# Designing Compatible Analog Circuits for Equilibrium Propagation: Implementations Using The Adjoint Method and Reciprocity Principles

Mohamed Watfa, Alberto Garcia-Ortiz, Gilles Sassatelli

# Designing Compatible Analog Circuits for Equilibrium Propagation: Implementations Using The Adjoint Method and Reciprocity Principles

Mohamed Watfa*†, Alberto Garcia-Ortiz†, Gilles Sassatelli*
*LIRMM, University of Montpellier, CNRS, Montpellier, France
†ITEM, University of Bremen, Bremen, Germany
{mohamed.watfa, sassatelli}@lirmm.fr, agarcia@item.uni-bremen.de

***Abstract*—Equilibrium Propagation (EP) offers an energy-efficient alternative to backpropagation for training analog neural networks, enabling on-device learning in Edge AI applications. However, practical hardware implementations of EP are hindered by stringent circuit requirements and challenges in computing loss gradients for common functions like crossentropy without resorting to expensive data converters. In this paper, we address these challenges by identifying the necessary conditions that an analog circuit must satisfy to implement EP, leveraging the adjoint method and the reciprocity principle of analog circuits, and developing two novel circuit architectures that fully realize EP in hardware. The first architecture employs the sparsemax activation function, offering a practical means to implement a crossentropy-like loss function while circumventing the complexities associated with the softmax function. The second architecture introduces a simpler topology that aligns with analog processing principles by enforcing output values to reside in a simplex, eliminating the need for calculating probabilities explicitly. We validate our designs on two benchmark datasets, demonstrating that our circuits can effectively and efficiently train analog neural networks using EP within practical hardware constraints.**

***Index Terms*—analog neural networks, equilibrium propagation, circuit design, spice simulation, training**

## I. Introduction

The increasing demand for energy-efficient *on-device training* has driven significant interest in analog neural networks (ANNs), particularly those based on in/near-memory computing (IMC) architectures [1]. These architectures integrate computation directly with memory, circumventing the von Neumann bottleneck and offering substantial energy savings for resource-constrained devices [2]. Additionally, on-device training addresses the device-to-device variability inherent in analog components, especially in memristors [1], [2], by allowing the system to be retrained online.

However, training ANNs efficiently is challenging. Conventional algorithms like backpropagation require global error signals, which are fundamentally incompatible with the locality and parallelism of IMC architectures [3]. Alternative local-learning frameworks [4]–[6], such as Equilibrium Propagation (EP), have been introduced to address this issue. EP [4], in particular, is uniquely suited to analog implementations as it embeds learning directly into the system's dynamics without separate forward and backward passes. EP operates in two phases, called *free* and *nudged*, and computes gradients from local differences in energy. For instance, when weights are stored in a linear resistor, the gradient is proportional to the difference in the square of the voltage drop across that resistor between the two phases [7].

Despite this alignment with analog systems, a key unresolved question is: *what class of analog circuits can support EP-compatible training dynamics?* To answer this, we derive a general condition using the adjoint method [8], showing that EP emerges naturally in circuits with reciprocal small-signal models. This insight eliminates the need for external gradient computation circuits, which are often needed for backpropagation [1], and guides the design of fully trainable analog architectures using either voltage or current processing.

A second challenge in analog training lies in the implementation of loss functions. While the gradient of simple losses such as the mean squared error (MSE) are easily implementable in analog using subtractors [9], [10], probabilistic losses like crossentropy, commonly-used in classification tasks, require the softmax function, which involves exponentials and normalization. These operations are difficult to realize in analog hardware and are typically implemented in the digital domain [1], necessitating high-precision, power-intensive data converters that consume up to 98% of total area and power [11]. As a result, fully-analog probabilistic gradient computation circuits remain an unsolved problem.

