on a breadboard. (b) The circuit diagram for a single edge, which houses the circuitry for both the free and the clamped network. Comparators and an XOR gate compute the direction of resistance change based on the relative voltage drops across the free and clamped variable resistors (digipots), and the XOR output is stored in a  $D$ -flop before being fed back into the up-down input of the potentiometers. (c) The global clock circuitry. The control circuit receives an ascending or descending edge from the data-acquisition card (computer) into the “Trigger” port. This produces a cascading effect through the debouncer, changing output 1, then 2, then 3, which are fed into XOR gates. (d) This cascade results in a descending edge in the digipot chip select and  $D$ -flop clock signal, then a descending edge in the digipot clock signal, and finally a return to high for both signals. As a result, the XOR output of the edge circuit shown in (b) is sampled and stored by the  $D$ -flop ahead of the digipot clock, triggering a change in resistance. This avoids feeding the potentially fluctuating XOR signal directly into the digipot. (e) The average resistance change as a function of the comparator voltages  $V_C$  and  $(V_C - V_F)$ . Ideally, we would have step functions jumping at  $(V_C - V_F) = 0$  V. Noise spreads out the transition and, in this edge, comparator bias shifts the curves to the right.

edges of the free and clamped networks always have the same resistance and are housed on the same breadboard (Fig. 7A).

The absolute-value comparison in Eq. (C1) is still non-trivial to evaluate electronically. A comparator produces a signed comparison  $\Delta V_i^C > \Delta V_i^F$  but this will yield the *opposite* of our desired value if both drops are negative,

which we cannot rule out *a priori*. We can, however, assume that the two voltage drops have the same sign. Empirically, we find this is nearly always the case, especially for  $\eta \ll 1$ . We can then use a second comparison,  $\Delta V_i^C < 0$ , to determine if  $\Delta V_i^C > \Delta V_i^F$  is equivalent to  $|\Delta V_i^C| > |\Delta V_i^F|$  (positive voltage) or its inverse (negative voltage). Our learning rule can now be written using only

functions of common logical circuit components:

$$\Delta R_i = \begin{cases} +\delta R & \text{if xor} [\Delta V_i^C > \Delta V_i^F, 0 < \Delta V_i^C], \\ -\delta R & \text{otherwise.} \end{cases} \quad (\text{C2})$$

We implement Eq. (C2) with two comparators (LM339AN), one XOR gate (SN74ALS86N), and one  $D$ -flop (TI CD74HC73E JK flop plus SN74ALS86N XOR gate) on every edge [Fig. 7(b)]. On each edge, the output of the XOR gate is stored in the  $D$ -flop and fed back into the up-down input of the digital potentiometers in both the free and the clamped networks. During training, the resistance updates of every variable resistor are triggered by the descending edge of a global clock signal fed into the digital potentiometers. A switch debouncer or delay (MC14490PG) circuit and three XOR gates are wired to generate two sequential descending edge signals [red and green in Figs. 7(b)–7(d)]. The first descending edge is used to trigger a  $D$ -flop (TI CD74HC73E JK flop plus SN74ALS86N XOR gate) to store the output of the XOR gate [Eq. (C2)]. Because the learning machine naturally moves the voltage of the free and clamped networks toward each other, this XOR output ( $U/D$  signal) will typically become dominated by noise by the end of training and will oscillate rapidly. Storing the value in the  $D$ -flop ensures a clean signal in the  $U/D$  port of the digital potentiometers, as shown in Fig. 7(b). Finally, the variable resistors (digipots) used in our system (AD5220 100k) have a slight bias in their logical evaluation. As a result, the update rule [Eq. (C2)] is imperfectly evaluated at similar free and clamped voltage drops, as shown in Fig. 7(d). These incorrect evaluations do not prevent our system from functioning but do limit the error floor.

---

[1] B. A. Richards, *et al.*, A deep learning framework for neuroscience, *Nat. Neurosci.* **22**, 1761 (2019).  
 [2] U. Hasson, S. A. Nastase, and A. Goldstein, Direct fit to nature: An evolutionary perspective on biological and artificial neural networks, *Neuron* **105**, 416 (2020).  
 [3] A. Tavanaei and A. Maida, BP-STDP: Approximating backpropagation using spike timing dependent plasticity, *Neurocomputing* **330**, 39 (2019).  
 [4] A. Ganguly, R. Muralidhar, and V. Singh, in *20th International Symposium on Quality Electronic Design (ISQED)* (Institute of Electrical and Electronics Engineers, New York, New York, 2019), p. 335.  
 [5] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature* **521**, 436 (2015).  
 [6] A. Tero, R. Kobayashi, and T. Nakagaki, *Physarum* solver: A biologically inspired method of road-network navigation, *Physica A* **363**, 115 (2006).  
 [7] K. Alim, N. Andrew, A. Pringle, and M. P. Brenner, Mechanism of signal propagation in *Physarum polycephalum*, *Proc. Natl. Acad. Sci.* **114**, 5136 (2017).

