Simulating Decentralized, Physics-Driven Learning

Ethan Seiz
eseiz24@avenues.org

Abstract

Since the popularization of deep neural networks (DNNs), the computational requirements for artificial neural networks (ANNs) have increasingly expanded. In attempts to increase the efficiency of DNNs, application-specific integrated circuits (ASICs) have been developed to execute specific mathematical operations at a high throughput. In contrast, another more novel approach seeks to eliminate this bottleneck completely, mimicking how neurons and synapses in the brain self-adjust based on local information without a central processor. One of the first physically realized versions of these systems uses two identical networks of adjustable resistors that model the behavior of neurons and employs a learning algorithm that enables the local comparison and subsequent updating of each resistor. While finalized implementations of these networks can be manufactured cheaply at the nanoscale using existing CMOS fabrication techniques, larger, more expensive circuit components are better suited for exploring the ambiguous relationships between network parameters, making large-scale experiments difficult. This paper explores the hyperparameter relationships of these networks (i.e. the effect that the configuration and amount of resistors have on accuracy) in a resource-light way, by rendering similar resistor-based neural networks in circuit simulation software. By nature of the method of circuit simulation, a fundamentally adapted version of the previous learning algorithm is tested, and it is revealed that the original, more complex algorithm is desirable at the smaller scale. However, while demonstrating that there are certain requirements for network size and complexity based on the desired number of parameters, it is also indicated that a physically larger and more complex network may be successful at hosting the adapted learning algorithm, which could enable the use of cheaper electronic components in the network.

1. Introduction

Since the demonstration of applying backpropagation to efficiently train large deep neural networks (DNNs) by Krizhevsky et al. [1] in 2012, the use of DNNs, which have proven to be highly accurate for tasks in computer vision, speech recognition, and robotics, has skyrocketed. Consequently, an increasingly large demand for computational power has surfaced due to the complexity of the training algorithms, which is quickly being met by the bounds of

traditional CPU computing conventionally supporting DNNs [2]. In response to this deficit, a new branch of application-specific integrated circuits (ASICs), DNN accelerators, were inspired to make up for this shortfall in hardware energy efficiency [3]. These hardware accelerators are circuits designed specifically to carry out the mathematical operations in DNNs at a high throughput. For instance, a series of Multiplier Accumulator components (MACs) can be wired to efficiently compute matrix multiplications, considerably increasing the speed of Convolutional Neural Networks (CNNs), a type of DNN, which perform several matrix multiplications per layer [4]. More conventionally, hardware accelerators have been used in the inference phase of learning, which can account for up to 90% of energy costs in deep learning models, or in the training phase [2].

While DNN accelerators can increase the performance of certain machine learning tasks by several orders of magnitude, they lack application generalizability and only temporarily address this computational bottleneck [5]. That is, a processing unit (e.g. CPU) is still required to train the network, and external memory is still required to store this information [5], [6]; only a certain part of the processing may be aided with an accelerator. Therefore, topical research has focused on transitioning towards learning systems that are more akin to the brain and other biological systems, which can function at adequate speeds despite operating at significantly slower signal propagation rates and possessing far more learning parameters [5], [7], [8]. Such systems, physical neural networks (PNNs), bypass digitally computing standard machine learning algorithms, instead exploiting the physical response of materials to perform computation faster and more energy efficiently than their digital counterparts [2].

Another even less computationally heavy method is utilizing contrastive learning algorithms in conjunction with PNNs as a substitution for cost function minimization altogether, thereby replacing the need for any external processing [9]. Instead, contrastive learning compares the differing responses produced by two different sets of input and/or output conditions to adjust degrees of freedom (i.e. weights) locally [5], [9], [10]. This entails only simple boolean comparisons between the responses of two corresponding regions of interest, followed subsequently by adjustment. The introduction of equilibrium propagation into this learning scheme nudges the network towards the desired solution instead of imposing it directly, enabling decentralized learning [5].

Recent work [5] fully realized this method in the laboratory by assembling two identical networks and imposing each with a different input and output state simultaneously. This thus allowed the comparisons to occur concurrently, ridding the need for any memory storage which had previously restricted any fully *in situ* implementations. [5] exploited an electrical-induced physical imperative to minimize power dissipation, constructing two identical networks composed of variable resistors. Each resistor in the network acts as a degree of freedom that forms a node by intersecting with other resistors. Analogous to weights and nodes in a conventional ANN, the resistance of a given resistor indicates the strength of the correlation between the nodes it connects. When voltage is applied to a network of these resistors, forward propagation occurs as the system naturally minimizes energy dissipation, and the output is calculated within nanoseconds. Then, the learning algorithm is applied as the voltage drop across each corresponding resistor is compared and the resistance is updated in small steps according to the learning rule.

So far, it has been demonstrated that this type of network is successful at completing a variety of machine learning tasks at a smaller scale, with a maximum of three inputs and three outputs. Even preliminarily, this proves to be very promising; however, constructing larger networks to experiment with different configurations and sizes is imperative to fully understand the capacity and relationships at play in this novel approach [5]. Yet, a cost and sizing barrier limits realizing systems on a significantly larger scale: the 16-resistor implementation in [5] was several square feet in size, and currently costs roughly $13 an edge (Appendix A). With existing nanofabrication techniques [11], a comparable 10 million-edge network could be implemented in $10mm^2$ [5], but this method is not suitable for prototyping which is the phase that this work is currently in.

This paper demonstrates two implementations of this physics driven network in two different simulation softwares, Falstad Circuit Simulator and Circuitlab, which enable the rapid and cheap prototyping of larger systems of this nature. The first implementation in Falstad proves to have more functionality and a higher accuracy, as it allows for a direct adaptation of the learning rate presented in the prior research. However, the second implementation in Circuitlab has the capacity to render much larger networks and run more complex simulations. In addition to providing a blueprint for the digital realization of this novel neural network, this research explores unique network configurations with an emphasis on CNN-inspired architecture. Ultimately, this research presents an alternative approach for prototyping networks of this design before their eventual permanent realizations in silicon.

2. Related Works

The Hardware Accelerator Pipeline

In recent work, Wright et al. [2] demonstrated how to apply backpropagation to train multiple physical systems to execute DNNs. First, an example PNN was introduced, where an optical pulse was generated based on input data encoded into its emission spectrum and propagated through a crystal, producing effects of physical computation once the output spectrum is measured. The second experiment used physics-aware training (PAT), which is a hybrid *in situ-in silico* algorithm that involves a processor to estimate the gradient loss with respect to the model's controllable parameters. A more complex image classification task was then used, where the existing PNN from the first experiment were trained with PAT to achieve a 99.1% accuracy on MNIST (Modified National Institute of Standards and Technology database), a benchmark digit classification task. The main advantage PAT displayed was the ability to leverage physical hardware to complete the forward pass, which runs inferences and produces an error, increasing efficiency.

