


Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering
(34.th cycle)

Dynamic Neural Networks and Brain-inspired Computing

Gianluca Zoppo

* * * * * *

Supervisor
Prof. Fernando Corinto, Supervisor

Doctoral Examination Committee:
Prof. Mauro Forti, Referee, Università Degli Studi Di Siena
Prof. Qiangfei Xia, Referee, University of Massachusetts
Prof. Michele Bonnin, Politecnico di Torino
Prof. Ronald Tetzlaff, Technische Universität Dresden
Prof. Yuchao Yang, Peking University

Politecnico di Torino
May 6th, 2022

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Gianluca Zoppo
Turin, May 6th, 2022

Summary

In the last decade, machine learning has drawn considerable interest in the Information Technology industry as a consequence to the promising performance achieved by graphics processing units. Although the current capabilities of artificial intelligence are constantly increasing, the energy required to train models is becoming unfeasible due to the use of conventional von Neumann architectures. This characteristic appears to be particularly critical in the context of Deep Learning where the networks parameters have to be routed from memory to processors to compute gradients, and subsequently routed back to memory to perform the update. This limitation suggests to move beyond the current architectural approach and take inspiration from the brain to rethink of novel architectures. Among the different emerging technologies, the memristor stands out as a promising candidate.

Neurological research suggests that neural representation is highly dynamic, encoding multiple types of tasks and stimuli by the joint activity of interconnected populations of neurons. Among the many proposed models, dynamic neural networks are able to mimic these biological complex phenomena and have been extensively studied from the hardware implementation's point of view.

This thesis explores the field of dynamic neural networks and aims to exploit memristor's programmability to implement the equilibrium point learning technique known as Equilibrium Propagation to train continuous-time recurrent neural networks and weakly-coupled oscillatory neural networks in solving associative memory and classifications tasks. Since the neural connectivity matrix of these models can be implemented in memristive crossbars, a detailed analysis of noise and systematic contributions of this fundamental building block is provided. The overall goal of this work is to evaluate the performance of dynamic neural networks mapped on simulated analogue memristor-based platforms that take into account device and circuits' non-idealities.

Acknowledgements

I had the privilege to pursue my doctoral studies at Politecnico di Torino to conduct research in the emerging field of neuromorphic computing and combining my mathematical background with the new frontiers in AI. The research results here presented would have not been possible without the supervision I received from my advisor Prof. Fernando Corinto. I am grateful for your unconditional support and invaluable guidance. You have given me the freedom to pursue various projects without objection and helped me growing both as a scientist and a person. Thanks for being such a positive influence, always with a smile and a friendly professional attitude.

Research is exciting, but can be often discouraging, nevertheless, Francesco Marrone has always been by my side. Your presence has been very important in this journey. We shared moments of difficulties but also of big excitement. I really wish you all the best for your future career! A warm hug goes to my colleagues and great friends Riccardo Gervasi, Andrea Napolitano and Ada Palamà that have supported me during my ups and downs and with whom I had the best coffee breaks. You always had a word of encouragement and this really motivated me to keep going and finish my studies. I am also very grateful to Andrea Costamagna for his valuable help in extending my research projects and his inspiring enthusiasm. Last but not least, I can not forget to thank Jacopo Secco. You have been both a friend and a mentor with a charismatic personality teaching me that it is never too late to jump into new adventures and being always ambitious.

An additional big thank is due to Prof. R. Stanley Williams and his group at Texas A&M for co-advising, and for opening the way to amazing projects and ideas I never imagined to be part of. I really appreciated the time you and Prof. Samuel Palermo have committed to providing me with detailed and constructive feedback. Your professional guidance throughout my research career have been much appreciated. A special thank goes to Anil Korkmaz. I am really grateful for all the interesting discussions and exchanges we had. Your attitude to work and your friendly and professional commitment have been extremely priceless.

A warm thank goes to both Profs. Michele Bonnin and Marco Gilli for their fruitful feedback and help with collaborations and extensions of my work. An additional thank goes to Profs. Marcello Delitala and Sergio Rolando for their

invaluable support in my unforgettable experience as a teaching assistant.

Some special words of gratitude go to all my friends for being so much present, giving me their invaluable supports during the most difficult moments of my PhD program. Giulia, Martin, Martina and Aurora, you have always been my major source of encouragement.

Finally, I thank my family for their unconditional support and love. I am thankful to my parents who have always believed in me and given me everything I need to follow my dreams. Ale, you are my role model and my best friend. You and Martina have taught me to never let any obstacles stop me from making my dreams a reality. I am so lucky to have you by my side. Franci thank you for being an important part of my story. You have always supported, encouraged, and helped me to get through this important journey in the most positive way. You have given me the extra strength and motivation to get things done.

Finally, the last words go to my grandparents, my rocks that I miss so much. This thesis is dedicated to you.

"If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake."

Yann LeCun

Contents

List of Tables

List of Figures

Introduction

Motivations

In the last decade, as a result of the great performance enhancement achieved by Graphics Processing Units (GPUs), machine learning has drawn considerable interest in the Information Technology (IT) industry [1]. A GPU is a specialized processor designed to accelerate multiple and simultaneous computations by taking advantage of its parallel processing architecture. This peculiarity significantly facilitates machine learning operations and speeds up data pipelines. Although the state-of-the-art capabilities of Artificial Intelligence (AI) models constantly increase, the energy required to train and exploit those models becomes unsustainable. This is mainly due to the conventional architecture adopted by the current computers that separate the processing unit from the memory unit (i.e. von Neumann architecture). This characteristic appears to be particularly critical in the context of deep learning.

The Back-Propagation algorithm was developed in 1986 by David Rumelhart, Geoffrey Hinton and Ronald Williams for training multi-layer neural networks [2]. The algorithm consists in an iterative training scheme that aims to minimize the difference between the system output and a desired target. This technique makes use of the chain rule of differentiation for composite functions and compute gradients by recursively propagate the error signal from the output layer to the network stratification. Besides its extraordinary ability, the learning process needs to shuttle back and forth large amounts of data between memory and processing units making the algorithm computationally demanding and energy consuming. At each iteration, the networks parameters have to be routed from memory to processors to compute gradients, and subsequently routed back to memory to perform the update. This important limitation suggests to move beyond the current architectural approach and take inspiration from the brain to rethink of new, unconventional non-von Neumann architectures.

To this end, the search of innovative computing platforms that could offer new, low power processing methods and architectures has intensified [3, 4]. A neuromorphic computing approach aims to overtake the current conventional digital processing by exploiting complex dynamics and nonlinear phenomena emerging from the

physics of non-volatile memory devices such as memristors [5, 6].

Recently, memristors have played a key role as synaptic devices for bio-plausible neural networks: their peculiar switching properties and signal storing capability are well suited for emulating synapse functionality of biological neural networks [7, 8]. Neurological research suggests that neural representation is highly dynamic, encoding multiple types of tasks and stimuli by the joint activity of interconnected populations of neurons. A neuromorphic computation attempts to mimic these biological complex phenomena by modelling and implementing each part of the neuronal network into hardware with the aim of running machine learning algorithms.

Hardware approaches have remarkably accelerated the inference process of neural networks showing promising performances in terms of energy efficiency, speed and accuracy. However, despite their fascinating potential for neuromorphic applications, memristive devices reveal undesirable properties when it comes to programming their conductance [9, 10]:

1) The conductance update is inherently stochastic exhibiting cycle-to-cycle and device-to-device variabilities;

2) Memristive devices can be programmed up to a finite conductance range;

3) Memristors have a non-linear conductance response.

These effects can be mitigated by performing an offline training on conventional computers and then transferring the computed weights onto memristive hardware by means of complex tuning protocols. However, in the case of in situ learning, such techniques can not be used since devices have to be repeatedly programmed throughout the process.

A second challenge to face is the non-locality of most training algorithms: weight update do not exclusively depend on the pre- and post- synaptic neurons. Among the many proposed bio-plausible learning techniques [11], the novel learning framework known as Equilibrium Propagation (EP) has drawn particular interest in the neuromorphic computing community due to the locality of its weight update rule [12]. Recently, various works have proposed hardware implementations of EP for training recurrent and spiking neural networks paving the way for building fully analog neural networks supporting on-chip learning [13, 14, 15, 16, 17, 18, 19].

Thesis organization

This thesis is structured as follows. Chapter 1 introduces the field of Machine Learning focusing on learning techniques used in Dynamic Neural Networks. Chapter 2 introduces the emerging field of Neuromorphic computing and describes

simulated analogue computing platforms that exploit memristors' conductance programmability to implement the novel algorithm Equilibrium Propagation to train recurrent neural networks and weakly-coupled oscillatory neural networks in solving associative memory and classifications tasks. Chapter 3 provides a detailed analysis of memristive crossbars to assess the system performance. Chapter 4 evaluates in terms of random and systematic errors the proposed memristor-based computing systems used to perform linear computations in probabilistic models. Chapter 5 concludes the thesis and gives some possible future lines of the work.

Main Contributions

This thesis explores the concept of dynamic neural network trained with equilibrium point learning rules. This class of dynamic system is modelled from a circuital implementation point of view providing the mathematical tools to analyse its ability to solve machine learning tasks. The key contributions of this work are summarized as follows:

1. A unified formulation of the three different equilibrium point learning rules, Recurrent Back-Propagation, Contrastive Learning and Equilibrium Propagation was used to simplify the different notations found in the current literature;

2. The Spike-timing-dependent plasticity (STDP)-compatible weight change proposed by the authors in [20] for training asymmetric neural networks is mathematically justified by the analysis of geometrical properties of the lagrangian multiplier used to solve the constrained optimization problem;

3. The symmetric learning rule for energy-based models is generalized to the case of gradient-like dynamics;

4. A unified formulation of the output random noise distributions of memristive crossbars is reported;

5. The wire resistance contribution when solving a matrix-vector multiplication using memristive crossbars is given in a closed mathematical expression;

6. The calibration stage for the recovering step used in memristive circuits is generalized to the case of feedback loops.

Most of these results are based on previous works:

- Zoppo, Gianluca and Marrone, Francesco and Bonnin, Michele and Corinto, Fernando". "Equilibrium Propagation and (Memristor-based) Oscillatory Neural Networks". In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. (Accepted)

- Zoppo, Gianluca and Korkmaz, Anil and Marrone, Francesco and Palermo, Samuel and Corinto, Fernando and Williams, R. Stanley. "Analog Solutions of Discrete Markov Chains via Memristor Crossbars". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2021), pp. 4910-4923.

- Zoppo, Gianluca and Marrone, Francesco and Corinto, Fernando. "Local learning in Memristive Neural Networks for Pattern Reconstruction". In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE (2021), pp. 1-5.

- Zoppo, Gianluca and Marrone, Francesco and Corinto, Fernando. "Equilibrium propagation for memristor-based recurrent neural networks". In: *Frontiers in neuroscience* (2020), p. 240.

- Zoppo, Gianluca and Marrone, Francesco and Corinto, Fernando. "A Continuous-time Learning Rule for Memristor–based Recurrent Neural Networks". In: *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (2019), pp. 494-497.

Chapter 1

Dynamic Neural Networks for Optimization

In the biological systems framework, the term "learning" refers to the result of progressive modifications that occur at the neuronal synapses level. Synaptic changes and global tasks live at separate levels, i.e. a single synapse is not aware of the overall purpose of the learning process. There must exist some underlying principles, governing the local synaptic modifications, that allow the system to accomplish the overall learning activity. However, these fundamental rules remain largely unknown.

A pioneer of this open field of research is Donald O. Hebb [21] who suggested in 1962 that synaptic changes occur whenever a coincidence of pre- and post-synaptic activity yields. The connection between two neurons is strengthened if they both simultaneously activate under an external input, and reduces if they activate separately. Although this is a basic local learning rule, it is well known that it can lead to powerful self-organization effects in relatively simple neural networks models.

From a different perspective, gradient descent offers a more sophisticated technique that provides to the system some guiding directions to reorganize the connections changes throughout the network. Among the large variety of different algorithms that have been proposed, Back-Propagation [2] and Back-Propagation through time [22] stand from the crowd. Even though these methods seem to be biologically implausible, they represent the workhorse of the AI era [23].

Inspired by computational neuroscience works, contemporary approaches aim to emulate the surprising efficiency of the Back-Propagation learning scheme by means of local learning rules [24, 25]. Since neurological research shows that neurons perform leaky integration and synapses are updated through local mechanisms, models based on Dynamic Neural Networks seem adapted to capture this complex biological dynamical behaviour. Thus, recent strategies have been used to investigate the learning phase of brain-inspired architectures [26]. The next sections introduce the mathematical tools used to analyse the stability of continuous-time neural network

and the principles of learning.

1.1 Nonlinear Dynamical Systems

Consider a generic nonlinear dynamical system:

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \theta) \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases} \tag{1.1}$$

where $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is a continuously differentiable vector-value function. The n-dimensional vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ represents the state variables of the system and θ is a system parameter. For sake of simplicity, this section considers $\theta \in \mathbb{R}$ but the same analysis can be generalized to $\boldsymbol{\theta} \in \mathbb{R}^m$ with $m > 1$.

Definition 1. *A solution $\hat{\boldsymbol{x}}$ of the algebraic equation:*

$$\mathbf{f}(\hat{\boldsymbol{x}}, \theta) = 0$$

is defined as equilibrium point, or fixed point of system (1.1). The equilibrium point $\hat{\boldsymbol{x}}$ is said to be

- *Stable, if for each $\epsilon > 0$, there exists $\delta = \delta(\epsilon) > 0$ such that*

$$\|\boldsymbol{x}(0) - \hat{\boldsymbol{x}}\| < \delta \Rightarrow \|\boldsymbol{x}(t) - \hat{\boldsymbol{x}}\| < \epsilon, \quad \forall t \geq 0$$

- *Locally asymptotically stable, if it is stable and there exists δ such that*

$$\|\boldsymbol{x}(0) - \hat{\boldsymbol{x}}\| < \delta \Rightarrow \lim_{t \to \infty} \|\boldsymbol{x}(t) - \hat{\boldsymbol{x}}\| = 0$$

An equilibrium point $\hat{\boldsymbol{x}}$ is said to be unstable if it is not stable.

Figure 1.1 shows the topological nature of the dynamic system (1.1) near an equilibrium point $\hat{\mathbf{x}}$. If $\hat{\mathbf{x}}$ is asymptotically stable, we may find a constant δ such that any trajectories that start in its neighborhood $B(\hat{\mathbf{x}}, \delta) = \{\mathbf{x} : \|\mathbf{x} - \hat{\mathbf{x}}\| < \delta\}$ will eventually tend to $\hat{\mathbf{x}}$.

Definition 2. *The region of attraction of an equilibrium point is the set of all starting points $\boldsymbol{x}(0) = \boldsymbol{x}_0$ such that the solution $\boldsymbol{x}(t, \boldsymbol{x}_0)$ of the system (1.1) satisfies*

$$\lim_{t \to \infty} \|\boldsymbol{x}(t, \boldsymbol{x}_0) - \hat{\boldsymbol{x}}\| = 0.$$

The concept of stability of an equilibrium point of dynamic systems is of crucial importance in the analysis of nonlinear dynamic systems. However, the analytical calculation of the exact region of attraction might be difficult or even impossible. Let us define as

$$\mathbf{J_f}(\hat{\mathbf{x}}) = (J_{ij}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\hat{\mathbf{x}}, \theta) & \dots & \frac{\partial f_1}{\partial x_n}(\hat{\mathbf{x}}, \theta) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\hat{\mathbf{x}}, \theta) & \dots & \frac{\partial f_n}{\partial x_n}(\hat{\mathbf{x}}, \theta) \end{bmatrix} \tag{1.2}$$

the Jacobian matrix of the system evaluated at the equilibrium point $\hat{\mathbf{x}}$.

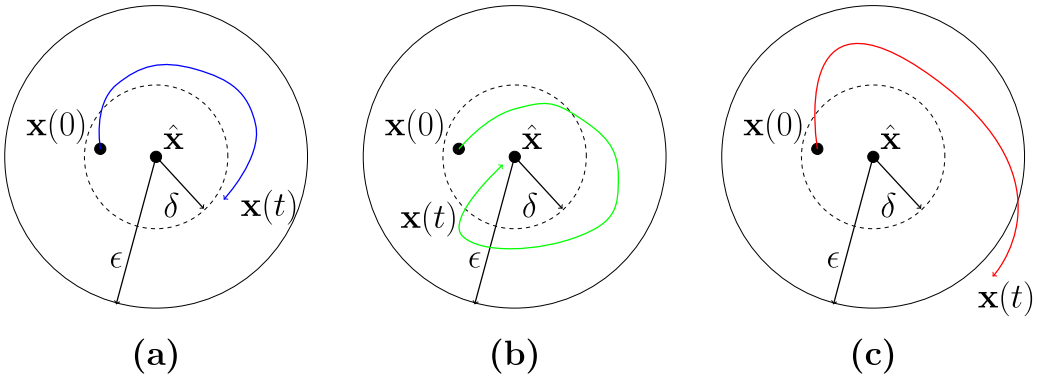


Figure 1.1: Definition of (a) stable, (b) asymptotically stable and (c) unstable equilibrium point $\hat{\mathbf{x}}$.

Lyapunov's first method makes use of the eigenvalues of $\mathbf{J_f}(\hat{\mathbf{x}})$ to examine the local asymptotic stability of $\hat{\mathbf{x}}$:

Theorem 1. *(Lyapunov's first Method)*
Let $\hat{\boldsymbol{x}}$ be an equilibrium point of the dynamical system (1.1).

- *$\hat{\boldsymbol{x}}$ is asymptotically stable if the real parts of all the eigenvalues λ_i $\forall i = 1, \dots, n$ of $\mathbf{J_f}(\hat{\boldsymbol{x}})$ are negative, i.e. $Re(\lambda_i) < 0$ $\forall i = 1, \dots, n$;*
- *$\hat{\boldsymbol{x}}$ is unstable if there exists at least one k with $0 < k < n$ such that $Re(\lambda_k) > 0$.*

In large scale dynamical systems, the direct computation of the eigenvalues of the Jacobian is usually complicated. The following theorem provides some useful conditions that can be used to verify the nature of the jacobian matrix's eigenvalues.

Theorem 2. *(Gershgorin Circle Theorem)*
Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$. Then every eigenvalue of $\boldsymbol{A}$ lies in one of the Gershgorin circles

$$D_i = \{z \in \mathbb{C} : |z - A_{ii}| \leq S_i\} \quad \vee \quad D_j = \{z \in \mathbb{C} : |z - A_{jj}| \leq S_j\} \tag{1.3}$$

where $S_i = \sum_{j=1, j \neq i}^{n} |A_{ij}|$ and $S_j = \sum_{i=1, i \neq j}^{n} |A_{ij}|$.

As a result of this theorem the following property holds:

Corollary 1. *Let $\hat{\boldsymbol{x}}$ be an equilibrium point of the system (1.1), $\boldsymbol{J_f}(\hat{\boldsymbol{x}})$ be the jacobian evaluated at the equilibrium point and $S_i = \sum_{j=1,j\neq i}^{n} |J_{ij}|$. If $J_{ii} < 0$ and $S_i < |J_{ii}|$ for $i = 1, \ldots, n$ then $\hat{\boldsymbol{x}}$ is a locally asymptotically stable equilibrium point.*

Proof. The Gershgorin circles for $\mathbf{J_f}(\hat{\mathbf{x}})$ are given by:

$$D_i = \{z \in \mathbb{C} : |z - (J_{ii}, 0)| \leq S_i\}. \tag{1.4}$$

Theorem 2 guarantees that all eigenvalues of $\mathbf{J_f}(\hat{\mathbf{x}})$ lie in D_i for some $1 \leq i \leq n$. Since $J_{ii} < 0 \ \forall i = 1, \ldots, n$ then the centers $(J_{ii}, 0)$ are all located on the negative real half-axis of the complex plane. $|J_{ii}|$ is the distance between each center and the origin of the complex plane and the circles radius $S_i < |J_{ii}|$ for $i = 1, \ldots, n$. This implies that every D_i is entirely contained in the negative real half-axis of the complex plane. Thus, the real part of all eigenvalues is negative and $\hat{\mathbf{x}}$ is asymptotically stable for Theorem 1. ■

In some applications, one may require that a system has a unique asymptotically stable equilibrium. The equilibrium $\hat{\mathbf{x}}$ is said to be globally asymptotically stable, whenever the trajectory $\mathbf{x}(t, \mathbf{x}_0)$ approaches $\hat{\mathbf{x}}$ as $t \to \infty$ regardless of the initial condition $\mathbf{x}_0$. The following theorem is a well-established result used to verify the global stability of equilibria.

Theorem 3. *(Lyapunov's second Method) Let $\hat{\boldsymbol{x}}$ be an equilibrium point for the system in Eq. (1.1). Let $V : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function such that:*

a) $V(\hat{\boldsymbol{x}}) = 0$ *and* $V(\boldsymbol{x}) > 0 \ \forall \boldsymbol{x} \neq \hat{\boldsymbol{x}}$*;*

b) $\dfrac{dV}{dt}(\boldsymbol{x}) < 0, \forall \boldsymbol{x} \neq \hat{\boldsymbol{x}}$*;*

Then $\boldsymbol{x} = \hat{\boldsymbol{x}}$ is asymptotically stable. If $\dfrac{dV}{dt}(\boldsymbol{x}) \leq 0 \ \forall \boldsymbol{x} \neq \hat{\boldsymbol{x}}$ then $\boldsymbol{x} = \hat{\boldsymbol{x}}$ is stable. In addition, if it yields:

c) $V(\boldsymbol{x}) \to \infty$ *when* $\|\boldsymbol{x}\| \to \infty$*.*

then $\boldsymbol{x} = \hat{\boldsymbol{x}}$ is globally asymptotically stable and is the unique equilibrium point.

The Lyapunov theorem provides an alternative to the Gershgorin's conditions for the analysis of locally asymptotically stable equilibria. However, it does not say anything about the form and the way to construct suitable functions V, but at least gives sufficient conditions to examine. Figure 1.2 shows the geometric relation between the system trajectories in the neighborhood of a stable equilibrium and the Lyapunov function as described in Theorem 3.

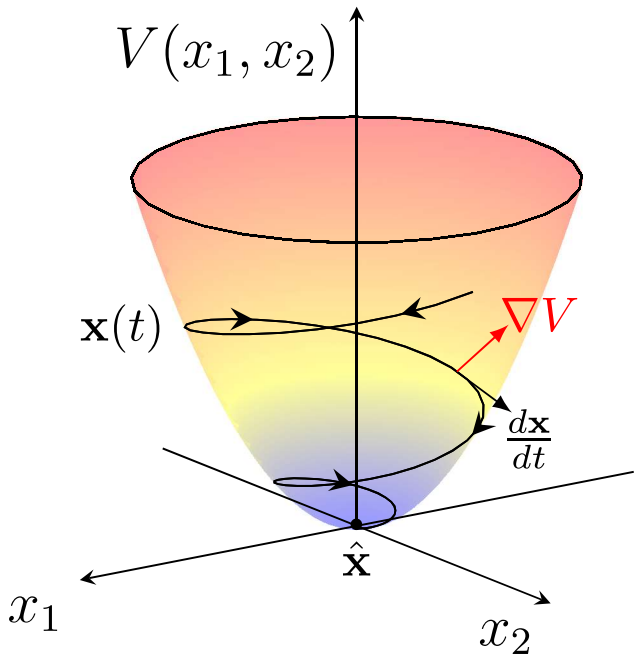


Figure 1.2: Geometric meaning of the Lyapunov function: the inner product of the Lyapunov function gradient ∇V and the tangent vector $\frac{d\mathbf{x}}{dt}$ is constantly negative. This means that the angle between these the two vectors is larger than 90 degrees and $V(x_1, x_2)$ monotonically decreases along trajectories.

Gradient-like systems, an important class of dynamic systems, have a natural candidate for the definition of a Lyapunov function. These kind of vector fields can be defined in the following compact matrix form:

$$\frac{d\mathbf{x}}{dt} = -\mathbf{A}(\mathbf{x}, t)\nabla_{\mathbf{x}} E(\mathbf{x}, \theta) \tag{1.5}$$

where $\mathbf{A}(\mathbf{x}, t)$ is a symmetric positive definite matrix whose entries may depend on the time t and the state vector $\mathbf{x}$ and $E : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is a differentiable scalar function. Whenever $\mathbf{A}(\mathbf{x}, t) = \mathbb{I}$, the vector field (1.5) defines a gradient (or conservative) system with potential function $U = -E$. These systems are well known in the classical mechanics framework.

Let us assume that θ is fixed and use the notation $E(\mathbf{x}) = E(\mathbf{x}, \theta)$ for sake of simplicity. The stability of the system (1.5) depends on the critical points' nature of the function E. A point $\mathbf{x}^*$ is a critical point of E whenever $\nabla_{\mathbf{x}} E(\mathbf{x}^*) = 0$. If all the second-order partial derivatives of E exist, $\mathbf{x}^*$ is defined a local minimum of E

if the hessian matrix

$$\mathbf{H}_E(\mathbf{x}^*) = \begin{bmatrix} \frac{\partial^2 E}{\partial x_1^2}(\mathbf{x}^*) & \dots & \frac{\partial^2 E}{\partial x_1 \partial x_n}(\mathbf{x}^*) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial x_n \partial x_1}(\mathbf{x}^*) & \dots & \frac{\partial^2 E}{\partial x_n^2}(\mathbf{x}^*) \end{bmatrix}$$

is positive definite, i.e. $\mathbf{x}^T \mathbf{H}_E(\mathbf{x}^*)\mathbf{x} > 0 \ \forall \mathbf{x} \in \mathbb{R}^n$.

