Efficient Machine Learning in a Memristor Network

Monica Conte[1]

[1]*Soft Condensed Matter & Biophysics, Debye Institute for Nanomaterials Science, Utrecht University, Princetonplein 1, 3584 CC Utrecht, The Netherlands.*

(Dated: May 28, 2025)

Inspired by the brain's energy-efficient processing, we propose a physical implementation of artificial neural networks to surpass digital systems in speed and energy efficiency. We consider a network with nodes connected by conical microfluidic memristor channels filled with aqueous electrolyte [1]. These memristors exhibit tunable conductance, responsive to electrical, chemical, and mechanical stimuli, as well as channel geometry. Based on this adaptability, we develop a hybrid training algorithm that adjusts either channel geometry or external stimuli as weights. Physical relaxation replaces the feedforward pass, while a combined physical-numerical gradient descent drives the backward pass. This approach enables tasks such as input-output voltage mapping and linear regression.

I. INTRODUCTION

Modern computing algortihms are able to efficiently perform complicated nuemrical and symbolic operations, but their sequential structure limits their performance on more perceptual tasks. Inspired by the architecture and the functioning of the human brain that performs tasks via distributed and parallel fashion, artificial neural network (ANN) algorithms outreached traditional computing achieving remarkable success in challenging tasks ranging from image recognition to natural language processing and became central to many technological advancements in the field of artificial intelligence.

Recent advances in AI systems have been powered by the silicon-based computing power of the Graphics Processing Units (GPUs) and by an unprecedented abundance of data. With a clear trend of larger ANN models, their the energy requirements have grown significantly, negatively impacting their efficiency and scalability[**?** **?**]. With the rapid increase in computation requirements of AI training and the slower improvemnts in computing hardware traditionally predicted by Moore's Law, the necessity of new computing paradigms became more relevant. As an alternative to ANN performed strictly in-silico, physical neural networks (PNN) have been introduced, where physical processes rather than mathematical operations are trained[**?**]. Since physical processes can be faster and more energetically efficient than their digital representation, this direction constitutes a promosing research direction to address this issue.

Applications of PNNs have been explored across diverse fields, including optics [**?**], electronics [**?**], and fluidic and mechanical systems [], among others. Many existing training algorithms rely entirely on in silico methods, which often face challenges when transferring the learned model to real physical devices. To address this, hybrid training approaches have been proposed, where the inference phase is executed physically while the backward phase is computed numerically. Other methods, based on coupled learning, implement local learning rules at the level of each connection or edge. However, these approaches still face the need for two identical physical systems to apply different boundary conditions. In this work, we propose a novel algorithm in which both inference and backpropagation are carried out through physical processes, specifically leveraging the natural relaxation dynamics of the system in response to external stimuli and weight modifications.

A key advantage of PNNs is their ability to more closely emulate certain functional aspects of biological neural systems. One of the defining features of the brain is its capacity to process and integrate multiple types of stimuli—such as electrical, chemical, and mechanical inputs—simultaneously. This multimodal responsiveness is believed to contribute significantly to the brain's efficiency and adaptability []. Inspired by this, we focus on neural networks constructed from memristive elements. Specifically, we utilize ionic microfluidic channels in aqueous solutions[**?**], which exhibit the unique ability to respond concurrently to diverse physical stimuli. This characteristic is central to the operation of our proposed training algorithm.

The paper is organized as follows. Section 2 reviews related work on memristor networks, machine learning algorithms, and physical learning methods. Section 3 describes the methods used in our study, including the network model, machine learning algorithm, and the physical learning approach. In Section 4, we present our proposed method for coupling memristors with capacitors and discuss the necessary properties for effective training. Section 5 details our experimental setup and implementation. The results and analysis are provided in Section 6, followed by conclusions and suggestions for future work in Section 7.

II. MEMRISTOR NETWORK

The PNN we consider is an electrical network made of nodes connected by resistive elements. As resistive elements, we consider conical microfluidic channels, schematically represented in **??**. An azimuthally symmetric conical channel connects two reservoirs that contain an incompressible aqueous 1:1 electrolyte with viscosity $\eta = 1.01 mPas$ containing ions with diffusion coefficients $D = 1.75 \mu m^2 ms^{-1}$ and charge $\pm e$, with e the proton charge. The cone has a base radius of $R_b = 200$nm, a tip radius of $R_t = 50$nm and has length $L = 10\mu$m, which sets it in the long channel limit, where entrance and exit effects can be neglected. The walls of the channel are homogeneously charged and generate a surface potential ψ_0 that attracts the opposite-sign ions present in the solution due to the dissociation of the electrolyte. The electric potential decreases exponentially away from the surface due to the electrostatic screening of the attracted ions: an electric double layer (EDL) forms on the charged surfaces.

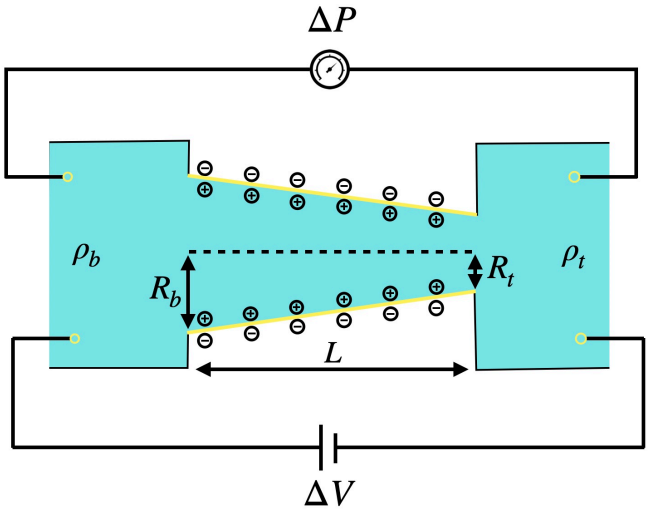

FIG. 1. A schematic representation of the conical fluidic memristor. A cone of lenght L, base radius R_b and tip radius R_t with charged surface connnects two bulk reservoirs of a 1:1 electrolyte in an acqueous solution. The bulk density in the base ρ_b differs from the bulk density in the tip ρ_t. At the far side of the memristor, an electrical potential and pressure drop are applied. Overall, this results in a chemical, electrical and mechanical sensistive system.