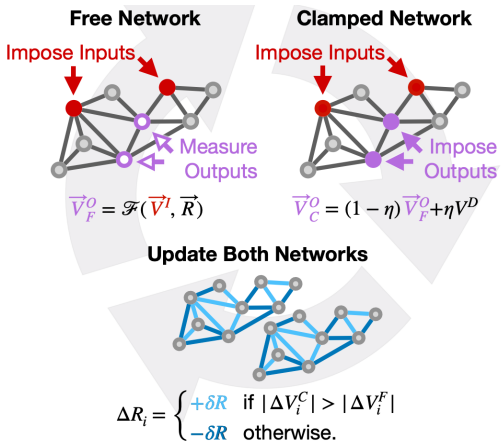
Building upon [2], Kendall et al. [9] trained PNNs relying on local rules aided by an external processor, in contrast to PAT which instead minimized a global cost function. In [9], a deep analog neural network of three input and output nodes, and two hidden layers of three neurons each, connected by several adjustable resistors (represented as edges), was realized. Notably, on each edge, was a nonlinear transfer function modeled by a pair of antiparallel

diodes. This produced a sigmoidal activation function which scales input values to a number between 1 and -1, giving the network nonlinear properties. The fundamental difference in this work to work done prior was the use of contrastive learning and equilibrium propagation to train the network. This was done by first imposing a set of training inputs to the network ('free state'), measuring its response, and then comparing this to the response of the network when the outputs are clamped at desired output voltages ('clamped' state). The error gradient of each edge was calculated by comparing the voltage drop across each resistor in the two different 'states', which then determined the direction to update each resistor (in resistance), infinitesimally updating the network towards the target solution. This process is then repeated several times, determined by the amplitude of the nudge, until the network outputs the desired solution. An important distinction is that this system requires memory, as the voltage drops across each must be temporarily stored so that they can be compared between the two states. This caveat therefore restricted the entire system to a hybrid *in situ-in silico* form.

Most recently, Dillavou et al. [5] solved this conundrum by constructing two identical twin networks to simultaneously measure the responses of the same physical system to two different sets of boundary conditions (i.e. states). Similar to previous research, the same physical nature of the hardware was exploited, and both contrastive learning and equilibrium propagation were utilized. But this network was able to be fully realized *in situ* as no external processing or memory is required; the model can enact its learning completely asynchronously.

3. Background

Vanguard training procedure developed by Dillavou et al. [5]:



- Two identical twin **networks** are assembled, allowing for the simultaneous comparison between the edges of networks imposed with different boundary conditions. The free network is identical to the clamped network; they differ in terms of their input and output conditions (states). Both networks are composed of a series of digital potentiometers (digipots), which can be electronically triggered to change their resistance, allowing for

automatic updates. In this paper, we experiment with using networks composed of digipots and potentiometers (i.e. resistors that require a manual input to be adjusted).

- In the **free network**, Input voltages V^{Input} are applied to the free network and the output response is measured V_f^{Output}.
- In the **clamped network**, the same input voltages V^{Input} are applied to the clamped network, but the outputs V_c^{Output} are clamped closer to the desired output voltages than V_f^{Output} determined by the equation:

$$V_c^{Output} = nV^{Desired} + (1 - n) V_f^{Output}$$

(Equation 1)

- The original **learning rule**,

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

(Equation 2)

where the change in resistance is a function of the boolean comparison between the voltage drops ΔV_i^C and ΔV_i^F in corresponding nodes. Note that the resistance changes ΔR_i^c and ΔR_i^f in each network by the same amount δR each time, and the increment is determined by the quality of the digipot (i.e. the number of possible discrete resistance values or steps).

4. Approach

Falstad circuit simulator was the first simulation tool used due to its large component library and relative ease of use. In Falstad we constructed networks composed solely of digipots, our goal to digitally render the networks realized in [5] as accurately as possible. First, in our implementation, an adapted version of the original learning rule Eq. 2, Eq. 3,

$$\Delta R_i^C = \Delta R_i^F = \begin{cases} +\delta R & if AND[V_{node}^C > V_{node}^F, 0 < V_{node}^C] \\ -\delta R & otherwise \end{cases}$$

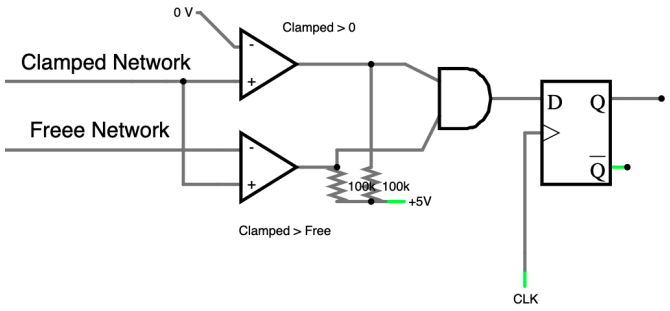
(Equation 3)

is evaluated by a circuit (Fig. 1a) housed on each edge of the network, sending an ‘update’ signal to the digipot on its respective edge, thus enabling the network to self-adjust its “learning degrees of freedom.” Eq. 3 enables the absolute value comparison in Eq. 2 to be electronically executed by two comparators and an AND gate, and a subsequent D-Flop stores and sends the output signal to a digipot every clock cycle to eliminate noise. [5] used an XOR gate after two

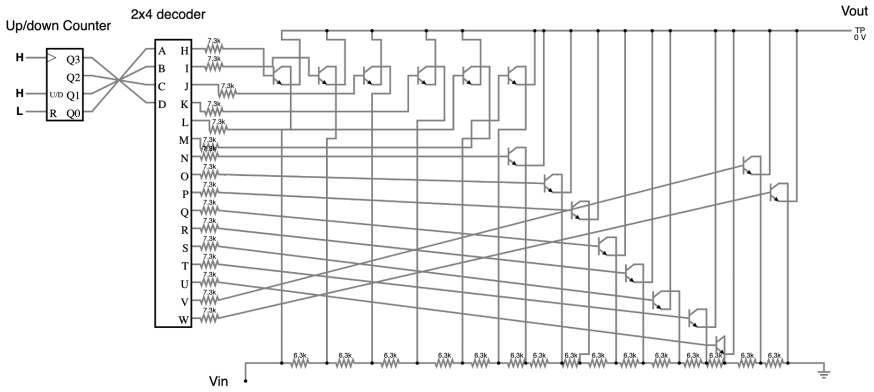
comparators in series. We use an AND gate instead, because we compare the *voltages* of two corresponding nodes instead of the *voltage drops* across the edges. This introduces added simplicity that is discussed later on.

The signal that is outputted by this evaluation circuit determines which direction to update its corresponding 100kΩ digital potentiometer, which has 16 discrete ~6.3kΩ steps. We constructed a custom digipot as the component does not exist in Flastad (and in most other circuit simulators), which is created by wiring 16, ~6.3kΩ resistors in series after V_{in} (Fig. 1b). Another set of wires, each of which connect to V_{out} through n-type bipolar junction transistors run perpendicular to this array, intersecting between each resistor. Therefore, 'adding' resistance in constant increments of ~6.3kΩ can be achieved simply by triggering the neighboring transistor. A 4x4 up/down counter and 4x16 decoder do this automatically, receiving either a source (up) or ground (down) signal from the D-Flop in the upstream evaluation circuit.

Fig.1



a) Evaluation circuit. The Clamped Network wire connects to a node in the clamped network and the Free Network wire connects to the corresponding node in the free network.



b) One 16-step 100kΩ digipot. Each transistor is connected to the 4x16 decoder with a 7.3kΩ resistor to minimize the voltage drop (0.1V).