This paper addresses both challenges by developing the theory and circuits needed to support EP-based training entirely within analog hardware. The main contributions are as follows:

- **Foundational Framework**: We derive a general framework that characterizes the class of analog circuits compatible with EP. Using the adjoint method, we show that EP arises directly from the circuit equations when the small-signal model is reciprocal.
- **Sparsemax Computation Circuit (SCC)**: We present an analog-friendly implementation of the sparsemax function [12], [13], avoiding the computational complexities of softmax and enabling efficient analog probabilistic computation.
- **Voltage Simplex Network (VSN)**: We propose a novel circuit architecture embedding gradient computations

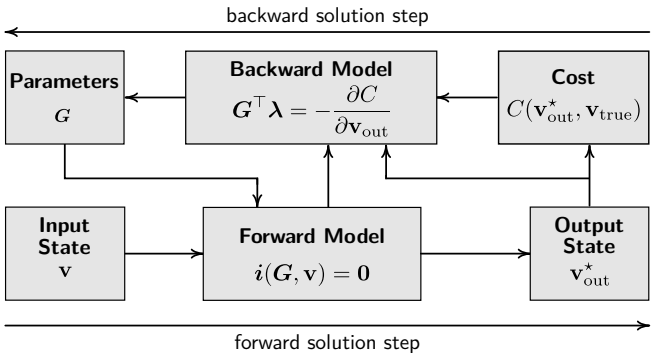


Fig. 1: Adjoint method for implicitly-defined constraints.

through a simple resistor-diode network that inherently enforces simplex constraints via Kirchhoff's laws, significantly simplifying analog EP training.

The remainder of the paper is organized as follows. In Section II, we establish the conditions under which analog circuits support EP-compatible training using the adjoint method. Section III presents the sparsemax function and its analog implementation. The Voltage Simplex Network is introduced in Section IV. Section V reports simulation results validating both architectures.

## II. Conditions For Designing Equilibrium Propagation Compatible Analog Circuits

Designing analog circuits for EP first requires identifying the properties the circuit must satisfy to enable learning. By formulating the training problem as a constrained optimization problem governed by Kirchhoff's laws and applying the adjoint method, our method reveals three key points: (1) learning rules that resemble those of EP emerge directly from the solution, (2) reciprocity in the small-signal model is the necessary condition for in-circuit gradient computation, and (3) EP-compatible training can be extended to current-mode circuits, which is often overlooked in other works [7], [10].

### A. Adjoint Method

Training a supervised model involves minimizing a cost $C$ over parameters $\boldsymbol{G} \in \mathbb{R}^{n \times n}$, subject to a constraint that defines how predictions $\mathbf{v}_{\text{out}}$ are obtained. In analog, this constraint is defined implicitly via Kirchhoff's current law (KCL):

$$\min_{\boldsymbol{G}} \quad C(\mathbf{v}, \mathbf{v}_{\text{true}}) \quad \text{s.t.} \quad \boldsymbol{i}(\boldsymbol{G}, \mathbf{v}) = \mathbf{0} \tag{1}$$

where $\boldsymbol{i} \in \mathbb{R}^n$ are the branch currents, $\mathbf{v} \in \mathbb{R}^n$ are the steady-state node voltages, and $\mathbf{v}_{\text{true}}$ are desired voltages.

To train the circuit, we must compute $\mathrm{d}C/\mathrm{d}\boldsymbol{G}$, but $\mathbf{v}$ depends implicitly on $\boldsymbol{G}$ through the circuit equations. The adjoint method computes this gradient efficiently without explicitly differentiating $\mathbf{v}$ with respect to $\boldsymbol{G}$ [8]. The steps are presented in Algorithm 1 and illustrated graphically in Figure 1.

**Algorithm 1** Adjoint Method

1: **Forward Pass:** Using an initial guess for $\boldsymbol{G}$, solve the nonlinear equation to obtain the solution $\mathbf{v}^\star$:

$$\boldsymbol{i}(\boldsymbol{G}, \mathbf{v}) = \mathbf{0} \tag{2}$$

2: **Backward Pass:** Calculate the Lagrange multiplier $\boldsymbol{\lambda} \in \mathbb{R}^n$ by solving the linear system:

$$\left(\frac{\partial \boldsymbol{i}}{\partial \mathbf{v}}\right)^\top \Big|_{\mathbf{v}=\mathbf{v}^\star} \boldsymbol{\lambda} = -\frac{\partial C}{\partial \mathbf{v}}\Big|_{\mathbf{v}=\mathbf{v}^\star} \tag{3}$$

3: **Gradient Computation:** Plug $\boldsymbol{\lambda}$ into the equation below to compute the parameter gradient $\mathrm{d}C/\mathrm{d}\boldsymbol{G}$:

$$\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{G}} = \boldsymbol{\lambda} \left(\frac{\partial \boldsymbol{i}}{\partial \boldsymbol{G}}\right)^\top \Big|_{\mathbf{v}=\mathbf{v}^\star} \tag{4}$$

### B. Interpretation in the Context of Analog Circuits

The forward pass of the adjoint method corresponds to the circuit naturally settling to its steady state $\mathbf{v}^\star$. This solves the constraint in (1) directly in hardware, eliminating the need for iterative nonlinear solvers required in digital implementations.

