Adjoint Method: The Connection between Analog-based Equilibrium Propagation Architectures and Neural ODEs

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Analog neural networks (ANNs) hold significant potential for substantial reductions in power consumption in modern neural networks, particularly when employing the increasingly popular Energy-Based Models (EBMs) in tandem with the local Equilibrium Propagation (EP) training algorithm. This paper analyzes the relationship between this family of ANNs and the concept of Neural Ordinary Differential Equations (Neural ODEs). Using the adjoint method, we formally demonstrate that ANN-EP can be derived from Neural ODEs by constraining the differential equations to those with a steady-state response. This finding opens avenues for the ANN-EP community to extend ANNs to non-steady-state scenarios. Additionally, it provides an efficient setting for NN-ODEs that significantly reduces the training cost.

1 Introduction

The aspiration to empower IoT devices at the edge with real-time adaptive learning capabilities has been one of the primary motivators for advancing efficient neural network training methods. This endeavor goes beyond merely substituting backpropagation with alternative strategies; it represents a shift in the computational principles underlying model design, training, and inference. Such a transformation requires the incorporation of innovative models like Energy-Based Models (EBMs), which diverge significantly from conventional paradigms. This shift is pivotal for leveraging the unique attributes of diverse computing paradigms, such as photonic and neuromorphic computing.

In light of the evolving computational landscape, the advent of Neural Ordinary Differential Equations [1] (Neural ODEs) and Equilibrium Propagation [2] (EP) have emerged as promising contenders. Neural ODEs extend neural network models into the continuous-time domain, allowing the incorporation of differential equations to represent the evolution of the network states. EP, in contrast, provides a biologically plausible learning framework that integrates the concept of a cost function, a notable departure from the training methodologies of conventional EBMs.

The adjoint method serves as a pivotal link between EP and Neural ODEs, providing a framework that allows for the efficient computation of gradients in continuous-time models. This paper posits that EP can be construed as a special case of Neural ODEs, unified by the underlying adjoint method. This connection is not merely theoretical but has practical implications, especially in the realm of analog computation. By leveraging an analog circuit, we will demonstrate how the Lambda ($\boldsymbol{\lambda}$) variable in Neural ODEs corresponds precisely to the variation of the node voltages of the circuit. The exploration of such connections is vital for the development of low-cost and efficient solutions in neural networks, aligning with the broader goals of advancing biologically plausible and energy-efficient machine learning models.

Submitted to the First Workshop on Machine Learning with New Compute Paradigms at NeurIPS (MLNPCP 2023). Do not distribute.

$$\min_{\boldsymbol{\theta}} \quad C(\boldsymbol{y}, \boldsymbol{y}_{\text{true}})$$
$$\text{s.t.} \quad \boldsymbol{y} = \boldsymbol{U}_n(\cdots(\boldsymbol{U}_0(\boldsymbol{x}, \boldsymbol{\theta})\cdots))$$

(a) Output defined explicitly as a composition of functions.

$$\min_{\boldsymbol{\theta}} \quad C(\boldsymbol{s}, \boldsymbol{s}_{\text{true}})$$
$$\text{s.t.} \quad \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{0}$$

(b) Output defined implicitly via a nonlinear implicit equation.

$$\min_{\boldsymbol{\theta}} \quad C(\boldsymbol{s}, \boldsymbol{s}_{\text{true}})$$
$$\text{s.t.} \quad \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{s}(t), t, \boldsymbol{x}, \boldsymbol{\theta})$$

(c) Output defined implicitly via a differential equation.

Figure 1: Supervised learning optimization problem definitions

2 Formulation of Training as an Optimization Task

The goal of training any supervised machine learning model is to find the optimal parameters $\boldsymbol{\theta}$ that associate low cost values C to input-output pairs $\{\boldsymbol{x}, \boldsymbol{y}_{\text{true}}\}_{i=1}^{N}$ from the training dataset. This objective is usually formulated in the form of an optimization problem, where the constraints define how the output is obtained.