Theorem 4. *The following properties hold:*

a) E decreases in time along system's trajectories;

b) The set of critical points of E is identical to the set of equilibrium points of the system (1.5);

c) Let $\boldsymbol{x}^*$ be an isolated local minimum of $E(\boldsymbol{x})$. Then $\boldsymbol{x}^*$ is a locally stable equilibrium point of the system (1.5);

Proof. a) $E(\mathbf{x})$ decreases in time since its derivative along the system's trajectories is:

$$\frac{dE}{dt} = \nabla_{\mathbf{x}} E^T \frac{dx}{dt} \overset{(1.5)}{=} -\nabla_{\mathbf{x}} E^T \mathbf{A}(\mathbf{x}, t) \nabla_{\mathbf{x}} E < 0 \tag{1.6}$$

under the condition that the matrix $\mathbf{A}$ is symmetric and positive definite.

b) The system converges to one of the possibly many local minima of the function E:

$$\frac{dE}{dt} = -\nabla_{\mathbf{x}} E^T \mathbf{A}(\mathbf{x}, t) \nabla_{\mathbf{x}} E = 0 \Leftrightarrow \nabla_{\mathbf{x}} E = \mathbf{0} \Leftrightarrow \frac{d\mathbf{x}}{dt} = \mathbf{0} \tag{1.7}$$

Thus, $\frac{dE}{dt} = 0$ only at the equilibrium points of the system (1.5).

c) Let us define the function

$$V(\mathbf{x}) = E(\mathbf{x}) - E(\mathbf{x}^*)$$

for $\forall \mathbf{x} \in B(\mathbf{x}^*, \delta)$. Observe that $V(\mathbf{x}^*) = E(\mathbf{x}^*) - E(\mathbf{x}^*) = 0$. The second order Taylor expansion about the equilibrium point gives:

$$V(\mathbf{x}) = V(\mathbf{x}^*) + \nabla_{\mathbf{x}} V(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) + (\mathbf{x} - \mathbf{x}^*)^T \mathbf{H}_V(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*).$$

For an isolated local minimum $\mathbf{x}^*$ it yields:

1) $\nabla_{\mathbf{x}} V(\mathbf{x}^*) = \nabla_{\mathbf{x}} E(\mathbf{x}^*) = \mathbf{0}$,
2) $\mathbf{H}_V(\mathbf{x}^*) = \mathbf{H}_E(\mathbf{x}^*)$ is positive definite.

It follows that

$$V(\mathbf{x}) = V(\mathbf{x}^*) + (\mathbf{x} - \mathbf{x}^*)^T \mathbf{H}_V(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) > V(\mathbf{x}^*) = 0$$

for all $\mathbf{x} \in B(\mathbf{x}^*, \delta)$ and $\mathbf{x} \neq \mathbf{x}^*$. Therefore, $V(\mathbf{x})$ is a Lyapunov function of the system:

1) $V(\mathbf{x}^*) = 0$ and $V(\mathbf{x}) > 0 \ \forall \mathbf{x} \in B(\mathbf{x}^*, \delta)$
2) $\frac{dV}{dt} = \frac{dE}{dt} < 0$

and $\mathbf{x}^*$ is a locally stable equilibrium point of the system (1.5).

■

The above proposition guarantees that $E(\mathbf{x})$ decreases in time and the system converges to an asymptotically stable equilibrium that coincides with a local minimum of $E(\mathbf{x})$. Since there may exist multiple equilibria, the system converges to the nearest local minimum close to the initial starting position. The speed of convergence depends on the choice of $\mathbf{A}$ [27].

Remark 1. *Theorem 4 provides a way to construct a Lyapunov function given the scalar function of a gradient-like system. If the scalar function is positive, i.e.* $E(\boldsymbol{x}) > 0 \ \forall x \neq x^*$ *then it coincides with the (global) Lyapunov function.*

1.1.1 Gradient Descent

The attraction behaviors of equilibria provide the basis for solving many different applications. Let us consider the following unconstrained optimization problem: find a vector $\mathbf{x} \in \mathbb{R}^n$ that minimizes the scalar function $E = E(\mathbf{x})$. This function is often called cost, energy or objective function. Optimization problems require the uniqueness of the equilibrium to avoid convergence towards local minima (undesired spurious responses) and hence ensure global optimization. One of the simplest and popular method used in optimization, namely gradient descent algorithm, makes use of the gradient system defined in Eq. (1.5) with $\mathbf{A}(\mathbf{x}, t) = \eta \mathbb{I}$:

$$\frac{d\mathbf{x}}{dt} = -\eta \nabla_{\mathbf{x}} E(\mathbf{x}) \tag{1.8}$$

where $\eta > 0$ is usually called the learning parameter. As a result of Theorem 4, the evolution of the system state variable $\mathbf{x}$ results in the minimization of $E(\mathbf{x})$ as the time flows. The discrete-time version of the steepest-descent can be formulated as:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \tilde{\eta} \nabla_{\mathbf{x}} E(\mathbf{x}_k), \quad k \geq 1 \tag{1.9}$$

where $\tilde{\eta}$ is the learning rate that (implicitly) depends on the integration time. Equation (1.9) represents the fundamental building block of the famous Back-Propagation algorithm.

1.1.2 Dynamic Neural Networks

Let the n-dimensional vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ represents the state variables of the continuous-time dynamic neural networks given by:

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}), \quad \mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^n \\ \mathbf{y} = \mathbf{g}(\mathbf{x}) \end{cases} \tag{1.10}$$

where $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^{n \times n} \times \mathbb{R}^n \to \mathbb{R}^n$ is a continuously differentiable vector field, $g(\cdot)$ is a monotone increasing activation function such that $g'(x) \neq 0 \; \forall x \in \mathbb{R}$, $\mathbf{W}$ is the matrix of connecting weights W_{ij}, and $\mathbf{I}$ is a vector of external inputs. Dynamic neural networks require to have many asymptotically stable equilibrium points whenever they are employed to simulate neural associative memories or to implement feature extraction in image processing [28, 29]. For Theorem 1, this happens whenever one of the following condition hold:

- The system (1.10) is a gradient-like (or gradient) system with energy E;
- The jacobian matrix of the system (1.10) yields all the hypotheses defined in the Corollary 1 of the Gershgorin Circle Theorem.

1.2 Dynamic Neural Learning

There are two main concepts of dynamic neural learning: equilibrium point learning and dynamic temporal learning [30, 31]. The former is designed for implementing neural associative memories and aims to teach the system to converge to prescribed equilibria. The requirement of this dynamic learning is the stability of the equilibrium points. The latter adjusts the parameters of a neural system such that the system state follows desired trajectories in time. It is interesting to observe that this last learning process can be considered as the generalization of the equilibrium point learning algorithm. When the time becomes long enough, the system state will reach a steady state defined by the prescribed trajectory. This section focuses on the former method.

The objective of an equilibrium point learning scheme is to find the weights W_{ij} so that, for a given initial condition $\mathbf{x}^0$, the fixed point of the system corresponds to a desired target value $\mathbf{T}$.

This is achieved by minimizing a suitable cost $\mathbf{C}(\mathbf{T}, \mathbf{g}(\hat{\mathbf{x}}))$ that measures the discrepancy between the target and the current output of the system at equilibrium $\hat{\mathbf{y}} = \mathbf{g}(\hat{\mathbf{x}})$ by iteratively moving in the direction of the steepest descent as defined by:

$$\Delta W_{ij} = -\eta \frac{\partial \mathbf{C}}{\partial W_{ij}}(\mathbf{T}, \mathbf{g}(\hat{\mathbf{x}})) = -\eta \nabla_{\mathbf{y}} \mathbf{C}^T G'(\mathbf{x}) \frac{\partial \hat{\mathbf{x}}}{\partial W_{ij}}. \tag{1.11}$$

where $G'(\hat{\mathbf{x}}) = \text{diag}(g'(\hat{x}_1), \ldots, g'(\hat{x}_n))$ is an invertible matrix.

In general, it is usually preferable to learn a set of target patterns and minimize $\mathcal{C}(\mathcal{T}, \hat{\mathbf{x}}) = \sum_j \mathbf{C}(\mathbf{T}^j, \hat{\mathbf{x}})$ with $\mathcal{T} = \{\mathbf{T}^1, \ldots, \mathbf{T}^m\}$. However, due to the linearity of differentiation, it is sufficient to analyse the estimation of the gradient for one single pattern. Depending on the choice of $\mathbf{W}$, network's units can be split in three sets: input, hidden and output. For sake of simplicity, the following analyses focus on the case where input and output units coincide. However, the following algorithms work either in the case of input-hidden-output hierarchy with feedbacks connections or fully connected networks.

1.2.1 Recurrent Back-Propagation

Recurrent Back-Propagation was first introduced by Almeida and Pineda who independently obtained the same results and developed an iterative scheme to adjust the synaptic weight matrix of a dynamic neural network [32, 33]. The idea is to force the neural network to converge, for fixed input and initial state, to a desired fixed–point attractor. As for feedforward neural network, this is achieved by minimizing a particular loss function associated to the neural network parameters. The novelty of this method is that the error signal is "back-propagated" by introducing an analog side network (i.e. an associated differential equation). This avoids the direct computation of the gradient by exploiting the second nonlinear dynamics of a side network.

The learning process consists in minimizing the loss function $\mathbf{C}$ which measures the difference between the desired fixed point $\mathbf{T}$ and the output of the system at the actual fixed point $\hat{\mathbf{x}}$. One way to drive the system to converge to a desired attractor is to let it evolve in the weight parameter space along trajectories which have opposite direction of the gradient as defined in Eq. (1.11). The derivative of $\hat{\mathbf{x}}$ with respect to W_{ij} is obtained by observing that the fixed points of (1.10) must satisfy the nonlinear equation:

$$\mathbf{f}(\hat{\mathbf{x}}, \mathbf{W}, \mathbf{I}) = \mathbf{0}. \tag{1.12}$$

Differentiating (1.12) with respect to W_{ij} one obtains:

$$\mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}}{\partial W_{ij}} + \frac{\partial \mathbf{f}^i}{\partial W_{ij}} = \mathbf{0} \tag{1.13}$$

where

$$\frac{\partial \mathbf{f}^i}{\partial W_{ij}} = [0, \ldots, 0, \underset{\substack{\downarrow \\ i^{th}}}{\frac{\partial f_i}{\partial W_{ij}}}, 0, \ldots, 0]^T$$

.

Solving (1.13) in terms of $\frac{\partial \hat{\mathbf{x}}}{\partial W_{ij}}$, the gradient descent defined in Eq. (1.11) becomes:

$$\Delta W_{ij} = -\eta \nabla_{\mathbf{y}} \mathbf{C}^T G'(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}}{\partial W_{ij}} = \eta \nabla_{\mathbf{y}} \mathbf{C}^T G'(\hat{\mathbf{x}}) \mathbf{J}_{\mathbf{f}}^{-1}(\hat{\mathbf{x}}) \frac{\partial \mathbf{f}^i}{\partial W_{ij}} \tag{1.14}$$

Unfortunately, (1.14) requires a matrix inversion to compute the error signal but, considering

$$\hat{\mathbf{z}} = [\mathbf{J}_{\mathbf{f}}^T(\hat{\mathbf{x}})]^{-1} G'(\hat{\mathbf{x}}) \nabla_{\mathbf{y}} \mathbf{C} \tag{1.15}$$

one can avoid this inversion by introducing an associated dynamical system. Indeed, (1.15) is equivalent to:

$$\mathbf{J}_{\mathbf{f}}^T(\hat{\mathbf{x}}) \hat{\mathbf{z}} - \nabla_{\mathbf{x}} \mathbf{C} = 0 \tag{1.16}$$

where $\nabla_{\mathbf{x}} \mathbf{C} = G'(\hat{\mathbf{x}}) \nabla_{\mathbf{y}} \mathbf{C}$. This equation can be seen as the steady state of the following associated network:

$$\frac{d\mathbf{z}}{dt} = \mathbf{J}_{\mathbf{f}}^T(\hat{\mathbf{x}}) \mathbf{z} - \nabla_{\mathbf{x}} \mathbf{C}. \tag{1.17}$$

Therefore, the system of differential equations is completely defined by:

$$\begin{cases} \frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}) \\ \frac{d\mathbf{z}}{dt} = \mathbf{J}_{\mathbf{f}}^T(\hat{\mathbf{x}}) \mathbf{z} - \nabla_{\mathbf{x}} \mathbf{C} \\ \Delta W_{ij} \propto \hat{z}_i \frac{\partial f_i}{\partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{W}) \end{cases} \tag{1.18}$$

Observe that the weights' update depends on the corresponding fixed points of the first two equations.

In conclusion, the learning system defined in Eqs. (1.18) minimizes the target cost function $\mathbf{C}$ by introducing an associated differential equation that computes the backpropagated error signal by avoiding the jacobian inverse's computation. However, the necessity of a side network for the propagation of error derivatives makes this technique highly different from emulating the brain complex computation. In any case, this method has established the basis for future works on this direction as shown in the next sections.

1.2.2 Contrastive Learning

Contrastive Learning is a supervised learning algorithm for training dynamic neural networks that admit an energy function governing the dynamics [34, 35]. The weights' update is proportional to the difference between the energy evaluated at the equilibrium points of two different running phases: free, without teacher signal, and forced, with teacher signal.

Let us assume that a dynamic neural network can be rewritten as a gradient-like (or possibly gradient) system governed by the free energy function $E^f(\mathbf{y}, \mathbf{W}, \mathbf{I})$:

$$\frac{d\mathbf{x}}{dt} = -G'(\mathbf{x})^{-1}\nabla_{\mathbf{x}}E^f(\mathbf{x}, \mathbf{W}, \mathbf{I}) = -\nabla_{\mathbf{y}}E^f(\mathbf{y}, \mathbf{W}, \mathbf{I}) \tag{1.19}$$

where $\mathbf{y} = \mathbf{g}(\mathbf{x})$ is the output of the system and $\nabla_{\mathbf{x}}E^f = G'(\mathbf{x})\nabla_{\mathbf{y}}E^f$. The goal of the learning algorithm is to adjust the network parameters so that for a given initial condition and an external input, the output of the system converges to a target equilibrium $\mathbf{T}$. In order to train the system, the algorithm considers a second gradient-like system:

$$\frac{d\mathbf{x}}{dt} = -\nabla_{\mathbf{y}}E^t(\mathbf{y}, \mathbf{W}, \mathbf{I}, \mathbf{T}) \tag{1.20}$$

with energy $E^t(\mathbf{y}, \mathbf{W}, \mathbf{I}, \mathbf{T}) = E^f(\mathbf{y}, \mathbf{W}, \mathbf{I}) + F(\mathbf{y}, \mathbf{W}, \mathbf{I}, \mathbf{T})$ where the function F denotes a teacher forcing term such that $F(\mathbf{y}, \mathbf{W}, \mathbf{I}, \mathbf{T}) = 0$ if the output of the system coincides with $\mathbf{T}$. Let us define the cost (or contrast) function

$$\mathbf{C}(\mathbf{W}, \mathbf{I}, \mathbf{T}) = E^t(\hat{\mathbf{y}}^t, \mathbf{W}, \mathbf{I}, \mathbf{T}) - E^f(\hat{\mathbf{y}}^f, \mathbf{W}, \mathbf{I})$$

and define the contrastive learning rule as:

$$\begin{aligned}
\Delta W_{ij} &= -\eta\frac{\partial C}{\partial W_{ij}} = -\eta\frac{\partial}{\partial W_{ij}}[E^t(\hat{\mathbf{y}}^t, \mathbf{W}, \mathbf{I}, \mathbf{T}) - E^f(\hat{\mathbf{y}}^f, \mathbf{W}, \mathbf{I})] \\
&= -\eta\left[\left((\nabla_{\mathbf{y}}E^t)^T\frac{\partial\hat{\mathbf{y}}^t}{\partial W_{ij}} + \frac{\partial E^t}{\partial W_{ij}}\right) - \left((\nabla_{\mathbf{y}}E^t)^T\frac{\partial\hat{\mathbf{y}}^f}{\partial W_{ij}} + \frac{\partial E^f}{\partial W_{ij}}\right)\right] = \\
&= -\eta\left(\frac{\partial E^t}{\partial W_{ij}} - \frac{\partial E^f}{\partial W_{ij}}\right)
\end{aligned} \tag{1.21}$$

where $\nabla_{\mathbf{y}}E^t(\hat{\mathbf{y}}^t) = \nabla_{\mathbf{y}}E^f(\hat{\mathbf{y}}^f) = 0$ at the equilibria of systems (1.19) and (1.20). This method aims to reduce the difference between the free and forced dynamics by recursively changing the systems' parameters according to the different energy configurations obtained. Even though it has been proven that for a specific case of neural networks this simple methodology equals the Back-Propagation algorithm efficiency [36], the objective function has some theoretical problems. It may happen that $\mathbf{C}$ assumes negative values if the free dynamics stabilizes in a minimum of the energy function that has a lower value than the one found by the forced dynamics [37]. This may result in a sudden change in the weight update and a temporary "unlearning" phase.

1.2.3 Equilibrium Propagation

Recently, Bengio and the authors in [12] proposed an alternative solution to the use of a side network by introducing Equilibrium Propagation, a learning technique used for energy-based models. The advantage of this approach is indeed the

requirement of just one kind of neural computation for the training phase of the network. This method is more flexible than the Contrastive Learning algorithm because the cost function $\mathbf{C}$ can change and the gradient descent is performed by locally perturbing the system. This avoids the undesirable theoretical problems outlined in Section 1.2.2. Firstly, inputs are clamped and the network relaxes to a fixed point which corresponds to a local minimum of the energy function. Secondly, after a small external error signal is injected, the network relaxes to a new but nearby fixed point which now corresponds to a rather lower cost value. The method recasts the approach proposed by Pineda and Almeida into the equivalent constrained optimization problem:

$$\min_{\mathbf{W}} \ \mathbf{C}(\mathbf{T}, \mathbf{y}) \quad \text{subject to} \quad \mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}) = \mathbf{0} \tag{1.22}$$

The constrained optimization problem defined in Eq. (1.22) can be solved by means of the Lagrange Multipliers' method [12, 38]. Let us introduce a new variable $\boldsymbol{\lambda} \in \mathbb{R}^N$, the Lagrange multiplier, and consider the Lagrangian function defined by:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{W}, \mathbf{T}, \mathbf{I}) = \mathbf{C}(\mathbf{T}, \mathbf{g}(\mathbf{x})) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}). \tag{1.23}$$

Let us keep the entries of the weight matrix $\mathbf{W}$ constant and solve for $(\boldsymbol{\lambda}, \mathbf{x})$:

$$\begin{cases} \nabla_{\boldsymbol{\lambda}} \mathcal{L} = \mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}) = \mathbf{0} \quad \rightarrow \quad \mathbf{x} = \hat{\mathbf{x}} \text{ equilibrium of (1.10)} \\ \nabla_{\mathbf{x}} \mathcal{L} = G'(\mathbf{x}) \nabla_{\mathbf{y}} \mathbf{C} + \mathbf{J}_{\mathbf{f}}^T(\mathbf{x}) \boldsymbol{\lambda} = \mathbf{0} \end{cases} \tag{1.24}$$

In order to find the solution $\hat{\boldsymbol{\lambda}}$ of the system (1.24), Bengio and his co-workers suggested to consider the following augmented dynamical system:

$$\frac{d\mathbf{x}^\beta}{dt} = \mathbf{f}^\beta(\mathbf{x}^\beta, \mathbf{W}, \mathbf{I}) = \mathbf{f}(\mathbf{x}^\beta, \mathbf{W}, \mathbf{I}) - \beta \nabla_{\mathbf{y}} \mathbf{C}(\mathbf{T}, \mathbf{y}^\beta) \tag{1.25}$$

where $\beta > 0$ is a forcing parameter and $\mathbf{y}^\beta = g(\mathbf{x}^\beta)$. A fixed point $\hat{\mathbf{x}}^\beta$ of the new dynamical system satisfies

$$\mathbf{f}^\beta(\hat{\mathbf{x}}^\beta, \mathbf{W}, \mathbf{I}) = \mathbf{0}. \tag{1.26}$$

Since (1.26) is constant $\forall \beta$, the total derivative with respect to β evaluated at $\beta = 0$ gives:

$$\mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}^\beta}{\partial \beta}\bigg|_{\beta=0} - \nabla_{\mathbf{y}} \mathbf{C}(\mathbf{T}, \hat{\mathbf{y}}) = \mathbf{0}. \tag{1.27}$$

where $\hat{\mathbf{x}}^{\beta=0} = \hat{\mathbf{x}}$. Let us now compare the two following equations:

$$\begin{cases} \mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}^\beta}{\partial \beta}\Big|_{\beta=0} - \nabla_{\mathbf{y}} \mathbf{C} = \mathbf{0} \\ \mathbf{J}_{\mathbf{f}}^T(\hat{\mathbf{x}}) \hat{\boldsymbol{\lambda}} + G'(\hat{\mathbf{x}}) \nabla_{\mathbf{y}} \mathbf{C} = \mathbf{0} \end{cases} \tag{1.28}$$

and by solving the system in terms of $\hat{\boldsymbol{\lambda}}$, it follows that:

$$\hat{\boldsymbol{\lambda}} = -[\mathbf{J}_{\mathbf{f}}^T(\hat{\mathbf{x}})^{-1} G'(\hat{\mathbf{x}}) \mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}})] \frac{\partial \hat{\mathbf{x}}^\beta}{\partial \beta}\Big|_{\beta=0}. \tag{1.29}$$

Since the remaining system parameters W_{ij} do not appear to have a similar straightforward closed formulation, a gradient descent on the Lagrangian gives the following iterative solution:

$$\Delta W_{ij} = -\eta \frac{\partial \mathcal{L}}{\partial W_{ij}} \propto -\hat{\boldsymbol{\lambda}}^T \frac{\partial \mathbf{f}^i}{\partial W_{ij}} = -\hat{\lambda}_i \frac{\partial f_i}{\partial W_{ij}} \tag{1.30}$$

where $\eta > 0$ is the learning rate. Unfortunately, Eq. (1.30) requires the inverse of the jacobian matrix and multiple matrix multiplications. However, the following results show some useful geometric properties of the lagrangian multiplier $\hat{\boldsymbol{\lambda}}$ that can be used to simplify the computation of the gradient descent.

Proposition 1. *The two following properties hold:*

1) The two gradient descents defined in Eqs. (1.11) and (1.30) have the same direction $\frac{\partial \mathcal{L}}{\partial W_{ij}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}, \boldsymbol{W}, \boldsymbol{T}, \boldsymbol{I}) = \frac{\partial \mathbf{C}}{\partial W_{ij}}(\boldsymbol{T}, \hat{\boldsymbol{y}})$;

2) It yields that $\nabla_{\boldsymbol{y}} \mathbf{C}^T \hat{\boldsymbol{\lambda}} = \nabla_{\boldsymbol{y}} \mathbf{C}^T \left(-\frac{\partial \hat{\boldsymbol{y}}^\beta}{\partial \beta}\Big|_{\beta=0} \right)$.

Proof. 1) Let us start computing the gradient of the Lagrangian function with respect to W_{ij}:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{ij}}(\hat{\mathbf{x}}, \hat{\boldsymbol{\lambda}}, \mathbf{W}, \mathbf{T}, \mathbf{I}) &\overset{(1.30)}{=} \hat{\boldsymbol{\lambda}}^T \frac{\partial \mathbf{f}}{\partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{W}) = \\
&\overset{(1.28)}{=} -\nabla_{\mathbf{y}} \mathbf{C}^T G'(\hat{\mathbf{x}}) \mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}})^{-1} \frac{\partial \mathbf{f}}{\partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{W}) = \\
&\overset{(1.13)}{=} \nabla_{\mathbf{y}} \mathbf{C}^T G'(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}}{\partial W_{ij}} = \\
&\overset{(1.11)}{=} \frac{\partial \mathbf{C}}{\partial W_{ij}}(\mathbf{T}, \hat{\mathbf{x}})
\end{aligned}$$

2) Let us transpose the second equation of system (1.28) and get:

$$\begin{cases} \mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}^\beta}{\partial \beta}\Big|_{\beta=0} - \nabla_{\mathbf{y}} \mathbf{C} = \mathbf{0} \\ \hat{\boldsymbol{\lambda}}^T \mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}}) + \nabla_{\mathbf{y}} \mathbf{C}^T G'(\hat{\mathbf{x}}) = \mathbf{0}^T \end{cases} \tag{1.31}$$

By multiplying the first equation with the row vector $\hat{\boldsymbol{\lambda}}^T$ it yields:

$$\begin{cases} \hat{\boldsymbol{\lambda}}^T \mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}^\beta}{\partial \beta}\Big|_{\beta=0} - \hat{\boldsymbol{\lambda}}^T \nabla_{\mathbf{y}} \mathbf{C} = 0 \\ \hat{\boldsymbol{\lambda}}^T \mathbf{J}_{\mathbf{f}}(\hat{\mathbf{x}}) + \nabla_{\mathbf{y}} \mathbf{C}^T G'(\hat{\mathbf{x}}) = 0 \end{cases} \tag{1.32}$$

By substituting the second equation into the first and using the symmetry property of the dot product it follows:

$$\nabla_{\mathbf{y}} \mathbf{C}^T \hat{\boldsymbol{\lambda}} = -\nabla_{\mathbf{y}} \mathbf{C}^T G'(\hat{\mathbf{x}}) \frac{\partial \hat{\mathbf{x}}^\beta}{\partial \beta}\Big|_{\beta=0} = \nabla_{\mathbf{y}} \mathbf{C}^T \left(-\frac{\partial \hat{\mathbf{y}}^\beta}{\partial \beta}\Big|_{\beta=0} \right) \tag{1.33}$$

where $\frac{\partial \hat{\mathbf{y}}^\beta}{\partial \beta} = G'(\mathbf{x}^\beta) \frac{\partial \hat{\mathbf{x}}^\beta}{\partial \beta}$.