At the far side of both reservoirs, electrical potential, pressure and ion concentration are imposed. As a result, the memristor feels a potential drop $\Delta V = V_b - V_t$ defined as the difference between the potential in the base and in the tip. The ions within the EDL region generate an electro-osmotic fluid flow $Q_V = (\pi R_b R_t \epsilon \psi_0 / L\eta)\Delta V$. Meanwhile, the pressure drop $\Delta P = P_b - P_t$, given by the difference between the base and the tip pressure, induces a Poiseuille-like flow $Q_P = (\pi R_b^3 R_t^3 / 8 L \eta \langle R^2 \rangle)\Delta P$. Finally, the ions experience a concentration gradient $\Delta\rho = \rho_b - \rho_t$, defined by the difference between the ion concentration in the reservoir connected to the base of the channel, ρ_b, and the concentration in the reservoir connected to the tip, ρ_t. Consequently, a diffusive fluid flow arises. However, we consider only the pressure and potential contributions to the total fluid flow, $Q = Q_P + Q_V$, assuming the diffusive flow to be negligible. The resulting fluid flow generates non-trivial ion concentration profiles which, in turn, directly infulence the channel's conductance g.

In the steady-state, the channel is exposed to constant chemical, pressure and electrical signal, the corresponding conductance $g_\infty(\mathbf{\Delta})$ depends on the three stimuli $\mathbf{\Delta} = (\Delta V, \Delta P, \Delta\rho)$ and is determined by the ion concentration profiles according to

$$\frac{g_\infty(\mathbf{\Delta})}{g_0} = \int_0^L \frac{\bar{\rho}(x, \mathbf{\Delta})}{2\rho_b L} dx$$

where g_0 is the Ohmic conductance and $\bar{\rho}(x)$ the radially averaged salt concentration as a function of $x \in [0, L]$[**?**]. By solving the stationary condition of the total salt flux, an expression for the average salt concentration can be found and reads

$$\frac{\bar{\rho}(x, \mathbf{\Delta})}{2\rho_b} = 1 - \frac{\Delta\rho}{\rho_b}\mathcal{I}(x, \Delta V, \Delta P) + \frac{\rho_{in}(\Delta V)}{2\rho_b \mathrm{Pe}}\left[\mathcal{F}(x) - \mathcal{I}(x, \Delta V, \Delta P)\right] \quad (1)$$

where we defined, for the sake of simplicity and readability, two functions

$$\mathcal{I}(x, \Delta V, \Delta P) = \frac{e^{\frac{x}{L}\frac{R_t^2}{R_b R(x)}\mathrm{Pe}} - 1}{e^{\frac{R_t}{R_b}\mathrm{Pe}} - 1}, \qquad \mathcal{F}(x) = \frac{x}{L}\frac{R_t}{R(x)}.$$

Moreover, we introduced an expression for the concentration inhomogeneity

$$\rho_{in}(\Delta V) = 2\frac{(R_b - R_t)\sigma}{R_t^2}\frac{e\Delta V}{k_B T}$$

and the Péclet number $\mathrm{Pe} = QL/D\pi R_t^2$ which quantifies the importance of the flow Q over diffusion [**?**].

When the external stimuli vary dynamically in time, they generate ion accumulation and deplition in the channel, changing consequently the conductivity. We assume the dynamics of the time-dependent conductance $g(t)$ as a single exponential decay towards the steady-state conductance g_∞ with a diffusion-like timescale $\tau = L^2/12D$, as in Ref. [**?**]. The equation for the conductance reads

$$\frac{\partial g(\mathbf{\Delta}(t), t)}{\partial t} = \frac{g_\infty(\mathbf{\Delta}(t)) - g(\mathbf{\Delta}(t), t)}{\tau} \quad (2)$$

with $\mathbf{\Delta}(t) = (\Delta V(t), \Delta P(t), \Delta\rho(t))$. As in Ref. [**?**], we consider the time scale relative to voltage-driven ion accumulation or depletion in the channel. We justify this choice by stating that pressure-driven dynamics

is faster and recalling that we neglect chemically-driven transport. Due to the fact that the conductance depends on the history of the signals applied, this system is called memristor. Aside from the direct dependence of the conductance on external stimuli, there is also an implicit dependence on geometrical properties, such as length L and base radius R_b, that we address in this work.

The PNN is composed of N nodes, an example is shown in **??**. Each node n $\in [1, \mathrm{N}]$ is characterized by a value

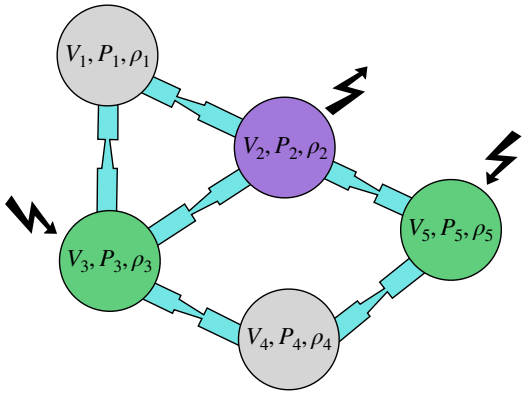

FIG. 2. An example of a network of five nodes connected by six memristors. The nodes are divided in input nodes (green circles), hidden nodes (gray nodes) and output nodes (purple circles) each of them is characterized by a triplet of physical quantities, representing the potential, pressure, and density. Th resulting stimuli on the memristor is given by the difference between base and tip signal, it thus depends on the oreintation of the memristor. In the case of the memristor that connects the first and the second node, for example, the exteranl stimuli are $\Delta V = V_2 - V_1$, $\Delta P = P_2 - P_1$ and $\Delta\rho = \rho_2 - \rho_1$. The zig-zag lines directed towards the input nodes indicate where the electrical signal is applied, while the zig-zag lines pointing outwards from the nodes indicate where the electrical signal is measured.

of electrical potential V_n, pressure P_n and bulk density ρ_n. Even though each node in the network is characterized by three physical properties, only electrical current flows through the network. This current propagates via the memristive elements, enabling modifications to the electric potential at other nodes. In contrast, pressure and density are externally imposed at each node and do not propagate through the network. The set of nodes is divided in thee sets. A node belongs to the set of *input nodes* if it is used to imput current in the circuit, its value of voltage is known and fixed. After the imputs are imposed, the electrical signal travels throught the network and the resulting output potential is read in the nodes belonging to *output nodes*. The remaining nodes, that do not function as input or output, are included in the set of *hidden nodes*.