[8] R. A. McGovern, A. N. V. Moosa, L. Jehi, R. Busch, L. Ferguson, A. Gupta, J. Gonzalez-Martinez, E. Wyllie, I. Najm, and W. E. Bingaman, Hemispherectomy in adults and adolescents: Seizure and functional outcomes in 47 patients, *Epilepsia* **60**, 2416 (2019).  
 [9] B. Sengupta and M. B. Stemmler, Power consumption during neuronal computation, *Proc. IEEE* **102**, 738 (2014).  
 [10] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, Towards biologically plausible deep learning, *ArXiv:1502.04156* (2015).  
 [11] Y. Bengio and A. Fischer, Early inference in energy-based models approximates back-propagation, *ArXiv:1510.02777* (2016).  
 [12] Y. Bengio, A. Fischer, T. Mesnard, S. Zhang, and Y. Wu, From STDP towards biologically plausible deep learning, <https://www.semanticscholar.org/paper/From-STDP-towards-Biologically-Plausible-Deep-Bengio-Fischer/bed43f7328b51590433dead30116ba1ff5fb7602> (2015).  
 [13] D. Marković, A. Mizrahi, D. Querlioz, and J. Grollier, Physics for neuromorphic computing, *Nat. Rev. Phys.* **2**, 499 (2020).  
 [14] M. Ernout, J. Grollier, and D. Querlioz, Using memristors for robust local learning of hardware restricted Boltzmann machines, *Sci. Rep.* **9**, 1851 (2019).  
 [15] J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, Training end-to-end analog neural networks with equilibrium propagation, *ArXiv:2006.01981* (2020).  
 [16] E. Martin, M. Ernout, J. Laydevant, S. Li, D. Querlioz, T. Petrisor, and J. Grollier, EqSpike: Spike-driven equilibrium propagation for neuromorphic implementations, *iScience* **24**, 102222 (2021).  
 [17] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, and Q. Xia, Efficient and self-adaptive *in-situ* learning in multi-layer memristor neural networks, *Nat. Commun.* **9**, 2385 (2018).  
 [18] 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).  
 [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] 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, in *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)* (Institute of Electrical and Electronics Engineers, New York, New York, 2016), p. 1.  
 [21] Z. Wang, C. Li, W. Song, M. Rao, D. Belkin, Y. Li, P. Yan, H. Jiang, P. Lin, M. Hu, J. P. Strachan, N. Ge, M. Barnell, Q. Wu, A. G. Barto, Q. Qiu, R. S. Williams, Q. Xia, and J. J. Yang, Reinforcement learning with analogue memristor arrays, *Nat. Electr.* **2**, 115 (2019).  
 [22] W. Zhang, B. Gao, J. Tang, P. Yao, S. Yu, M.-F. Chang, H.-J. Yoo, H. Qian, and H. Wu, Neuro-inspired computing chips, *Nat. Electr.* **3**, 371 (2020).  
 [23] Y. Arima, M. Murasaki, T. Yamada, A. Maeda, and H. Shinohara, A refreshable analog VLSI neural network chip with 400 neurons and 40 K synapses, *IEEE J. Solid-State Circuits* **27**, 1854 (1992).