■

Equation (1.33) shows that along the gradient's direction of the cost function $\mathbf{C}$ at $(\mathbf{T}, \mathbf{g}(\hat{\mathbf{x}}))$ the projection of the vector $-\frac{\partial \hat{\mathbf{y}}^\beta}{\partial \beta}\Big|_{\beta=0}$ equals the projection of the vector $\hat{\boldsymbol{\lambda}}$. This means that the information provided by Eq. (1.29) is redundant and one can simply consider $\hat{\boldsymbol{\lambda}} = -\frac{\partial \hat{\mathbf{y}}^\beta}{\partial \beta}\Big|_{\beta=0}$. Using Prop. 1, the update rule for the system weights is:

$$\begin{aligned} \Delta W_{ij} &= -\hat{\lambda}_i \frac{\partial f_i}{\partial W_{ij}} = \\ &= \eta \left(\frac{\partial \hat{y_i}^\beta}{\partial \beta}\Big|_{\beta=0} \right) \frac{\partial f_i}{\partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{W}) \end{aligned} \tag{1.34}$$

The training process can be therefore summarized in the following steps:

1) First, the input is clamped and the network follows the free dynamics defined by the gradient system (1.10) relaxing to the free fixed point $\hat{\mathbf{x}}$;

2) Secondly, a small perturbation of amplitude $-\beta \nabla C(\mathbf{T}, \mathbf{x})$ is introduced to the system (1.10) allowing the neural network (1.25) to relax to a new fixed point $\hat{\mathbf{x}}^\beta$;

3) Lastly, the weights of the matrix $\mathbf{W}$ are changed according to Eq. (1.34).

Since the update is non-symmetric, the learning process that follow the gradient descent defined in Eq. (1.34) is referred in the next sections to as Asymmetric Equilibrium Propagation.

The authors in [12] showed that gradient systems (or equivalently energy-based models) have a symmetric update rule. The following result shows that the asymmetric update rule defined in Eq. (1.34) is equivalent to a symmetric update whenever the systems admits a gradient-like formulation with $\mathbf{A}(\mathbf{x}, t) = G'(\mathbf{x})^{-1}$:

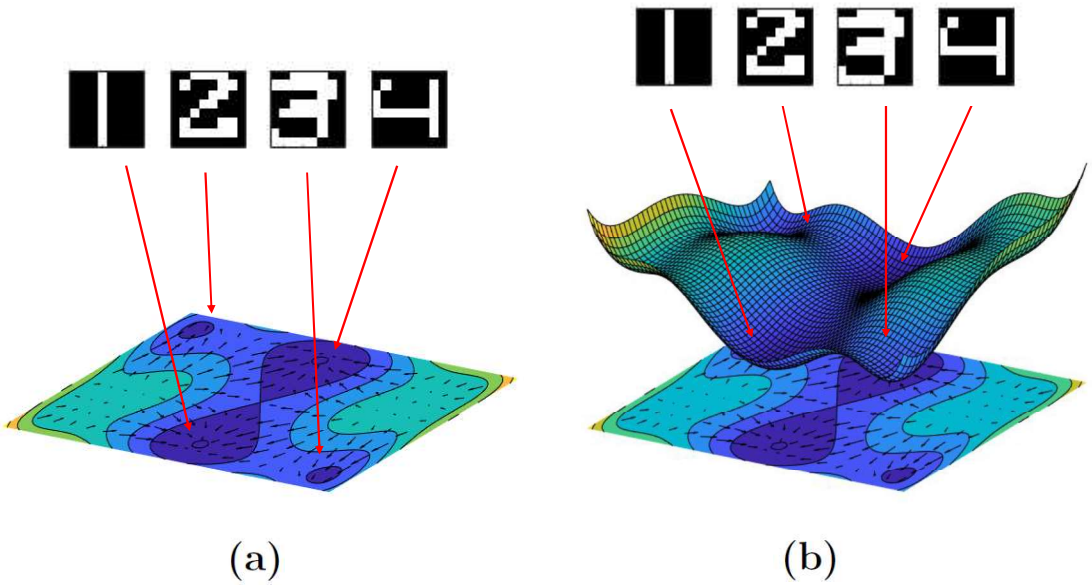


Figure 1.3: The objective of an equilibrium point learning scheme is to find the system parameters so that, for a given initial condition, the fixed point of the system corresponds to a desired target value. (a) Illustration of a generic dynamical system whose stable equilibria correspond to the learnt associated memories. (b) Illustration of a gradient-like system whose equilibria/memories correspond to the minima of the associated energy function.

Theorem 5. *Let the dynamical system defined in Eq. (1.10) be a gradient-like system of the form:*

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}) = -G'(\mathbf{x})^{-1} \nabla_{\mathbf{x}} E(\mathbf{x}, \mathbf{W}, \mathbf{I}) \\ \mathbf{y} = \mathbf{g}(\mathbf{x}) \end{cases} \tag{1.35}$$

where $E : \mathbb{R}^N \to \mathbb{R}$ *is a twice differentiable function. Let* $F(\boldsymbol{x}, \boldsymbol{T}, \boldsymbol{W}, \boldsymbol{I}, \beta) = E(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{I}) + \beta \boldsymbol{C}(\boldsymbol{T}, \boldsymbol{y})$ *be the augmented system energy, the learning rule defined in Eq. (1.34) is equivalent to:*

$$\Delta W_{ij} \propto -\lim_{\beta \to 0} \frac{\dfrac{\partial F}{\partial W_{ij}}(\hat{\mathbf{x}}^\beta, \mathbf{W}, \mathbf{T}, \mathbf{I}, \beta) - \dfrac{\partial F}{\partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{T}, \mathbf{W}, \mathbf{I}, 0)}{\beta} \tag{1.36}$$

where $\hat{\boldsymbol{x}}^\beta$ *is the fixed point of the system:*

$$\frac{d\mathbf{x}^\beta}{dt} = \mathbf{f}^\beta(\mathbf{x}^\beta, \mathbf{W}, \mathbf{I}) = -G'(\mathbf{x})^{-1} \nabla_{\mathbf{x}} F(\mathbf{x}^\beta, \mathbf{T}, \mathbf{W}, \mathbf{I}, \beta) \tag{1.37}$$

Proof. From Eq. (1.34), the gradient descent is:

$$\Delta W_{ij} = \eta \left(\frac{\partial \hat{y}_i{}^{\beta}}{\partial \beta}\Big|_{\beta=0} \right) \frac{\partial f_i}{\partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{W})$$

The $i-$th component of the vector field defined in Eq. (1.35) is

$$f_i = -\frac{1}{g'(x_i)} \frac{\partial E}{\partial x_i}(\mathbf{x}, \mathbf{W}, \mathbf{I}) = -\frac{\partial E}{\partial y_i}(\mathbf{x}, \mathbf{W}, \mathbf{I}) \tag{1.38}$$

for every $i = 1, \ldots, n$.

$$\begin{aligned} \Delta W_{ij} &= \eta \left(\frac{\partial \hat{y}_i{}^{\beta}}{\partial \beta}\Big|_{\beta=0} \right) \frac{\partial f_i}{\partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{W}) = \\ &= -\eta \left(\frac{\partial \hat{y}_i{}^{\beta}}{\partial \beta}\Big|_{\beta=0} \right) \frac{\partial^2 E}{\partial W_{ij} \partial y_i}(\hat{\mathbf{x}}, \mathbf{W}, \mathbf{I}) = \\ &= -\eta \left(\frac{\partial \hat{y}_i{}^{\beta}}{\partial \beta}\Big|_{\beta=0} \right) \frac{\partial^2 E}{\partial y_i \partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{W}, \mathbf{I}) \end{aligned}$$

where in the last equality we used the symmetry property of the mixed second-order partial derivatives of twice differentiable functions. Let us now observe that $E(\hat{\mathbf{x}}, \mathbf{W}, \mathbf{I}) = F(\hat{\mathbf{x}}, \mathbf{T}, \mathbf{W}, \mathbf{I}, 0)$, thus it follows:

$$\begin{aligned} \Delta W_{ij} &= -\eta \left(\frac{\partial \hat{y}_i{}^{\beta}}{\partial \beta}\Big|_{\beta=0} \right) \frac{\partial^2 F}{\partial y_i \partial W_{ij}}(\hat{\mathbf{x}}, \mathbf{T}, \mathbf{W}, \mathbf{I}, 0) = \\ &= -\eta \frac{d}{d\beta} \left[\frac{\partial F}{\partial W_{ij}}(\hat{\mathbf{x}}^{\beta}, \mathbf{T}, \mathbf{W}, \mathbf{I}, \beta) \right]_{\beta=0} \end{aligned} \tag{1.39}$$

■

Figure 1.3(a) shows the geometric interpretation of a generic dynamical system whose stable equilibria correspond to some associated memories. If the system admits a gradient-like formulation, the equilibria/memories correspond to the minima of the energy function as shown in Fig. 1.3(b). The next sections illustrates some examples of application of the previously presented learning rules.

1.3 Continuous-time Recurrent Neural Networks

A continuous-time recurrent neural network with n dynamical neural units is described by the following nonlinear differential equations in matrix form:

$$\begin{cases} \dot{\mathbf{x}} = -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} \\ \mathbf{y} = \mathbf{g}(\mathbf{x}) \end{cases} \tag{1.40}$$

where $\mathbf{I} \in \mathbb{R}^n$ is an input vector and $\mathbf{W} \in \mathbb{R}^{n \times n}$ is the interconnecting weight matrix. The neural output is $\mathbf{y} = \mathbf{g}(\mathbf{x})$ where $g(\cdot)$ is a continuous, monotone and increasing activation function. Using a matrix notation, $\mathbf{g}(\mathbf{x}) = [g(x_1), \dots, g(x_n)]^T \in \mathbb{R}^n$. For the local asymptotic stability of the equilibrium point $\hat{\mathbf{x}}$, all the eigenvalues of the jacobian matrix at the equilibrium must be examined. The jacobian of the dynamical system defined in Eq. (1.40) evaluated at the equilibrium is:

$$\mathbf{J_f}(\hat{\mathbf{x}}) = -\mathbb{I} + \mathbf{W}G'(\hat{\mathbf{x}}) \tag{1.41}$$

where $G'(\hat{\mathbf{x}}) = \text{diag}(g'(\hat{x_1}), \dots, g'(\hat{x_n}))$. As a result of Corollary 1, $\hat{\mathbf{x}}$ is an asymptotically stable equilibrium point of the neural network (1.40) if

$$W_{ii}g'(\hat{x}_i) < 1 \quad \text{and} \quad \sum_{j=1, j\neq i}^{n} |W_{ij}g'(\hat{x}_j)| < |-1 + W_{ii}g'(\hat{x}_i)|$$

for all $i = 1, \dots, n$.

Let $C(\mathbf{T}, \mathbf{x})) = \frac{1}{2}\|\mathbf{T} - \mathbf{g}(\hat{\mathbf{x}})\|^2$ be the cost function that measures the euclidean difference between the target vector and the output of the system at the equilibrium. The next two subsections reformulate Recurrent Back-Propagation and Equilibrium Propagation for the specific case of the neural network defined in Eq. (1.40).

A) Recurrent Back-Propagation

By computing and evaluating the jacobian of the system at the equilibrium point of the system we get that $\mathbf{J_f^T}(\hat{\mathbf{x}}) = -\mathbf{I} + G'(\hat{\mathbf{x}})\mathbf{W}^T$ and $\dfrac{\partial f_i}{\partial W_{ij}} = g(\hat{x}_j) = \hat{y}_j$. The learning system defined in Eq. (1.18) becomes

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} \\ \dfrac{d\mathbf{z}}{dt} = -\mathbf{z} + G'(\hat{\mathbf{x}})[\mathbf{W}^T\mathbf{z} + \mathbf{T} - \mathbf{g}(\hat{\mathbf{x}})] \\ \Delta W_{ij} \propto \hat{z}_i\hat{y}_j \end{cases} \tag{1.42}$$

As it is pointed out by [32], the stability of the dynamical system defined in Eq. (1.40) is a sufficient condition for the stability of the second equation defined in Eq. (1.42). To prove this, it is enough to observe that the two systems' jacobians $\mathbf{J_f}$ and $\mathbf{J_f^T}$ have the same eigenvalues, hence it follows that the fixed points of (1.40) must also be locally stable if the fixed points of (1.42) are locally stable.

B) Asymmetric Equilibrium Propagation

Let us define the extended vector field:

$$\begin{aligned} \frac{d\mathbf{x}}{dt} &= -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} - \beta\nabla_{\mathbf{y}}\mathbf{C}(\mathbf{T}, \mathbf{y}) = \\ &= -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} + \beta[\mathbf{T} - \mathbf{g}(\mathbf{x})] \end{aligned} \tag{1.43}$$

Let $\hat{\mathbf{y}} = \mathbf{g}(\hat{\mathbf{x}})$ be the output of the free dynamics ($\beta = 0$) at the equilibrium point and $\hat{\mathbf{y}}^\beta = g(\hat{\mathbf{x}}^\beta)$ the output of the nudged phase ($\beta > 0$). According to Eq. (1.34), the learning scheme is

$$\Delta W_{ij} \propto \frac{(\hat{y}_i^\beta - \hat{y}_i)}{\beta}\hat{y}_j. \tag{1.44}$$

1.4 Hopfield Neural Networks

A Hopfield Neural Network is a continuous-time recurrent neural network with n dynamical neural units and symmetric weights, i.e. $W_{ij} = W_{ji}$ with possibly $W_{ii} = 0$. The system can be described by the following nonlinear differential equations in matrix form:

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} \\ \mathbf{y} = \mathbf{g}(\mathbf{x}) \end{cases} \tag{1.45}$$

where $\mathbf{I} \in \mathbb{R}^n$ is an input vector and $\mathbf{W} \in \mathbb{R}^{n\times n}$ is the interconnecting symmetric weight matrix. The stability of the neural network (1.45) can be analysed by using the following energy:

$$E(\mathbf{x}, \mathbf{W}) = \sum_{i=1}^{N} \int_0^{x_i} sg'(s)ds - \frac{1}{2}\mathbf{g}(\mathbf{x})^T\mathbf{W}\mathbf{g}(\mathbf{x}) - \mathbf{I}^T\mathbf{g}(\mathbf{x}). \tag{1.46}$$

It is possible to verify that:

$$\nabla_{\mathbf{x}} E(\mathbf{x}, \mathbf{W}) = G'(\mathbf{x})\left[\mathbf{x} - \mathbf{W}\mathbf{g}(\mathbf{x}) - \mathbf{I}\right] \overset{(1.45)}{=} -G'(\mathbf{x})\dot{\mathbf{x}} \tag{1.47}$$

Hence, the dynamic equation of the continuous-time neural network (1.45) can be expressed in the form of a gradient-like system:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}) = -G'(\mathbf{x})^{-1}\nabla_{\mathbf{x}} E(\mathbf{x}, \mathbf{W}, \mathbf{I}). \tag{1.48}$$

Intuitively, the trajectory of the system defined in Eq. (1.48) moves downhill along the surface described by $E(\mathbf{x})$. In the case of sigmoidal or hyperbolic tangent activation functions, the matrix $G'(\mathbf{x})^{-1}$ defines a Riemannian metric which changes with time and influences the speed of convergence of the network. However, it does not alter the equilibrium points of the system. Thus, equilibria of (1.48) are minima of the energy function defined in Eq. (1.46):

$$\mathbf{f}(\mathbf{x}, \mathbf{W}, \mathbf{I}) = \mathbf{0} \Rightarrow \nabla_{\mathbf{x}} E(\mathbf{x}, \mathbf{W}, \mathbf{I}) = \mathbf{0} \tag{1.49}$$

The next subsection reformulates Contrastive Learning and Equilbrium Propagation for the case of Hopfield Neural Networks using the results found in Theorem 5.

A) Contrastive Learning

Let us rewrite the Hopfield energy defined in Eq. (1.46) in terms of the system's output $\mathbf{y} = \mathbf{g}(\mathbf{x})$ as:

$$E^f(\mathbf{y}, \mathbf{W}, \mathbf{I}) = \sum_{i=1}^{N} \int_0^{y_i} sds - \frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{I}^T\mathbf{y}.$$

If $F(\mathbf{y}, \mathbf{W}, \mathbf{I}, \mathbf{T}) = \frac{\gamma}{2}\|\mathbf{T} - \mathbf{y}\|^2$ with $\gamma \in \mathbb{R}_{-\{0\}}$, we can define the teaching energy as

$$E^t(\mathbf{y}, \mathbf{W}, \mathbf{I}, \mathbf{T}) = E^f(\mathbf{y}, \mathbf{W}, \mathbf{I}) + F(\mathbf{y}, \mathbf{W}, \mathbf{I}, \mathbf{T}).$$

Let $\mathbf{C}(\mathbf{W}, \mathbf{I}, \mathbf{T}) = E^t(\hat{\mathbf{y}}^t, \mathbf{W}, \mathbf{I}, \mathbf{T}) - E^f(\hat{\mathbf{y}}^f, \mathbf{W}, \mathbf{I})$ be the cost function, the learning dynamics is defined as the following set of differential equations:

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} \\ \dfrac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} + \gamma[\mathbf{T} - \mathbf{g}(\mathbf{x})] \\ \Delta W_{ij} = -\eta\left(\dfrac{\partial E^t}{\partial W_{ij}} - \dfrac{\partial E^f}{\partial W_{ij}}\right) = g(\hat{x}_i^t)g(\hat{x}_j^t) - g(\hat{x}_i^f)g(\hat{x}_j^f) \end{cases} \tag{1.50}$$

where $\hat{\mathbf{x}}^f$ and $\hat{\mathbf{x}}^t$ are the equilibrium points of the free and teaching dynamics, respectively.

B) Symmetric Equilibrium Propagation

Let $\mathbf{C} = \frac{1}{2}\|\mathbf{T} - \mathbf{g}(\mathbf{x})\|^2$ be the cost function that measures the euclidean distance between the target $\mathbf{T}$ and the output of the system $\mathbf{g}(\mathbf{x})$ at the equilibrium. Let us define the following augmented energy function:

$$F(\mathbf{x}, \mathbf{T}, \mathbf{W}, \beta) = E(\mathbf{x}, \mathbf{W}) + \beta\mathbf{C}(\mathbf{T}, \mathbf{x}) \tag{1.51}$$

where $\beta > 0$ is the forcing parameter and consider the corresponding gradient-like system:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}^\beta(\hat{\mathbf{x}}^\beta, \mathbf{W}) = -G'(\mathbf{x})^{-1}\nabla_{\mathbf{x}}F(\mathbf{x}, \mathbf{T}, \mathbf{W}, \beta) \tag{1.52}$$

or equivalently,

$$\frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{I} + \beta[\mathbf{T} - \mathbf{g}(\mathbf{x})]. \tag{1.53}$$

A fixed point $\hat{\mathbf{x}}^\beta$ of (1.53) corresponds to a local minimum of the augmented energy function. Observe that if $\beta = 0$ then systems (1.45) and (1.53) coincide. From Eq. (1.36), the learning scheme is defined as:

$$\begin{aligned} \Delta W_{ij} \propto -\frac{d}{d\beta}\left[\frac{\partial F}{\partial W_{ij}}(\hat{\mathbf{x}}^\beta, \mathbf{T}, \mathbf{W}, \mathbf{I}, \beta)\right]_{\beta=0} = \\ = \frac{d}{d\beta}\left(g(\hat{x}_i^\beta)g(\hat{x}_j^\beta)\right)\Big|_{\beta=0} \approx \frac{g(\hat{x}_i^\beta)g(\hat{x}_j^\beta) - g(\hat{x}_i)g(\hat{x}_j)}{\beta} \end{aligned} \tag{1.54}$$

with $\beta \approx 0$.

1.5 Kuramoto Model

The appearance of synchronized oscillations in the brain has prompted many studies using oscillatory networks to perform temporal coding of information such as associative memories. Generally, networks consist of coupled oscillators interacting with each other according to suitable learning rules, and the information is coded as phase-locked oscillations. The Kuramoto oscillator stand from the crowds as one of the most used mathematical models for oscillatory associative memory. The dynamics of the network is described in terms of the phase equations:

$$\frac{dx_i}{dt} = \sum_{j=1}^{n} W_{ij} \sin(x_j - x_i) \quad \forall i = 1, \ldots, n \tag{1.55}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state variable vector and $\mathbf{W} \in \mathbb{R}^{n \times n}$ is the coupling weight matrix. Whenever the interconnections are assumed to be symmetric, i.e. $W_{ij} = W_{ji}$ and $W_{ii} = 0$, the state stability of the system dynamics can be deduced by reformulating it as a gradient system whose potential function is:

$$E(\mathbf{x}, \mathbf{W}) = -\frac{1}{2} \sum_{i,j=1}^{N} W_{ij} \cos(x_i - x_j) \tag{1.56}$$

It is easy to verify that the dynamic equation of the dynamic neural network (1.55) can be expressed in the form:

$$\frac{dx_i}{dt} = -\frac{\partial E}{\partial x_i}(\mathbf{x}, \mathbf{W}) \tag{1.57}$$

for all $i = 1, \ldots, n$. This means that equilibria of (1.55) are minima of the potential function defined in Eq. (1.56). The next subsection reformulates Contrastive Learning and Equilbrium Propagation for the case of the phase dynamics defined by the Kuramoto model using the results found in Theorem 5.

A) Contrastive Learning

Let us define the potential function defined in Eq. (1.56) as the free energy $E^f(\mathbf{x}, \mathbf{W}, \mathbf{I})$. If $F(\mathbf{x}, \mathbf{W}, \mathbf{I}, \mathbf{T}) = \frac{\gamma}{2} \sum_{i=1}^{n} \cos(T_i - x_i)$ with $\gamma \in \mathbb{R}_{-\{0\}}$, we can define the teaching energy as

$$E^t(\mathbf{x}, \mathbf{W}, \mathbf{I}, \mathbf{T}) = E^f(\mathbf{x}, \mathbf{W}, \mathbf{I}) + F(\mathbf{x}, \mathbf{W}, \mathbf{I}, \mathbf{T}).$$

Let $\mathbf{C}(\mathbf{W}, \mathbf{I}, \mathbf{T}) = E^t(\hat{\mathbf{x}}^t, \mathbf{W}, \mathbf{I}, \mathbf{T}) - E^f(\hat{\mathbf{x}}^f, \mathbf{W}, \mathbf{I})$ be the cost function, the learning dynamics is defined as the following set of differential equations:

$$\begin{cases} \dfrac{dx_i}{dt} = \sum_{j=1}^{n} W_{ij} \sin(x_j - x_i) \\ \dfrac{dx_i}{dt} = \sum_{j=1}^{n} W_{ij} \sin(x_j - x_i) + \gamma \sin(T_i - x_i) \\ \Delta W_{ij} = -\eta \left(\dfrac{\partial E^t}{\partial W_{ij}} - \dfrac{\partial E^f}{\partial W_{ij}} \right) = \cos(\hat{x}_i^t - \hat{x}_j^t) - \cos(\hat{x}_i^f - \hat{x}_j^f) \end{cases} \tag{1.58}$$

for all $i, j = 1, \ldots, n$. The outputs $\hat{\mathbf{x}}^f$ and $\hat{\mathbf{x}}^t$ are the equilibrium points of the free and teaching dynamics, respectively.

B) Symmetric Equilibrium Propagation

The learning rule tries to modify the weights W_{ij} so that, for a given initial condition $\mathbf{x}_0$, the output $\hat{\mathbf{x}}$ of the system (1.55) corresponds to a desired target value $\mathbf{T}$. This is achieved by minimizing the cost $\mathbf{C}$ for a single pair of points $\mathbf{T}$ and $\mathbf{x}$:

$$C(\mathbf{T}, \mathbf{x}) = n - \sum_{i=1}^{n} \cos(T_i - x_i) \tag{1.59}$$

which measures the distance between a desired target $\mathbf{T}$ and the output state of the system $\mathbf{x}$. Observe that $C(\mathbf{T}, \mathbf{x}) \geq 0$ and $g(\cdot)$ is the identity function. Let us define the following augmented energy function:

$$F(\mathbf{x}, \mathbf{T}, \mathbf{W}, \beta) = E(\mathbf{x}, \mathbf{W}) + \beta C(\mathbf{T}, \mathbf{x}) \tag{1.60}$$

where $\beta \geq 0$ is the forcing parameter and consider the corresponding gradient system:

$$\frac{dx_i}{dt} = -\frac{\partial F}{\partial x_i}(\mathbf{x}, \mathbf{T}, \mathbf{W}, \beta) = \sum_{j=1}^{N} W_{ij} \sin(x_j - x_i) + \beta \sin(T_i - x_i) \tag{1.61}$$

for all $i = 1, \ldots, n$. From Eq. (1.36), the learning scheme is defined as:

$$\begin{aligned} \Delta W_{ij} &\propto -\frac{d}{d\beta} \left[\frac{\partial F}{\partial W_{ij}}(\hat{\mathbf{x}}^\beta, \mathbf{T}, \mathbf{W}, \mathbf{I}, \beta) \right]_{\beta=0} = \\ &= \frac{d}{d\beta} \cos(\hat{x}_i^\beta - \hat{x}_j^\beta) \Big|_{\beta=0} = \lim_{\beta \to 0} \frac{\cos(\hat{x}_i^\beta - \hat{x}_j^\beta) - \cos(\hat{x}_i - \hat{x}_j)}{\beta} \end{aligned} \tag{1.62}$$

Thus, the learning rule can be simply approximated in:

$$\Delta W_{ij} \propto \frac{\cos(\hat{x}_i^\beta - \hat{x}_j^\beta) - \cos(\hat{x}_i - \hat{x}_j)}{\beta}, \tag{1.63}$$

with $\beta \simeq 0$.

1.6 Conclusion

This chapter gave a mathematical treatment of gradient descent learning algorithms for neural networks using the general framework of nonlinear dynamical systems.

The system defined in Eq. (1.10) with initial conditions $\mathbf{x}_0$ is assumed to be convergent during the learning process. This happens when either the system is a gradient-like (or gradient) system or the jacobian matrix of the system yields all the hypotheses defined in the Corollary 1 of the Gershgorin Circle Theorem. Accordingly, the learning rules considered depend on the output of the system evaluated at equilibrium. The fixed point $\hat{\mathbf{x}}$ is a function of the system parameters $\mathbf{W}$ for fixed initial conditions and external inputs, the aim is to modify $\mathbf{W}$ so that $\hat{\mathbf{x}}$ approaches a desired target $\mathbf{T}$. In terms of gradient (or gradient-like) systems this results in potential functions that have as local minima the stable equilibria of the system.
The key contributions of this chapter are:

1. Three different learning rules, Recurrent Back-Propagation, Contrastive Learning and Equilibrium Propagation are presented in a unified formulation, as learning algorithms to continuous-time dynamic neural networks.