The nodes in the PNN are connected by M memristive elements. Consequently, a memristor connecting two nodes experinces potential drop, pressure drop and concentration gradient given by the difference between the node where the base of the memristor is connected to, and the node to which the tip is linked. The memristor is characterized by its conductance state g_m with m $\in [1, \mathrm{M}]$, consequence of both the geomtery parameters and the external stimuli.

To solve numerically the resulting circuit, it is necessary to simultaneously address the circuit equations and the time evolution equation for the memristors' conductance. The circuit equations are handled using the open-source circuit solver AHKAB[], which computes the electric potentials at the nodes and the currents through the edges discretizing the differential equations in timesteps Δt. To incorporate the memristive elements, we approximate the time evolution of the conductances using a discretized form of **??**, employing the Euler-forward method. Specifically, the conductance at a given timestep is calculated from its value at the previous timestep as:

$$g_\mathrm{m}\left(\mathbf{\Delta}(t), t\right) = g_\mathrm{m}\left(\mathbf{\Delta}(t), t - \Delta t\right) + \frac{g_{\mathrm{m},\infty}\left(\mathbf{\Delta}(t) - g_\mathrm{m}\left(\mathbf{\Delta}, t\right)\right)}{\tau}\Delta t. \quad (3)$$

In this work, we only apply constant voltage supplies to the input nodes of the circuit. Nonetheless, we notice that solving the dynamical equations is needed due to the dynamical change in the conductance of the memristors that, in turn, results in time-dependent potential drops across them. A more detailed illustration of the algorithm that solves the circuit, along with an analysis of the relaxation to the steady state of the memristor network, is provided in **??**.

III. TRAINING ALGORITHM FOR A MEMRISTOR NETWORK

We developed a training algorithm that makes use of the physical dependencies of memristor conductivity on both external stimuli and geometric properties to implement a physical neural network. These dependencies allow us to interpret such physical parameters as tunable weights, analogous to standard artificial neural networks, and to encode learning directly into the physical structure of the system. As in traditional ANNs, the goal of the training algorithm is to modify these weights so that the network maps a given set of inputs to a corresponding set of desired outputs. In the case of the memristor network considered here, training aims to produce specific voltage values at designated output nodes, given fixed voltage inputs at the input nodes. Crucially, the physical quantities that influence memristor conductance fall into two distinct categories: external stimuli, such as pressure or concentration, that are applied to the nodes, and geometric parameters, such as channel length or base radius, that are defined on the edges. As a result, we distinguish between two types of training strategies: *node-defined weight training*, as illustrated in **??**(a), and *edge-defined weight training*, as shown in **??**(b).

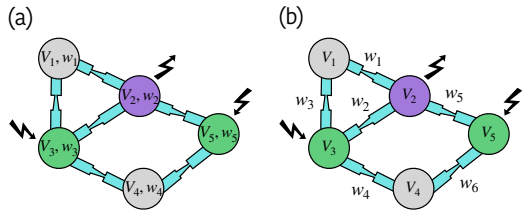

FIG. 3. (a) Memristor network with weights defined in the nodes. (b) Memristor network where wegiths characterize each edge.

A modification to any of these weights affects the network's behavior by altering the functional form of the memristor's steady-state conductance, $g_\infty(\boldsymbol{\Delta})$, which depends on the local potential, pressure, and concentration drops across the device. Importantly, the *entire functional shape* of g_∞ plays a role in determining the network's dynamics and ultimate steady state, as illustrated in **??**.

In this work, we focus on training a single property as the weight, while keeping all other parameters fixed as initially defined. For example, we may select an external stimulus, such as the imposed pressure or the density at each node. Alternatively, we can choose the length or base radius of the channel. The training process is carried out by iterating a scheme, illustrated in **??**, which consists of two phases: a forward pass and a backward pass. In the forward phase, the network is fed with specific electric potentials in the input nodes. Following both circuit laws and memristor's dynamics, the network reaches a steady state where electrical potentials can be measured in the output nodes. Clearly, the output of the network depends on the set of weights imposed on the network $\boldsymbol{w}$, regardless them being node or edge defined. The value of the outputs can be used to quantify the status of the network, meaning how much its answer differs from the desired one. Specifically, we define a cost function that computes the difference squared between the voltage in the output nodes and the values of desired voltages

$$C\left(\boldsymbol{w}\right)=\sum_{i\in\Lambda}\left(V_i\left(\boldsymbol{w}\right)-V_i^D\right)^2,$$

where Λ represents the set of indeces belonging to the output nodes, $V_i\left(\boldsymbol{w}\right)$ the voltage measured on the output nodes and V^D is a vector containing the values of desired output nodes.

The backward phase has the fundamental purpose of finding a set of weights that minimize the cost function. We follow the method of the *steepest descent*, for which the update of the weights from one training step s to the following $s+1$,

$$\boldsymbol{w}^{s+1}=\boldsymbol{w}^s-\alpha\boldsymbol{\nabla}C\left(\boldsymbol{w}\right), \tag{4}$$

results in a decrease in the cost funtion until a minimum is reached, for sufficienlty small values of α [**?**]. The gradient in the update rule is computed using a *finite differences method*

$$\boldsymbol{\nabla}C\left(\boldsymbol{w}\right)=\frac{C\left(\boldsymbol{w}+\boldsymbol{\Delta w}\right)-C\left(\boldsymbol{w}\right)}{\Delta w}, \tag{5}$$

where $\boldsymbol{\Delta w}$ is a vector with the same number of entries as the weight vector, but every entry has a value Δw.

In the training algortihm, we first initialize the system with the memristor parameters specified in **??**, an initial pressure of $1bar$ and concentration of $0.2mM$ at each node. With the knowledge of input and desired output voltage values, it is possible to performe the first forward pass, from which we obtain the cost function $C\left(\boldsymbol{w}\right)$, that we store since it is needed in the computation its gradient. Subsequently, we proceed with the backward pass. We compute each entry of the gradient in **??** by selecting the corresponding weight, a node or an edge parameter, increment its value of Δw and applying a forward step to obtain the corresponding entry of $C\left(\boldsymbol{w}+\boldsymbol{\Delta w}\right)$. With the stored value of $C\left(\boldsymbol{w}\right)$, we are able to compute the gradient in **??** and use it to calculate the weight update in **??**.