Implemented together, these components act as a single edge in the network. As a proof of concept, we constructed a simple three-edge sequential network by wiring three of these integrated circuits together (Fig. 2a). Since the network is sequential, meaning that each digipot is wired in series, a simpler voltage comparison scheme can occur where only the preceding pair of nodes, before their downstream resistors, need to be compared in order to determine the proper evolution for each digipot (Eq. 3). Hence, the comparators for each edge are wired to the appropriate free and clamped nodes of the network instead of implementing additional circuitry to measure voltage drop.

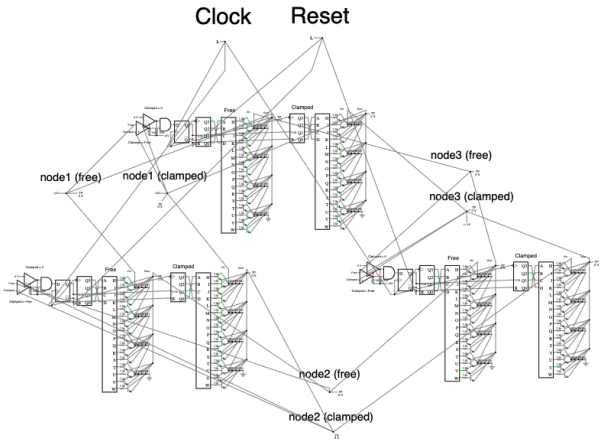
We trained the network by asking it to produce a single desired voltage $V^{Desired}$ = 1.5V at Node 3 (free) in Fig. 2a**,** while constantly holding input nodes Node 1 (free) and Node 1 (clamped) at 5V. Here we implemented Eq. 1 with $n = 1$, meaning that V_c^{Output} = $V^{Desired}$ = 1.5V and remained constant throughout training steps. After four training steps (four clock cycles), V_f^{Output} approached 1.5V, oscillating above and below $V^{Desired}$ by an average of +/- 0.263V (Fig. 2b). This error value is dependent on the quality of digigpots used; in a comparable network and task with a 128-step digipot this value was approximately +/- 0.1V [5].

Secondly, a five-edge network was simulated (Fig. 2c), where certain nodes are connected multiple times. This added significant density, and challenged the network to minimize energy dissipation as different potentiometers evolved in different directions, which mimicked the complexity of the 16-edge model which was our goal to simulate. Although the network was more complex, the same learning rule (Eq. 3) was implemented, where the voltage of each node was being compared instead of the voltage drop. This caused a redundant voltage comparison (Fig. 2c), as there were more edges than nodes. Note that on which node the redundant voltage comparison occurs was chosen completely at random.

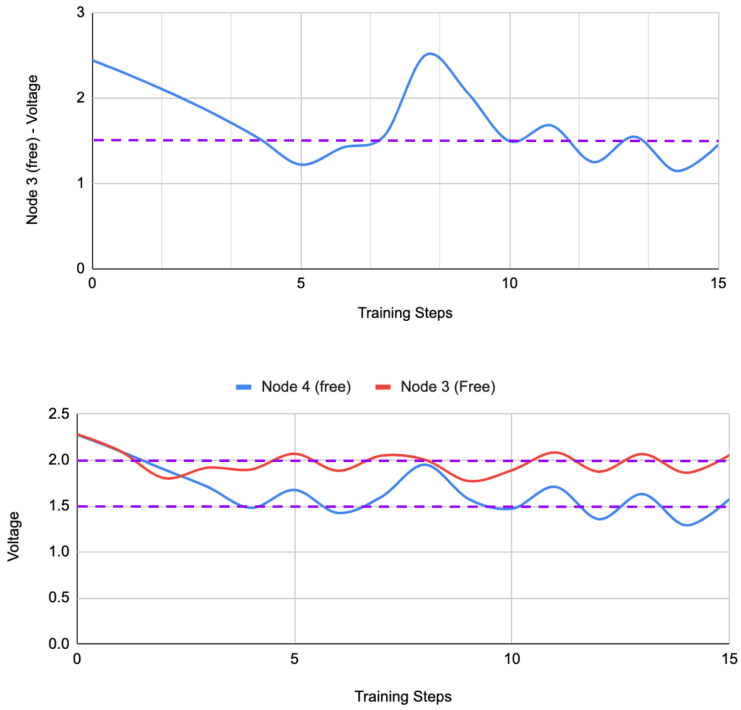
The same procedure as we used for the three-edge network was followed; however, out of the additional fourth nodes, Node 4 (clamped) $V_c^{Output\ (2)}$ was clamped at 2V $V^{Desired(2)}$. After three training steps, V_f^{Output} and $V_f^{Output\ (2)}$ reached $V^{Desired}$ and $V^{Desired(2)}$ respectively. The average error flow that followed was +/- 0.14V (Fig. 2b).

The arduous nature of building out each edge, which each contain approximately 40 components, becomes too complex for Falstad to simulate once the network exceeds approximately 8 edges. We considered dropping each edge of its evaluation circuit and having a centralized one instead (Fig. 2d), but ultimately concluded that this would minimally limit complexity while severely slowing down the entire system, as each edge would have to be evaluated one by one which is contrary to the purpose of the contrastive algorithm. A useful feature of certain circuit simulation softwares is the ability to perform parameter sweeps, where a certain value of a component, say the potential gradient (K) of a potentiometer, can be cycled through a given range at a specified step size. In other words, a simple potentiometer can be given digipot-like properties (i.e. the ability to update its resistance automatically) by being able to control its potential gradient in simulation. This essentially exchanges our 40-component part in Falstad with a single potentiometer, solving the problem of having a model that is too large. Falstad does not provide this functionality, so we instead transition to building models in Circuitlab.