After achieving steady-state, using another circuit, the output node voltages $\mathbf{v}^\star_{\text{out}}$ are compared against the desired voltages $\mathbf{v}_{\text{true}}$, resulting in the term $\partial C/\partial \mathbf{v}_{\text{out}}$ that specifies how the output node voltages should be modified from the steady state to reduce the cost $C$. This is then translated into $\mathrm{d}C/\mathrm{d}\boldsymbol{G}$, which the adjoint method accomplishes in two steps:

- **First**, it computes the voltage change required at each (internal) node of the circuit. This corresponds to solving for $\boldsymbol{\lambda}$ in Equation (3).
- **Second**, it translates the change required at each node to a change in the parameters of the circuit using Equation (4).

While the direct application of the adjoint method could potentially be used to train analog neural networks, it would require a separate circuit to compute $\partial C/\partial \mathbf{v}_{\text{out}}$ and another circuit to solve for $\boldsymbol{\lambda}$. The computation of $\partial C/\partial \mathbf{v}_{\text{out}}$ is not an issue. However, since $\partial \boldsymbol{i}/\partial \mathbf{v}$ in Equation (3) evaluates to $\boldsymbol{G}$, a circuit that computes $\boldsymbol{\lambda}$ must have the same set of parameters as the original circuit that computes $\mathbf{v}_{\text{out}}$. This is impossible to achieve in practice due to matching considerations. Nevertheless, under some conditions, it is possible avoid matching issues while also requiring no additional circuit to compute $\boldsymbol{\lambda}$.

A key observation is that if a circuit has a smooth, equivalent small-signal model and the small-signal circuit is reciprocal, the computation of $\boldsymbol{\lambda}$ can be performed directly within the circuit that computes $\mathbf{v}_{\text{out}}$. This is based on the following insights:

- **First**, given that the small-signal model, $\boldsymbol{G}\mathbf{v} = \boldsymbol{I}$ (in Y-parameter form), of any analog circuit is linear, Equation (3) can be computed on the small signal circuit.
- **Second**, if we set $\mathbf{v}$ to $\boldsymbol{\lambda}$ and $\boldsymbol{I}$ to $-\partial C/\partial \mathbf{v}_{\text{out}}$, and ensure that the $\boldsymbol{G}$ matrix is reciprocal, i.e., $\boldsymbol{G}^\top = \boldsymbol{G}$, then the small-signal model implements Equation (3) directly.

Thus, the Lagrange multiplier $\boldsymbol{\lambda}$ can be interpreted in this context to represent the voltage variation at the internal nodes of the circuit when a current of $-\partial C/\partial \mathbf{v}_{\text{out}}$ is injected in the output nodes, effectively operating the circuit in reverse.

The computation of $\boldsymbol{\lambda}$ uses the derivatives $(\partial \boldsymbol{i}/\partial \mathbf{v})^\top$ and $\partial C/\partial \mathbf{v}_{\text{out}}$ at the point $\mathbf{v} = \mathbf{v}^\star$, which are constant. By injecting current into the circuit, we move away from $\mathbf{v}^\star$. This is not an issue if the circuit is linear, because all entries in $(\partial \boldsymbol{i}/\partial \mathbf{v})^\top$ and $\partial C/\partial \mathbf{v}_{\text{out}}$ are scaled by the same amount, causing the scale factors to cancel out. However, if the circuit is nonlinear, which is the case, the entries are scaled nonlinearly, and the computed $\boldsymbol{\lambda}$ deviates from the true value at steady-state. This is one of the drawbacks of using a single circuit to compute $\mathbf{v}_{\text{out}}$ and $\boldsymbol{\lambda}$. To minimize the distortion, the injected currents $-\partial C/\partial \mathbf{v}$ must be kept as small as possible.