2. The authors in [20] assumes a priori a STDP-compatible weight change similar to the one found in Eq. (1.34) and show that the scalar product between this vector field and the gradient descent $\frac{\partial C}{\partial W}$ is negative. This guarantees the optimization of the objective function. The results found in Prop. 1 prove that the projection of the lagrange multiplier along the direction of minimization coincides with the contribution given by $-\frac{\partial \hat{\mathbf{y}}^\beta}{\partial \beta}$. This mathematically justifies the choice of the lagrangian multiplier as $\boldsymbol{\lambda} = -\frac{\partial \hat{\mathbf{y}}^\beta}{\partial \beta}$ and proves that the STDP-compatible weight change compute the exact gradient.

3. Theorem 5 generalizes Equilibrium Propagation for energy-based model to the case of gradient-like dynamics by considering a modified version of the energy used in [12]. This choice was dictated by the need of a Hopfield-like neural network that could be easily implemented by an electric circuit.

Chapter 2

Dynamic Neural Networks and Brain-inspired Computing

Neuromorphic computing is an emergent field of technology combining artificial intelligence, neuroscience, computing architecture and material sciences [39]. While modern computers had achieved notable performance, the ever-growing pressure for big data keeps on demanding a rapid development of sophisticated computing platforms. The execution of various computational tasks in current von-Neumann architectures needs to shuttle back and forth large amounts of data between memory and processing units. This requires enormous computing hardware resources and large power consumption during operations. For these reasons, the desire to replicate the biological neural activity to transport and process information has attracted significant attention from different research groups. Inspired by the human brain, neuromorphic systems have demonstrated remarkable computational efficiency improvements paving the way for a novel class of customized hardware architectures.

The availability of different types of emerging devices has pushed neuromorphic research into two broad areas of investigation:

1) bio-plausible computing: focused on emulating dynamics of biological synapses;

2) bio-inspired computing: focused on developing electronic systems that make use of algorithms inspired from biological mechanisms.

The former computing research has focused in building algorithms for Spiking Neural Networks that closely mimic biologically observed phenomena, such as spike-timing-dependent plasticity (STDP). The latter has focused on efficiently computing mathematical functions. A workhorse of this class is the feed-forward artificial neural network architecture. This thesis is focused on artificial synapses for bio-inspired neuromorphic computing systems that enable fast and sustainable machine intelligence.

Most neuromorphic approaches are based on pure CMOS technology. However, in the last decade several emerging nanoscale devices have been explored for implementing synapses in bio-inspired neural networks. Among these, the memristor stands out as a promising candidate.

The overall goal of this chapter is to review the most important characteristics of memristive devices and propose simulated analogue computing platforms that exploit memristors' conductance programmability to implement equilibrium point learning in dynamic neural networks. In particular, this section focuses on the novel algorithm Equilibrium Propagation used to train recurrent neural networks and weakly-coupled oscillatory neural networks in solving associative memory and classifications tasks. During the training stage, the network oscillates between the two phases defined in Chapter 1 and computes the gradient of the associated cost function. The results of this chapter are based on previous works "Equilibrium propagation for memristor-based recurrent neural networks" by Zoppo et al [13], "Equilibrium Propagation for Recurrent Neural Networks based on Resistive Switching devices: from circuit implementation to supervised machine learning" by Costamagna [40] and "Local Learning in Memristive Neural Networks for Pattern Reconstruction" by Zoppo et al [41].

2.1 Memristors

The memristor (contraction of memory resistor) was theoretically presented for the first time in 1970 by [5] as the fourth ideal circuit element, in addition to the capacitor, the resistor and the inductor.

The original definition was given in terms of a relation of the form $F(\phi, q) = 0$ where ϕ (voltage momentum or flux) and q (current momentum or charge) are the time integrals in $(-\infty, t]$ of the port voltage and current respectively. Under mild regularity assumptions, it is possible to express the voltage in terms of the current (or vice versa) in the forms:

$$\begin{cases} v = R(q)i \\ R(q) = \frac{d\phi}{dq} \end{cases} \qquad \begin{cases} i = G(\phi)v \\ G(\phi) = \frac{dq}{d\phi} \end{cases} \tag{2.1}$$

The quantities $R(q)$ and $G(\phi)$ depend on the entire history of the input variable and are defined as the memristor's *memristance* and *memductance* respectively. The concept of memristor was consequently generalized to the concept of *extended memristor* [42]. An extended memristor is defined in terms of the voltage v and the current i by

$$\begin{cases} i & = G(\phi, v, \mathbf{x})v \\ \dot{\mathbf{x}} & = g(\phi, v, \mathbf{x}) \\ \dot{\phi} & = v \end{cases} \tag{2.2}$$

where $\mathbf{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ is an internal vector with n state variables, $G(\phi, v, \mathbf{x})$ is the memductance and $g : \mathbb{R}^{n+2} \to \mathbb{R}^n$ is a continuous function that governs the evolution of vector $\mathbf{x}$. A similar definition for the voltage-controlled extended memristor can be obtained by interchanging the current momentum and the current with the voltage momentum and voltage. The peculiarities that discern a memristor within the set of all nonlinear dynamical systems, are:

- any zero-mean periodic input (voltage or current) yields a current-voltage loop in the first and the third quadrants of the $i - v$ plane;
- the loop is pinched at the origin (other additional intersections may occur);
- the loop has a shape varying with both the amplitude and frequency.

The choice of particular materials or specific physical mechanisms do not define memristors. This is clearly reported in [43] "all 2-terminal non-volatile memory devices based on resistance switching are memristors, regardless of the device material and physical operating mechanisms". The next section reports a brief introduction to some of the current memristive technologies.

2.1.1 Device Technologies

In the last two decades, memristive switching devices have been realized using many different physical mechanisms of material systems. Commonly, the process of programming into the high-conductance state is referred to as "SET" and programming into the low-conductance is "RESET". Among all the several variants, this thesis rapidly introduces Phase-Change Memories (PCMs), Conductive-Bridging (CBRAMs) and Resistive (ReRAMs) random access memories. For an extensive review on memristive physical devices, see [9, 10].

Phase-Change Memory RAM

PCMs are based on reversible crystallization (high-conductance state) and amorphization (low-conductance state) of the so-called phase change materials [44]. Potentiation and depression in PCM occur by modulating crystalline-to-amorphous volume ratio. The change of state of a PCM device is highly influenced by the temperature reached by the phase-change material when applying an electrical pulse to the device. RESET tends to be an abrupt procedure whereas the SET operation can be made incremental by repeated pulses that slowly crystallize the high-resistance plug within the device. The amorphous material gradually transforms into the crystalline state until a single crystal is reached. The typical crystallization temperature is about $500 - 600K$ [45]. In the SET operation, it is possible to program multiple resistance levels by varying the width of the pulse. In general, PCM-based artificial synapses have exhibited large scale integration, high scalability, and good

stability, making them promising candidates for neuromorphic platforms. However, the amorphization mechanism results in an inherently asymmetric weight update in PCM synapses due to the abrupt RESET that can not be easily gradually implemented. This may significantly affect network performances.

Conductive-Bridging RAM

The basic cell of CBRAMs, is a metal-ion conductor-metal system [46]. If an oxidation reaction occurs (i.e. loss of electrons) at an electrode, then the electrode is classified as an anode. If instead a reduction reaction (i.e. gain of electrons) occurs, then the electrode is classified as a cathode. Active electrodes are metals that participate in reactions that occur in the electrolyte in order to transport the electricity. Inert electrode are instead metals that do not interfere in any chemical reaction. The working principles of these synaptic devices rely on the formation and rupture of conduction channels composed of metal. During the potentiation process, a positive voltage is applied to the active electrode to oxidize metal into ions and electrons. Following the electric field, cations migrate toward the cathode and get reduced inside the electrolyte leading to different filament growth modes. During depression, a negative potential is applied to the active electrode, and the metal conduction channel destabilizes and breaks. CBRAMs have demonstrated to be promising candidates as next-generation non-volatile memory devices due to their fast speed, scalability, and ultra-low power consumption. However, they may suffer from limitations in terms of weight precision and update linearity/symmetry due to their inherent stochasticity.

Resistive RAM

Resistive RAM (ReRAM) is a non-volatile memory similar to CBRAM, except that the filament through the insulating thin-film is a chain of defects within an oxide [47]. The underlying metal-insulator–metal structure is compact, CMOS-compatibile and highly scalable, and the energy consumption per synaptic operation and programming currents can be made ultralow. However, the filament formation process is difficult to control: large variability through Poissonian statistics is inevitable. Once the filament is formed, it may be broken resulting in an high resistance state or re-formed resulting in a lower resistance state by triggering an electrical field and/or a local temperature increases. To date, ReRAM-based memristors are commercially available memory chips but similar to CBRAM devices, the fundamental filament formation and dissolution processes lead to unavoidable device variations, and the weight update linearity/symmetry requires optimizations for neuromorphic systems.

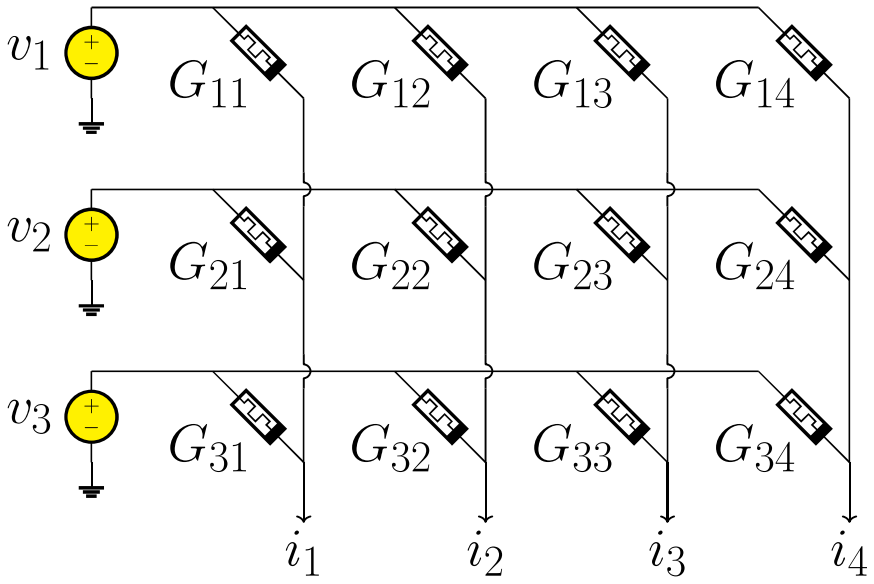


Figure 2.1: Illustration of a 3×4 memristive crossbar used to perform matrix-vector multiplication.

2.1.2 Crossbars

The product of a matrix and a vector is a fundamental operation widely used in a large variety of tasks that make use of linear algebra computing. However, in digital computing system this computation is a tedious and hardware intensive task. Its computation complexity grows as $O(nm)$, where n and m are the matrix dimensions. In contrast to using digital binary numbers, an analog accelerator encodes numbers using the continuously real valued circuit voltages and positive conductances of programmable resistive devices. The computation complexity can be reduced from $O(nm)$ to $O(1)$. The use of memristors for performing Matrix-Vector Multiplications (MVMs) is depicted in Fig. 2.1. An $n \times m$ crossbar array consists of n horizontal wires (word lines) and m vertical wires (bit lines). Each bit line connects to each word line through a memristor cell for a total of nm intersections. Let R_{jk} and G_{jk} denote the resistance and conductance of the $(jk)-$cell, where $G_{jk} = 1/R_{jk}$. Let the cells of the $k-$th column be programmed to their conductance values $G_{1k}, G_{2k}, \ldots, G_{nk}$. As a result of the Ohm's Law, by applying the voltages $v_1, v_2, \ldots, v_n$ to the n rows, a current of $G_{jk}v_j$ passes from the $(jk)-$th cell into the bit line. Then, from Kirchhoff's law, the total current from the $k-$th bit line is the sum of currents flowing through the column, as shown in Fig. 2.1. The output current i_k is the dot-product of input voltages at each row $\mathbf{v} = (v_1, v_2, \ldots, v_n)^T$ and cell conductances $\mathbf{G}_k = (G_{1k}, G_{2k}, \ldots, G_{nk})^T$ in a column, that is, $i_k = \mathbf{G}_k^T\mathbf{v}$. The total current $\mathbf{i}$ is the vector whose components correspond

to each bit line current i_k $\forall k = 1, \ldots, m$ and $\mathbf{i} = \mathbf{G}^T\mathbf{v}$ where $\mathbf{G} \in \mathbb{R}^{n\times m}$. For construction, the crossbar performs the product between the transpose of the matrix $\mathbf{G}$ and the column vector $\mathbf{v}$. For this reason, this operation may also be referred to as Vector-Matrix Multiplication (VMM), i.e. $\mathbf{i}^T = \mathbf{v}^T\mathbf{G}$.

The current output can be either converted into voltage by a transimpedance amplifier (TIA), or digitalized by ADCs for usage in digital systems, or forwarded to other analog systems. It is important to highlight that as opposed to traditional application specific integrated circuit (ASIC) matrix multipliers, the MVM and the matrix entries' storage take place in the same location. Thus, computation using memristor crossbars can be significantly faster and more power efficient compared to conventional digital computation due to this high intrinsic parallelism [48, 49, 50, 51, 52, 53].

2.2 Conductance Range

A challenge in crossbar computing is that a memristor can only be programmed within an available conductance range, i.e. $G_{ij} \in G_{range} = [G_{min}, G_{max}]$ $\forall i = 1, \ldots, n$, $\forall j = 1, \ldots, m$. G_{min} and G_{max} are the available minimum and the maximum memristor conductances. Currently, there are two different approaches to map matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ into the so-called conductance matrix $\mathbf{G} \in \mathbb{R}^{n\times m}$ [48, 50].

The matrix $\mathbf{A}$ can be transformed into the conductance matrix using the following linear transformation:

$$\begin{aligned} \gamma &= \frac{G_{max} - G_{min}}{A_{max} - A_{min}} \\ \delta &= G_{max} - \gamma A_{max} \\ \mathbf{G} &= \gamma \mathbf{A}^T + \delta \mathbf{1}_n \mathbf{1}_m^T \end{aligned} \tag{2.3}$$

where A_{min} and A_{max} correspond to the minimum and maximum entries in the matrix $\mathbf{A}$ and $\mathbf{1}_k$ is a column vector of k ones. This linear transformation can be done in a digital system before the analog computation.

Once the output vector from the crossbars shown in Fig. 2.1 is measured, one can get the correct result by means of the following inverse operations:

Proposition 2. *Let* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{v} \in \mathbb{R}^n$. *If* $\boldsymbol{i} = \boldsymbol{A}\boldsymbol{v}$ *and* $\tilde{\boldsymbol{i}} = \boldsymbol{G}^T\boldsymbol{v}$ *then*

$$\mathbf{i} = \frac{1}{\gamma}\left[\tilde{\mathbf{i}} - \left(\delta \sum_k v_k\right)\mathbf{1}_m\right]$$

Proof. Let us use the definition of the conductance matrix $\mathbf{G}$ and get:

$$\begin{aligned}\tilde{\mathbf{i}} &= \mathbf{G}^T\mathbf{v} = (\gamma\mathbf{A}^T + \delta\mathbf{1}_n\mathbf{1}_m^T)^T\mathbf{v} = \\ &= \gamma\mathbf{A}\mathbf{v} + \delta\mathbf{1}_m\mathbf{1}_n^T\mathbf{v} = \\ &= \gamma\mathbf{i} + \delta\mathbf{1}_m\mathbf{1}_n^T\mathbf{v} = \\ &= \gamma\mathbf{i} + \left(\delta\sum_{k=1}^{n} v_k\right)\mathbf{1}_m\end{aligned} \tag{2.4}$$

where $\mathbf{1}_n^T\mathbf{v} = (\sum_{k=1}^{n} v_k)$. Let us now solve for the correct output $\mathbf{i}$ and find the recovering linear transformation:

$$\mathbf{i} = \frac{1}{\gamma}\left[\tilde{\mathbf{i}} - \left(\delta\sum_{k=1}^{n} v_k\right)\mathbf{1}_m\right]. \tag{2.5}$$

■

Thus, after the analog to digital conversion, the output of the MVM can be recovered using Prop. 2 in a digital system.

Alternatively, a second approach usually employed is to represent one matrix element using the conductance difference of two memristors, i.e. a differential pair. The input voltage signals on two adjacent rows must have the same amplitude with opposite polarity. The differential calculation is performed by the direct current summation:

$$i_k = \sum_{j=1}^{n} G_{jk}^{+}v_j + G_{jk}^{-}(-v_j) = \sum_{j=1}^{n}(G_{jk}^{+} - G_{jk}^{-})v_j \quad \forall k = 1, \dots m \tag{2.6}$$

where $G_{jk}^{+} - G_{jk}^{-}$ is the scaled mapped matrix element A_{kj} of the matrix $\mathbf{A}$. Observe that this method enables to have negative values and provides a level of defect tolerance to the calculation: mitigate stuck device issues by adjusting the conductance of the coupled device. However, the use of differential pairs doubles the number of required memristor cells and thus the chip area.

The next sections show how memristive crossbars can be used to implement equilibrium point learning in dynamic neural networks.

2.3 Continuous-time Recurrent Neural Networks

Let us consider the nonlinear system defined in Eq. (1.40). It determines a continuous-time Recurrent Neural Network that can be implemented by an analog RC network circuit at the input of each amplifier [54] as shown in Figure 2.2. Here, the system state variable is directly fed back as input by a connection between the neurons' outputs and the input lines. Most of the algorithms used to

train neural networks require both positive and negative synaptic weights. This can be easily handled by doubling the crossbars in the network, as highlighted in Section 2.2 and shown in Fig. 2.2(a). This approach requires the use of the dual output neurons $y_k = g(v_k)$, $k = 1, \ldots, n$, with the two output terminals providing y_k and $-y_k$ as exemplified in Figure 2.2(b). The crossbar representing positive weights $\{G^+_{kj}\}_{k,j=1,\ldots,n}$ is connected to the positive output's neurons while the negative counterpart $\{G^-_{kj}\}_{k,j=1,\ldots,n}$ is connected to the complementary neurons outputs. Let v_k be the input voltage of the k-th neuron, one may write the current equation of the input node of the k-th neuron using Kirchhoff's current law as follows:

$$C_k \frac{dv_k}{dt} + \frac{v_k}{R_k} = \sum_{j=1}^{N} (G^+_{kj} - G^-_{kj}) y_j + I_k. \tag{2.7}$$

By defining the neural weights $W_{jk} = G^+_{kj} - G^-_{kj}$ one would get:

$$C_k \frac{dv_k}{dt} = -\frac{v_k}{R_k} + \sum_{j=1}^{N} W_{kj} y_j + I_k \tag{2.8}$$

which corresponds to the k-th component of system (1.40) with $C_k = R_k = 1$ and $v_k = x_k \ \forall k = 1, \ldots, n$.

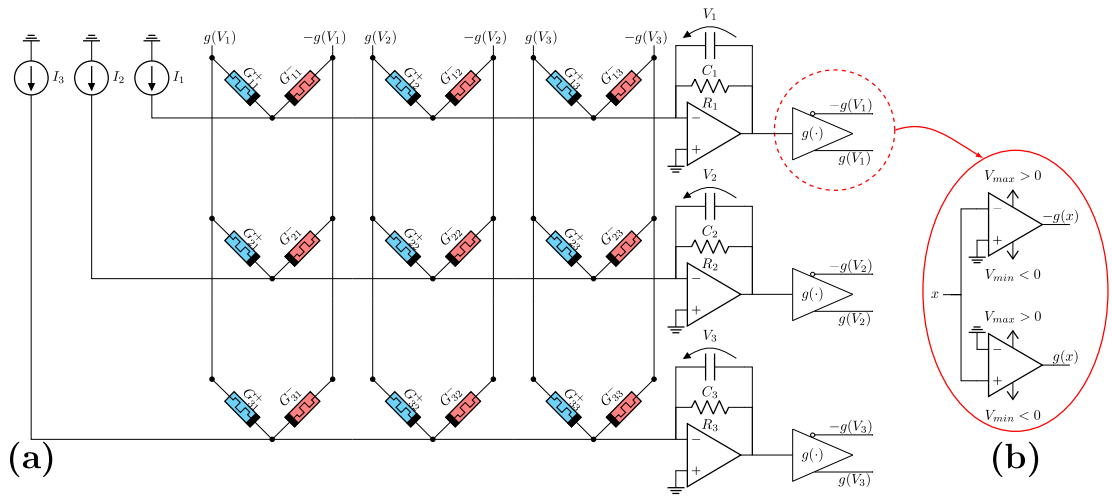


Figure 2.2: a) An example of a 3×3 Recurrent Neural Network with dual-crossbar bipolar weights implementation. b) A simple neuron implementation. Taken from [41] ©2021 IEEE.

Equilibrium Propagation relies on two different phases, a free phase and a forced phase. For the sake of implementation, it is advantageous to have a unique dynamical system that alternates between the two stages. This can be realized by means

of a digital clock with suitable frequency, range and duty cycle which alternates between the two dynamics. This is formally equivalent to considering the parameter β to be a time-dependent function

$$\beta(t) = \begin{cases} 0, & \text{free phase} \\ \beta, & \text{forced phase} \end{cases} \tag{2.9}$$

that implements the desired clock.

Memristor-based neural networks allows to adapt the synaptic weights W_{ij} continuously during an online training phase. Memristors' tunability is met by increasing the architecture complexity and adding appropriate supplementary circuit elements which enable the application of a series of discrete programming pulses that perform the weights update [55, 56].

Figure 2.3: Dataset of the 10 different resized 8×8 images taken from MNIST dataset [57].

2.3.1 Associative Memory

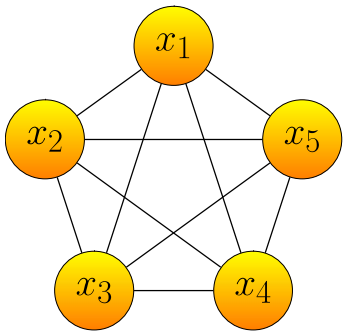
The memories/patterns to learn are the $m = 10$ different resized 8×8 images taken from MNIST dataset [57] depicted in Fig. 2.3. The network architecture consists of a fully connected neural network with no self-loops as shown in Fig. 2.4. Each image is reshaped into the column vectors $\mathbf{T}_k \ \forall k = 1, \ldots, m$ of size $n = 64$.

Generally, a standard recurrent neural network model is trained using the unsupervised Hebbian learning rule:

$$\mathbf{W} = \frac{1}{m} \sum_{k=1}^{m} (\mathbf{T}_k \mathbf{T}_k^T - \mathbb{I}_{n \times n})$$

where m is the number of memories and $\mathbf{T}_k \in \mathbb{R}^n$ is the $k-$th image. The resulting weight matrix is symmetric and the network defines a Hopfield model. This kind of model trained on uncorrelated patterns has an approximate capacity of $0.14n$ but this factor decreases significantly when patterns are correlated. Storkey [58] overcome this problem by introducing a correction in the update rule that reduces the correlation among patterns by recursively building $\mathbf{W} = \mathbf{W}_{m+1}$ as follows:

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \frac{1}{n}[(\mathbf{T}_k \mathbf{T}_k^T - \mathbb{I}_{n \times n}) - \mathbf{T}_k \mathbf{T}_k^T \mathbf{W}_k^T - \mathbf{W}_k \mathbf{T}_k \mathbf{T}_k^T]$$



(a)

	x_1	x_2	x_3	x_4	x_5
x_1	–	W_{12}	W_{13}	W_{14}	W_{15}
x_2	W_{21}	–	W_{23}	W_{24}	W_{25}
x_3	W_{31}	W_{32}	–	W_{34}	W_{35}
x_4	W_{41}	W_{42}	W_{43}	–	W_{45}
x_5	W_{51}	W_{52}	W_{53}	W_{54}	–

(b)

Figure 2.4: (a) Illustration of a fully connected neural network with 5 nodes. (b) Weights matrix $\mathbf{W}$ representing each edge influence in the network.

for each $k = 1, \ldots, m$. The Storkey rule equally distributes the memories in the parameter space increasing the capacity of the network. The asymptotic capacity is $n/\sqrt{2 \ln n}$, which represents an improvement over the Hebbian rule.

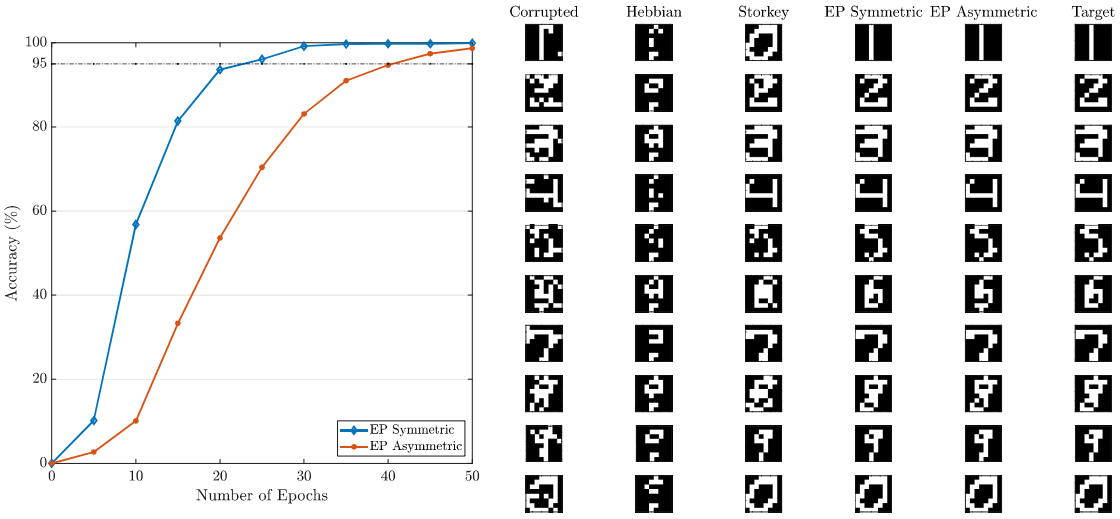
A small amount of noise can be useful for escape from local minima (i.e. spurious state) and look for the global minimum of the system. A popular approach for accelerating the learning is to start with a large contribution of noise and gradually decreases it as the learning proceeds. This process of adaption is similar to that in simulated annealing [59]. However, such addition of noise does not change dramatically the qualitative performance of the network for large dimensions.