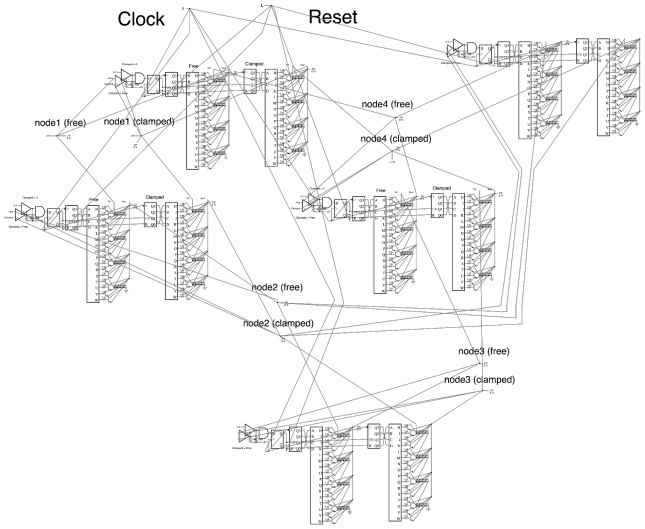
Fig. 2



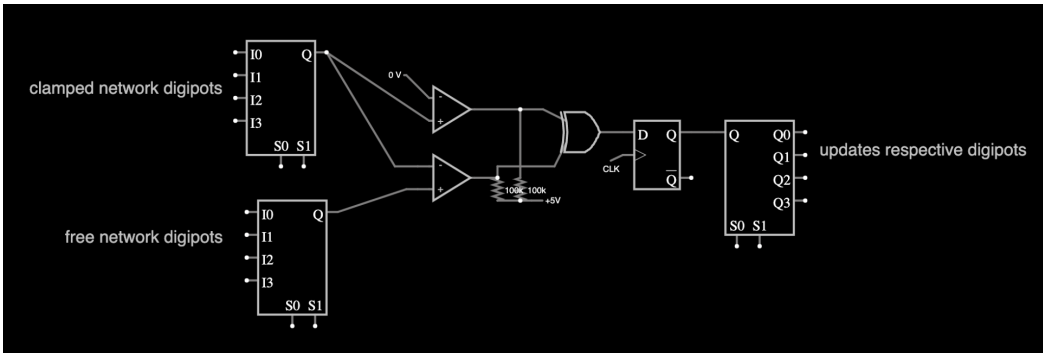
a) A three-edge network constructed in Falstad circuit simulator. All the circuitry required for one edge is integrated into a compact component. Each component has its two comparators wired to the appropriate free and clamped nodes before it (clockwise).



b) The graph on the top is the voltage of Node 3 (V_f^{Output}) in the free network updated every clock cycle for the three-edge network. The graph on the bottom is the voltage of Node 3 and 4 ($V_f^{Outputs}$) in the free network updated every clock cycle for the five-edge network. The dotted lines in each graph show the respective $V^{Desired}$ for each node. Once V_f^{Output} reaches ~$V^{Desired}$, V_f^{Output} oscillates indefinitely.



c) On the left is the five-edge network in Falstad. The edge circuit on the top right is the redundant edge, and it samples the voltage of Node 2.



d) Conceptual design of a centralized evaluation circuit. Multiplexers are attached to each node of the clamped and free networks, allowing for each pair of nodes to be cycled through the base comparator circuit, and subsequently each digipot updated through another multiplexer.

Our first goal in Circuitlab was to simulate the circuits we had built in Falstad. We started first with the three-edge sequential model, wiring three 100kΩ potentiometers in series for each network (Fig. 3a). Since we no longer had the evaluation circuits to compute Eq. 2/3 automatically, an extension program was developed in Python that automatically determined which direction each potentiometer should evolve in (Appendix B). A notable difference is that this program compared the power dissipations of corresponding potentiometers instead of the

voltage drops across them. Thus, the learning rule was instead,

$$\Delta R_i^C = \Delta R_i^F = \begin{cases} \uparrow R & if |P^C_{potentiometer}| > |P^F_{potentiometer}|, \\ \downarrow R & otherwise. \end{cases}$$

(Equation 4)

where $P^C_{potentiometer}$ and $P^F_{potentiometer}$ represent the respective power dissipations of a free and clamped edge pair, and the updated resistance is denoted by $\uparrow R$ and $\downarrow R$ to indicate that once the direction of the resistance evolution is determined by Eq. 4, it does not change, unlike with the previous learning rule (Eq. 2). Hence, while Eq. 4 adopts contrastive learning, it does not update in infinitesimal nudges and therefore is not equilibrium propagation like Eq. 2. The power dissipation metric used in Eq. 4 is analogous to measuring the voltage drop over an edge done in Eq. 2.

Once the directions for the resistance evolutions for each potentiometer were found, each potentiometer was manually set with one of the following expressions:

1. K = -R → Voltage up
2. 1-K = +R → Voltage down
3. s = 0.5 = R → Voltage stays the same

K is the parameter that gets sweeped from $K_{Current}$ to 1 at an adjustable step rate, S. At the start of training, $K_{Current}$ starts off as 0.5, $K_{initial}$, which means all edges are initialized at the center of their resistance ranges (~50kΩ). As K is sweeped, Exp. 1 follows the same trajectory, Exp. 2 changes inversely, and Exp. 3 remains constant as illustrated in Fig. 4. These three expressions give us all the characteristics needed to model the change given by Eq. 4: increase resistance (Exp. 1), decrease resistance (Exp. 2), or keep resistance the same (Exp. 3). Exp. 3 gives us a new feature, no resistance change, that we were unable to implement in the Falstad model.

As shown in our Falstad simulations of the five- and three-edge networks (Fig. 2b), once V_f^{Output} reached ~$V^{Desired}$, V_f^{Output} fluctuated up and down by a slight amount as the digitpots entered a loop of concurrent up and down signals. In the Circuitlab model this did not happen; once the power dissipations of corresponding edges were the same, they stopped evolving and stayed at that same value. Likewise, the noise was significantly better. In addition, the adjustable rate gave us added flexibility, allowing us to model the step size of virtually any digital potentiometer, and even get logarithmic or pseudo-logarithmic spacings of the resistance values, which can improve network flexibility and reduce error [5], [M. Stern].

Once each potentiometer was set to the appropriate expression, a parameter sweep which plotted K with respect to the voltage of the free network output node(s) $\mathbf{V_f^{Output}}$ was run, and the value of K that satisfied $V_c^{Output} = V_f^{Output}$ was updated to $K_{Current}$. This process was repeated $1/n$ times until $V_c^{Output} = V_f^{Output} = V^{Desired}$.

We confirmed the success of this network's implementation in Circuitlab by applying the same input and output values used for the three-edge network in Falstad. An important difference is that the procedure in Circuitlab compares the power dissipation between states

versus Falstad which compares the node voltages; however, this should not matter since the network is sequential. In fact, we confirmed this does not have an impact by modifying our procedure to compare between the voltages of corresponding nodes as well. In both situations, we were able to achieve ~98% accuracy with a step rate of S= 0.03125 which matched the step size of the digipots used in Falstad (128 steps). The step rate in order to match a certain step size can be found using the equation,

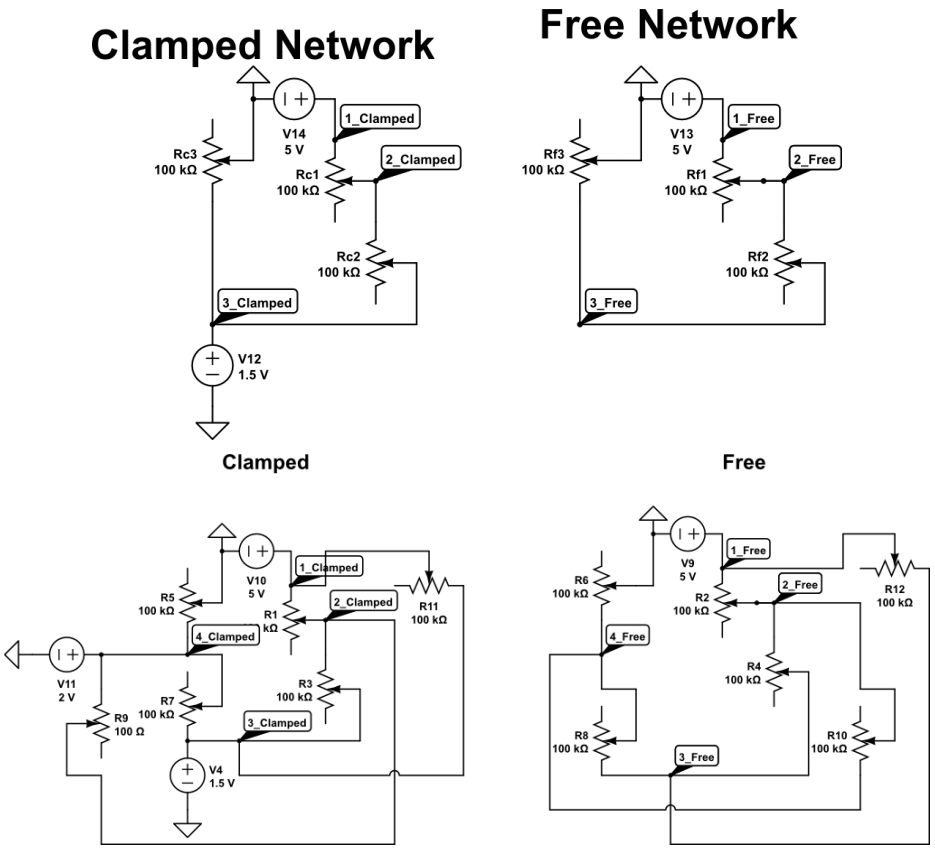
$$S = \frac{K_{max} - K_{initial}}{r_{desired}}$$