### C. Comparison with EP

The update obtained from the adjoint method closely resembles the EP rule. For example, consider a resistor with conductance $g$. EP gives the update rule as

$$\left[\frac{\mathrm{d}C}{\mathrm{d}g}\right]_{\text{EP}} \propto \Delta v_{\text{nudge}}^2 - \Delta v_{\text{free}}^2, \tag{5}$$

where $\Delta v_{\text{free}}$ and $\Delta v_{\text{nudge}}$ are the steady-state voltage drops across $g$ in the free and nudged phases, respectively.

In the case of the adjoint method, the update rule is given by:

$$\left[\frac{\mathrm{d}C}{\mathrm{d}g}\right]_{\text{Ad}} \propto (\lambda_{\text{nudge}} - \lambda_{\text{free}}) \cdot \Delta v_{\text{free}}. \tag{6}$$

The EP and adjoint updates are equivalent except for an additional $\lambda^2$ term present in EP. This term is negligible when the perturbation is small ($\lambda \approx 0$), but becomes significant as voltage variations increase [14]. In contrast, the adjoint method provides an exact gradient regardless of the perturbation size.

### D. Current-Voltage Duality

While the above discussion has used voltages as the main state variable, analog circuits, with the duality of currents and voltages, open up the possibility of a different circuit topology where current is the state variable. Instead of a constraint of the form $\boldsymbol{i}(\boldsymbol{G}, \mathbf{v}) = \mathbf{0}$, we can consider a constraint of the form $\boldsymbol{v}(\boldsymbol{R}, \mathbf{i}) = \mathbf{0}$, where $\boldsymbol{R}$ denotes the Kirchoff resistance matrix and $\mathbf{i}$ the branch currents. In this case, the output of the network is obtained by measuring the current through the output ports. The nudging is performed using voltages that change the output currents, and by satisfying the reciprocity principle, the voltage applied at the output ports will correspond to current variations in the internal branches, allowing the system to be trained.

## III. Sparsemax Circuit

The conditions established in the last section are essential for designing analog circuits that can be trained through current or voltage excitation at the output nodes. We have shown how $\boldsymbol{\lambda}$ can be computed without the need for any additional circuit. However, the computation of $\partial C/\partial \mathbf{v}_{\text{out}}$ (the loss gradient) requires a dedicated circuit. This section introduces, for the first time, the Sparsemax Computation Circuit (SCC), which offers an analog-friendly alternative to the crossentropy loss.

### A. Challenges with Softmax Implementation in Analog Hardware

The crossentropy loss, used for classification tasks, uses the softmax function to transform output scores into probabilities.

$$\text{softmax}(\mathbf{v})_i = \frac{e^{v_i}}{\sum_j e^{v_j}} \tag{7}$$

The softmax function requires exponentiating each input, which is difficult to implement accurately in analog hardware due to temperature sensitivity, device mismatch, and the need for large dynamic range [15]. A common workaround is to compute softmax digitally, but this requires high-precision data converters to interface with the analog domain, introducing significant overhead in power and area [1].

It is important to note that the softmax is only needed during training, when the gradient of the loss function is required. For inference, the network's output is typically determined by selecting the class with the highest score, a functionality efficiently implemented in the analog domain using Winner-Take-All (WTA) circuits [16]. However, since the WTA function is discontinuous, it cannot be directly differentiated, posing a problem for gradient-based methods.

To address this, we adopt sparsemax [12], [13], a piecewise-linear alternative to softmax that avoids exponentials and is far more suitable for analog implementation.

### B. Introducing the Sparsemax

The sparsemax function maps an input vector $\mathbf{v} \in \mathbb{R}^k$ onto the probability simplex $\Delta^{K-1}$ and finds the point $\mathbf{p}$ closest to $\mathbf{v}$ in terms of Euclidean distance.

$$\text{sparsemax}(\mathbf{v}) = \underset{\mathbf{p} \in \Delta^K}{\operatorname{argmin}} \parallel \mathbf{p} - \mathbf{v} \parallel_2^2 \tag{8}$$

Using the loss function defined in [12], the gradient with respect to $\mathbf{v}_{\text{out}}$ is

$$\frac{\partial C}{\partial \mathbf{v}_{\text{out}}}(\mathbf{v}_{\text{out}}, \mathbf{v}_{\text{true}}) = -\boldsymbol{\delta}_{\mathbf{v}_{\text{true}}} + \text{sparsemax}(\mathbf{v}), \tag{9}$$

where $\boldsymbol{\delta}_{\mathbf{v}_{\text{true}}} \in \mathbb{R}^k$ is a vector that is zero everywhere except at the index corresponding to the true class. Notably, this gradient expression mirrors that of the crossentropy loss but without the computational complexities associated with exponentials.