From a different perspective, as described in Section 1.2, gradient descent offers a more sophisticated technique that provides to the system some guiding directions to reorganize the connections changes throughout the network. As described in Section 1.2.3, Equilibrium Propagation provides a more flexible learning technique compared to Recurrent Back-Propagation and Contrastive learning, thus Eqs (1.34) and (1.39) are used to descend along the gradient. The update of the weights is performed after each epoch by averaging the weight changes by the total number of examples m. This approach aims to avoid getting stuck in local minima and allows to reduce the total number of weights' updates. This is of particular interest in the case of in-situ learning in hardware implementations.

The activation function used is the hyperbolic tangent and the gains of the units are kept constant to 0.45. The learning rate is $\eta = 0.45$. In these simulations, patterns are presented by means of a constant input vector $\mathbf{I}$ in a cyclic fashion since random presentation of the patterns did not show any particular improvement. The teaching parameter is set to $\beta = 0.1$ and the number of epoch is 50. Time spans for the simulation of the dynamic system are chosen in order to guarantee the convergence of the state variables.

In order to assess the validity of the novel supervised algorithm for solving

associative memory's tasks, the same neural network was used and trained using Hebbian, Storkey, Symmetric and Asymmetric Equilibrium Propagation learning rules. Fig. 2.5(a) shows that the symmetric update rule easily recovers the target patterns already in the first 30 epochs. On the contrary, the asymmetric technique results in a slightly slower convergence. As shown in Fig. 2.5(b), results provide evidence that Equilibrium Propagation is perfectly able to reconstruct even in the presence of correlated patterns whereas the two unsupervised techniques converge to some local minima.

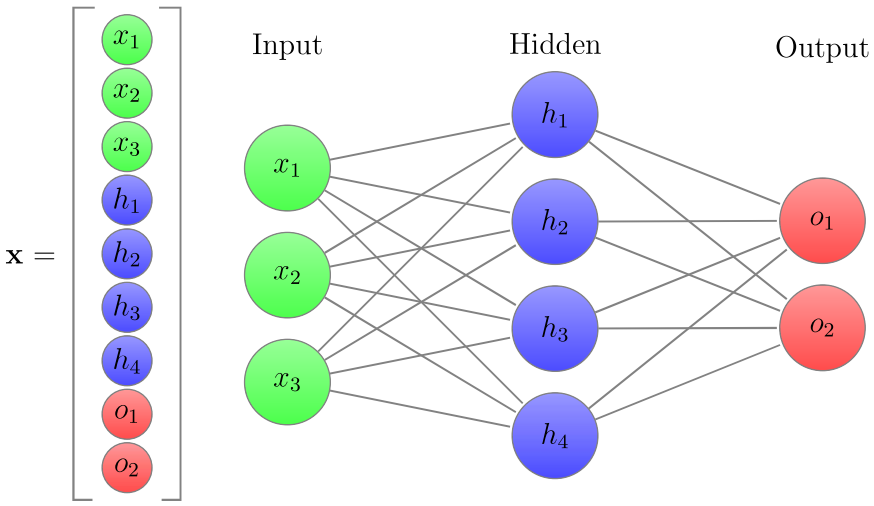


(a) (b)

Figure 2.5: (a) Comparison between the learning curves of a Recurrent Neural Network trained with both the symmetric version of Equilibrium Propagation (in blue) and with the asymmetric counterpart (in orange). (b) From the left column: ten corrupted patterns with probability $p = 0.1$, reconstructed pattern with Hebbian, Storkey, Symmetric and Asymmetric Equilibrium Propagation learning rules. The last column shows the target patterns.

2.3.2 Classification

The classification of the MNIST data-set [57] is a omnipresent standard for assesing the correct functioning of a machine learning classifier. This dataset contains 60000 images of the ten classes of handwritten digits with size 28×28. The objective is to classify each image with the correct number class. For reducing the computational burden and train the network within a reasonable time using the whole dataset, a nearest-neighbor interpolation was performed to obtain 14×14



(a)

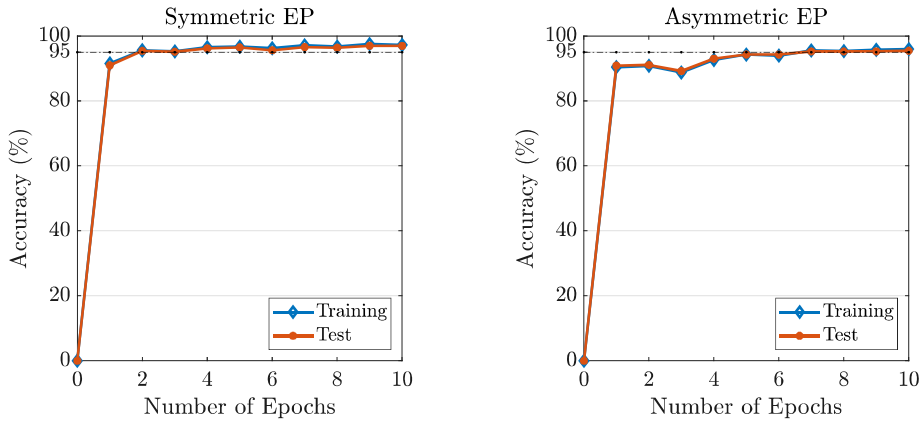
	x_1	x_2	x_3	h_1	h_2	h_3	h_4	o_1	o_2
x_1	–	–	–	W_{14}	W_{15}	W_{16}	W_{17}	–	–
x_2	–	–	–	W_{24}	W_{25}	W_{26}	W_{27}	–	–
x_3	–	–	–	W_{34}	W_{35}	W_{36}	W_{37}	–	–
h_1	W_{41}	W_{42}	W_{43}	–	–	–	–	W_{48}	W_{49}
h_2	W_{51}	W_{52}	W_{53}	–	–	–	–	W_{58}	W_{59}
h_3	W_{61}	W_{62}	W_{63}	–	–	–	–	W_{68}	W_{69}
h_4	W_{71}	W_{72}	W_{73}	–	–	–	–	W_{78}	W_{79}
o_1	–	–	–	W_{84}	W_{85}	W_{86}	W_{87}	–	–
o_2	–	–	–	W_{94}	W_{95}	W_{96}	W_{97}	–	–

(b)

Figure 2.6: (a) Illustration of recurrent neural network with 9 nodes used to solve a classification task. (b) Weights matrix $\mathbf{W}$ representing edge's influence between neurons. This choice of matrix establishes a hierarchy between neurons: input (green), hidden (blue) and output (red) nodes.

equivalent images. These images are then successively reshaped into column vector of size 196×1.

Many different possible architectures can be used to solve this image classification task, but all of them must be characterized by input, hidden and output layers as shown in Fig. 2.6(a) with green, blue and red nodes, respectively.



(a) (b)

Figure 2.7: (a) Accuracy of the network trained with Symmetric Equilibrium Propagation on the MNIST dataset. (b) Accuracy of the network trained with Asymmetric Equilibrium Propagation on the MNIST dataset.

As proposed by the authors in [12], the network considered is a three layers network with $n = n_i + n_h + n_o$ neurons, in which the $n_i = 196$ input neurons are fully connected to $n_h = 128$ hidden neurons, in turn fully connected to the $n_o = 10$ output neurons as exemplified by the 9×9 weight matrix in Fig. 2.6(b). No connections between the neurons in the same layer are considered. The activation function used was the sigmoid and the hyperparameters $\eta = 0.01$ and $\beta = 0.2$ were optimized with a grid search approach. The network parameters were sampled using a random uniform initialization and the model was trained on the whole dataset. Images are introduced to the network by clamping the input nodes of the system and the external input vector $\mathbf{I}$ is used as a bias term. In 10 epochs, the model reaches an accuracy of 96.4%. In Figs. 2.7(a) and (b) are reported the accuracy of the network trained with Symmetric and Asymmetric Equilibrium Propagation over the different epochs. Even though the authors in [12, 20] achieved a better accuracy with an analogous network, it is important to highlight that the proposed system has the following differences with respect to the original model:

1. The dynamical system defined in Eq. (1.40) is different from the original Energy-based model. This choice was dictated by the need of a Hopfield-like neural network that could be easily implemented by an electric circuit [54];

2. The asymmetric version of Equilibrium Propagation is slightly different from the original formulation in [20];

3. The network is trained on resized 14×14 images instead of the original 28×28 images to accelerate the calculations;

4. A single learning parameter η is used for both layers;
5. A smaller amount of epochs is used to train the network;
6. The system dynamics is simulated using the built-in `MatLAB` routine `ode45` instead of the Runge-Kutta numerical integration.

Table 2.1 shows the hyperparameters chosen to solve the MNIST classification using both the hyperbolic tangent and sigmoid functions. In both simulations, the network was trained on the entire dataset, (i.e. $N_{train} = 60000$ and $N_{test} = 10000$). For a fair comparison, the architecture used had 196 input nodes, 128 hidden nodes and 10 output nodes. The training process was performed for 10 epochs, with mini-batches of 20 data as suggested in [12]. As shown in the table, results do not show any remarkable difference on the choice of the activation function.

$g(\cdot)$	η	β	N_{train}	Mini-batch	# of Epochs	ACC_{train}	ACC_{test}
sigmoid	0.01	0.2	10000	20	10	97.5%	97.0%
tanh	0.005	0.1	10000	20	10	96.4%	96.1%

Table 2.1: Parameters and results of the optimized network with either sigmoid or hyperbolic tangent activation functions for the MNIST classification task.

2.4 Weakly-Coupled Oscillatory Neural Network

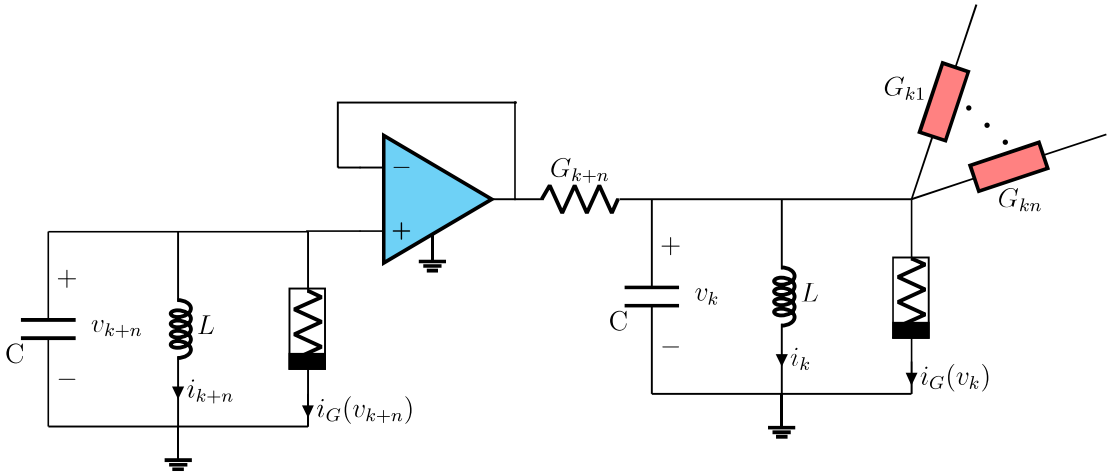
Consider a network of n weakly connected van der Pol oscillators whose states are individually controllable through the coupling with an additional driving oscillator unit as shown in Fig. 2.8(a). The following analysis may be generalized to other similar oscillators. Let us assume that each single nonlinear system admits an asymptotically stable, T periodic limit cycle. Let $\epsilon \ll 1$ be the interaction strength, by applying Kirchhoff's law to the network in Fig. 2.8(a), the following equation yields:

$$\begin{cases} C\frac{dv_k}{dt} = -i_k - i_G(v_k) - \epsilon f(v_1, \ldots, v_{2n}) \\ L\frac{di_k}{dt} = v_k \\ C\frac{dv_{k+n}}{dt} = -i_{k+N} - i_G(v_{k+n}) \\ L\frac{di_{k+n}}{dt} = v_{k+n} \end{cases} \tag{2.10}$$

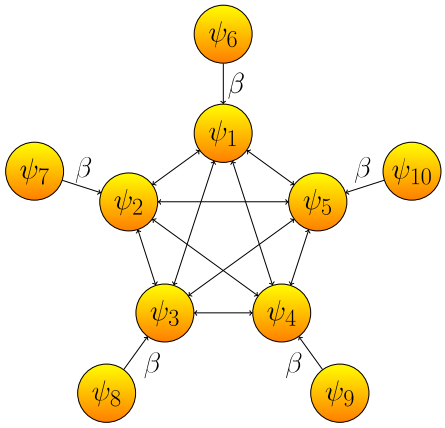
where

$$f(v_1, \ldots, v_{2n}) = \sum_{j=1}^{n} G_{kj}(v_k - v_j) + G_{k+n}(v_k - v_{k+n})$$

is the coupling function describing the interactions. By assuming $i_G(v_k) = -g_a v_k + g_b v_k^3$ and introducing the adimensional time $\tau = \frac{t}{LG}$, and state variables $x_k = \frac{v_k}{v_0}$,



(a)



(b)

	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5
ψ_1	–	W_{12}	W_{13}	W_{14}	W_{15}
ψ_2	W_{12}	–	W_{23}	W_{24}	W_{25}
ψ_3	W_{13}	W_{23}	–	W_{34}	W_{35}
ψ_4	W_{14}	W_{24}	W_{34}	–	W_{45}
ψ_5	W_{15}	W_{25}	W_{35}	W_{45}	–

(c)

Figure 2.8: (a) Master-slave configuration of the k-th van der Pol oscillator with the corresponding $(k+n)$-th driving unit. The connection between the two oscillators is unidirectional. (b) Illustration of the Kuramoto model with 10 nodes in a master-slave configuration. Masters are connected to slaves with strength β. (c) The symmetric weights matrix $\mathbf{W}$ representing edge's influence between the nodes.

$y_k = \frac{i_k}{Gv_0}$, where G is a generic reference conductance and v_0 is a reference voltage,

the state equations can be rewritten in the adimensional form

$$\begin{cases} \frac{dx_k}{d\tau} = -\alpha y_k + \delta x_k - \gamma x_k^3 - \epsilon \tilde{f}(x_1, \ldots, x_{2n}) \\ \frac{dy_k}{d\tau} = x_k \\ \frac{dx_{k+n}}{d\tau} = -\alpha y_{k+n} + \delta x_{k+n} - \gamma x_{k+n}^3 \\ \frac{dy_{k+n}}{d\tau} = x_{k+n} \end{cases} \tag{2.11}$$

where

$$\tilde{f}(x_1, \ldots, x_{2n}) = \sum_{j=1}^{N} \Gamma_{kj}(x_k - x_j) + \Gamma_k(x_k - x_{k+n})$$

is the normalized coupling function and $\alpha = \frac{LG^2}{C}$, $\delta = g_a \frac{LG}{C}$, $\gamma = g_b v_0^2 \frac{LG}{C}$, $\Gamma_{kj} = G_{kj} \frac{LG}{C}$ and $\Gamma_k = G_{k+N} \frac{LG}{C}$.

As a result of a recently developed description of weakly-coupled oscillatory neural networks dynamics in terms of amplitude and phase variables [60, 61, 62], it is possible to show that the phase dynamics of the system defined in Eq. (2.11) coincides locally in the neighborhood of the limit cycle with the asymptotic phase dynamics defined by Kuramoto model described in Section 1.5.

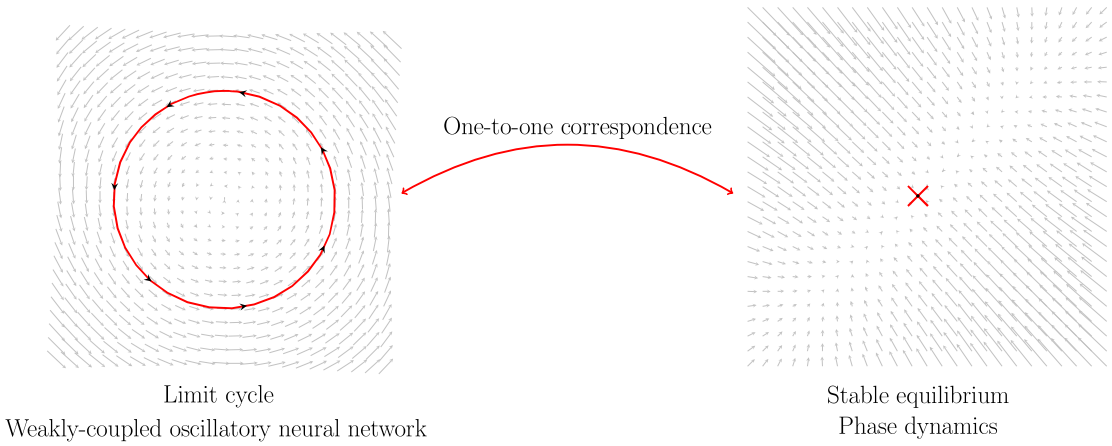


Figure 2.9: The one-to-one correspondence between the oscillatory sinusoidal behaviors of a weakly-coupled oscillatory neural network with the stable equilibria of the phase dynamics described the Kuramoto model.

The phase deviation equation for the k-th state:

$$\begin{cases} \frac{d\psi_k}{dt} = \tilde{\epsilon} \left[\sum_{j=1}^{n} \Gamma_{kj} \sin(\psi_j - \psi_k) + \Gamma_k \sin(\psi_{k+N} - \psi_k) \right] \\ \frac{d\psi_{k+n}}{dt} = 0 \Rightarrow \psi_{k+n}(t) = \psi_{k+n}(0) \end{cases} \tag{2.12}$$

where $\tilde{\epsilon} = \epsilon\frac{\sqrt{\alpha}}{2}$.
Assuming $W_{kj} = \tilde{\epsilon}\Gamma_{kj}$, $\beta = \tilde{\epsilon}\Gamma_k$ and $\psi_{k+n}(0) = T_i$, Eq. (2.12) can be rewritten as:

$$\frac{d\psi_k}{dt} = \sum_{j=1}^{n} W_{kj} \sin(\psi_j - \psi_k) + \beta \sin(T_k - \psi_k) \tag{2.13}$$

which is equivalent to the Kuramoto model defined in Eq. (1.61) with $\psi_k = x_k$ $\forall k = 1, \ldots, n$. This important results allows to create a one-to-one correspondence between the oscillatory sinusoidal behaviors of the system defined in Eq. (2.11) with the stable equilibria of the phase dynamics described by system (2.13) as depicted in Fig. 2.9. Therefore, the parameters of the oscillatory model can be update using the equilibrium point learning algorithms defined in Section 1.5.

2.4.1 Associative Memory

The network architecture consists of a fully connected WCON with $n = 64$ oscillators and symmetric weights as described in Fig. 2.8(b). Weights are randomly initialized by sampling from a uniform distribution and are mapped to desired memductance values of the memristive interconnections. The update of the weights is performed by averaging the back propagated errors over the total number of training images shown in Fig. 2.3. This approach allows us to lower the amount of total updates of the weight matrix. The learning rate is $\eta = 0.0001$ and it decreases during the iterations using a step decay schedule. The forcing parameter is set to $\beta = 0.1$ and the training process ends whenever a prefixed accuracy is reached. Time spans for the simulation of the dynamic system are chosen in order to guarantee the convergence of the state variables.

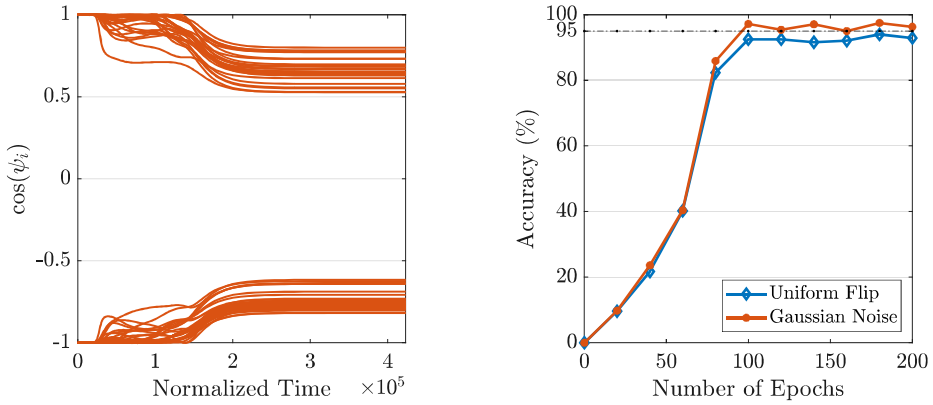
In the simulations, the driving unit oscillators have two different important roles:

- to set the phase of each oscillator as equal to the perturbed/target pattern ($\beta \gg 0$);
- to model the teaching signal of the second phase of Equilibrium Propagation as defined in Eq. (2.13).

Since all target patterns are composed by 0 and/or π, they represent equilibria of the free dynamical system defined in Eq. (2.8) with $\beta = 0$. To let the system be able to escape from a constant evolution of the phase dynamics, a small perturbation is added to the phase initial conditions of the oscillators.

In order to evaluate the effectiveness of Symmetric Equilibrium Propagation for training oscillatory networks in associative memory's tasks, the novel algorithm is compared with the unsupervised Hebbian learning rule:

$$\mathbf{W} = \frac{1}{m}\sum_{i=1}^{m}[\cos(\mathbf{T}_i)\cos(\mathbf{T}_i)^T - \mathbb{I}_{n\times n}]$$



(a) (b)

Figure 2.10: (a) Phase dynamics trajectories. (b) Accuracy of the network trained with Equilibrium Propagation to reconstruct perturbed patterns taken from the MNIST dataset. In blue, pixels are perturbed by flipping the pixels from white to black and vice versa with probability $p = 0.1$. In orange, the same patterns are corrupted with a Gaussian Noise.

where $\cos(\mathbf{T}_i) = [\cos(T_i^1), \ldots, \cos(T_i^n)]^T$. Patterns are corrupted using either a uniform flipping of the pixels with probability $p = 0.1$ or an additive Gaussian noise with standard deviation $\sigma = 0.5$. As can be observed in Fig. 2.10(a) phase trajectories do not converge to multiple values of 0 and π. This is probably due to the cosinusoidal combination in the potential function that allows the system to have many possible equilibria. However, as expected, simulations show that the longer is the training, the smaller is the difference with the target patterns even though the system is really sensible to perturbations. After convergence, the cosine of the output phase differences is computed and results are saturated to the closest values -1 or 1. A pattern is recognized as correctly reconstructed if the Hamming difference between the reconstruction and the target image is zero. As shown in Fig. 2.10(b), results provide evidence that the WCON trained with EP is perfectly able to reconstruct the corrupted patterns. However, the network needs a larger amount of epochs to be able to reach comparable results with the continuous-time recurrent neural networks counterpart.

2.5 Conclusion

This chapter aimed at investigating dynamic neural networks in solving pattern recognition tasks and validate the theoretical results found in Chapter 1. The use of ideal resistive switching components to model synapses is extremely beneficial

to be able to sequentially update and adjust the synaptic weights. Simulations showed convincing results that the two learning rules have significant capabilities in solving image reconstruction and classification tasks motivating further research in this direction. The symmetric version of Equilibrium Propagation is also used to train the phase dynamics of weakly-coupled oscillatory neural networks to perform as associative memory model. Even though the training phase requires more iterations compared to the conventional Hopfield neural networks, the system is able to remarkably improve the current retrieval performance known in the literature.

From an implementation perspective, the symmetrical version of Equilibrium Propagation could detrimentally limit the potential of the model in terms of analog implementations due to the intrinsic stochastic asymmetry that may arise in the programming phase. On the other hand, even though the asymmetric version is potentially more general, the convergence of the system is not always guaranteed and requires further assumptions. The next chapter aims to analyze systematic and random errors arising from the use of memristive crossbars that can influence the previously introduced neural systems' performance.

Chapter 3

Memristive Crossbars: System Evaluation and Configurations

In-memory computing is a non-von-Neumann approach where certain operational tasks are performed directly at the memory level. Computing is carried out by exploiting the physical properties of the memory devices and their array-level organization. Besides reducing latency and energy cost associated with data migration, the massive parallelism obtained by dense arrays of nanoscale memory devices have dramatically improved the computational time complexity associated with certain tasks. However, despite their fascinating potential for neuromorphic applications, memristive devices reveal undesirable properties due to their inherently stochastic cycle-to-cycle and device-to-device variabilities. This limitation combined with the physical properties of the circuit components further introduce random fluctuations and systematic errors:

- the random noise limits the system precision,
- the systematic errors degrade the output accuracy.

In recent papers, different methods have been used to estimate the output precision and accuracy of the memristor based crossbars, such as bit-precision [50], relative error [63], and cosine similarity [64]. This chapter aims to provide a novel analysis of both random and systematic errors by presenting non-application specific mathematical tools to systematically evaluate precision and accuracy of memristive crossbars.

3.1 Active and Passive arrays

Sneak paths are undesired paths for current, which are parallel to the intended target. This phenomenon causes cross-talk interference between adjacent memory cells and result in distortions that influence the operation of memristor crossbars.

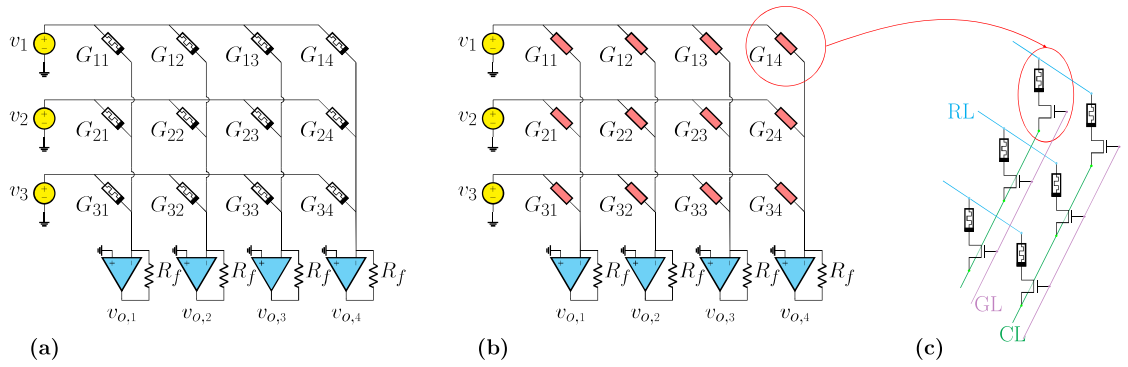


Figure 3.1: (a) Illustration of a 3×4 passive crossbar used to perform matrix-vector multiplication. (b) Illustration of a 3×4 active crossbar with 1T1R cells. (b) The one-transistor one-memristor (1T1R) cell at each junction in a crossbar. GL: Gate Line, RL: Row Line, CL: Column Line.