(Equation 5)

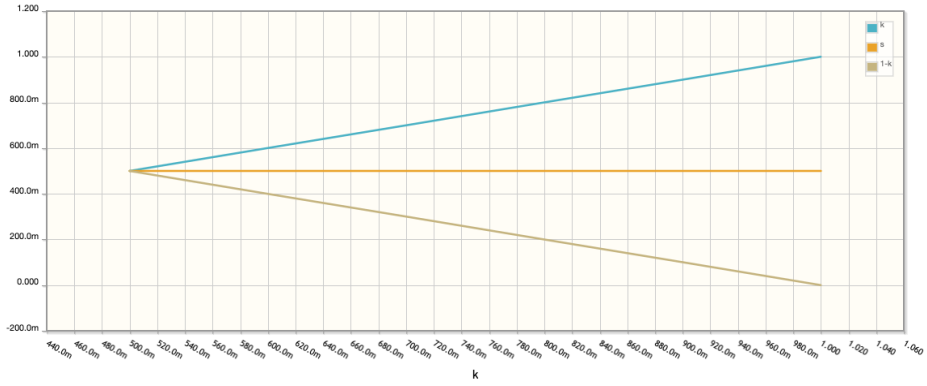
where r is the desired amount of steps to be modeled. A smaller S will yield a higher quality, as the potentiometer (or digipot) cycles through more possible values. We demonstrated this with S = 0.0001, and are able to achieve ~100% accuracy with the same network (Fig. 3c).

Having established a method for implementing contrastive learning in Circuitlab, we then constructed the more complicated, five-edge model to test the method's robustness. Similar to its Falstad implementation, the network had five-edges and four output nodes in the exact same configuration; however, this model did not have a redundant voltage comparison because the contrastive algorithm was implemented differently. We performed the exact same procedure as with the five-edge network in Falstad, and got a ~61.25% accuracy. We observed that Node 3 of the network had trouble reaching the range of $V^{Desired}$(1.5V), which created large discrepancies between the values of K that satisfied $V^{Desired}$ for both nodes (Fig. 3d). This was likely due to limited complexity associated with the network, as it had trouble finding an adequate value for K that satisfied both nodes.

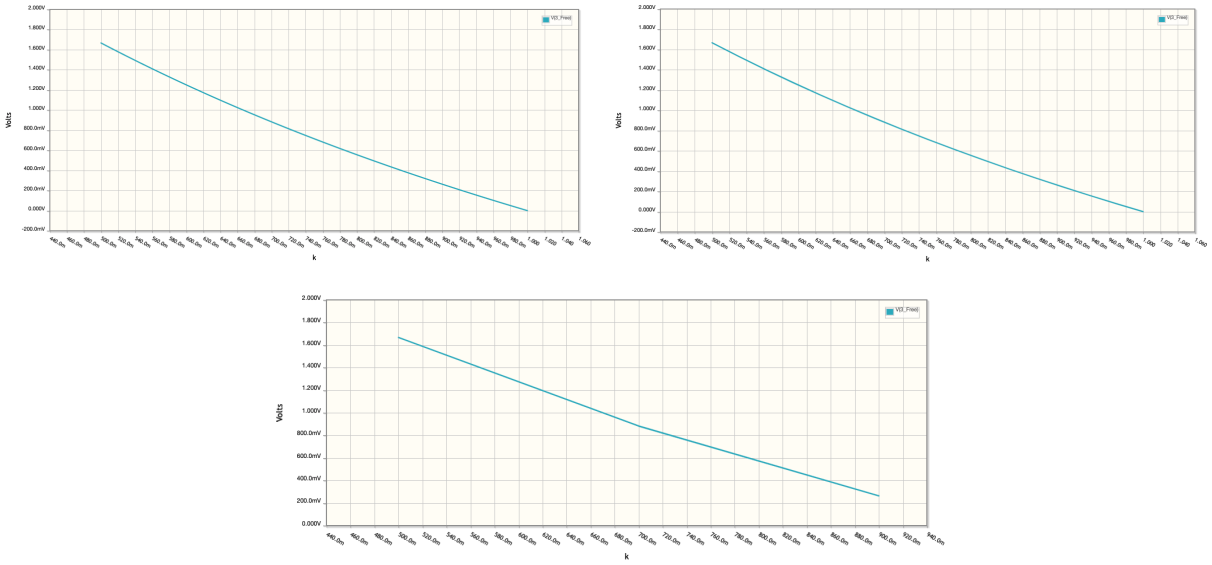
Fig. 3



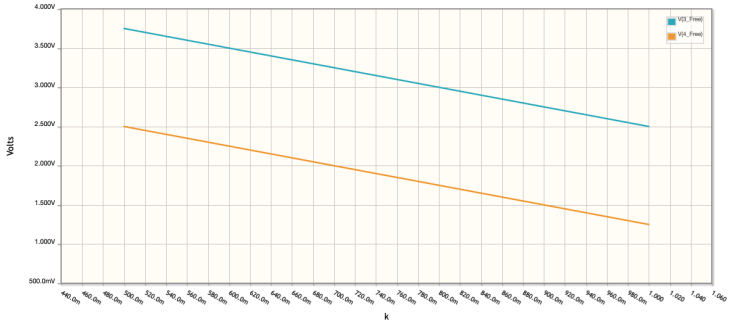
a) The 3 (top) and 5 (bottom) edge networks in Circuitlab. Here we use potentiometers instead of digipots, which are existing components in the Circuitlab library. Thus, the model is far simpler, and we implement the free and clamped networks separately, side by side.



b) The behavior of the three expressions. Plot of K versus the output of each expression.



c) The top left and right plots are of the voltage of Node 3 in the three-edge free network with respect to K. The left plot is with S= 0.03125 and the right plot S= 0.0001. The accuracy improves by ~2% with the smaller step rate, as the graph is higher resolution with more discrete values for K and the voltage of Node 3. Visually, this does not seem that apparent; the third graph with S= 0.2 provides a better contrast.



d) Graph of the third and fourth node voltages in the five-edge simulation. As described previously, Node 3 has trouble entering the proximity of $V^{Desired}$ (1.5V) at all, while Node 4 reaches $V^{Desired}$ far earlier.