### C. Sparsemax Algorithm

Computing the sparsemax function involves finding the appropriate threshold $\tau$ to project the input vector onto the probability simplex. The algorithm, presented in Algorithm 2 [13], effectively zeros out the entries of $\mathbf{v}$ that are below the threshold $\tau$, resulting in a sparse output vector that sums to one.

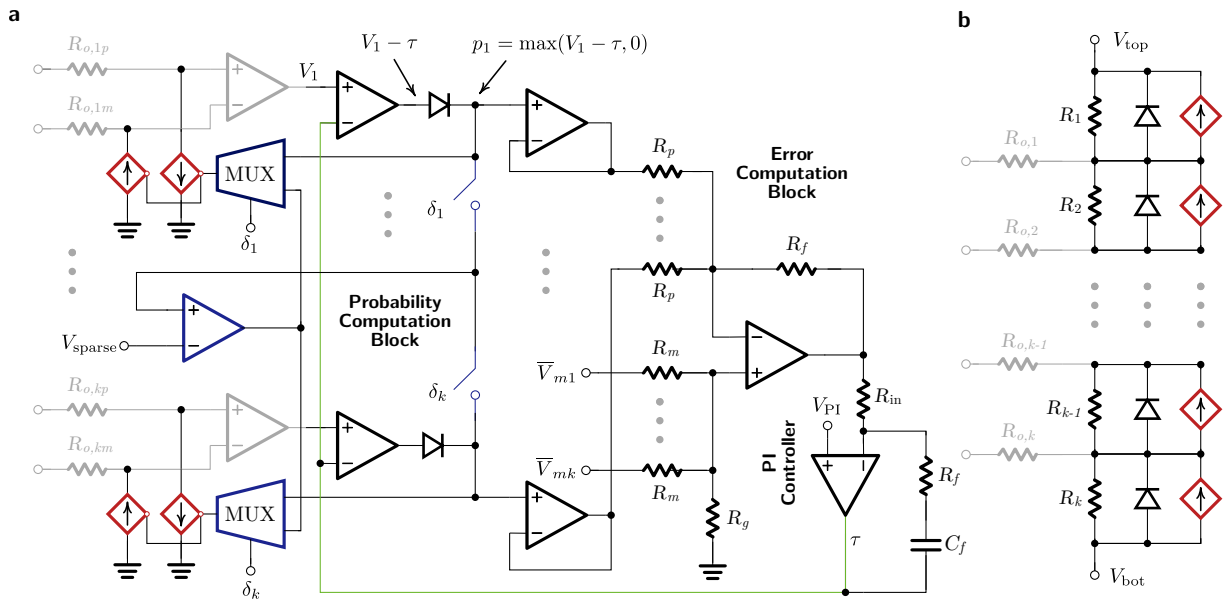


Fig. 2: Proposed circuits: (a) Sparsemax computation circuit, (b) Voltage simplex layer.

**Algorithm 2** Sparsemax Algorithm

1: Sort the input vector $\mathbf{v}$ in descending order:

$$v_{(1)} \geq v_{(2)} \geq \cdots \geq v_{(k)} \tag{10}$$

2: Compute cumulative sums of the sorted vector:

$$S_k = \sum_{i=1}^{k} v_{(i)} \tag{11}$$

3: Find the largest $k^*$ satisfying:

$$v_{(k)} > \frac{1}{k}(S_k - 1) \tag{12}$$

4: Compute the threshold value $\tau$:

$$\tau = \frac{1}{k^*}(S_{k^*} - 1) \tag{13}$$

5: Compute the output probabilities:

$$p_i = \max(v_i - \tau, 0) \tag{14}$$

### D. Problem Reformulation for Analog Implementation

The main challenge in implementing the sparsemax function, as presented in Algorithm 2, lies in computing $\tau$ without explicit sorting, which is impractical in analog hardware. To eliminate the need for sorting, we reformulate the problem as:

$$\sum_{i=1}^{k} \max(v_i - \tau, 0) = 1 \tag{15}$$

This equation seeks a value of $\tau$ such that the sum of the thresholded inputs equals one. By interpreting this as a root-finding problem, we can implement a feedback loop in analog hardware that adjusts $\tau$ until this condition is satisfied.