The conventional passive arrays have no transistors in the memristor crossbar as shown in Fig. 3.1(a). Even though this characteristic is really attractive from a circuit implementation perspective, sneak path current issues usually prohibit an appropriate programming and/or reading of memristors' conductances. Common solutions to mitigate this problem is to engineer the memristor I–V non linearity itself or to connect in series a two-terminal selector device in each cell (1S1R). In the latter case, memristor and selector can be stacked on top of each other reducing the cell footprint of 1T1R schemes [65].

Nowadays, the most practical approach to realize large memristor crossbars is to integrate memristors with MOS transistors in active arrays as shown in Fig. 3.1(b). By connecting a transistor to a memristor in series (1T1R cell) as depicted in Fig. 3.1(c), the current flowing through unselected cells can be effectively shrunk to enable accurate memristors' reading and programming. In addition, the third terminal in the transistors offers the possibility to control the conductance's update in a linear and symmetric manner. During the inference process, all transistors in the 1T1R array are in the ON state whereas during memristor conductance programming, transistors are partially ON to allow for precise weight updating. Even though this solution seems appealing, it increases both the power consumption and the packing density.

3.2 Random Errors

Analog computation suffers from reduced robustness to noise as compared with digital computation. This section focuses on the modeling of memristive device stochastic noise including programming noise, thermal noise, and shot noise. These

results enable computation of the system SNR and output bit precision providing a comparison between the analog systems and their digital computing counterparts [66]. The overall SNR for the $k-$th output is:

$$SNR_k = \frac{k\text{-}th\ output\ signal\ power}{\sigma^2_{TOT_k}}, \tag{3.1}$$

where $\sigma^2_{TOT_k}$ is the $k-$th output noise power and the output bit precision can be calculated from:

$$k-\text{th Output Bit Precision} = \frac{10\log(SNR_k) - 1.76}{6.02}. \tag{3.2}$$

For evaluating the total output precision, we define a vector with all total output noise standard deviations $\sigma_{TOT_i}\ \forall i = 1, \ldots, m$ and evaluate the relative output dispersion from the ideal value $\mathbf{x}$ by means of the coefficient $\frac{\|\boldsymbol{\sigma}_{TOT}\|}{\|\mathbf{x}\|}$ where $\|\cdot\|$ is a vector norm. This work assumes $\sigma^2_{TOT_k}$ as the sum of the output voltage programming $\overline{v^2_{P,k}}$, thermal $\overline{v^2_{T,k}}$ and shot $\overline{v^2_{S,k}}$ noises. The next sections evaluate the magnitude of these values. Other noise contributions can be also considered.

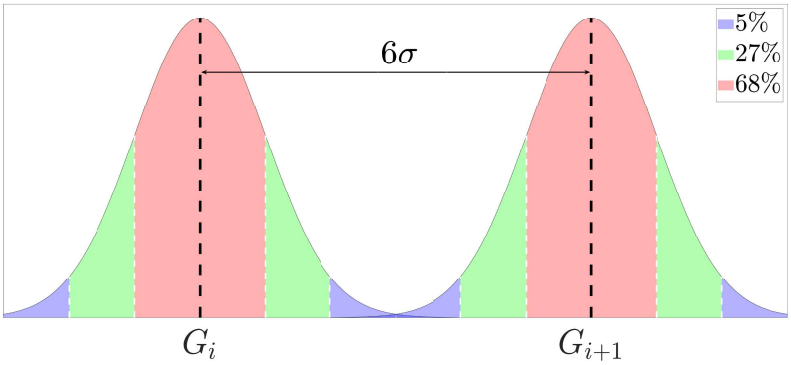


Figure 3.2: Normally distributed programming noise model that illustrates the intended write distribution between two conductance levels separated by the pre-defined bit precision. Taken from [67] ©2021 IEEE.

3.2.1 Programming Noise

Memristors are programmable to any conductance in the range G_{range}, within a pre-defined tolerance [55, 68] determined by an iterative read-write process, with improved precision coming at the expense of additional steps and energy expenditure. In order to compare the analog precision to digital implementations, G_{range} is divided into 2^{N_b} intervals (levels) to compute the equivalent bit precision of the

programmable levels, where N_b is the number of desired bits. At the time of writing, the best experimental results available in the literature reported up to 256 (8 bits) reliably programmable conductance levels [69].

The cycle-to-cycle programming variations of the memristor conductance yield an essentially Gaussian distribution, and thus introduces a stochastic fluctuation that further affects the system precision and will be referred to as programming noise. Observe that whenever the devices are programmed, the error becomes constant until the next programming stage. If the same conductance matrix is repeatedly used for an iterative calculation, the error can be considered as a systematic level shift in the mean conductance affecting the output accuracy.

In most experimental reports, the programming noise is modeled by a normal distribution with a standard deviation σ_P [48, 49, 50, 64]. To study the effects of the programming noise on output precision, a normal distribution $N(0, \sigma_P^2)$ was used to perturb each ideal memristor conductance. The programmed levels of the memristor should be distinct from each other with high confidence as in Fig. 3.2, in other words $\pm 3\sigma_P$ should be found in each interval of width $\Delta L = \frac{G_{range}}{(2^{N_b}-1)}$. For this reason, $\sigma_P = \Delta L/6$.

Let us assume a random perturbation of the matrix $\mathbf{G}$ by sampling from a Gaussian distribution with zero mean and variance σ_P^2:

$$\tilde{\mathbf{G}} = \mathbf{G} + \mathbf{Z}_P \tag{3.3}$$

where $\mathbf{Z}_P = (Z_{ij}) \sim N(0, \sigma_P^2)$. Thus the ideal computation $\mathbf{i} = \mathbf{G}^T\mathbf{v}$ transforms into the perturbed system

$$\tilde{\mathbf{i}} = \tilde{\mathbf{G}}^T\mathbf{v} = (\mathbf{G} + \mathbf{Z}_P)^T\mathbf{v}. \tag{3.4}$$

where $\tilde{\mathbf{i}}$ is the perturbed output. The k-th component of Eq. (3.4) is a linear combination of independent Gaussian random variables and therefore it holds [70]:

$$\begin{aligned} \tilde{i}_k - i_k &= Z_{1k}v_1 + \ldots + Z_{nk}v_n = \\ &\sim v_1 N(0, \sigma_P^2) + \ldots + v_n N(0, \sigma_P^2) = \\ &\sim N(0, v_1^2\sigma_P^2) + \ldots + N(0, v_n^2\sigma_P^2) = \\ &\sim N(0, (v_1^2 + \ldots + v_n^2)\sigma_P^2) = \\ &\sim N(0, \|\mathbf{v}\|_2^2\sigma_P^2) \quad \forall k = 1, \ldots m \end{aligned} \tag{3.5}$$

where $\|\cdot\|_2$ is the euclidean norm. This means that if one randomly perturbs all the entries of a matrix by sampling from a Gaussian distribution $N(0, \sigma_P^2)$, then the k-th output current noise will be normally distributed. Since the output variance is given in terms of current noise, it can be then converted into output voltage noise by multiplying it by the square of the feedback resistance: $\overline{v_{P,k}^2} = \|\mathbf{v}\|_2^2\sigma_P^2 R_f^2$ $\forall k = 1, \ldots, m$.

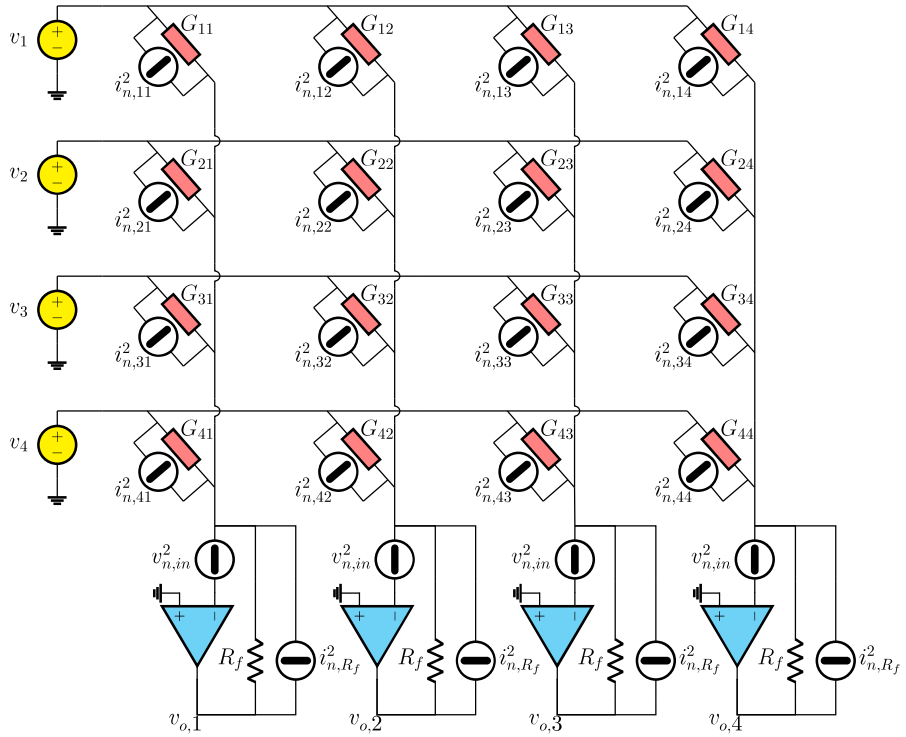


Figure 3.3: A schematic illustration of a 4×4 crossbar with the associated noise sources.

3.2.2 Thermal Noise

Thermal agitation of charged particles in a resistive device gives rise to random fluctuations in the voltage across its terminals, known as thermal noise. As shown in Fig. 3.3, the root mean square current noise $i_{n,jk}$ generated in a single memristor with conductance G_{jk} is given as follows [71]:

$$i_{n,jk} = \sqrt{4K_B T f G_{jk}}, \tag{3.6}$$

where f is the operating frequency of the crossbar, K_B is the Boltzmann constant, T is the temperature in Kelvin. The thermal noise is modeled as a Gaussian distribution with zero mean and standard deviation $i_{n,jk}$. Then, the equivalent standard deviation of the $(jk)-$th memconductance can be described as:

$$\sigma_T = \frac{i_{n,jk}}{|v_j|} = \sqrt{\frac{4K_B T f G_{jk}}{v_j^2}} \tag{3.7}$$

where v_j is the voltage drop across the two terminal. In the case of active crossbars, transistor thermal noise in the 1T1R structure can be neglected with proper sizing [72].

Let us assume a random perturbation of the matrix $\mathbf{G}$ by sampling from a Gaussian distribution with zero mean and variance σ_T^2:

$$\tilde{\mathbf{G}} = \mathbf{G} + \mathbf{Z}_T \tag{3.8}$$

where $\mathbf{Z}_T = (Z_{jk}) \sim N(0, \sigma_T^2)$. Thus the ideal computation $\mathbf{i} = \mathbf{G}^T\mathbf{v}$ transforms into the perturbed system

$$\tilde{\mathbf{i}} = \tilde{\mathbf{G}}^T\mathbf{v} = (\mathbf{G} + \mathbf{Z}_T)^T\mathbf{v}. \tag{3.9}$$

where $\tilde{\mathbf{i}}$ is the perturbed output. The k-th component of Eq. (3.9) is a linear combination of independent Gaussian random variables and therefore it holds [70]:

$$\begin{aligned}
\tilde{i}_k - i_k &= Z_{1k}v_1 + \ldots + Z_{nk}v_n = \\
&\sim v_1 N(0, \sigma_T^2) + \ldots + v_n N(0, \sigma_T^2) = \\
&\sim N\left(0, v_1^2 \frac{4K_BTfG_{1k}}{v_1^2}) + \ldots + N(0, v_n^2 \frac{4K_BTfG_{nk}}{v_n^2}\right) = \\
&\sim N\left(0, 4K_BTf\left(\sum_{j=1}^{n} G_{jk}\right)\right) \\
&\sim N\left(0, 4K_BTf\|\mathbf{G}_k\|_1\right) \quad \forall k = 1, \ldots m
\end{aligned} \tag{3.10}$$

where $\|\mathbf{G}_k\|_1 = \sum_{j=1}^{n} |G_{jk}| = \sum_{j=1}^{n} G_{jk}$ is the 1-norm of the $k-th$ column of the conductance matrix $\mathbf{G}$. This means that if one randomly perturbs all the entries of a matrix by sampling from a Gaussian distribution $N(0, \sigma_T^2)$, then the k-th output current noise will be normally distributed. Since the output variance is given in terms of current noise, it can be then converted into output voltage noise by multiplying it by the square of the feedback resistance: $\overline{v_{T,k}^2} = 4K_BTf\|\mathbf{G}_k\|_1 R_f^2 + 4K_BTfR_f^2 \; \forall k = 1, \ldots, m$ where the last term is the voltage noise variance of the feedback resistor.

3.2.3 Shot Noise

Shot noise is an electronic noise that originates from the discrete independent charged particles in the current flow. Roughly speaking, electrical current does not behave like water: uniform flow and smooth variation in time. Since the motion of individual electrons is unpredictable, it is commonly computed the average number of electrons drifting past a particular section per time interval. The variation about the mean value of this quantity is defined as shot noise. For large numbers, the Poisson distribution approaches a normal distribution about its mean as a result of the central limit theorem and the equivalent standard deviation of the $(jk)-$th memconductance can be described as:

$$\sigma_S = \sqrt{\frac{2qfG_{jk}|v_j|}{v_j^2}} = \sqrt{\frac{2qfG_{jk}}{|v_j|}} \tag{3.11}$$

where q is electron charge, and v_j is the voltage drop across the generic memristor with conductance G_{jk}.

Let us assume a random perturbation of the matrix $\mathbf{G}$ by sampling from a Gaussian distribution with zero mean and variance σ_S^2:

$$\tilde{\mathbf{G}} = \mathbf{G} + \mathbf{Z}_S \tag{3.12}$$

where $\mathbf{Z}_S = (Z_{jk}) \sim N(0, \sigma_S^2)$. Thus the ideal computation $\mathbf{i} = \mathbf{G}^T\mathbf{v}$ transforms into the perturbed system

$$\tilde{\mathbf{i}} = \tilde{\mathbf{G}}^T\mathbf{v} = (\mathbf{G} + \mathbf{Z}_S)^T\mathbf{v}. \tag{3.13}$$

where $\tilde{\mathbf{i}}$ is the perturbed output. The k-th component of Eq. (3.13) is a linear combination of independent Gaussian random variables and therefore it holds [70]:

$$\begin{aligned}
\tilde{i}_k - i_k &= Z_{1k}v_1 + \ldots + Z_{nk}v_n = \\
&\sim v_1 N(0, \sigma_S^2) + \ldots + v_n N(0, \sigma_S^2) = \\
&\sim N\left(0, v_1^2 \frac{2qfG_{1k}}{|v_1|}) + \ldots + N(0, v_n^2 \frac{2qfG_{nk}}{|v_n|}\right) = \\
&\sim N\left(0, 2qf \sum_{j=1}^{n} |i_{jk}|\right) \quad \forall k = 1, \ldots m
\end{aligned} \tag{3.14}$$

where i_{jk} is the current flowing out from the $(jk)-th$ memristor. This means that if one randomly perturbs all the entries of a matrix by sampling from a Gaussian distribution $N(0, \sigma_T^2)$, then the k-th output current noise will be normally distributed. Since the output variance is given in terms of current noise, it can be then converted into output voltage noise by multiplying it by the square of the feedback resistance: $\overline{v_{S,k}^2} = 2qf \sum_{j=1}^{n} |i_{kj}| + 2qfR_f^2 \ \forall k = 1, \ldots, m$ where the last term is the voltage noise variance of the feedback resistor.

3.3 Systematic errors

Precision is insufficient to describe the system performance. To measure the discrepancy between the ideal and the corrupted results, the overall output accuracy can be computed in terms of the normwise relative error. Let $\|\cdot\|$ be a vector norm, the normwise relative error E_r between the measured output vector $\tilde{\mathbf{x}}$ and the output provided by a digital software $\mathbf{x}$ can be defined as:

$$E_r = \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \tag{3.15}$$

Common norms used are the euclidean norm, $1-$norm and $\infty-$norm.

Recently, [49] proposed to reduce wire resistances effect by estimating linear scaling factors for each column output. This method may be used to compensate further systematic contributions. In general, other systematic errors such as finite gain, input and output resistance and capacitance of the op-amp, voltage offsets and op-amp non-linearity can affect system accuracy. The next sections focus on the analytical computations of the wire resistance and op-amp's finite gain systematic contributions.

3.3.1 Wire Resistance

Let us consider the MVM operation using a memristive crossbar with conductance matrix $\mathbf{G} \in \mathbb{R}^{n\times m}$, and introduce the following notation for each column $k = 1, \ldots, m$:

- $\mathbf{i}_k = (i_{1k}, \ldots, i_{nk})^T$ the vector of n currents through the n memristors in the $k-$th column with the presence of wire resistances;
- $\mathbf{e}_k = (e_{1k}, \ldots, e_{nk})^T$ the vector including the potentials of nodes corresponding to the upper terminals of memristors in the $k-$th column;
- $\hat{\mathbf{e}}_k = (\hat{e}_{1k}, \ldots, \hat{e}_{nk})^T$ the vector including the potentials of nodes corresponding to the lower terminals of memristors in the $k-$th column;
- R is the wire resistance;
- $\mathbf{\Gamma}_k = R\mathbf{G}_k = \text{diag}(G_{1k}, \ldots, G_{nk})$ a diagonal matrix where each entry is the product of the wire resistance and the memductance G_{jk} of the $k-$th column's memristors.

A circuital analysis of the $k-$th column depicted in Fig. 3.4 gives

$$\begin{cases} \mathbf{V}\hat{\mathbf{e}}_k = \mathbf{L}\mathbf{\Gamma}_k\hat{\mathbf{e}}_k - \mathbf{L}\mathbf{\Gamma}_k\mathbf{e}_k \\ \mathbf{i}_k = \mathbf{G}_k(\mathbf{e}_k - \hat{\mathbf{e}}_k) \end{cases} \tag{3.16}$$

where

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 1 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & 1 & \ldots & 1 \end{pmatrix} \quad \mathbf{V} = \begin{pmatrix} -1 & 1 & \ldots & 0 \\ 0 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 0 & \ldots & 0 & -1 \end{pmatrix} \tag{3.17}$$

Solving the first equation of the system (3.16) in terms of $\hat{\mathbf{e}}$, it yields:

$$\begin{aligned} \hat{\mathbf{e}}_k &= (\mathbf{L}\mathbf{\Gamma}_k - \mathbf{V})^{-1}\mathbf{L}\mathbf{\Gamma}_k\mathbf{e}_k \\ \Rightarrow \hat{\mathbf{e}}_k &= (\mathbb{I}_{n\times n} - \mathbf{\Gamma}_k^{-1}\mathbf{L}^{-1}\mathbf{V})^{-1}\mathbf{e}_k \\ \Rightarrow \hat{\mathbf{e}}_k &= (\mathbb{I}_{n\times n} + \mathbf{\Gamma}_k^{-1}\mathbf{V}^T\mathbf{V})^{-1}\mathbf{e}_k \end{aligned} \tag{3.18}$$

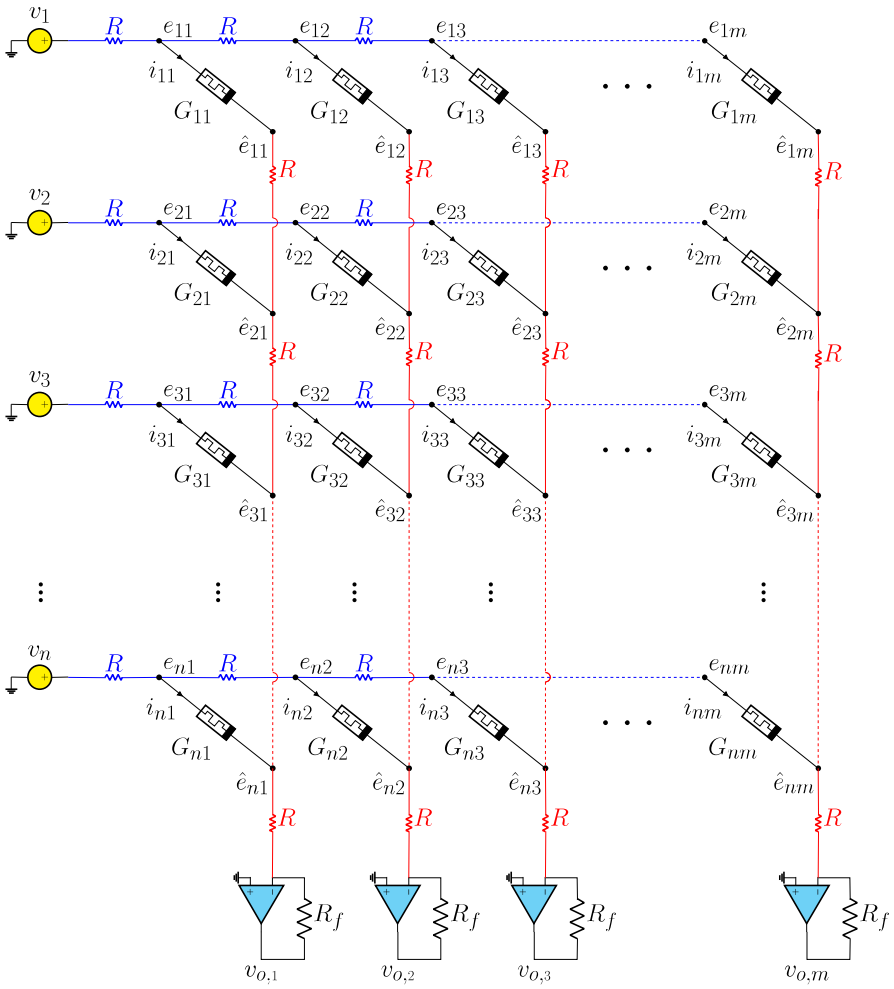


Figure 3.4: The schematic of a memristive crossbar with the presence of wire resistance R. Wire resistance is modeled by small-value resistors on vertical lines and horizontal lines.

where $\mathbf{L}^{-1} = -\mathbf{V}^T$. The Woodbury formula [73] and Eq. (3.18) allow us to calculate the potential distribution over the vertical wire connecting the $k-$th column's memristors to the operational amplifier:

$$\begin{aligned} \hat{\mathbf{e}}_k &= [\mathbb{I}_{n\times n} - (\mathbb{I}_{n\times n} + \mathbf{L}^T\mathbf{L}\mathbf{\Gamma}_k)^{-1}]\mathbf{e}_k \\ \Rightarrow \hat{\mathbf{e}}_k &= \mathbf{e}_k - (\mathbb{I}_{n\times n} + \mathbf{L}^T\mathbf{L}\mathbf{\Gamma}_k)^{-1}\mathbf{e}_k \\ \Rightarrow \mathbf{e}_k - \hat{\mathbf{e}}_k &= (\mathbb{I}_{n\times n} + \mathbf{L}^T\mathbf{L}\mathbf{\Gamma}_k)^{-1}\mathbf{e}_k \end{aligned} \tag{3.19}$$

By substituting Eq. (3.19) in the second equation of the system (3.16), it follows

that the vector of memristor currents is

$$\begin{aligned}\mathbf{i}_k &= \mathbf{G}_k(\mathbb{I}_{n\times n} + \mathbf{L}^T\mathbf{L}\mathbf{\Gamma}_k)^{-1}\mathbf{e}_k \\ \mathbf{i}_k &= [(\mathbb{I}_{n\times n} + \mathbf{L}^T\mathbf{L}\mathbf{\Gamma}_k)\mathbf{G}_k^{-1}]^{-1}\mathbf{e}_k \\ \mathbf{i}_k &= (\mathbf{G}_k^{-1} + R\mathbf{L}^T\mathbf{L})^{-1}\mathbf{e}_k\end{aligned} \tag{3.20}$$

This analysis yields for each column of a two dimensional crossbar $\mathbf{G} \in \mathbb{R}^{n\times m}$. Let us now analyse the horizontal wire effect by introducing the following notation:

- $\mathbf{G}_{diag} = \text{diag}(\mathbf{G}_1, \ldots, \mathbf{G}_m) \in \mathbb{R}^{nm\times nm}$;
- $\underline{\mathbf{e}} = (\mathbf{e}_1, \ldots, \mathbf{e}_m)^T \in \mathbb{R}^{nm}$;
- $\underline{\mathbf{u}} = (\mathbf{v}, \mathbf{0}, \ldots, \mathbf{0})^T \in \mathbb{R}^{nm}$;
- $\underline{\mathbf{i}} = (\mathbf{i}_1, \ldots, \mathbf{i}_m)^T \in \mathbb{R}^{nm}$.