5. Experiments

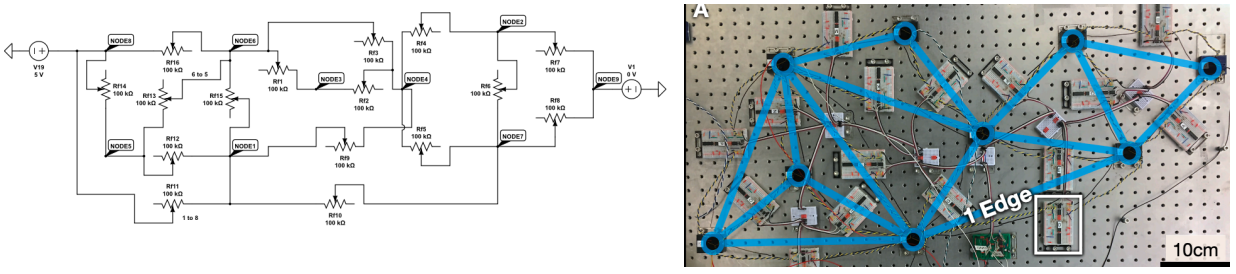
Our initial goal was to simulate the 16-edge network realized by Dillavou et al [5]. [5] trained the network to successfully perform three tasks: allostery, regression, and classification. We started with the most straightforward task, allostery, which is inspired by a biological feature of proteins where a molecule may bind to a certain region of an enzyme, influencing the shape or function of it in some other region [5], [12]. In ‘flow allostery,’ a pressure drop across input

arteries allows the vascular system to deliver blood flow and more oxygen to other parts of the brain by causing pressure drops in other locations, influencing output locations which can be far away from the input arteries [13]. [5] mimicked allostery in an electrical network, by asking the model to produce output voltages at specified nodes in response to given input voltages imposed at other nodes.

We constructed two 16-edge networks identical to the ones in [5] (i.e. same configurations of nodes and edges), and in our first set of experiments, followed the input specifications given by allostery tasks ii and iv (Fig. 4b). We started with task ii, where there is one input node imposed with 5V, another node held at ground, and a third output node clamped at 4V in only one of the two networks (the clamped network). Using a nudge of $n = 1$ and$\mathsf{S}$ = 0.03125 (128 steps), our network was able to deliver approximately 4V at the output node in response to an input voltage of 5V, with a precision of ~99.8%. In the same task, we were able to output exactly 4V volts by increasing the steps size$\mathsf{S}$ = 0.00390625 (256 steps). [5] used a nudge of $n = 0.5$ for all allostery tasks. We found that this had no effect on the accuracy of our model, so out of simplicity we instead used $n = 1$ for all allostery tasks.

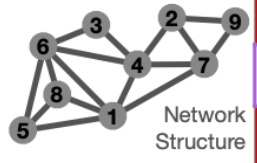
Next, we implemented the more complicated task iv, where there are two input nodes held at 5V and 1V, a ground node, and three output nodes clamped at 3V each. Using the same nudge and step size, we got an accuracy of ~93%, which is significantly lower than the high precision our model achieved on the previous task. We infer this is because of our Circuitlab model's unique learning algorithm (Eq. 4), which employs contrastive learning, but unlike the Falstad model and [5], does not use equilibrium propagation. In other words, achieving a higher accuracy in this case is not a matter of finding more discrete resistance values which we could easily implement by decreasing the step size further, but is instead an issue of finding a more unique set of combinations for the resistance values of each potentiometer, something not possible with this learning scheme. Ultimately, this is a major concern that we encounter multiple times while testing this model and is something that can significantly hinder accuracy. Expanding the size of the network may be a way to add the additional complexity needed to find better combinations to satisfy higher accuracies.

Fig. 4

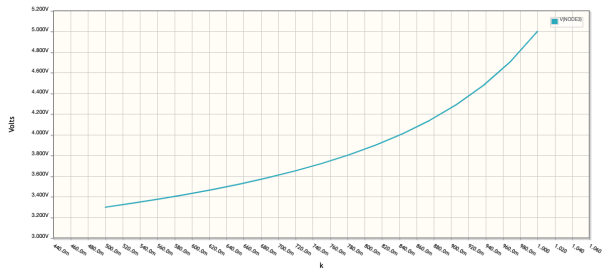


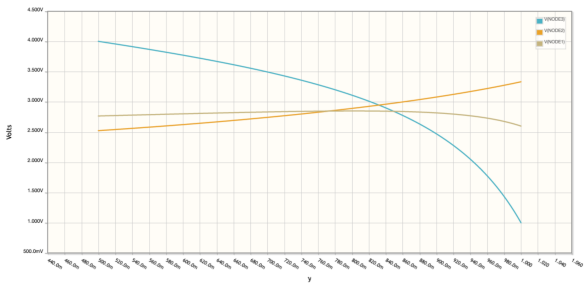
a) Our 16-edge network in Circuitlab compared to the 16-edge physical network realized in [5].

	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	2.5V
ii	Allostery				4V					5V	0V
iii	Allostery				4V	1V				2V	0V
iv	Allostery		3V	3V	3V	5V				1V	0V
v	Allostery		3V		2V		4V	5V	1V		0V



b) Voltage specifications for each allostery task and the classification task given in [5]. We only test Allostery task ii and iv and the classification task.





c) The graph on the left is of allostery task ii, and the one on the right is of allostery task iv. Allostery task ii achieves ~100% accuracy, while allostery task iv is only ~93% accurate. Upon visual observation of the graph, we noticed that two of the nodes reach $V^{Desired}$ at starkly different values of K, while one of the nodes, Node 1, has trouble reaching its $V^{Desired}$ at all. This discrepancy between results on the two allostery tasks is likely because task ii is less complicated (less input and output nodes), leaving the network with more empty nodes for additional processing. Task iv on the other hand is much more complex and has all but three nodes occupied.

After demonstrating our simulation's ability to model the 16-edge network allostery, we attempted to test it on the classification task. [5] uses a benchmark dataset of three species of iris flowers, and tasks the network with classifying these flowers based on input measurements of petal and sepal length and width. We used the same dataset [14] and randomly extracted 30 of the 150 flowers as a training set, separating the remaining 130 flowers into a test set. Four input nodes were designated for each measurement: one node for ground and three output nodes. Since the network is linear, we cannot designate one node for each class and train the network to produce a high value at the node corresponding to the correct class. Instead, a custom output scheme inspired by previous work [5] was utilized, where $V^{Desired}$ is determined every 30 training steps (one epoch) by the free network's response to the average input values from each species of flower in the training set. Therefore, as the network trains, $V^{Desired}$ changes but eventually settles on consistent values. The error can be calculated in between

training steps by running the entire test set through the model and comparing $V_f^{Outputs}$ with $V^{Desired}$. [5] was able to achieve over 95% accuracy on this task with $n = 1$.