Althought performed numerically in this work, we highlight that the feedforward pass is a physical process in which the system natually relaxes to its steady state. Moreover, also the backward pass we use can be translated to a physical event since involves performing the forward pass a number of time as the number of the weights. Once the gradient of the cost function is determined and stored, only the simple algebric calculations of **??** are left to be computed. We propose this method as a hybrid physical and in-silico training algorithm, as a future experimental approach to the probelm.

A. Training a Memristor Voltage Divider

A voltage divider is an electrical circuit made of three nodes connected in series by two resistances, that returns a fraction of the input voltage. Given the simplicity of its structure, we use it as an initial playground for the training algorithm described above. Specifically, we replace the resistances with memristor channels, to be trained such that giving $V_1=5V$ and $V_3=0V$ as inputs, we get $V_2=4V$ as output. In **??** a schematic of the memristor voltage divider is shown.

We initialize the system with pressures $P_1=P_2=1bar$ and ion concetration $\rho_1=\rho_2=0.2mM$ on each node, length $L_1=L_2=10\mu m$ and base radius $R_{b1}=R_{b2}=200nm$ on each edge. After choosing one of these four physical quantities as training parameter $\boldsymbol{w}=(w_1,w_2)$, we implement the training as descibed in **??** that modifies them, while the other parameters are kept constant. In **??** we show that the cost function normalized to its initial value decreases of many order of magnitude during the training for each of the choosen weight. The values of the choosen set of weights simultaneouly changes from the initial condition, as shown in Figures [], adjusting the

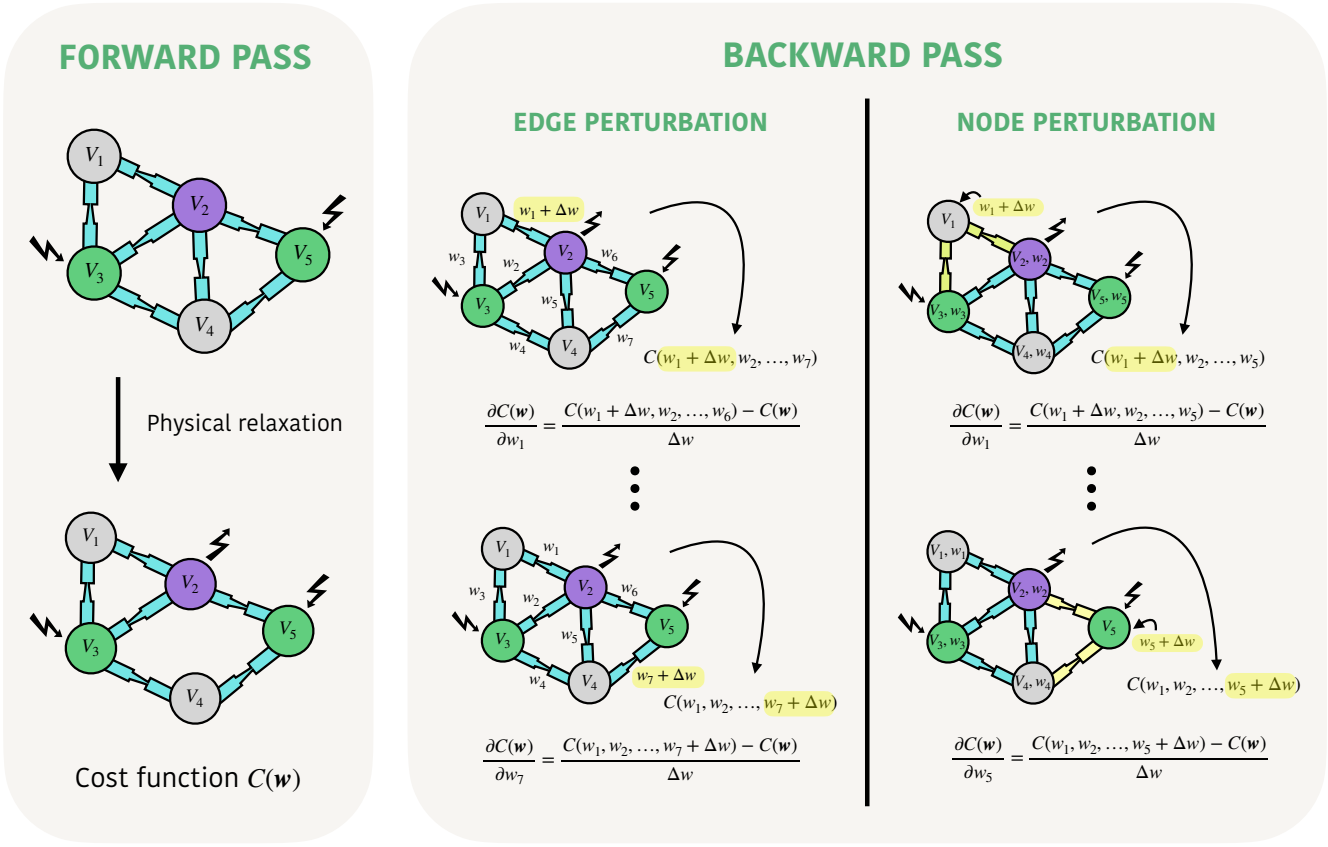

FIG. 4. Scheme representing the two phases of the training algorithm, they are iteraitvely performed for a selected number of training steps.

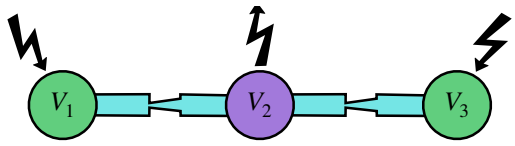

FIG. 5.

behaviour of the network to the desired one. If the values of a choosen type of weight at the end of the training are imposed to the netowork together with the voltage inputs, the network will autonomously realx to a staedy state that coincides with the desired state. To exemplify this point, the inset in Figure shows the physical relaxation of the voltage divider when the trained values of pressure, which correspond to the three values in the last training step, are imposed on the nodes together with the constant inputs voltages. The inset shows that the output voltage V_2 autonomously relaxes to the desired value of $V_D = 4V$ in a time scale characteristic of the memristor.