### E. Circuit Implementation

Based on the reformulated problem, we propose an analog circuit design, shown in Figure 2a, that computes the sparsemax function and its corresponding gradients without any complex computations. The circuit comprises two main blocks: a sparsemax computation circuit (black) implementing Equation (8), and a gradient computation circuit (blue) implementing Equation (9).

*1) Sparsemax Computation Circuit:* This consists of three main blocks:

- **Probability Computation Block:** Computes the sparsemax probabilities from the input voltages, as defined in Equation (14). Subtractions are performed using differential amplifiers, and diodes enforce non-negativity by clipping negative values.
- **Error Computation Block:** Ensures the probabilities sum to one by comparing the total output $\sum_i p_i$ to a reference value. This is done using a sum-difference amplifier that outputs an error signal proportional to the difference.
- **PI Control Block:** Adjusts the threshold $\tau$ based on the error signal. Implemented using a differential amplifier with resistive-capacitive feedback, it drives the system toward satisfying the simplex constraint.

Together, these blocks form a closed-loop system that converges to the threshold $\tau$ satisfying the constraint $\sum_i p_i = 1$, thereby solving the projection problem in (15).

*2) Gradient Computation Circuit:* Once the sparsemax probabilities are computed, the gradients defined in (9) are

implemented using current mirrors or transconductance amplifiers (shown in red), along with multiplexer circuits. For output nodes corresponding to incorrect classes ($\delta_i = 0$), currents proportional to $p_i$ are injected or sunk. For the correct class node ($\delta_i = 1$), a current of $1 - p_i$ is applied. To preserve reciprocity, these currents are injected before the subtractor stage, as subtractors violate the condition $\boldsymbol{G} = \boldsymbol{G}^\top$.

## IV. Voltage Simplex Network

The Sparsemax Computation Circuit can be integrated with neural network architectures beyond Energy-Based Models (EBMs). While the design achieves its intended purpose and offers strong theoretical guarantees [12], it incorporates multiple processing blocks and, to some extent, processes signals in a digital-like manner. To develop a solution that is more aligned with the principles of EBMs and fully exploits analog processing, we introduce a different and simpler topology, the Voltage Simplex Network (VSN), which integrates the gradient computation units directly into the main circuit design.

### A. Circuit Description

The design of the VSN closely resembles that of the sparsemax network but incorporates two key modifications: First, the output layer is not duplicated. Second, the output ports are no longer floating but are connected to a Voltage Simplex Layer (VSL), as depicted in Figure 2b. The VSL consists of a series of resistors connected between the supply voltages $V_{\text{top}}$ and $V_{\text{bot}}$. Furthermore, diodes are connected across the resistors to ensure that their voltage is (approximately) positive.

By construction, the sum of the voltage drops across these resistors equals $V_{\text{top}} - V_{\text{bot}}$ by Kirchhoff's laws, and it is (approximately) positive by the action of the diodes; thus, the voltage drops across the resistors can be associated with class probabilities.

By connecting the previously floating output nodes of the neural network to the nodes of the VSL, the neural network can influence the voltage drops across the resistors in the VSL and hence adjust the predicted probabilities.

An important characteristic of this topology is that by enabling the network to directly produce probabilities, it simplifies the requirements for generating loss gradients. By defining the cost function as the mean squared error (MSE), we retain the benefits of using crossentropy loss, such as effectively penalizing deviations from the true class, while adopting a cost function that is straightforward to implement in analog hardware. The use of MSE in classification tasks is not unconventional. Studies have shown that MSE can be effectively used with probabilistic outputs in classification tasks, achieving performance comparable to crossentropy loss [17].

In terms of the current to be injected for training, with the MSE loss $C = \frac{1}{2}\|\mathbf{v}_R - \boldsymbol{\delta}\|_2^2$, where $\boldsymbol{\delta}$ is the one-hot encoded vector representing the true class, and $v_{R_i} = v_{R_i}^+ - v_{R_i}^-$ being the voltage drop across the $i$-th resistor in the VSL, it can be shown that the gradient with respect to the node voltage $v_{n_j}$ (for node $j$ below node $i$) is

$$\frac{\mathrm{d}C}{\mathrm{d}v_{nj}} = (v_{Rj} - \delta_j) - (v_{Ri} - \delta_i). \tag{16}$$

Implementing this gradient requires adding two current sources to ground at every node. However, by noting that node $i$ requires a current proportional to $+(V_{R_i} - \delta_i)$, we can apply Kirchhoff's Current Law (KCL) to use only one current source for each node. These current sources are connected in parallel across the resistors, as illustrated in Figure 2b.