Figure 1 shows three classes of optimization problems in machine learning. In conventional feedforward networks (Fig. 1a), the output $\boldsymbol{y}$ is derived explicitly through the composition of the functions $\boldsymbol{U}_n$ through $\boldsymbol{U}_0$, where $\boldsymbol{U}_i$ are usually the layers of the model. Conversely, equilibrium models represent $\boldsymbol{y}$ as a function of the network's state variable $\boldsymbol{s}$. In the case of EP (Fig. 1b), this function is expressed through an nonlinear implicit equation, whereas in Neural ODEs (Fig. 1c), it is defined through a differential equation.

Regardless of the method employed to generate the output, the supervised learning process comprises three main stages: (1) computation of $\boldsymbol{y}$ or $\boldsymbol{s}$, (2) computation of the loss gradient $\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{y}}$ or $\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{s}}$, and (3) computation of the parameter gradient $\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{\theta}}$.

Computing $\boldsymbol{y}$, often referred to as the inference/forward step, is simpler in feedforward networks due to the explicit form of $\boldsymbol{y}$. For equilibrium models, this necessitates the use of a nonlinear solver, such as the Newton-Raphson method or Euler's method for ODEs.

In feedforward networks, the parameter gradient $\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{\theta}}$ is typically computed via the backpropagation algorithm, which uses the chain rule to propagate gradients from the output layer towards the input layer. However, when $\boldsymbol{y}$ is defined implicitly, calculating $\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{\theta}}$ requires a different approach. This is the subject of the next two section.

3 Neural ODEs

3.1 Motivation

Neural ODEs are a family of deep neural network models that can be interpreted as a continuous equivalent of Residual Networks (ResNets). In ResNets, the hidden state $\boldsymbol{s}[t+1]$ at layer $t+1$ is derived from the hidden state $\boldsymbol{s}[t]$ at layer t according to the equation $\boldsymbol{s}[t+1] = \boldsymbol{s}[t] + \boldsymbol{f}(\boldsymbol{s}[t], \boldsymbol{x}, \boldsymbol{\theta})$, where $\boldsymbol{f}(\cdot)$, for many applications, is a simple feedforward network, and $\boldsymbol{\theta}$ are its parameters. This transformation can be viewed as a discretization of a derivative $\boldsymbol{s}'(t)$ with timestep $\Delta t = 1$. Upon taking Δt to zero, this leads to an ordinary differential equation: $\frac{\mathrm{d}\boldsymbol{s}(t)}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{s}(t), t, \boldsymbol{x}, \boldsymbol{\theta})$. The output layer $\boldsymbol{s}[T]$ is then defined as the solution of this ODE, starting from an initial state $\boldsymbol{s}[0]$.

3.2 Adjoint Method for ODEs

Consider the optimization problem presented in Fig. 1c, where the goal is to find the parameters $\boldsymbol{\theta}$ that minimize the cost C at the output layer $\boldsymbol{s}(T)$. The adjoint method for ODEs provides a way to backpropagate through a model whose output is defined implicitly through a differential equation. The algorithm is provided in Alg 1. For its derivation, see [1].

The solution of equation (2) requires knowing the value of $\boldsymbol{s}(t)$ along its entire trajectory. Storing these evaluations in step 1 is impractical for large systems or long time spans due to memory

Algorithm 1 Adjoint method for constraints defined via ODEs

1: Solve the (forward) differential equation for $0 \leqslant t \leqslant T$ to get $\boldsymbol{s}(t)$.

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{s}(t), t, \boldsymbol{x}, \boldsymbol{\theta}) \\ \boldsymbol{s}(t=0) = \boldsymbol{s}_0 \end{cases} \tag{1}$$

2: Using the $\boldsymbol{s}(T)$ from step 1, find $\boldsymbol{\lambda}(T)$. Then solve the (backward) differential equation from $t = T$ to $t = 0$.