The circuital analysis of the circuit shown in Fig. 3.4 gives:

$$\begin{cases}\mathbf{e}_{m-1} = R\mathbf{i}_m + \mathbf{e}_m \\ \mathbf{e}_{m-2} = R(\mathbf{i}_m + \mathbf{i}_{m-1}) + \mathbf{e}_{m-1} \\ \quad\vdots \\ \mathbf{e}_1 = R(\mathbf{i}_m + \ldots + \mathbf{i}_2) + \mathbf{e}_2 \\ \mathbf{e}_0 = R(\mathbf{i}_m + \ldots + \mathbf{i}_1) + \mathbf{e}_1\end{cases} \tag{3.21}$$

where $\mathbf{e}_0 = \mathbf{v}$. This set of m vector equations can be written in matrix form by introducing the following matrices:

$$(\mathbf{L} \otimes \mathbb{I}_{n\times n}) = \begin{pmatrix} \mathbb{I}_{n\times n} & 0 & \ldots & 0 \\ \mathbb{I}_{n\times n} & \mathbb{I}_{n\times n} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ \mathbb{I}_{n\times n} & \mathbb{I}_{n\times n} & \ldots & \mathbb{I}_{n\times n} \end{pmatrix}$$

$$(\mathbf{V} \otimes \mathbb{I}_{n\times n}) = \begin{pmatrix} -\mathbb{I}_{n\times n} & \mathbb{I}_{n\times n} & \ldots & 0 \\ 0 & -\mathbb{I}_{n\times n} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbb{I}_{n\times n} \\ 0 & \ldots & 0 & -\mathbb{I}_{n\times n} \end{pmatrix}$$

$$(\mathbb{I}_{n\times n} \otimes \mathbf{A}) = \begin{pmatrix} \mathbf{A} & 0 & \cdots & 0 \\ 0 & \mathbf{A} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A} \end{pmatrix}, \forall \mathbf{A} \in \mathbb{R}^{n\times n}$$

where $\otimes$ is the kronecker product notation.

Remark 2. *The kronecker product has the following properties:*

$$\begin{aligned}(\mathbf{A}\otimes\mathbf{B})^T &= (\mathbf{A}^T\otimes\mathbf{B}^T)\\ (\mathbf{A}\otimes\mathbf{B})^{-1} &= (\mathbf{A}^{-1}\otimes\mathbf{B}^{-1})\\ (\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) &= (\mathbf{AC}\otimes\mathbf{BD})\end{aligned} \tag{3.22}$$

From Remark 2, it follows that (3.20) and (3.21) can be rewritten as:

$$\begin{cases}(\mathbf{V}^T\otimes\mathbb{I}_{n\times n})\underline{\mathbf{e}} = R(\mathbf{L}^T\otimes\mathbb{I}_{n\times n})\underline{\mathbf{i}} - \underline{\mathbf{u}}\\ \underline{\mathbf{i}} = \mathbf{H}^{-1}\underline{\mathbf{e}}\end{cases} \tag{3.23}$$

where $\mathbf{H} = \text{diag}(\mathbf{G}_1^{-1} + R\mathbf{L}^T\mathbf{L}, \dots, \mathbf{G}_m^{-1} + R\mathbf{L}^T\mathbf{L})$.
From the second equation of system (3.23) it yields that $\underline{\mathbf{e}} = \mathbf{H}\underline{\mathbf{i}}$ and by substituting this equality in the first equation one gets:

$$\begin{aligned}\mathbf{H}\underline{\mathbf{i}} &= R(\mathbf{V}^T\otimes\mathbb{I}_{n\times n})^{-1}(\mathbf{L}^T\otimes\mathbb{I}_{n\times n})\underline{\mathbf{i}} - (\mathbf{V}^T\otimes\mathbb{I}_{n\times n})^{-1}\underline{\mathbf{u}}\\ &= -R(\mathbf{L}\mathbf{L}^T\otimes\mathbb{I}_{n\times n})\underline{\mathbf{i}} + (\mathbf{L}\otimes\mathbb{I}_{n\times n})\underline{\mathbf{u}}\\ &\Rightarrow [\mathbf{H} + R(\mathbf{L}\mathbf{L}^T\otimes\mathbb{I}_{n\times n})]\underline{\mathbf{i}} = (\mathbf{L}\otimes\mathbb{I}_{n\times n})\underline{\mathbf{u}}\end{aligned} \tag{3.24}$$

Observing that

$$(\mathbf{L}\otimes\mathbb{I}_{n\times n})\underline{\mathbf{u}} = \begin{pmatrix}\mathbb{I}_{n\times n} & 0 & \dots & 0\\ \mathbb{I}_{n\times n} & \mathbb{I}_{n\times n} & \ddots & \vdots\\ \vdots & \vdots & \ddots & 0\\ \mathbb{I}_{n\times n} & \mathbb{I}_{n\times n} & \dots & \mathbb{I}_{n\times n}\end{pmatrix}\begin{pmatrix}\underline{\mathbf{v}}\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\end{pmatrix} = \begin{pmatrix}\underline{\mathbf{v}}\\ \underline{\mathbf{v}}\\ \vdots\\ \underline{\mathbf{v}}\end{pmatrix} = \underline{\underline{\mathbf{v}}} \tag{3.25}$$

we have that system (3.24) can be simplified in:

$$\underline{\mathbf{i}} = [\mathbf{G}_{diag}^{-1} + R(\mathbb{I}_{m\times m}\otimes\mathbf{L}^T\mathbf{L}) + R(\mathbf{L}\mathbf{L}^T\otimes\mathbb{I}_{n\times n})]^{-1}\underline{\underline{\mathbf{v}}} \tag{3.26}$$

or in the compact form:

$$\underline{\mathbf{i}} = [\mathbf{G}_{diag}^{-1} + \mathbf{R}_W]^{-1}\underline{\underline{\mathbf{v}}} \tag{3.27}$$

where $\mathbf{R}_W = R(\mathbb{I}_{m\times m}\otimes\mathbf{L}^T\mathbf{L}) + R(\mathbf{L}\mathbf{L}^T\otimes\mathbb{I}_{n\times n})$ represents the vertical and horizontal wire resistance contribution in a memristive crossbar. Figure 3.5 shows an example with $R = 1$. In conclusion, the $k-$th component of the current output vector $\mathbf{i}$ is

$$i_k = \underline{\mathbf{a}}_k^T\underline{\mathbf{i}} = \underline{\mathbf{a}}_k^T[\mathbf{G}_{diag}^{-1} + \mathbf{R}_W]^{-1}\underline{\underline{\mathbf{v}}} \tag{3.28}$$

where the column vector $\underline{\mathbf{a}}_k = [\mathbf{0}_n^T, \dots, \mathbf{0}_n^T, \underset{\substack{\downarrow\\ k^{th}}}{\mathbf{1}_n^T}, \mathbf{0}_n^T, \dots, \mathbf{0}_n^T]^T \in \mathbb{R}^{nm}$.

Remark 3. *If $R \to 0$ then $\boldsymbol{R}_W = \boldsymbol{0}$ and $\underline{\boldsymbol{i}} = \boldsymbol{G}_{diag}\underline{\boldsymbol{v}}$ or equivalently $\boldsymbol{i} = \boldsymbol{G}^T\boldsymbol{v}$.*

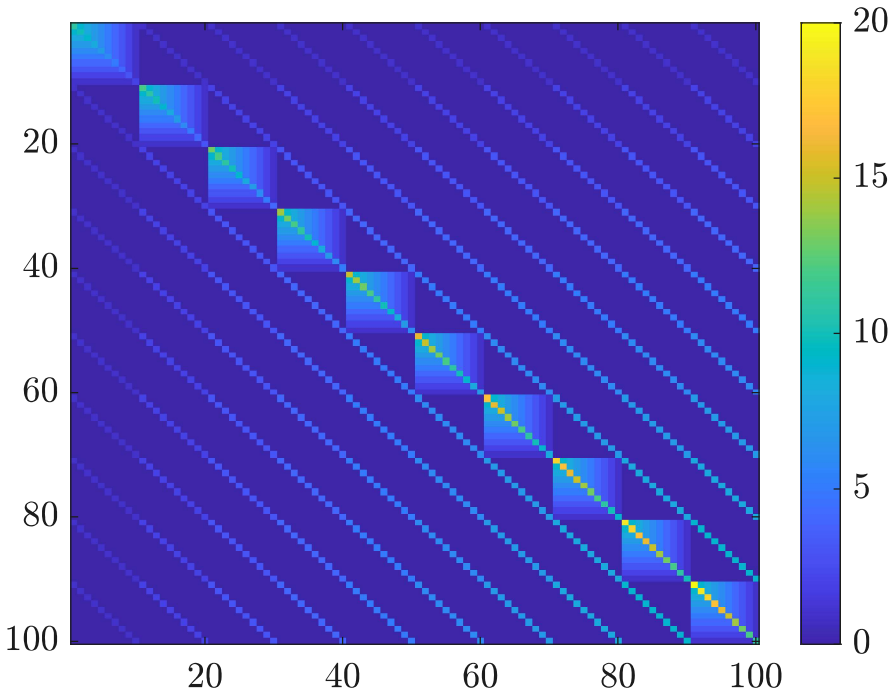


Figure 3.5: The 100×100 block matrix $\mathbf{R}_W$ representing the horizontal and vertical wire resistance contribution.

3.3.2 Finite gain of the Op-Amp

The current output can be converted into voltage by using transimpedance amplifiers (TIAs). A TIA is required instead of a resistor for current to voltage conversion in order to decrease the impedance seen from the crossbar with the following equation [74]:

$$R_i = \frac{R_f}{A_o + 1} \approx \frac{R_f}{A_o} \tag{3.29}$$

where A_o is the op-amp's gain and R_i is the input resistance of the TIA. In computation mode, the memristors should be operated in their linear range [49]. The voltage drop on the memristors, v_m, should also be kept well below the switching threshold voltage

$$v_m < v_j - v_{ref} \tag{3.30}$$

where v_j is the crossbar input voltage for the j-th row and v_{ref} is the TIA virtual reference voltage. Because of the op-amp's finite gain, a systematic error will be introduced on each output voltage. This is caused by the virtual reference node of each TIA not being equal to v_{ref}. A post-measurement error correction can be implemented to compensate for this systematic error. This is found from the

analysis of a non-ideal voltage adder. After measurement, the k-th output voltage can be corrected by the multiplicative correction factor

$$G_f + \frac{(\sum_{j=1}^{n} G_{jk} + G_{in} + G_f)(G_f + G_o)}{G_o A_o - G_f} \tag{3.31}$$

where $\sum_{j=1}^{n} G_{jk}$ is the sum of the memristor conductances connected to the k-th TIA, G_{in} and G_o are the op-amp's input and output conductances respectively, and G_f is the TIA feedback conductance.

3.4 Feedback Loops

Memristive crossbars enable fast MVMs calculation in one step using Ohm and Kirchhoff's laws. On the other hand, recent works have taken this a step further by demonstrating that linear algebraic systems can also be solved in a single time step using similar hardware with feedback [64, 75, 76]. Hereinafter, the crossbar depicted in Fig. 3.1 will be referred to as open-loop crossbar to discriminate it from crossbars that make use of feedback loops.

3.4.1 Linear Systems $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$

The feedback crossbar in Fig. 3.6 can be used to solve systems of linear equations such as

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.32}$$

in which $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$ are known but $\mathbf{x} \in \mathbb{R}^n$ is an unknown vector to be determined. In this structure, TIAs formed by an op-amp and crossbar resistances in feedback are used for two purposes. They create a virtual reference node that enables the distribution of input current $\mathbf{i}$ over the crossbar in the feedback path and perform current to voltage conversion. As a result of Kirchhoff's Current Law, the following relation will hold at the outputs of the TIAs':

$$\mathbf{G}^T\mathbf{v} = \mathbf{i} \tag{3.33}$$

where $\mathbf{G}$ is the memristor conductance matrix of the crossbar.

In Fig. 3.6, resistors R_c are used as voltage to current converters to provide the input vector $\mathbf{i}$. The output voltage $\mathbf{v}$ in (3.33) is the result of the multiplication of the inverse of the matrix $\mathbf{G}$ and the input current vector $\mathbf{i} = R_c\mathbf{v}_i$. The advantage of using a crossbar to solve a linear equation is the fact that the computation of $\mathbf{A}^{-1}$ is not required. This significantly reduces the computational complexity.

To avoid doubling the crossbar using differential pairs whenever matrices have both positive and negative values, it is possible to rely on the linear transformation defined in Eq. (2.3). Unfortunately, the simple closed form relation between the

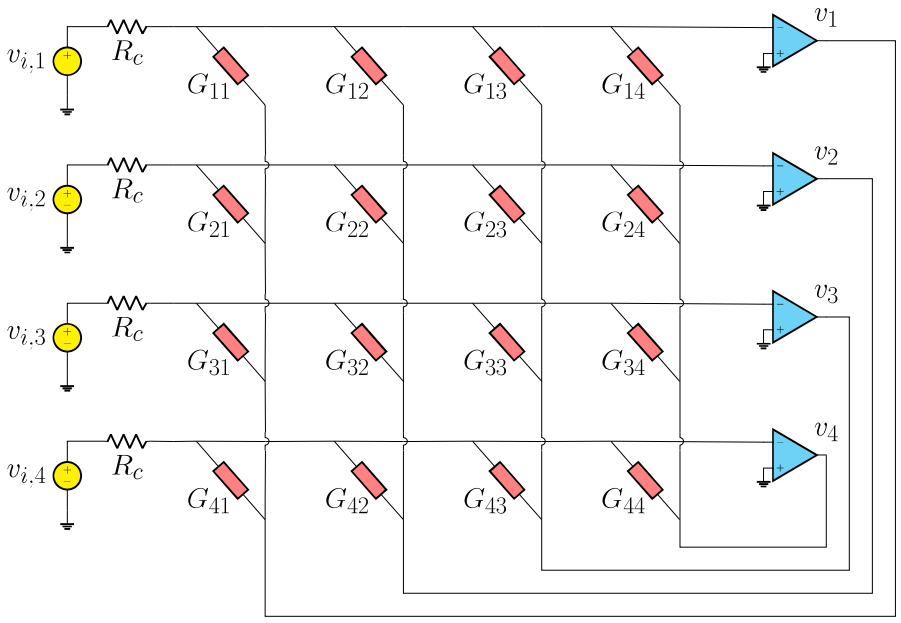


Figure 3.6: A schematic illustration of a 4×4 crossbar with feedback loops.

measured and the correct outputs defined in Eq. (2.5) can not be easily generalized to the case of linear systems. However, the following results show that the correct output can still be retrieved by simply adding a proper calibration step:

Proposition 3. *Let* $\boldsymbol{A} \in \mathbb{R}^{n\times n}, \gamma, \delta \in \mathbb{R}$. *Then:*

$$[\mathbf{G}^T]^{-1} = \frac{1}{\gamma}\mathbf{A}^{-1} - \frac{\delta}{\gamma}\left(\frac{\mathbf{A}^{-1}\mathbf{1}_n\mathbf{1}_n^T\mathbf{A}^{-1}}{\gamma + \delta\mathbf{1}_n^T\mathbf{A}^{-1}\mathbf{1}_n}\right) \tag{3.34}$$

Proof. Let us observe that

$$[\mathbf{G}^T]^{-1} = [(\gamma\mathbf{A}^T + \delta\mathbf{1}_n\mathbf{1}_n^T)^T]^{-1} = (\gamma\mathbf{A} + \delta\mathbf{1}_n\mathbf{1}_n^T)^{-1} \tag{3.35}$$

The thesis simply follows from the application of the Shermann-Morrison formula [73] to Eq. (3.35). ■

Theorem 6. *Let* $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{v} \in \mathbb{R}^n$ *and* $\boldsymbol{i} \in \mathbb{R}^n$. *If* $\boldsymbol{v} = \boldsymbol{A}^{-1}\boldsymbol{i}$ *and* $\tilde{\boldsymbol{v}} = [\boldsymbol{G}^T]^{-1}\boldsymbol{i}$ *then:*

$$\boldsymbol{v} = \gamma\left(\tilde{\boldsymbol{v}} + \frac{\delta\sum_{i=1}^n \tilde{v}_i}{1 - \delta\sum_{i=1}^n \tilde{w}_i}\tilde{\boldsymbol{w}}\right) \tag{3.36}$$

where $\tilde{\mathbf{w}} = [\mathbf{G}^T]^{-1}\mathbf{1}_n$.

Proof. Let us first consider an additional linear system $\mathbf{w} = \mathbf{A}^{-1}\mathbf{1}_n$ and its linearly transformed $\tilde{\mathbf{w}} = [\mathbf{G}^T]^{-1}\mathbf{1}_n$. By applying Prop. 3 we get:

$$\begin{aligned}
\tilde{\mathbf{w}} = [\mathbf{G}^T]^{-1}\mathbf{1}_n &= \left[\frac{1}{\gamma}\mathbf{A}^{-1} - \frac{\delta}{\gamma}\left(\frac{\mathbf{A}^{-1}\mathbf{1}_n\mathbf{1}_n^T\mathbf{A}^{-1}}{\gamma + \delta\mathbf{1}_n^T\mathbf{A}^{-1}\mathbf{1}_n}\right)\right]\mathbf{1}_n = \\
&= \frac{1}{\gamma}\mathbf{w} - \frac{\delta}{\gamma}\left(\frac{\mathbf{1}_n^T\mathbf{w}}{\gamma + \delta\mathbf{1}_n^T\mathbf{w}}\right)\mathbf{w} = \\
&= \frac{1}{\gamma}\mathbf{w} - \frac{\delta}{\gamma}\left(\frac{\sum_{i=1}^n w_i}{\gamma + \delta\sum_{i=1}^n w_i}\right)\mathbf{w} = \\
&= \frac{1}{\gamma}\frac{\gamma + \cancel{\delta\sum_{i=1}^n w_i} - \cancel{\delta\sum_{i=1}^n w_i}}{\gamma + \delta\sum_{i=1}^n w_i}\mathbf{w} = \\
&= \frac{\mathbf{w}}{\gamma + \delta\sum_{i=1}^n w_i}
\end{aligned}$$

Let us now apply Prop. 3 to the linear system

$$\begin{aligned}
\tilde{\mathbf{v}} = [\mathbf{G}^T]^{-1}\mathbf{i} &= \left[\frac{1}{\gamma}\mathbf{A}^{-1} - \frac{\delta}{\gamma}\left(\frac{\mathbf{A}^{-1}\mathbf{1}_n\mathbf{1}_n^T\mathbf{A}^{-1}}{\gamma + \delta\mathbf{1}_n^T\mathbf{w}}\right)\right]\mathbf{i} = \\
&= \frac{1}{\gamma}\mathbf{v} - \frac{\delta}{\gamma}\left(\frac{\mathbf{w}}{\gamma + \delta\sum_{i=1}^n w_i}\right)\mathbf{1}_n^T\mathbf{v} = \\
&= \frac{1}{\gamma}\mathbf{v} - \frac{\delta}{\gamma}\tilde{\mathbf{w}}\mathbf{1}_n^T\mathbf{v} = \\
&= \frac{1}{\gamma}(\mathbb{I} - \delta\tilde{\mathbf{w}}\mathbf{1}_n^T)\mathbf{v}
\end{aligned}$$

It follows that $\mathbf{v} = \gamma(\mathbb{I} - \delta\tilde{\mathbf{w}}\mathbf{1}_n^T)^{-1}\tilde{\mathbf{v}}$. By further applying Shermann-Morrison formula we get:

$$\begin{aligned}
\mathbf{v} &= \gamma\left(\mathbb{I} + \frac{\delta\tilde{\mathbf{w}}\mathbf{1}_n^T}{1 - \delta\mathbf{1}_n^T\tilde{w}}\right)\tilde{\mathbf{v}} = \\
&= \gamma\left(\tilde{\mathbf{v}} + \frac{\delta\tilde{\mathbf{w}}\mathbf{1}_n^T\tilde{\mathbf{v}}}{1 - \delta\mathbf{1}_n^T\tilde{w}}\right) = \\
&= \gamma\left(\tilde{\mathbf{v}} + \frac{\delta\sum_{i=1}^n \tilde{v}_i}{1 - \delta\sum_{i=1}^n \tilde{w}_i}\tilde{\mathbf{w}}\right)
\end{aligned}$$

■

This proposition shows that for computing the recovering step of the linear transformation (2.3), it is necessary to perform a calibration step, i.e. $\tilde{\mathbf{w}} = [\mathbf{G}^T]^{-1}\mathbf{1}_n$, whose result can be then used to find the correct solution of the computation.

In contrast to the open-loop crossbar, a careful gain and stability analysis must be performed for the feedback crossbar. The loop gain analysis of the circuit shows

that the diagonal elements of the inverse matrix $\mathbf{A}^{-1}$ must be positive. This ensures a negative gain for each loop and thus prevents the system from operating in an unstable positive feedback region [75]. The class of matrices that satisfy this constraint are $M-$matrices and positive definite matrices. To be able to provide the correct distribution of the currents over the crossbar, positive terminal of the op-amp needs to be as close as possible to the virtual reference voltage, thus requires a high open-loop gain.

The feedback crossbar depicted in Fig. 3.6 differs from the open-loop structure because it uses current instead of voltage inputs, which requires current noise variances to obtain the output distribution. Moreover, the magnitude of the random fluctuations strongly depends on the conductance matrix itself. If we consider the system $(\mathbf{G}+\mathbf{Z})^T(\mathbf{v}+\boldsymbol{\epsilon}) = (\mathbf{i}+\boldsymbol{\eta})$ and assume that $\|\mathbf{Z}\|\|\mathbf{G}^{-1}\| < 1$ for any induced norm, then:

$$\frac{\|\boldsymbol{\epsilon}\|}{\|\mathbf{v}\|} \leq \frac{K(\mathbf{G})}{1-\|\mathbf{Z}\|\|\mathbf{G}^{-1}\|}\left(\frac{\|\mathbf{Z}\|}{\|\mathbf{G}\|}+\frac{\|\boldsymbol{\eta}\|}{\|\mathbf{i}\|}\right) \tag{3.37}$$

where $K(\mathbf{G}) = \|\mathbf{G}\|\|\mathbf{G}^{-1}\|$ is the condition number [77] and $\boldsymbol{\epsilon}, \boldsymbol{\eta} \in \mathbb{R}^n$. Equation (3.37) states that if the matrix $\mathbf{G}$ is well-conditioned (i.e. $K(\mathbf{G})$ is close to 1), then small changes in $\mathbf{G}$ and $\mathbf{i}$ produce correspondingly small changes in the solution $\mathbf{v}$. If, on the other hand, $\mathbf{G}$ is ill-conditioned (i.e. $K(\mathbf{G})$ is large), then small changes in $\mathbf{G}$ and $\mathbf{i}$ may produce large changes in $\mathbf{v}$. This implies that the system precision strongly depends on the nature of the application [67]. Moreover, matrix properties also affect the convergence time and accuracy [78]. Thus, the same circuit can converge with different accuracy using different matrices. However, a high GBW (gain-bandwidth product) provides faster convergence for the case of badly conditioned matrices.

3.4.2 Linear transformation $\mathbf{z} = \mathbf{B}\mathbf{A}^{-1}\mathbf{c}$

In some applications that make use of linear algebra operations, another expensive computation relies on the following computation:

- solving a linear system $\mathbf{x} = \mathbf{A}^{-1}\mathbf{c}$,
- computing the MVM $\mathbf{z} = \mathbf{B}\mathbf{x}$.

Hereinafter, we define the combination of these two steps as the single *Matrix-Inverse-Vector Multiplication* (MIVM):

$$\mathbf{z} = \mathbf{B}\mathbf{A}^{-1}\mathbf{c} \tag{3.38}$$

with known $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{m\times n}$ and $\mathbf{c} \in \mathbb{R}^n$.

Thus feedback and open-loop crossbars can be used to perform inference process together and avoid the expensive computation of the inverse matrix $\mathbf{A}^{-1}$. In Fig.

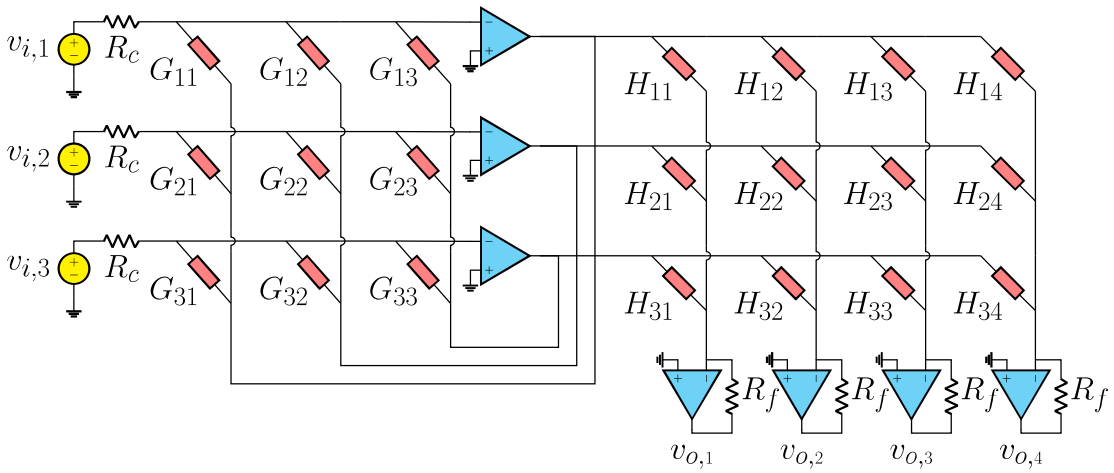


Figure 3.7: A schematic illustration of a 3×4 open-loop crossbar combined with with a 3×3 feedback crossbar.

3.7, feedback and open-loop structures are connected back-to-back where feedback crossbar solves the linear system $\mathbf{x} = \mathbf{A}^{-1}\mathbf{c}$ and open-loop crossbar uses this result as an input to compute the MVM $\mathbf{z} = \mathbf{B}\mathbf{x}$.