## V. Results

To validate the functionality of the proposed Sparsemax Computation Circuit, we designed a 4-input implementation using behavioral SPICE models operating within the voltage range of $\pm 0.6, \mathrm{V}$, matching the expected output range of the neural network. For a test input vector $\mathbf{v} = [0.1, -0.1, -0.3, 0.2]$, the expected output is $\boldsymbol{p} = [0.37, 0.17, 0, 0.46]$, which aligns with the simulation results shown in Figure 3a. The convergence behavior of the circuit is influenced by the PI controller's output $\tau(t)$, and the starting state of the circuit when the inputs are applied. For instance, starting from the zero state, it takes 20 $\mu$s for the outputs to converge. After that, the settling time is much shorter (5 $\mu$s), as can be seen in Figure 3a when $V_1$ is changed to 0.45V.

We evaluated our circuits using SPICE-level simulations on the MNIST and Iris datasets, following the setup in [18]. While often considered simple in digital settings, MNIST remains computationally intensive at the SPICE level due to the large number of nodes and the time required to simulate the circuit dynamics. For example, simulating one training epoch on the sparsemax topology takes nearly 1.5 days and required internal modifications to ngspice (spice simulator). The long training time is a consequence of simulating the circuit in SPICE, not a limitation of the hardware itself, which operates on sub-microsecond timescales [1].

In the first experiment, we trained the analog neural network using the sparsemax and the crossentropy loss (baseline). To accommodate the positive-only weights inherent in resistive components, we adopted a common technique in analog neural networks [1], duplicating the output layer and subtracting corresponding node values to obtain the final output. While this method effectively doubles the size of the output layer, it ensures accurate representation of both positive and negative weights in hardware.

As shown in Figure 3b, the network trained with the sparsemax loss function achieved a slightly better performance than the one with crossentropy (95% vs. 92%). This could be attributed to the fact that sparsemax produces sparse probabilities (some outputs are zero), whereas the softmax produces dense probabilities, causing the outputs to always be nudged even when the predictions are correct.

In the second experiment, we constructed the output layer without duplication, reducing network size but introducing a non-negativity constraint in Equation (1). Using the MSE loss as a baseline, the network achieved 81% accuracy after 6 epochs. By integrating the Voltage Simplex Layer (VSL)

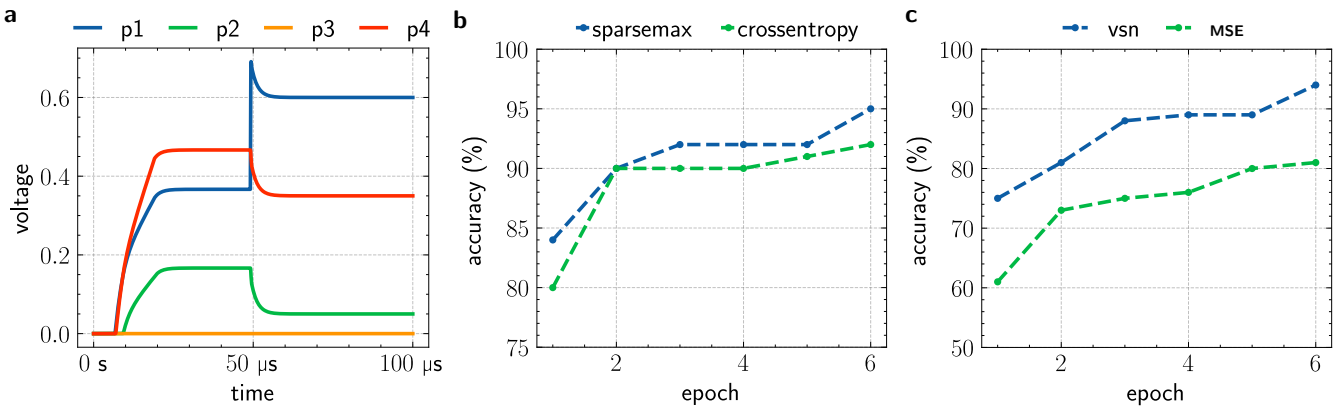


Fig. 3: Simulation plots: (a) sparsemax circuit output waveforms, (b) MNIST training with sparsemax vs. crossentropy (baseline), (c) MNIST training with VSL layer (vsn) vs. without (using MSE as baseline).