In order to investigate the adaptability of the algorithm, we trained the system with a set of three consequent desired voltages $V_D = \{1V, 3V, 4V\}$. In **??** we show how the output voltgage V_2 subsequently adapts to the desired voltages during the training of the pressures in the nodes, the evolution of which can be seen in **??**. It is evident that the system can be retrained at any stage to a new desired objective making it reusable. Any other weight type can be chosen, depending on the necessity.

B. Training a memristor network

To generalize the training objective, we consider a memristor network constructed with a more complex geometry. **??**(a) illustrates a network with three input nodes and two output nodes. The network is trained to produce output voltages of $[3, V, 4, V]$ when provided with input voltages of $[0, V, 2, V, 5, V]$. Reaching a desired input-output voltage mapping in a generic electric circuit can be useful for variuous electronic purpuses [**? ?**].

The result of the training is presented in **??**(b), which shows how the cost function responds to the wieght updates performed in the algorithm, for each of the four different weights. All of the possible different choices of weight result in a trained set that decreases the initial error of many orders of magnitudes.

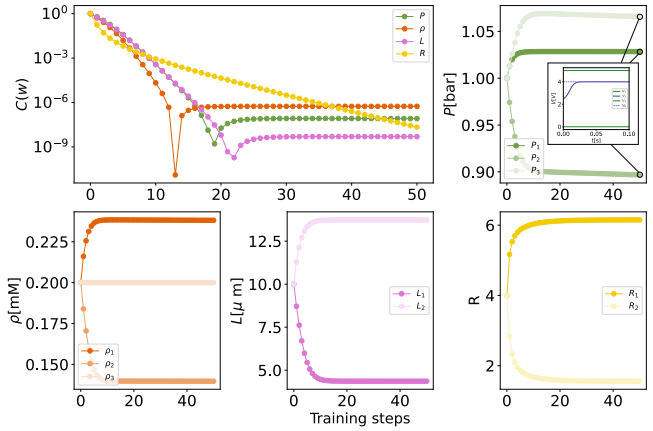

FIG. 6. (a) Behaviour of the cost function during the training of the voltage divider. Different colors indicate that the training is done with a specific choise of training weights, as indiacted in the legend. (b) Pressures imposed on the three nodes of the voltage divider during training. The inset shows the relaxation to the steady state of the output voltage V_2 to the desired value V_D when the pressures imposed are the values obtained in the last training step, and the input nodes are stimulated with fixed voltage. (c) Ion concentration desnities imposed on the three nodes of the voltage divider during training. (d) Length of the memristors during training. (e) Base radius of the memristors during training.

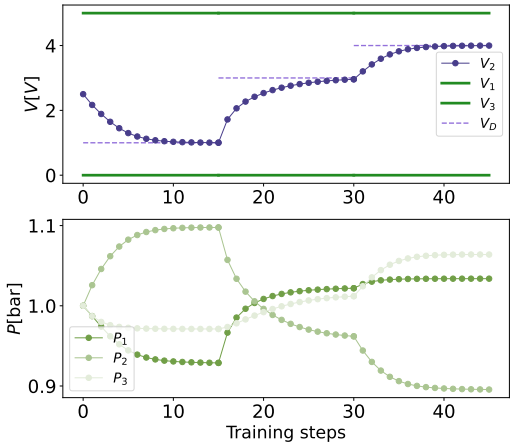

FIG. 7. (a) The trainiang of a memristor voltage divider to a set of desired voltage outputs $V_D = 1V, 3V, 4V$, indicated by dashed light purple lines. The output voltage V_2, indicated by dark purple dots, adapts to the desired output during the training. The input voltages V_1 and V_3 are constant at $5V$ and $0V$ respectively. (b) The dots with different shades of gray show the values of pressures, chosen as weights, imposed on the three nodes during training.

C. Linear Regression

We broaden the training capability of the network by including a linear regression task, where, in its simplest form, the network is trained to give potential output V_{out}

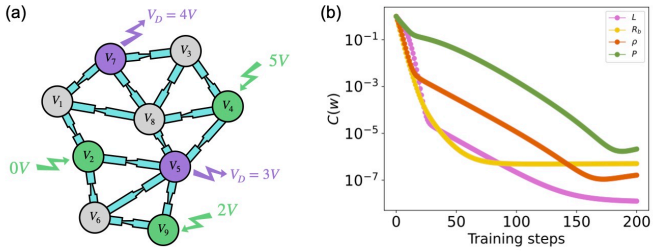

FIG. 8. (a) Generic network with 9 nodes and 15 edges with inputs $[V_2, V_4, V_9] = [0V, 2V, 5V]$ and desired outputs $[V_5^D, V_7^D] = [3V, 4V]$, delta weights a third of their intial value and learning rates $[0.001, 1, 0.0001, 0.001]$. (b) Cost funtion during training of generic graph for input-output mapping task. The different colors indicate different weight choice.

in a certain node of the network, related to the input potential V_{in} via a linear relationship

$$V_{\text{out}} = \mathrm{a} V_{\text{in}} + \mathrm{b}. \tag{6}$$

We generate two sets of data, a training set $\{V_{\text{in}}^{(j)}, V_{\text{out}}^{(j)}\}_{j=1}^{N_{\text{train}}}$ and a test set $\{V_{\text{in}}^{(j)}, V_{\text{out}}^{(j)}\}_{j=1}^{N_{\text{test}}}$, where N_{train} and N_{train} denote the number of training and test samples, respectively. To create the training set, a sequence of input voltages is randomly sampled from the interval $[1V, 4V]$, and the corresponding output voltages are computed using equation **??**, ensuring that the input-output pairs satisfy the desired relationship. The test set is constructed in a similar way, but uses $N_{\text{test}} = 20$ input values that are evenly spaced within the same interval. Their corresponding outputs are also computed using equation **??**. Note that the training and test sets contain different input values.

The training algorithm described in **??** can be slightly adapted to support this training objective. At each training step, an input-output voltage pair is selected from the training set and used as the input voltage and target output in the algorithm. This differs from the original algorithm, where the same input-output pair was presented at every step.