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{\lambda}}{\mathrm{d}t} = -[\boldsymbol{\lambda}(t)]^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{s}} \\ \boldsymbol{\lambda}(T) = \left[\frac{\partial C(\boldsymbol{s}(T), \boldsymbol{s}_{\text{true}})}{\partial \boldsymbol{s}}\right]^T \end{cases} \tag{2}$$

3: Plug $\boldsymbol{\lambda}(t)$ to get $\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{\theta}}$.

$$\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{\theta}} = -\int_T^0 [\boldsymbol{\lambda}(t)]^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} \mathrm{d}t \tag{3}$$

Algorithm 2 Adjoint method for constraints defined via implicit nonlinear equations

1: Given $\boldsymbol{\theta}$, solve the nonlinear equation given by the constraint to get $\boldsymbol{s}$.

$$\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{\theta}) = \mathbf{0} \tag{4}$$

2: Using the $\boldsymbol{s}$ from step 1, find $\boldsymbol{\lambda}$.

$$\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{s}}\right)^T \boldsymbol{\lambda} = -\left(\frac{\partial C}{\partial \boldsymbol{s}}\right)^T \tag{5}$$

3: Plug $\boldsymbol{\lambda}$ in the equation below to get $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}}$.

$$\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{\theta}} = \boldsymbol{\lambda}^T(\boldsymbol{\theta}) \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} \tag{6}$$

limitations. The authors of [1] addressed this issue by forming an augmented system of ODEs that recomputes $\boldsymbol{s}(t)$ concurrently with $\boldsymbol{\lambda}(t)$ during step 2.

While ODE solvers can be implemented in analog hardware, it is not obvious how the second and third steps could be realized. However, if we make some assumptions about the nature of the ODE, the resulting system has a direct interpretation in analog hardware. This is discussed next.

3.3 Bridging Neural ODEs to Physical Systems

Neural networks are well-known as universal function approximators, traditionally expressed in the form $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$. EP extends this paradigm to implicitly-defined neural networks. In such networks, the mapping from input to output is defined not directly but via an implicit equation, offering increased representational capacity.

Neural ODEs take this notion a step further by introducing a time component into the system. A unique feature of Neural ODEs is the ability to define a family of mappings between inputs and outputs. This is achieved by choosing different initial states for the system. After a time T, different initial states will generally lead to different final states, further enhancing the expressiveness of the model. This idea is depicted in Fig. 2.

The question arises: Can these Neural ODE systems be connected to stable physical systems, specifically analog circuits? In nonlinear resistive analog networks, for example, the system converges to the same stable state irrespective of the initial conditions ($\boldsymbol{s}$) as long as conductances ($\boldsymbol{\theta}$) and input voltages ($\boldsymbol{x}$) are fixed. Mathematically, this is represented by $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \mathbf{0}$. When a Neural ODE system reaches steady-state, at $t = \infty$, the differential equation $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{s}(t), t, \boldsymbol{x}, \boldsymbol{\theta})$ effectively reduces to an implicit nonlinear equation $\boldsymbol{f}(\boldsymbol{s}_\infty, \boldsymbol{x}, \boldsymbol{\theta}) = \mathbf{0}$. Furthermore, if the differential equation has a single equilibrium point at steady-state, then it can be represented by a nonlinear analog circuit. This idea is depicted in Fig. 3, which shows that for some specific ODEs, when $\boldsymbol{\theta}$ and $\boldsymbol{x}$ are fixed, the system always converges to the same steady-state irrespective of initial state of $\boldsymbol{s}$.

In the next section, we develop the algorithm for solving optimization problems in which the constraint is a nonlinear implicit equation, which is the steady-state solution of ODEs with one equilibrium state at $t = \infty$.

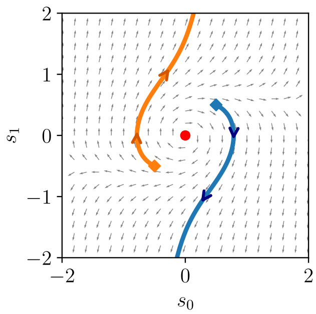

Figure 2: A example of a typical differential equation with no constraint on the trajectories.

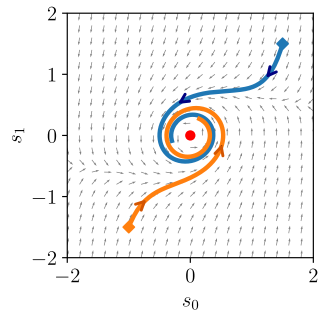

Figure 3: An example of a differential equation where all trajectories converge to the same equilibrium state.