Attempting this task in our simulated model demonstrated inherent bottlenecks in our procedure. As mentioned previously, since Circuitlab does not allow for statically evaluated parameters (e.g. potentiometers) to interface with plotting expressions in real analysis modes, we were unable to implement a system to automatically update the K values for each potentiometer after the power dissipation comparison occurs. Instead, for each training step in a classification task we had to follow this procedure:

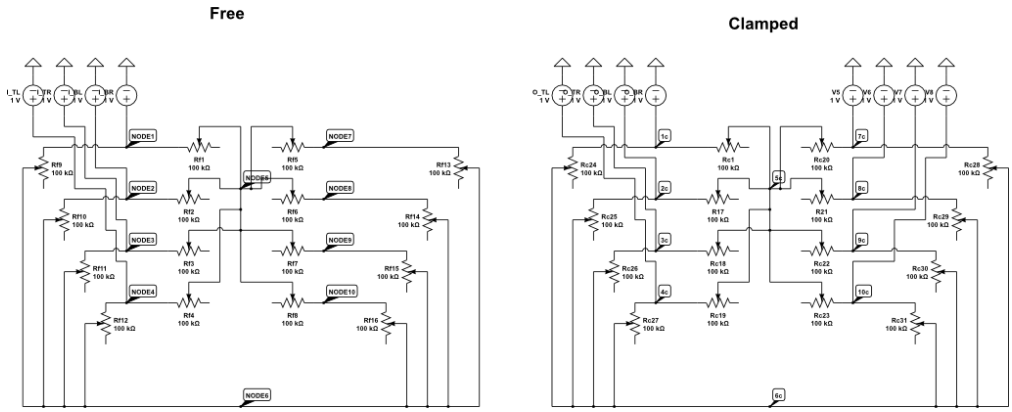
1. Find $V^{Desired}$ by imposing the free network inputs with class averages calculated from the training set. ($V^{Desired} = V_f^{Outputs}$).
2. Impose free network inputs with data points from the training set and measure $V_f^{Outputs}$.
3. Calculate V_c^{Output} for each clamped network output node via Eq. 1 with $V^{Desired}$ from step 1 and $V_f^{Outputs}$ from step 2. Impose $V_c^{Outputs}$ at the proper clamped network nodes.
4. Run the DC solver and export the csv to the external program.
5. Manually adjust a list of parameters (representing each potentiometer's resistance evolution expression) in Circuitlab according to the list generated by the external program. (This updates the direction of evolution for each potentiometer).
6. Run a parameter sweep and find the value of K that satisfies the $V^{Desired}$ for each free network output node.
7. Update $K_{Current}$. (One training step is completed).

In future work, this bottleneck may be reduced by implementing analog circuitry that automates the first three steps: following step 1 to calculate $V^{Desired}$ every epoch; step 2, a memory component that automatically feeds preloaded values from the training set into the free network inputs; and step 3, some sort of logic that calculates and imposes the $V_c^{Outputs}$. However, even this would not address the most significant source of inefficiency which occurs in steps 4 and 5, the manual work of updating each potentiometer according to the contrastive algorithm. Therefore, using an evaluation circuit at every edge (as done in [5] and in our Falstad implementations) provides a more optimal, unsupervised approach that is practical for training.

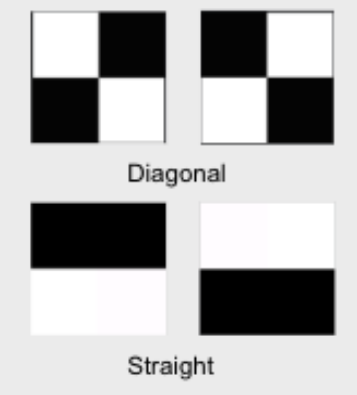
Lastly, we constructed a unique, CNN-inspired network to execute a simpler classification task. This classifier was tasked with differentiating between straight and diagonal lines, given a flattened 2x2 matrix of integer 'pixel' values represented as voltages. Fig. 5b illustrates the four possible input patterns the network could encounter and the data set distribution. We generated a dataset of 32 different input combinations, 16 per class, that contain one of the four patterns as an integer between zero to five and all the remaining values (not in the pattern) set at zero.

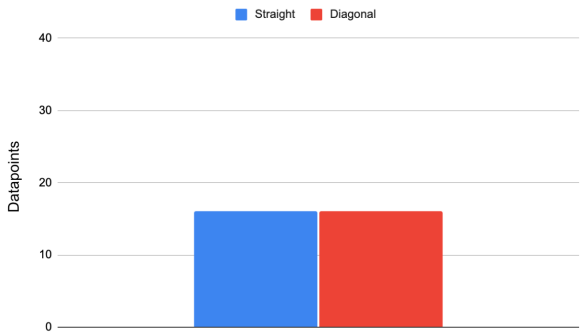
The network architecture was inspired by fully connected layers utilized in CNNs. According to Yoshua Bengio [15], hidden layer size should be between the size of the input layer and output layer. Thus, we constructed a model with four input neurons, two hidden layer neurons, and two output layers. As Fig. 3c depicts, every node in the input layer was connected to every node in the hidden layer, and the same applied for each node in the output layer. The same training procedure for the 16 edge network classification task was used including the nudge set to $n = 1$, and the same linear classification scheme qa also used to measure the outputs and error. Despite the significantly smaller dataset, the training process for this network was still very inefficient. Consequently, we ceased training the model after the fourth training step upon noticing behavior also observed in the Circuitlab simulation of the five-edge model and allostery task iv., where certain nodes in the network had trouble reaching $V^{Desired}$ or that the value of K that produced $V^{Desired}$ for each node differed dramatically (Fig. 5d). As deduced earlier when this issue was encountered, this is likely the result of limited complexity associated with the lack of equilibrium propagation in the learning algorithm (Eq. 4). Therefore, the next step would be to increase the hidden layer size of the network in an effort to increase the computational complexity. Although, it is worth noting that before training step four, the network was producing the desired behavior and training towards $V^{Desired}$ (Fig. 5e).

Fig 5.

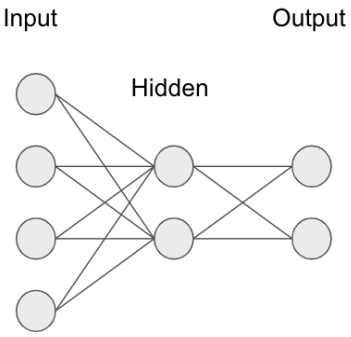


a) 16 edge ‘fully connected layer’ network.