- [24] C. Schneider and H. Card, Analog CMOS deterministic Boltzmann circuits, *IEEE J. Solid-State Circuits* **28**, 907 (1993).
- [25] 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 Lett.* **15**, 2203 (2015).
- [26] S. La Barbera, A. F. Vincent, D. Vuillaume, D. Querlioz, and F. Alibart, Interplay of multiple synaptic plasticity features in filamentary memristive devices for neuromorphic computing, *Sci. Rep.* **6**, 39216 (2016).
- [27] A. Serb, J. Bill, A. Khat, R. Berdan, R. Legenstein, and T. Prodromakis, Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses, *Nat. Commun.* **7**, 12611 (2016).
- [28] M. Stern and A. Murugan, Learning without neurons in physical systems, [arxiv:2206.05831](https://arxiv.org/abs/2206.05831) (2022).
- [29] N. Pashine, D. Hexner, A. J. Liu, and S. R. Nagel, Directed aging, memory, and nature's greed, *Sci. Adv.* **5**, eaax4215 (2019).
- [30] D. Hexner, N. Pashine, A. J. Liu, and S. R. Nagel, Effect of directed aging on nonlinear elasticity and memory formation in a material, *Phys. Rev. Res.* **2**, 043231 (2020).
- [31] J. R. Movellan, in *Connectionist Models*, edited by D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (Morgan Kaufmann, Cambridge, MA, 1991), p. 10.
- [32] N. Pashine, Local rules for fabricating allosteric networks, *Phys. Rev. Mater.* **5**, 065607 (2021).
- [33] B. Scellier and Y. Bengio, Equilibrium propagation: Bridging the gap between energy-based models and backpropagation, *Front. Comput. Neurosci.* **11**, 13 (2017).
- [34] B. Scellier, Ph.D. thesis, University of Montreal, school of arts and sciences, 2021. Available at [ArXiv:2103.09985](https://arxiv.org/abs/2103.09985).
- [35] M. Stern, D. Hexner, J. W. Rocks, and A. J. Liu, Supervised learning in physical networks: From machine learning to learning machines, *Phys. Rev. X* **11**, 021045 (2021).
- [36] C. P. Goodrich, A. J. Liu, and S. R. Nagel, The Principle of Independent Bond-Level Response: Tuning by Pruning to Exploit Disorder for Global Behavior, *Phys. Rev. Lett.* **114**, 225501 (2015).
- [37] J. W. Rocks, N. Pashine, I. Bischofberger, C. P. Goodrich, A. J. Liu, and S. R. Nagel, Designing allostery-inspired response in mechanical networks, *Proc. Natl. Acad. Sci.* **114**, 2520 (2017).
- [38] M. Stern, V. Jayaram, and A. Murugan, Shaping the topology of folding pathways in mechanical systems, *Nat. Commun.* **9**, 4303 (2018).
- [39] J. W. Rocks, H. Ronellenfitsch, A. J. Liu, S. R. Nagel, and E. Katifori, Limits of multifunctionality in tunable networks, *Proc. Natl. Acad. Sci.* **116**, 2506 (2019).
- [40] M. Ruiz-Garcia, A. J. Liu, and E. Katifori, Tuning and jamming reduced to their minima, *Phys. Rev. E* **100**, 052608 (2019).
- [41] D. Hexner, A. J. Liu, and S. R. Nagel, Periodic training of creeping solids, *Proc. Natl. Acad. Sci.* **117**, 31690 (2020).
- [42] M. Stern, M. B. Pinson, and A. Murugan, Continual learning of multiple memories in mechanical networks, *Phys. Rev. X* **10**, 031044 (2020).
- [43] M. Stern, C. Arinze, L. Perez, S. E. Palmer, and A. Murugan, Supervised learning through physical changes in a mechanical system, *Proc. Natl. Acad. Sci.* **117**, 14843 (2020).
- [44] D. Kappel, S. Habenschuss, R. Legenstein, and W. Maass, Network plasticity as Bayesian inference, *PLoS Comput. Biol.* **11**, e1004485 (2015).
- [45] Y. Feng and Y. Tu, The inverse variance-flatness relation in stochastic gradient descent is critical for finding flat minima, *Proc. Natl. Acad. Sci.* **118**, 9 (2021).
- [46] R. A. Fisher, The use of multiple measurements in taxonomic problems, *Ann. Eugen.* **7**, 179 (1936).
- [47] J. W. Rocks, A. J. Liu, and E. Katifori, Revealing structure-function relationships in functional flow networks via persistent homology, *Phys. Rev. Res.* **2**, 033234 (2020).
- [48] J. W. Rocks, A. J. Liu, and E. Katifori, Hidden Topological Structure of Flow Network Functionality, *Phys. Rev. Lett.* **126**, 028102 (2021).
- [49] Y. Yao, L. Rosasco, and A. Caponnetto, On early stopping in gradient descent learning, *Constr. Approx.* **26**, 289 (2007).
- [50] R. Tibshirani, Regression shrinkage and selection via the lasso, *J. R. Stat. Soc. Ser. B* **58**, 267 (1996).
- [51] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, Overcoming catastrophic forgetting in neural networks, *Proc. Natl. Acad. Sci.* **114**, 3521 (2017).
- [52] A. Holzinger, in *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)* (Institute of Electrical and Electronics Engineers, New York, New York, 2018), p. 55.
- [53] J. F. Wycoff, S. Dillavou, M. Stern, A. J. Liu, and D. J. Durian, Desynchronous learning in a physics-driven learning network, *J. Chem. Phys.* **156**, 144903 (2022).
- [54] M. Stern, S. Dillavou, M. Z. Miskin, D. J. Durian, and A. J. Liu, Physical learning beyond the quasistatic limit, *Phys. Rev. Res.* **4**, L022037 (2022).
- [55] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT Press, New York, NY, 2017).
- [56] G. Yeap, *et al.*, in *2019 IEEE International Electron Devices Meeting (IEDM)* (Institute of Electrical and Electronics Engineers, New York, New York, 2019), p. 36.7.1.
- [57] H. Barlow, Unsupervised learning, *Neural Comput.* **1**, 295 (1989).
- [58] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, edited by F. Bach, Adaptive Computation and Machine Learning Series (A Bradford Book, Cambridge, Massachusetts, 1998).
- [59] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, signSGD: Compressed optimisation for non-convex problems, *PMLR* **80**, 560 (2018).