4 Adjoint Method for Implicit Nonlinear Equations

Let $\boldsymbol{s}, \boldsymbol{s}_{\text{true}} \in \mathbb{R}^{n_s}$ and $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$. Suppose we have the function $C(\boldsymbol{s}, \boldsymbol{s}_{\text{true}}) : \mathbb{R}^{n_s} \times \mathbb{R}^{n_s} \to \mathbb{R}$, where $\boldsymbol{s}(\boldsymbol{\theta})$ is the solution of the implicit function $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{0}$ for a function $\boldsymbol{f} : \mathbb{R}^{n_s \times n_x \times n_\theta} \to \mathbb{R}^{n_s}$. What is $\frac{\mathrm{d}C}{\mathrm{d}\boldsymbol{\theta}}$?

C is a function of $\boldsymbol{s}$, which is a function of $\boldsymbol{\theta}$. Therefore:

$$\frac{\mathrm{d}\hat{C}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial C}{\partial \boldsymbol{s}} \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\boldsymbol{\theta}} \tag{7}$$

The differentiability of $\hat{C}$ with respect to $\boldsymbol{\theta}$ depends on the differentiability of $\boldsymbol{s}$ with respect to $\boldsymbol{\theta}$. By assuming that $\boldsymbol{s}(\boldsymbol{\theta})$ exists and is the solution of the implicit function $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{0}$, the implicit theorem guarantees the differentiability of $\boldsymbol{s}(\boldsymbol{\theta})$ [3]. Therefore, at the points $\boldsymbol{s} = \boldsymbol{s}(\boldsymbol{\theta})$, the following relationship holds:

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{s}} \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} \tag{8}$$

Since $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{\theta}) = \boldsymbol{0}$ everywhere, $\frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{s}(\boldsymbol{\theta}), \boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} = \boldsymbol{0}$. Assuming that $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{s}}$ is a non-singular matrix, the equation above can be expressed as:

$$\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\boldsymbol{\theta}} = -\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{s}}\right)^{-1} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} \tag{9}$$

Substituting equation (9) into equation (7) yields:

$$\frac{\mathrm{d}\hat{C}}{\mathrm{d}\boldsymbol{\theta}} = \left[-\frac{\partial C}{\partial \boldsymbol{s}} \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{s}}\right)^{-1}\right] \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} \tag{10}$$

Let $\boldsymbol{\lambda}(\boldsymbol{\theta}) \in \mathbb{R}^{n_s \times 1}$ be the solution of term inside the square brackets.

$$\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{s}}\right)^T \boldsymbol{\lambda}(\boldsymbol{\theta}) = -\left(\frac{\partial C}{\partial \boldsymbol{s}}\right)^T \tag{11}$$

Then the gradient can be written as:

$$\frac{\mathrm{d}\hat{C}}{\mathrm{d}\boldsymbol{\theta}} = \boldsymbol{\lambda}^T(\boldsymbol{\theta}) \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} \tag{12}$$

The algorithm for solving optimization problems with implicitly-defined constraints is summarized in Alg. 2. It is much simpler than Alg. 1 and has a direct interpretation in analog circuits.

5 Interpretation of the Adjoint Variable in Analog Circuits

In the context of analog circuits, we can think of $\boldsymbol{s}$ as the voltages $\boldsymbol{V}$ in the nodes of the circuit, $\boldsymbol{s}_{\text{true}}$ as the desired node voltages $\boldsymbol{V}_{\text{true}}$, $\boldsymbol{\theta}$ as the conductance matrix $\boldsymbol{G}$, $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{\theta}) = \boldsymbol{0}$ as the set of equations according to Kirchoff's current law, $\boldsymbol{s}(\boldsymbol{\theta})$ as the function that maps conductances to those voltages that satisfy Kirchoff's current law, and $\hat{C}(\boldsymbol{\theta}, \boldsymbol{s}_{\text{true}}) = C(\boldsymbol{s}(\boldsymbol{\theta}), \boldsymbol{s}_{\text{true}})$ as the function that evaluates the difference between the measured and desired voltages of the circuit.