Therefore, at each training step $j \in [1, N_{\text{train}}]$, the cost function used to compute the gradient in **??** is adapted to

$$C_j(\boldsymbol{w}) = \left[V_{\text{out}}(\boldsymbol{w}) - V_{\text{out}}^{(j)}\right]^2,$$

where $V_{\text{out}}(\boldsymbol{w})$ is the measured output voltage for the selected input, and $V_{\text{out}}^{(j)}$ is the corresponding target output in the training set. This expression corresponds to a simplified version of **??** for the case of a single output node.

To evaluate how well the algorithm performs during training, we compute the *test cost function* C_{test} over the entire test set. If $\boldsymbol{w}$ denotes the weight vector at a given

training step, the test cost is defined as

$$C_{\text{test}}(\boldsymbol{w}) = \sum_{j=1}^{N_{\text{test}}} \left[V_{\text{out}}(\boldsymbol{w}) - V_{\text{out}}^{(j)}\right]^2,$$

which sums the squared differences between the measured and target output voltages at each test point. For each input $V_{\text{in}}^{(j)}$ in the test set, the voltage $V_{\text{out}}(\boldsymbol{w})$ is the output produced by the network with weights $\boldsymbol{w}$, while $V_{\text{out}}^{(j)}$ is the desired output. This test cost is used for performance evaluation and is not involved in the training process.

Since the task is more demanding than a single input-output mapping, we use a larger network with more degrees of freedom. **??**(a) shows the graph selected for the regression task. It consists of 20 edges and 10 nodes, three of which are inputs and one is an output. Among the three input nodes, one is set to ground and another to a constant value, in order to enable the training of a linear function with an offset. For the training objective, we aim to train the network to fit the line $V_{\text{out}} = 0.2V_{\text{in}} + 0.3$ over the input voltage range $[1V, 5V]$.

The performance of the algorithm on this task is shown in **??**(b), where the learning rates for the different weights have been fine-tuned to prevent overshooting while still maintaining comparable convergence speed. From the behavior of the cost function, it is evident that the choice of weight type has a decisive impact on the network's ability to learn a complex task. Specifically, training the lengths of the channles on the edges results in successful convergence, whereas training the node pressures proves to be extremely inefficient. For a visual interpretation of the regression cost function, **??**(c) shows the state of the training at three different time steps.

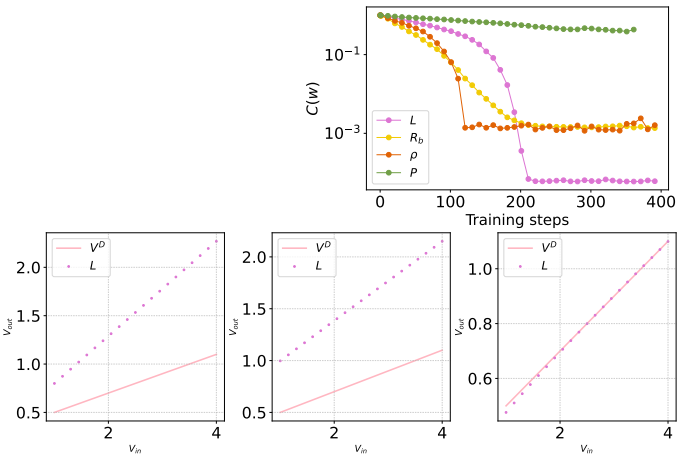

FIG. 9.

To study the influence of memristor orientation on training performance, we retained the same network geometry but inverted the orientation of some memristors, as shown in **??**(a). Specifically, if in the original configuration the base of a memristor pointed to one node and the tip to another, we reversed this direction.

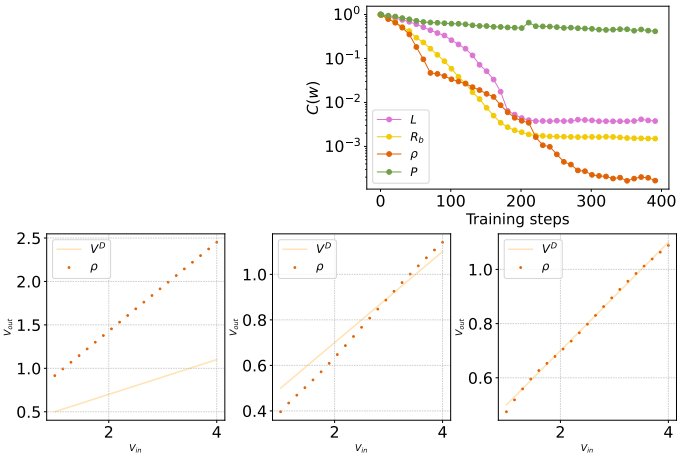

FIG. 10.

This modification significantly affects the behavior of the network. As shown in **??**(b), the new orientation alters the trainability of different weight types: in this configuration, training node concentration leads to better performance, while training edge length becomes less efficient. The underlying reason is that inverting the orientation of a channel also inverts the sign of the external stimulus applied to it, since the stimuli are defined node-wise and the difference is always taken as "base minus tip."

This reversal changes both the shape and the conductivity range over which the function $g_\infty(\boldsymbol{\Delta})$ is defined, thereby limiting or enabling the conductivity values the channel can attain. A network with well-oriented channels can therefore access conductivity ranges that make the training solvable, while a poorly oriented network may be constrained to regions where solutions are not found. To exemplify this point, Figure (X) of Ref. (Y) shows g_∞ as a function of pressure for positive and negative potential drops, assuming a zero concentration drop. The plot reveals two completely disjoint conductivity ranges depending on the sign of the potential drop, which explains why training on pressure alone may be challenging. This highlights the need for deeper exploration of network geometry when selecting the type of weight to train. In this regard, we observe that the edges whose orientation was inverted are directly connected to either the input or the output nodes. This suggests that particular attention should be given to these edges in future explorations, as they may have a bigger impact on trainability. As for the length-based training, **??**(c) illustrates how the network responds to the test set at three different stages of the training process.

Bigger networks have more degrees of freedom, and should allow for a bigger range of exploration of a solution. However, the experimantal realization would become more challenging. In this work, we show that an adequate geometry for a specific training goal can be sufficent to make a small network trainable. A deper analysis on the influence of geometry on the trainabiliy is needed

in the research contest of physical learning and can be explored in future research.