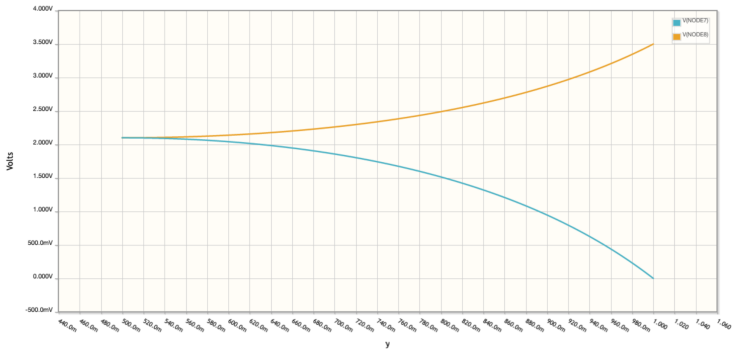




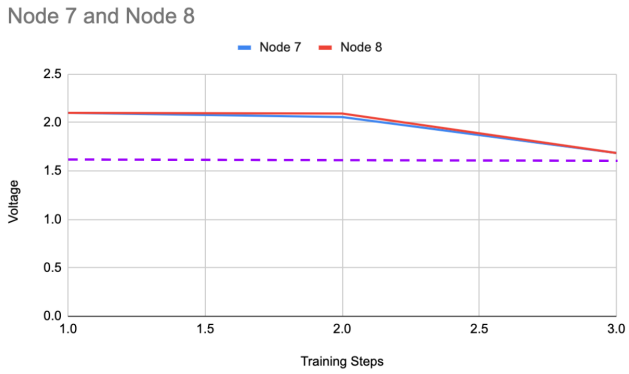
b) Two different input patterns for each class. Each class is represented equally in the data set.



c) Diagram of the network architecture inspired by the fully connected layer in a CNN.



d) At training step 4, we observed that the output node voltages started evolving in different directions, although both of the nodes $V_f^{Output} > V^{Desired}$.



e) Before training step 4, training is successful but very slow. The following graph depicts the voltages of each output node in the free network training towards $V^{Desired}$ (one epoch). $V^{Desired}$ is represented by the dotted purple line.

6. Conclusion

In this paper, we were able to simulate an avant-garde physics-driven learning circuit [5] that is difficult to prototype in its physical form. The system leverages physics to constantly adjust its node voltages (‘physical’ degrees of freedom), by nature minimizing energy dissipation, while its edge resistances (‘learning’ degrees of freedom) only adjust during training. Perhaps the most exciting aspect of this system is its computational ability [5]. Unlike in conventional neural networks where increasing the size of the network increases computation time, this system does not increase computation time when additional size is added because each edge acts independently, adjusting based off of local information, enabling the entire system to work completely in parallel [5]. In physical realizations [5], it has been demonstrated that this system can forwardly propagate and compute outputs within nanoseconds, while eliminating the computational complexity of backpropagation and the need for any processor or memory. On top of outpacing its ANN counterparts, this model of learning has many other applications such as in smart sensing, where its flexibility is highly desirable and provides a unique model of understanding for other neuromorphic systems [5], [2], [16].

In our simulations, we were ultimately able to determine that the use of equilibrium propagation is highly important in order to find discrete combinations of parameter values to satisfy the task at hand. Thus, simulation softwares that have the functionality to implement automatic contrastive learning circuitry (like Falstad) but also the computational capacity to render large models are desirable. However, the success of running the three-edge network and allostery task ii in Circuitlab proved that equilibrium propagation is not vital, and instead increasing the learning degrees of freedom of the network may suffice. In addition, simulation software that has the functionality to implement automatic contrastive learning locally on each

edge is imperative going forward to enable practical training speeds. Larger circuit simulations or physical realizations of these kinds of networks are necessary to better understand the optimal learning degrees of freedom required for a task of a certain complexity. Using the inherent advantages of circuit simulation software that we demonstrate in our research, future work could explore simulating similar networks with logarithmic resistance steps and implement elements of nonlinearity to the networks with diodes [5], [6]. All in all, our research was able to prove that modeling the complex nature of this decentralized, physics-driven learning system is possible in circuit simulation software, while shedding light on the relationships between task complexity and network size and performance.

Appendix

A.

Recent impacts to the global supply chain have dramatically influenced the price of the electrical components used in previous realizations of these networks. Below is a table that estimates the current cost of this network per edge based on the components used previously in [5].

Part ID	Name	Price	Qty
AD5220BNZ100	Digipot	7.4416	2
LM339AN	Comparator	1.18	2
SN74ALS86N	XOR Gate	0.651	1
CD74HC73E JK	Flip flop	0.577	1
ASD2009-R-T	Breadboard	2.97	1
Total		12.8196	

B.

All of the code generated for this project and all of the circuit schematics generated in Falstad Circuit Simulator can be accessed through this GitHub repository: https://github.com/hawkbsilleee/PioneerResearch.git.

C.

The circuit schematics designed for this project generated with Circuitlab can be accessed via the following links:

- Three-edge network: https://www.circuitlab.com/circuit/euj7ch648s83/autosave-2022-08-24t14_56_55_789z/
- Five-edge network: https://www.circuitlab.com/circuit/pv54vrcqua9q/5edge/
- 16 edge network for Allostery: https://www.circuitlab.com/circuit/q5ew26856z5k/allostery/
- 16 edge network for Iris Classification: https://www.circuitlab.com/circuit/2898d3qkkt95/16edge/
- CNN-inspired network: https://www.circuitlab.com/circuit/ygxzp87qgdd7/cnn4output/

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional Neural Networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] Wright, L.G., Onodera, T., Stein, M.M. et al. “Deep physical neural networks trained with backpropagation,” *Nature* 601, 549–555, 2019.

[3] V. Sze, Y.-H. Chen, T.J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[4] S. Roy, “Systolic matrix multiplier,” *Digital System Design*, 19-Feb-2021. [Online]. Available: https://digitalsystemdesign.in/systolic-matrix-multiplier/. [Accessed: 11-Sep-2022].

[5] S. Dillavou, M. Stern, A. J. Liu, and D. J. Durian, “Demonstration of decentralized physics-driven learning,” *Physical Review Applied*, vol. 18, no. 1, 2022.

[6] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] A. Tero, R. Kobayashi, and T. Nakagaki, “Physarum solver: A biologically inspired method of road-network navigation,” *Physica A: Statistical Mechanics and its Applications*, 363, 115 (2006).

[8] K. Alim, N. Andrew, A. Pringle, and M. P. Brenner, “Mechanism of signal propagation in Physarum polycephalum,” *Proceedings of the National Academy of Sciences* 114, 5136 (2017).

[9] J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, “Training End-to-End Analog Neural Networks with Equilibrium Propagation,” *Rain Nueromorphics,* 2022.

[10] J. R. Movellan, “Contrastive Hebbian learning in the continuous Hopfield model,” *Connectionist Models*, pp. 10–17, 1991.

[11] G. Yeap et al., "5nm CMOS Production Technology Platform featuring full-fledged EUV, and High Mobility Channel FinFETs with densest 0.021μm2 SRAM cells for Mobile SoC and High Performance Computing Applications," 2019 IEEE International Electron Devices Meeting (IEDM), 2019, pp. 36.7.1-36.7.4, doi: 10.1109/IEDM19573.2019.8993577.

[12] H. N. Motlagh, J. O. Wrabl, J. Li, and V. J. Hilser, “The ensemble nature of allostery,” *Nature*, vol. 508, no. 7496, pp. 331–339, 2014.

[13] J. Rocks, "Allosteric Functionality In Mechanical And Flow Networks," *Publicly Accessible Penn Dissertations* 3471, 2019.

[14] R. A. FISHER, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[15] Y. Bengio, "Practical recommendations for gradient-based training of Deep Architectures," *Lecture Notes in Computer Science*, pp. 437–478, 2012.

[16] F. Zhou and Y. Chai, "Near-sensor and in-sensor computing," *Nature Electronics*, vol. 3, no. 11, pp. 664–671, 2020.