Substituting $\boldsymbol{V}$ for $\boldsymbol{s}$, $\boldsymbol{G}$ for $\boldsymbol{\theta}$, and $\boldsymbol{I}$ for $\boldsymbol{f}$, the rest of the terms in Alg. 2 can be interpreted as follows:

- The terms $\frac{\partial \boldsymbol{I}}{\partial \boldsymbol{V}}$ is the derivative of the current with respect to the voltage, and therefore has a unit of Siemens.
- The term $\frac{\partial \boldsymbol{I}}{\partial \boldsymbol{G}}$ is the derivative of the current with respect to the conductance, and therefore has a unit of Volts.
- In most cases, the cost function C is the mean-squared-error. C therefore has units of Watts and $\frac{\partial C}{\partial \boldsymbol{V}}$ has a unit of Amperes.

With the link to analog circuits established, the adjoint algorithm can be interpreted as follows. The first step requires the solution of a nonlinear equation. This is exactly the steady-state (or DC) solution of an analog circuit. Let us denote these voltages as $\boldsymbol{V}^{\star}$. The current state of the circuit is compared against the desired state $\boldsymbol{V}_{\text{true}}$. The term $\frac{\partial C}{\partial \boldsymbol{V}^{\star}}$ measures how the node voltages should be modifed to reduce the discrepancy between the measured and the desired state. If we choose to inject a current of $\frac{\partial C}{\partial \boldsymbol{V}^{\star}}$ externally into the circuit, then since $\boldsymbol{\lambda} = -\left(\frac{\partial \boldsymbol{I}}{\partial \boldsymbol{V}^{\star}}\right)^{-T} \left(\frac{\partial C}{\partial \boldsymbol{V}^{\star}}\right)^{T}$ has units of Volts, $\boldsymbol{\lambda}$ is exactly the variation in the node voltages due to the injected current.

In a nonlinear analog circuit, the calculation of $\boldsymbol{\lambda}$ as presented above is impossible. Because of the presence of nonlinear elements, the injection of current into the circuit can completely change the state of a circuit. For example, a diode on the verge of conduction can switch on if the voltage across it increases as a result of the injected current. If this diode is connected across a large computational block, the block will be shorted to ground when the diode turns on, significantly altering the circuit between the two states. Since the calculation of $\boldsymbol{\lambda}$ requires the evaluation of the derivative $\frac{\partial \boldsymbol{I}}{\partial \boldsymbol{V}}$ at exactly the steady-state $\boldsymbol{V}^{\star}$ of the circuit, we have to ensure that the injected current does not disturb the state of the circuit. One approach is to scale the injected current $\left(\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{V}}\right)$ by some factor β to ensure that it is sufficiently small. Voltage variations in the nodes can then be measured and divided by β to obtain the unscaled version.

In summary, while the adjoint method can indeed be implemented on an analog circuit, the gradient calculated is an approximation since the act of injecting a current to measure the derivatives disturbs the circuit.

6 Connection to EP and Verification

To validate our analysis in the previous section, we designed a simple circuit, as an illustrative example, to learn the XOR dataset. The circuit has an input layer with dimensions 4×2, followed by a pair of up-down diodes to introduce nonlinearity, and an output layer with dimensions 4×2.

The XOR dataset has two inputs and one output. The inputs are represented by X_1 and X_2 and the output by the difference of Y_1 and Y_2. This is to account for the constraint that resistances cannot assume negative values. X_3 and X_4 represent the bias voltages and are of opposite polarity. We considered the node voltages V_1, V_2, Y_1, Y_2 as our state variables.

As an example, consider the conductance $G^{\dagger}$ (circled in the schematic). It is connected between nodes V_1 and Y_1. As a result, the voltage drop across it is $V^{\dagger} = V_1 - Y_1$. According to equation (6), $\left[\frac{\mathrm{d}C}{\mathrm{d}G^{\dagger}}\right]_{\text{Adjoint}} = \boldsymbol{\lambda}^{T} \frac{\mathrm{d}I}{\mathrm{d}G^{\dagger}}$. $\frac{\mathrm{d}I}{\mathrm{d}G^{\dagger}}$, in this example, is of size 1×4. Since $G^{\dagger}$ is connected to only V_1

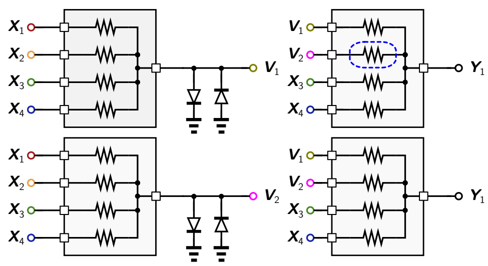

Figure 4: Schematic diagram for learning the XOR dataset.