Larger networks offer more degrees of freedom and can, in principle, enable broader exploration of the solution space. However, their experimental realization becomes increasingly complex. In this work, we showed that an appropriately chosen geometry adapted to a specific training objective can be sufficient to make even a small networks trainable. A deeper analysis of the role of geometry in determining trainability is still needed in the context of physical learning.

IV. CONCLUSIONS

We proposed a physical neural network composed of nodes connected by aqueous memristors. These devices exhibit conductance that responds to external stimuli of various physical natures, such as electrical, mechanical, and chemical. Additionally, their conductivity is sensitive to changes in the geometric parameters of their channel-like structure.

Our motivation for using these devices lies in the ability to exploit these dependencies to control the conductance of each memristor in the network. This enables us to manipulate an input electrical signal in a programmable way, according to a specific computational objective.

To this end, we developed a training algorithm that targets one of the physical parameters influencing conductivity and treats it as the analog of a weight in a standard machine learning framework. The electrical signal is used as both input and output, while other types of signals or geometric dependencies are used to adjust the network during training.

The training process consists of two phases. The first is an inference phase, in which input voltages are applied to the system and the circuit is allowed to physically relax to a steady state. At this point, the output voltages can be read. Ideally, these outputs should match a predefined set of target values reflecting the training goal. In the second phase, the update phase, the weights are adjusted to minimize a cost function based on the difference between the actual and desired outputs. By iteratively applying this procedure, the network is gradually tuned to achieve the desired input-output behavior.

The inference phase is entirely physical, as the system evolves naturally to its steady-state response when inputs are applied. In contrast, the update phase is only partially physical. The weight update relies on estimating the gradient of the cost function using a finite-differences method, following a steepest descent rule. Specifically, each weight is perturbed slightly, and the resulting change in the cost function is computed. The gradient is then approximated by taking the difference in cost between the perturbed and unperturbed configurations, divided by the magnitude of the perturbation. While an external computer is required to store and coordinate the gradients for each weight, the values themselves can be obtained through physical measurements of the system's output under the two conditions. This structure allows for a mostly physical implementation of learning, with digital support.

As a first simple application, we tested the algorithm on a voltage divider, training the network to produce a specific fraction of the input voltage at the output node. The training was successful across all four selected weight types: pressure drop, concentration gradient across the memristor, length of the memristor, and its base radius.After selecting a particular weight type, we further demonstrated that the network, once trained to achieve a specific objective, could be retrained to a different target with similar training speed. This result indicates that the algorithm is both adaptive to new training objectives and largely independent of the initial weight configuration.

Training larger and more complex memristor networks is also feasible with the algorithm we proposed. We demonstrated that a network composed of 9 nodes and 15 edges can be successfully trained for input-output voltage mapping using all the weight types considered.

We then challenged a more complex network, with 10 nodes and 20 edges, to perform a linear regression task with an offset. In this setting, we observed that the trainability of the network using a specific weight type strongly depends on the network's geometry. First, we showed that different geometries yield different final training accuracies when using the same weight type. Then, keeping the geometry fixed, we demonstrated that the orientation of the memristors also plays a crucial role: reversing the direction of just three memristors connected to either input or output nodes improved the training precision when using concentration as the trained weight by three orders of magnitude.

Given that the expressiveness of deep neural networks grows exponentially with depth[**?** **?**], we interpret the limitations encountered in training linear regression tasks as a consequence of the small size of the network. However, small-scale networks are experimentally attractive, and in such cases, a training error of a few orders of magnitude may be acceptable. A key challenge for scaling such networks lies in the physical control of individual weights, especially when the weight type involves mechanical or geometric parameters. Emerging technologies in microfluidics, soft robotics, and programmable matter may provide practical tools to overcome these limitations and scale the network while preserving control.

The core intention behind the algorithm is to enable a physically realizable neural network that requires minimal computational resources to operate. Thanks to the multiple physical dependencies of memristors, such as electrical, chemical, mechanical and geometrical factors, the algorithm supports a wide range of possible weight types. This flexibility makes it highly adaptable to various experimental scenarios, depending on which parameters are most accessible or controllable in a given setup. [.. examples of realizable system, cite literature].

Appendix A: Dynamical evolution to the steady state

In this section, we provide a more detailed explanation of how a memristor network relaxes to its steady state. Alongside outlining the algorithm used to solve the circuit, we highlight the importance of the dependence on the functional form of the steady-state conductance, which itself is determined by the values of the weights. Selecting appropriate weights shapes the staedy-state conductance function in such a way that the network naturally relaxes to a steady state aligned with the training objective.

For simplicity, we consider the voltage divider shown in **??**, where the memristors are labeled as m_1, connecting node 2 to node 1, and m_2, connecting node 3 to node 2. We also denote the corresponding potential drops across each memristor as $\Delta V_1 = V_2 - V_1$ and $\Delta V_2 = V_3 - V_2$, respectively. To simplify, we set both the pressure difference (ΔP) and concentration difference ($\Delta\rho$) to zero. In this example, we choose the channel length as the training weight.

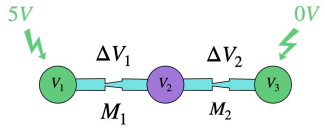

FIG. 11. Schematical representation of a memristor voltage divider. Two memristors, M_1 and M_2, connect two input nodes (green dots) with fixed voltages of 0 V and 5 V, respectively, to a single output node (purple dot). Each memristor is subject to one external stimulus: the electric potential difference between the nodes it connects.

The dynamical evolution of the conductance of each memristor, discretized in time-steps of size Δt, reduces to the update rule:

$$g_{\mathrm{m}}(t+\Delta t) = g_{\mathrm{m}}(t) + \frac{g_{\mathrm{m},\infty}(\Delta V_{\mathrm{m}}(t)) - g_{\mathrm{m}}(t)}{\tau}\Delta t, \quad \text{(A1)}$$

with initial condition $g_{\mathrm{m}}(t=0) = g_0$. The potential drops across the memristors, $\Delta V_1(t)$ and $\Delta V_2(t)$, are computed from Kirchhoff's laws. In the case of a simple voltage divider ,see **??**, these simplify to:

$$\Delta V_1(t) = \frac{5g_2(t)}{g_1(t)+g_2(t)}, \qquad \Delta V_2(t) = \frac{5g_1(t)}{g_1(t)+g_2(t)}, \quad \text{(A2)}$$

The simulation proceeds as follows:

- At $t = 0$, initialize: $g_1(0) = g_2(0) = g_0$.
- Compute $\Delta V_1(0)$ and $\Delta V_2(0)$ using **??**.
- Use **??** to update $g_1(\Delta t)$ and $g_2(\Delta t)$.
- Repeat the process at each time step until $|g_{\mathrm{m}}(t) - g_{\mathrm{m},\infty}(t)| < \epsilon$ for all memristors.