.

directly with the output nodes, accuracy increased to 94%. This is comparable to the 95% achieved with the sparsemax loss and duplicated output layers, demonstrating that the VSL effectively simplifies the circuit architecture without sacrificing performance. Similar improvements were observed with the Iris dataset: after 30 epochs, the baseline MSE simulation achieved 91% accuracy, while the network with the VSL reached 95%, closely matching the 96% obtained with the duplicated output layers.

## VI. Conclusion

We presented a theoretical framework for designing analog circuits that are compatible with the equilibrium propagation algorithm, and showed how the adjoint method can be leveraged to exploit the duality of voltage and current in analog systems for designing new topologies. By introducing the sparsemax function and providing its physical realization, we showed how to extend the processing in the analog domain as much as possible to avoid the use of expensive data converters for analog circuits beyond energy-based models. Furthermore, we presented the Voltage Simplex Network, a novel topology that computes probabilities directly within the circuit using only simple components. Empirical results on classification tasks validate our approach, marking a significant leap forward in the development of efficient, fully analog neural networks capable of on-chip learning.

## Acknowledgment

This work is supported by a public grant overseen by the French National Research Agency (ANR) as part of the 'PEPR IA France 2030' programme (Emergences project ANR-23-PEIA-0002).

## References

[1] T. P. Xiao, C. H. Bennett, B. Feinberg, S. Agarwal, and M. J. Marinella, "Analog architectures for neural network acceleration based on non-volatile memory," *Applied Physics Reviews*, vol. 7, 7 2020.

[2] A. S. Geoffrey W. Burr, Robert M. Shelby, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, 12 2016.

[3] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," *Frontiers in Neuroscience*, vol. 10, 7 2016.

[4] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in Computational Neuroscience*, vol. 11, 5 2017.

[5] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," p. 6572–6583, 2018.

[6] G. Hinton, "The forward-forward algorithm: Some preliminary investigations," 2022. [Online]. Available: https://arxiv.org/abs/2212.13345

[7] J. Kendall, R. Pantone, and K. Manickavasagam, "Training end-to-end analog neural networks with equilibrium propagation," 06 2020. [Online]. Available: http://arxiv.org/abs/2006.01981v2

[8] L. S. Pontryagin, E. F. Mishchenko, V. G. Boltyanskii, and R. V. Gamkrelidze, *Mathematical Theory of Optimal Processes*, 1962.

[9] O. Krestinskaya, K. N. Salama, and A. P. James, "Learning in memristive neural network architectures using analog backpropagation circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 2, pp. 719–732, 2019.

[10] S. Oh, J. An, S. Cho, R. Yoon, and K.-S. Min, "Memristor crossbar circuits implementing equilibrium propagation for on-device learning," *Micromachines*, vol. 14, 7 2023.

[11] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, "Switched by input," *Proceedings of the 53rd Annual Design Automation Conference*, 6 2016.

[12] A. F. T. Martins and R. F. Astudillo, "From softmax to sparsemax: A sparse model of attention and multi-label classification," 02 2016. [Online]. Available: http://arxiv.org/abs/1602.02068v2

[13] J. Duchi and S. Shalev-Shwartz, "Efficient projections onto the $l1$-ball for learning in high dimensions," *Proceedings of the 25th international conference on Machine learning - ICML '08*, 2008.

[14] A. Laborieux, M. Ernoult, B. Scellier, and Y. Bengio, "Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias," *Frontiers in Neuroscience*, vol. 15, 2021.

[15] C. R. Popa, *Synthesis of Computational Structures for Analog Signal Processing*. Springer New York, 2012.

[16] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, "Winner-take-all networks of o(n) complexity," in *Proceedings of the 1st International Conference on Neural Information Processing Systems*, ser. NIPS'88. Cambridge, MA, USA: MIT Press, 1988, p. 703–711.

[17] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," 2017. [Online]. Available: https://arxiv.org/abs/1702.05659

[18] M. Watfa, A. Garcia-Ortiz, and G. Sassatelli, "Energy-based analog neural network framework," *Front. Comput. Neurosci.*, vol. 17, p. 1114651, Mar. 2023.