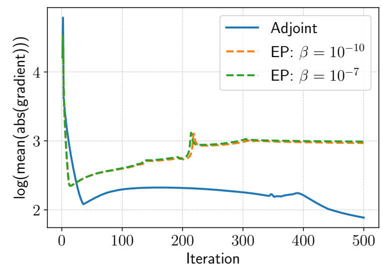

Figure 5: Plot of the gradients during training.

and Y_1, $\frac{\mathrm{d}I}{\mathrm{d}G^\dagger}$ will only have entries in the positions for nodes V_1 and Y_1, corresponding to λ_1 and λ_3, and these will be of opposite polarity. Thus, the evaluation of $\left[\frac{\mathrm{d}C}{\mathrm{d}G^\dagger}\right]_{\mathrm{Adjoint}}$ yields $(\lambda_1 - \lambda_3)V^\dagger$. This shows that the update to conductance $G^\dagger$ is the product of the voltage drop at steady-state and the variation due to the injected current across the two nodes it is connected to.

Let's now compare this result to the one from EP. In EP, the update equation for $G^\dagger$, as applied to this circuit is $\left[\frac{\mathrm{d}C}{\mathrm{d}G^\dagger}\right]_{\mathrm{EP}} \approx \frac{1}{2\beta}[(V_\beta^\dagger)^2 - (V_0^\dagger)^2]$, where $V_0^\dagger$ is the voltage drop across $G^\dagger$ before the current was injected, and $V_\beta^\dagger$ is the voltage drop after the current was injected. If we express $V_\beta^\dagger$ as $V_0^\dagger + \Delta V$, then $\left[\frac{\mathrm{d}C}{\mathrm{d}G^\dagger}\right]_{\mathrm{EP}} \approx \left(\frac{\Delta V}{\beta}\right)(V_0^\dagger) + \frac{1}{2\beta}(\Delta V)^2$. If the current injected is very small, $(\Delta V)^2 \to 0$, then $\left[\frac{\mathrm{d}C}{\mathrm{d}G^\dagger}\right]_{\mathrm{Adjoint}} \approx \left[\frac{\mathrm{d}C}{\mathrm{d}G^\dagger}\right]_{\mathrm{EP}}$, where $(\lambda_1 - \lambda_3) \approx \frac{\Delta V}{\beta}$.

The circuit was optimized using using both the adjoint and the EP methods [4]. A plot of the log of the mean absolute values of all the conductances across iterations is shown in Fig. 5. Initially, the gradients from both methods exhibit close proximity. Over time, however, the approximations inherent in the EP method introduce biases that cause its gradients to diverge from the true gradient. In contrast, the gradients obtained via the adjoint method continue to get smaller, indicating that the circuit is getting closer to the desired state.

7 Conclusion

A central theme of this work is the impact of constraints in optimization problems on neural network design and training rules. We explored two types of constraints: nonlinear implicit equations and differential equations. The adjoint method emerged as a unifying framework, linking Neural ODEs and EP and offering efficient gradient computation. This has practical implications, especially in analog computation, where we have interpreted the adjoint variable as the variation in node voltages in an analog circuit.

Looking ahead, we aim to focus on the efficient implementation of Neural ODEs in analog hardware. Additional future work could explore other types of constraints constraints that are more suitable for hardware design in systems other than analog circuits, opening up new avenues for efficient and adaptable neural network training.

References

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

[2] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation, 2017.

[3] Matthias Heinkenschloss. Implicit contraints, 2002. Available at `https://www.cmor-faculty.rice.edu/~cox/neuro/ImplConstraints.pdf`, accessed on 2023-09-29.

[4] Ebana framework. Available at `https://github.com/mawatfa/ebana`.