Algorithm 1 Relaxation of a Memristor Voltage Divider to Steady State

Input: Initial conductance g_0, time step Δt, convergence threshold ϵ, relaxation time τ
Initialize $g_1 \leftarrow g_0$, $g_2 \leftarrow g_0$, $t \leftarrow 0$
while not converged **do**
  Compute $\Delta V_1 \leftarrow \frac{5 \cdot g_2}{g_1+g_2}$
  Compute $\Delta V_2 \leftarrow \frac{5 \cdot g_1}{g_1+g_2}$
  $g_1 \leftarrow g_1 + \frac{g_\infty(\Delta V_1) - g_1}{\tau} \cdot \Delta t$
  $g_2 \leftarrow g_2 + \frac{g_\infty(\Delta V_2) - g_2}{\tau} \cdot \Delta t$
  if $|g_1 - g_\infty(\Delta V_1)| < \epsilon$ **and** $|g_2 - g_\infty(\Delta V_2)| < \epsilon$ **then**
    break
  end if
end while

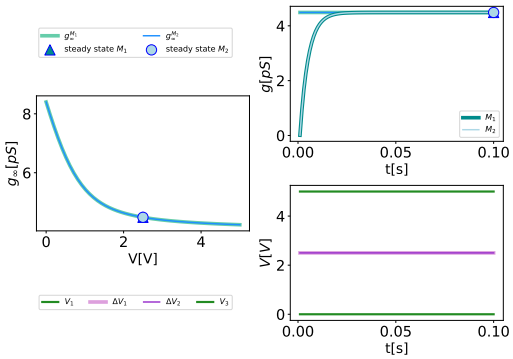

FIG. 12. Relaxation toward the steady state when $g_{1,\infty}(V) = g_{2,\infty}(V)$.

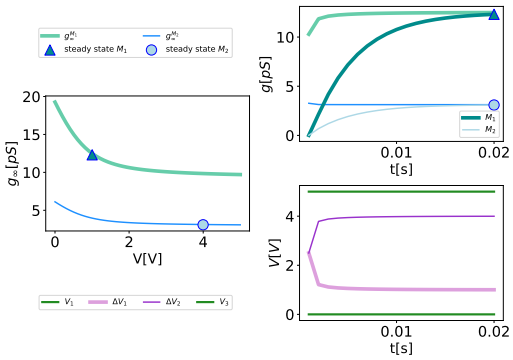

FIG. 13. Relaxation toward the steady state when the system has been trained to give $V_2 = 4V$. The training sets the two different functional forms of g_∞ seen in the right figure.

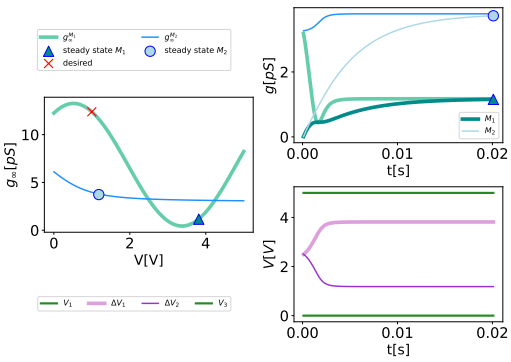

FIG. 14. We consider the trained system in Figure 3, but we define the funtion $g_{1,\infty}$ such that it passes througth the steady state point (triangle in Figure 3), now indicated with a red cross, but with a different functional form. Due to the different functinal form, the system reaches a steady state that doesn't coincide with the desired in Figure 3.

[1] Shih-Kai Chou, Jernej Hribar, Mihael Mohorčič, and Carolina Fortuna. The energy cost of artificial intelligence of things lifecycle, 2024.

[2] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021.

[3] Ali Momeni, Babak Rahmani, Benjamin Scellier, Logan G. Wright, Peter L. McMahon, Clara C. Wanjura, Yuhang Li, Anas Skalli, Natalia G. Berloff, Tatsuhiro Onodera, Ilker Oguz, Francesco Morichetti, Philipp del Hougne, Manuel Le Gallo, Abu Sebastian, Azalia Mirhoseini, Cheng Zhang, Danijela Marković, Daniel Brunner, Christophe Moser, Sylvain Gigan, Florian Marquardt, Aydogan Ozcan, Julie Grollier, Andrea J. Liu, Demetri Psaltis, Andrea Alù, and Romain Fleury. Training of physical neural networks, 2024.

[4] Xing Lin, Yair Rivenson, Nezih T. Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, September 2018.

[5] Hans Christian Ruiz Euler, Marcus N. Boon, Jochem T. Wildeboer, Bram van de Ven, Tao Chen, Hajo Broersma, Peter A. Bobbert, and Wilfred G. van der Wiel. A deep-learning approach to realizing functionality in nanoelectronic devices. *Nature Nanotechnology*, 15(12):992–998, December 2020.

[6] T. M. Kamsma, W. Q. Boon, T. ter Rele, C. Spitoni, and R. van Roij. Iontronic neuromorphic signaling with conical microfluidic memristors. *Phys. Rev. Lett.*, 130:268401, Jun 2023.

[7] Willem Q. Boon, Tim E. Veenstra, Marjolein Dijkstra, and René van Roij. Pressure-sensitive ion conduction in a conical channel: Optimal pressure and geometry. *Physics of Fluids*, 34(10), October 2022.

[8] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[9] Jaewook Kim, Tae-Kwang Jang, Young-Gyu Yoon, and SeongHwan Cho. Analysis and design of voltage-controlled oscillator based analog-to-digital converter. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(1):18–30, 2010.

[10] Selçuk Kose and Eby G. Friedman. Distributed on-chip power delivery. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(4):704–713, 2012.

[11] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos, 2016.

[12] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks, 2017.