As it is required by the feedback crossbar, the recovery step for this novel structure due to the linear transformation defined in Eq. (2.3) takes a two-step calculation as described in the following result:

Theorem 7. *Let $\boldsymbol{G}$ defined as in Eq. (2.3) and*

$$\boldsymbol{H} = \alpha \boldsymbol{B}^T + \beta \mathbf{1}_n \mathbf{1}_m^T, \quad \alpha = \frac{G_{max} - G_{min}}{B_{max} - B_{min}}, \quad \beta = G_{max} - \alpha B_{max}$$

The MIVM $\boldsymbol{z} = \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{c}$ with known $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{c}$ can be solved by performing the following two steps:

1) Solve $\tilde{\boldsymbol{q}} = [\boldsymbol{H}^T][\boldsymbol{G}^T]^{-1}\boldsymbol{1}_n$ with $\tilde{\boldsymbol{y}} = [\boldsymbol{G}^T]^{-1}\boldsymbol{1}_n$ (calibration step)

2) Solve $\tilde{\boldsymbol{z}} = [\boldsymbol{H}^T][\boldsymbol{G}^T]^{-1}\boldsymbol{c}$ with $\tilde{\boldsymbol{x}} = [\boldsymbol{G}^T]^{-1}\boldsymbol{c}$ (inference)

The correct solution $\boldsymbol{z}$ can be found by performing the following recovering step

$$\boldsymbol{z} = \frac{\gamma}{\alpha}[\tilde{\boldsymbol{z}} + (\delta k)\tilde{\boldsymbol{q}} - (\beta k)\boldsymbol{1}_m], \quad k = \frac{\sum_i \tilde{\boldsymbol{x}}_i}{1 - \delta \sum_i \tilde{\boldsymbol{y}}_i}. \tag{3.39}$$

Proof. Let $\mathbf{x} = \mathbf{A}^{-1}\mathbf{c}$ and $\mathbf{z} = \mathbf{B}\mathbf{x}$. From Theorem 6 we know that

$$\mathbf{x} = \gamma \left(\tilde{\mathbf{x}} + \frac{\delta \sum_{i=1}^n \tilde{x}_i}{1 - \sum_{i=1}^n \tilde{y}_i} \tilde{\mathbf{y}} \right)$$

where $\tilde{\mathbf{y}} = [\mathbf{G}^T]^{-1}\mathbf{1}_n$. We know that

$$\mathbf{H} = \alpha\mathbf{B}^T + \beta\mathbf{1}_n\mathbf{1}_m^T \Rightarrow \mathbf{B} = \frac{1}{\alpha}\left(\mathbf{H}^T - \beta\mathbf{1}_m\mathbf{1}_n^T\right)$$

It follows that:

$$\begin{aligned}
\mathbf{z} &= \gamma\mathbf{B}\left(\tilde{\mathbf{x}} + \frac{\delta\sum_{i=1}^n \tilde{x}_i}{1-\sum_{i=1}^n \tilde{y}_i}\tilde{\mathbf{y}}\right) = \\
&= \frac{\gamma}{\alpha}\left(\mathbf{H}^T - \beta\mathbf{1}_m\mathbf{1}_n^T\right)\left(\tilde{\mathbf{x}} + \frac{\delta\sum_{i=1}^n \tilde{x}_i}{1-\sum_{i=1}^n \tilde{y}_i}\tilde{\mathbf{y}}\right) = \\
&= \frac{\gamma}{\alpha}\left(\mathbf{H}^T\tilde{\mathbf{x}} + \frac{\delta\sum_{i=1}^n \tilde{x}_i}{1-\sum_{i=1}^n \tilde{y}_i}\mathbf{H}^T\tilde{\mathbf{y}} - \beta\delta\sum_{i=1}^n \tilde{x}_i\mathbf{1}_m - \beta\frac{\delta\sum_{i=1}^n \tilde{x}_i\sum_{i=1}^n \tilde{y}_i}{1-\sum_{i=1}^n \tilde{y}_i}\mathbf{1}_m\right) = \\
&= \frac{\gamma}{\alpha}\left[\tilde{\mathbf{z}} + (\delta k)\tilde{\mathbf{q}} - (\beta k)\mathbf{1}_m\right]
\end{aligned}$$

■

In the two-steps recovery process, the only important information provided by vector $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{x}}$ is their sums as shown in Eq. (3.39). By introducing an additional column line in the open-loop crossbar that works as an adder (e.g. $H_{mj} = G_{min}$, $\forall j = 1, \ldots, n$ is the $m-$th row of $\mathbf{H}$), the combination of the two structures can be used to perform the MIVM operation in a single platform.

The main bottleneck of the two structures is the feedback crossbar. The convergence time is a function of the matrix condition number and varies for each application. More details can be found either in Section 3.4.1 or in [67].

The open-loop structure is easier to implement compared to feedback crossbar. It is inherently stable, does not require a very high performance op-amp and its performance is not affected by the condition number.

3.5 Conclusion

The aim of this chapter was to provide a novel analysis of both random and systematic errors to establish the basis for future comparisons with similar systems that make use of linear algebra applications. It introduced the mathematical tools to compute the SNR and determine the output precision of the system. This method is non-application specific and allows the users to have a valid comparison between crossbars and their analog and digital counterparts for any practical use. The key contributions of this chapter are:

1. The unified formulation of the output noise distributions produced by the programming noise in Eq. (3.4), the thermal noise in Eq. (3.9) and the shot noise in Eq. (3.13);

2. The closed mathematical expression of the wire resistance contribution when solving a MVM operation defined in Eq. (3.27);

3. The formulation of a calibration stage for the recovering steps defined in Props. (6) and (7) for the feedback loops circuits depicted in Figs. 3.6 and 3.7, respectively.

Chapter 4

Analog Implementation of Probabilistic Models

This chapter aims to validate the theoretical results found in Chapter 3. The systems performance is evaluated in solving different applications of famous probabilistic models whose inference relies on linear algebra computations: Markov Chains (MCs) and Gaussian Processes (GPs). The different systems are evaluated using SNR, accuracy and energy efficiency. The results show how discrete MCs and GPs can meet the different circuit requirements by providing a detailed theoretical background to support the design study of the analog memristor crossbar. In particular, the goal is to provide a complete end-to-end description to rigorously map the inference operations characterizing these two probabilistic models into memristor crossbars and find their solutions. This approach enables the reader to clearly see the linear and inverse transformations, the output precision in terms of bits, and how MC and GP properties satisfy the feedback structure's requirements. The first part of this chapter is based on the article "Analog Solutions of Discrete Markov Chains via Memristor Crossbars" by Zoppo et al [67].

4.1 Discrete-Time Markov Chains

This section introduces discrete-time MCs, with the main focus on the Markov property and transition probability matrices. Further details can be found in [79]. A discrete random variable with finite state X is a measurable function $X : \Omega \to \Gamma$ where Ω is the set of possible outcomes and $\Gamma = \{1, \ldots, m\}$. A discrete time stochastic process with finite state is a sequence $\{X_k : k \in \mathbb{N}\}$ of discrete random variables with finite state.

Definition 3. *A first-order (time) homogeneous MC with finite state is a discrete time stochastic process that for all $k, h \in \mathbb{N}$ satisfies:*

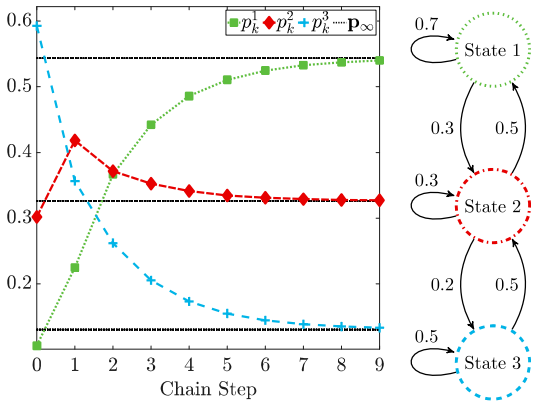


Figure 4.1: Graphical representation of a generic regular three-state MC together with Power Method iterations. Black dashed lines correspond to the limiting distribution. Taken from [67] ©2021 IEEE.

- *Markov property:*

$$p(X_{k+1} = x_{k+1}|X_k = x_k, \ldots, X_0 = x_0) = p(X_{k+1} = x_{k+1}|X_k = x_k) \tag{4.1}$$

- *Time homogeneity property:*

$$p(X_{k+1} = x_{k+1}|X_k = x_k) = p(X_{h+1} = x_{h+1}|X_h = x_h) \tag{4.2}$$

The transition probabilities can be represented in a compact form by the transition matrix $\mathbf{M}$:

$$M_{ij} = p(X_{k+1} = j|X_k = i). \tag{4.3}$$

$\mathbf{M}$ is a $m \times m$ non-negative probability matrix for which the column elements sum to 1. The matrix size denotes the number of states m of the MC. Starting from an initial distribution $\mathbf{p}_0 = [p(X_0 = 1), \ldots, p(X_0 = m)]^T$, the $k-$th distribution $\mathbf{p}_k$ can be deduced from the previous $k-1$ distributions by simply performing the following MVMs:

$$\mathbf{p}_k = \mathbf{M}\mathbf{p}_{k-1} \Rightarrow \mathbf{p}_k = \mathbf{M}^k\mathbf{p}_0 \tag{4.4}$$

where $\mathbf{M}^k$ is the k-th power of $\mathbf{M}$.

Definition 4. *A finite MC with transition matrix $\boldsymbol{M}$ is said to be irreducible if for all pairs of states (i, j), there exists some $r = r(i, j) \in \mathbb{N}$ such that $M_{ij}^r > 0$.*

Irreducibility guarantees that every state can be reached from any other state. A MC with transition probability matrix $\mathbf{M}$ is said to have a stationary distribution $\mathbf{p}_\infty = [p_\infty^1, \ldots, p_\infty^m]^T$ if

$$\mathbf{p}_\infty = \mathbf{M}\mathbf{p}_\infty, \quad \sum_{i=1}^{m} p_\infty^i = 1. \tag{4.5}$$

Whenever the chain starts in the stationary distribution, it remains in that distribution at any subsequent timestamp. There could be more than one stationary distribution corresponding to the eigenvectors of the transition matrix eigenvalue of 1. However, when the chain is irreducible, the uniqueness of the stationary distribution $\mathbf{p}_\infty$ is guaranteed and for every state i, p^i_∞ corresponds to the mean return time to that state. Each component of the stationary distribution can be regarded as the proportion of time spent by the chain in a given state.

Depending on the initial distribution, an irreducible MC does not necessarily converge towards its stationary distribution. Thus, it is necessary to introduce the concept of regular MCs.

Definition 5. *A matrix $\boldsymbol{M}$ is said to be primitive if there exists $r \in \mathbb{N}$ such that $M^r_{ij} > 0$ for all pairs of states (i, j). A finite MC with a primitive transition matrix $\boldsymbol{M}$ is regular.*

Regular MCs are always irreducible, but the reverse requires a further constraint:

Definition 6. *The period d_i of the state i is given by $d_i = gcd\{r \geq 1 : M^r_{ii} > 0\}$ where gcd denotes the greatest common divisor. A state i is said to be aperiodic if $d_i = 1$. The finite MC and its transition matrix $\boldsymbol{M}$ are called aperiodic if all states are aperiodic.*

Proposition 4. *The following properties hold:*

a) A finite MC is irreducible and aperiodic if and only if it is regular;

b) Let us consider an irreducible finite MC with transition matrix $\boldsymbol{M}$ and assume there exists $i \in \Gamma$ such that $M_{ii} > 0$, then the MC is regular;

c) If a finite MC is regular then it holds $\lim_{n\to\infty} p(X_n = i|X_0 = j) = p^i_\infty \ \forall j$.

For further details on the proofs, see [79]. Whenever a limiting distribution exists, the chain converges to it regardless of the initial condition. Each component of the limiting distribution can have two different meanings: long-run probabilities or proportion of time spent by the chain in a given state. The discrete-time evolution of a finite MC can be studied by successively iterating Eq. (4.4). Starting from an arbitrary initial state, one can predict the next most probable state of the chain by performing a MVM operation. By repeating this process, one can infer the most probable state after k steps. Figure 4.1 shows the evolution of the distribution vector of a generic regular MC at each iteration. After a certain number of transitions the distribution does not change. For this reason, it is interesting to analyse the stationary/limiting distribution of the chain.

4.1.1 Solving Discrete-Time Markov Chains

As can be inferred from Eq. (4.5), the stationary distribution of a MC can have two different interpretations: the solution of an eigenvector problem or of a linear system. As a result of this duality, a large variety of algorithms are available for solving finite MCs: direct and iterative methods [80]. The first class mostly requires the whole coefficient matrix to be stored and applications are often limited by storage capacities and computational efficiency (Gaussian Elimination, LU decomposition, GTH-algorithm, etc). The second class is preferable when dealing with large problems (Gauss-Seidel, Power Method, etc). Nevertheless, the time complexity is a polynomial function of the matrix size for both classes. Further details on space and time complexity can be found in [77].

Traditionally, the stationary distribution of a MC has been regarded as an eigenvector problem and the Power Method has been the popular method of choice. This algorithm starts with an initial distribution vector $\mathbf{p}_0$, which may be either random or an approximation to the dominant eigenvector. At every iteration k, the algorithm is a MVM between the transition matrix $\mathbf{M}$ and the current state $\mathbf{p}_k$ until a stopping criterion is satisfied.

Recently, a closed-loop circuit implementation was proposed that computes the dominant eigenvector of a matrix in a single time step [64]. This bypasses the successive MVMs characterizing the Power Method, but it precludes the possibility to compute the k-th step prediction that sometimes arises in MC applications. The irreducibility of the transition matrix $\mathbf{M}$ guarantees the existence of a unique stationary distribution. However, the convergence of both the closed-loop circuit implementation and the iterative Power Method is influenced by the primitivity of $\mathbf{M}$. If the stochastic matrix is not primitive, then it may have several eigenvalues on the unit circle, causing convergence problems. This results in long-run probabilities of the chain that strongly depend on the initial condition. For primitive matrices, the convergence to the unique dominant eigenvector is guaranteed. This class of matrices has only one eigenvalue on the unit circle and all other eigenvalues have modulus strictly less than 1. For this reason, starting from any probability vector, the iterative process is guaranteed to converge.

Remark 4. *One way to ensure that the problem has a solution when the primitivity condition is not verified is to consider a modified transition matrix* $\boldsymbol{Q} = \alpha\mathbb{I}_m + (1-\alpha)\boldsymbol{M}$ *where* $0 < \alpha < 1$ *and* $\mathbb{I}_m$ *is the* $m \times m$ *identity matrix. If* $\boldsymbol{p}_\infty = \boldsymbol{M}\boldsymbol{p}_\infty$ *then*

$$\begin{aligned}\boldsymbol{Q}\boldsymbol{p}_\infty &= [\alpha\mathbb{I}_m + (1-\alpha)\boldsymbol{M}]\boldsymbol{p}_\infty = \\ &= \alpha\mathbb{I}_m\boldsymbol{p}_\infty + (1-\alpha)\boldsymbol{M}\boldsymbol{p}_\infty = \\ &= \alpha\boldsymbol{p}_\infty + (1-\alpha)\boldsymbol{p}_\infty = \\ &= \alpha\boldsymbol{p}_\infty + \boldsymbol{p}_\infty - \alpha\boldsymbol{p}_\infty = \boldsymbol{p}_\infty.\end{aligned}$$

Thus, the two chains with transition matrices $\boldsymbol{M}$ *and* $\boldsymbol{Q}$ *share the same stationary distribution. Nevertheless, the latter is now a regular MC as highlighted in Prop.*

4(b) and therefore has a unique limiting distribution with the accuracy of the results influenced by the choice of α.

The alternative approach to the iterative method is the direct calculation of the eigenvector $\mathbf{p}_\infty$ by solving an associated linear system as shown in the following:

Proposition 5. *Let* $\boldsymbol{M}$ *be the transition matrix of a finite irreducible MC with stationary distribution* $\boldsymbol{p}_\infty$.
Let $\boldsymbol{A} = (\mathbb{I}_m - \boldsymbol{M} + \mathbf{1}_{m\times m})$ *where* $\mathbf{1}_{m\times m}$ *is the matrix with all entries equal to* 1 *and* $\mathbf{1}_m$ *is the column vector of ones. Then:*

a) $\boldsymbol{A}$ *is non-singular and* $A^{-1}_{ii} > 0\ \forall i = 1, \dots, m$;

b) $\boldsymbol{p}_\infty$ *is the unique solution of the system*

$$\boldsymbol{A}\boldsymbol{x} = \mathbf{1}_m. \tag{4.6}$$

c) The linear system defined in Eq. (4.6) is equivalent to the following system

$$\boldsymbol{G}^T\boldsymbol{v} = (\gamma + \delta)\mathbf{1}_m.$$

where $\boldsymbol{G}$ *is defined in Eq. (2.3).*

Proof. a) For details see [81, 82].
b) The objective is to find $\mathbf{p}_\infty$ such that $\mathbf{M}\mathbf{p}_\infty = \mathbf{p}_\infty$, subject to $\sum_{i=1}^m p^i_\infty = 1$. This constraint can be written as

$$\mathbf{1}_{m\times m}\mathbf{p}_\infty = \mathbf{1}_m. \tag{4.7}$$

By summing $\mathbf{p}_\infty - \mathbf{M}\mathbf{p}_\infty = 0$ to (4.7) and by collecting $\mathbf{p}_\infty$, the claim is obtained. For a), the irreducibility assumption ensures the non-singularity of the matrix, thus the uniqueness of the solution.
c) Consider the linear system $\mathbf{A}\mathbf{x} = \mathbf{1}_m$. By multiplying both sides by γ and then adding the vector $\delta\mathbf{1}_{m\times m}\mathbf{x}$, one obtains the equality:

$$\gamma\mathbf{A}\mathbf{x} + \delta\mathbf{1}_{m\times m}\mathbf{x} = \gamma\mathbf{1}_m + \delta\mathbf{1}_{m\times m}\mathbf{x} \tag{4.8}$$

By collecting $\mathbf{x}$ and observing that $\mathbf{1}_{m\times m}\mathbf{x} = \mathbf{1}_m$, one gets $\mathbf{G}^T\mathbf{x} = (\gamma + \delta)\mathbf{1}_m$. ■

In conclusion, (4.5) and (4.6) share the same solution. However, a direct computation of the linear system in (4.6) avoids the possible convergence issues arising in the iterative algorithms. Proposition 5(a) guarantees that the feedback system defined in Section 3.4.1 has a negative gain for each loop avoiding the system from operating in an unstable positive feedback region. Proposition 5(c) allows to circumvent the necessity of considering a recovering step to find the correct solution of the linear system (4.6) as defined in Prop. 6. In addition, the output can be obtained in two different ways:

- for an input vector $(\gamma + \delta)\mathbf{1}_m$, the circuit will provide the normalized eigenvector as the solution;
- for an input vector $\mathbf{1}_m$, the correct solution can be recovered by normalizing the output voltage vector.

The $(\gamma+\delta)$ term in the first approach can become a very small number based on the selection of memristor conductance range and matrix $\mathbf{A}$. This can impose physical limitations to circuit design. However, the second approach can be used for any conductance range. Thus, in the next section, an input vector $\mathbf{1}_m$ is introduced and the correct solution is obtained by normalizing the output voltage vector. Both configurations require ADCs (Analog-to-Digital Converters) after the crossbar TIAs to perform post-processing operations such as normalization and inverse linear transform in digital domain.

To review, the open-loop crossbar can be used either to predict the next state distribution vector via a simple MVM operation or to compute the stationary distribution of the regular chain by performing sequential updates by the Power Method. However, if the evolution of the system is not necessary, one can compute the stationary distribution of a MC in a single step using the feedback configuration.

4.1.2 Case Studies

In this section, two examples are illustrated in order to show systems performance on different applications. The two systems are evaluated using SNR, accuracy and energy efficiency. In the next results, the wire resistance contribution is assumed to be negligible since by performing the pre-processing proposed in [49] the errors can decreases as low as 5%. This allows to highlight the contribution given by the finite gain and the programming error.

Mouse in a Maze

The mouse in a maze shown in Fig. 4.2(a) is a classic example of a MC problem. Here there are nine rooms and each of them is connected to the adjacent rooms. At each timestamp, the mouse chooses the next room to visit based on the probability columns of the transition matrix shown in Fig. 4.2(b). Each room/state is labeled with numbers between 1 - 9 and the mouse is initially in room 3. Iterating Eq. (4.4), one can compute the probability distribution of the system and infer the most likely position of the mouse after 6 moves as shown in the histogram plotted in Fig. 4.2(c).

The mouse can only go from an odd-numbered room to an even-numbered room, and vice versa. Hence, the chain is not aperiodic: from any initial condition, the chain will alternately be in even- or odd-numbered states. However, the mouse is able to reach all the possible rooms, implying that the chain is irreducible. This

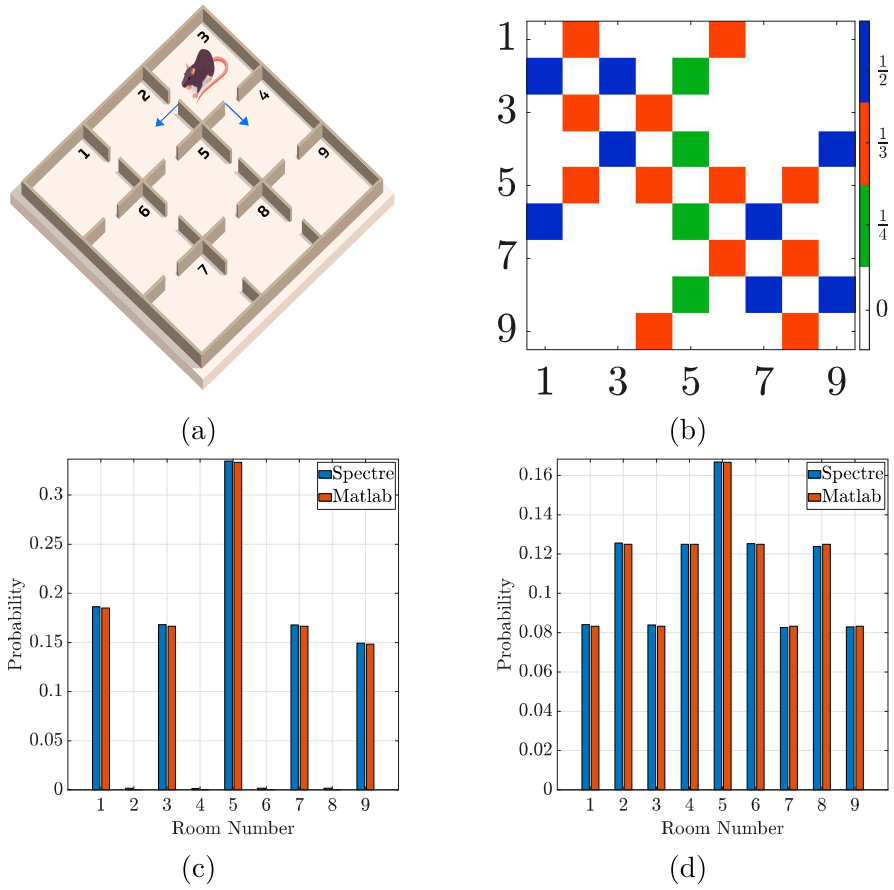


Figure 4.2: (a) Example of a mouse trapped in a maze. (b) Transition matrix representing the probability that the mouse goes from one room to another. (c) Comparison between the 6-th step probability distribution using `MatLAB` and the proposed open-loop crossbar with 8–bits precision memristors. (d) Comparison between the stationary distribution using `MatLAB` and the proposed feedback crossbar with 8–bits precision memristors. Taken from [67] ©2021 IEEE.

guarantees the uniqueness of the stationary distribution. By solving Eq. (4.6), one can analytically find

$$\mathbf{p}_\infty = \left[\frac{1}{12}, \frac{1}{8}, \frac{1}{12}, \frac{1}{8}, \frac{1}{6}, \frac{1}{8}, \frac{1}{12}, \frac{1}{8}, \frac{1}{12}\right]^T \tag{4.9}$$

plotted in the histogram of Fig. 4.2(d). Since the chain is not regular, the resulting vector does not coincide with the limiting distribution but provides the mouse's mean occupation times in each room.

The 6-th step probability distribution computation was implemented on the open-loop circuit in Fig. 3.1(b) using 8–bit precision memristors and $[100-1000]\mu S$ as memristor conductance range. The system was simulated in the Cadence/Spectre

software with default convergence settings, corner conditions, and a temperature of $27°C$. The `MatLAB` software was used to both implement the iterative process and normalize the collected results. The probability distribution vector is given in Fig. 4.2(c). The normwise relative error between the measured output and the true output given in Eq. (4.9) was measured to be 0.82%. The computation of the stationary distribution problem was implemented on the feedback circuit in Fig. 3.6. The stationary distribution $\mathbf{p}_\infty$ is given in Fig. 4.2(d). The normwise relative error of 0.61% was obtained from the circuit simulations. The accuracy for both architectures shows promising results, mainly due to the use of a small matrix and crossbar. The accuracy can be further improved by decreasing the systematic errors with higher gain op-amps and higher precision memristors. The SNR and energy efficiency for this type of random matrix problem will be evaluated in the next subsection by considering larger matrix sizes.

The Pagerank Algorithm

A very different type of MC problem is the core of Google's search engine [83]. Among all the different ranking algorithms [84], Pagerank considers a random walk on the web graph. It assigns a score that is proportional to the probability of being on that page after a large number of moves, i.e. the limiting distribution of the MC. Assuming there are m pages, one can generate an adjacency matrix $\mathbf{A}$ by setting each entry equal to 1 if node j has a link to node i, and 0 otherwise. Then, the transition matrix $\mathbf{W}$ can be constructed by normalizing each entry of the matrix to get sums of 1 for each column. Columns having all zeros entries (i.e. dangling nodes) are replaced by a uniform distribution vector in order to make $\mathbf{W}$ a well-defined stochastic matrix. This construction does not guarantee the MC's irreducibility and therefore the uniqueness of the stationary distribution. This happens when the web graph has some sinks: not all pages are reachable from any given arbitrary node. One way to implement this property consists in allowing the web surfer to jump from the current website to a new random website. The strategy is to perturb $\mathbf{W}$ in terms of a damping factor α that uniformly spreads part of the rank. This results in $\mathbf{M} = \alpha\mathbf{W} + (1-\alpha)\mathbf{R}$, where $\mathbf{R}$ is the matrix in which all entries are $\frac{1}{m}$. The irreducible transition matrix $\mathbf{M}$ is generally called the Google Matrix and a typical value for α is 0.85. $\mathbf{M}$ is now an entrywise positive matrix and thus primitive, but it is far from 'random'. This yields two possible interpretations of the Pagerank: long-run probabilities or mean occupation times.

The pagerank algorithm was implemented on both open-loop and feedback circuits using 8–bit precision memristors and $[100-1000]\mu S$ as memristor conductance range. In order to perform a realistic comparison with the current literature, the publicly available Mathworks dataset [85] was chosen. The adjacency matrix is shown in Fig. 4.3(a). The Cadence/Spectre software was used to simulate the circuit and collect the results.

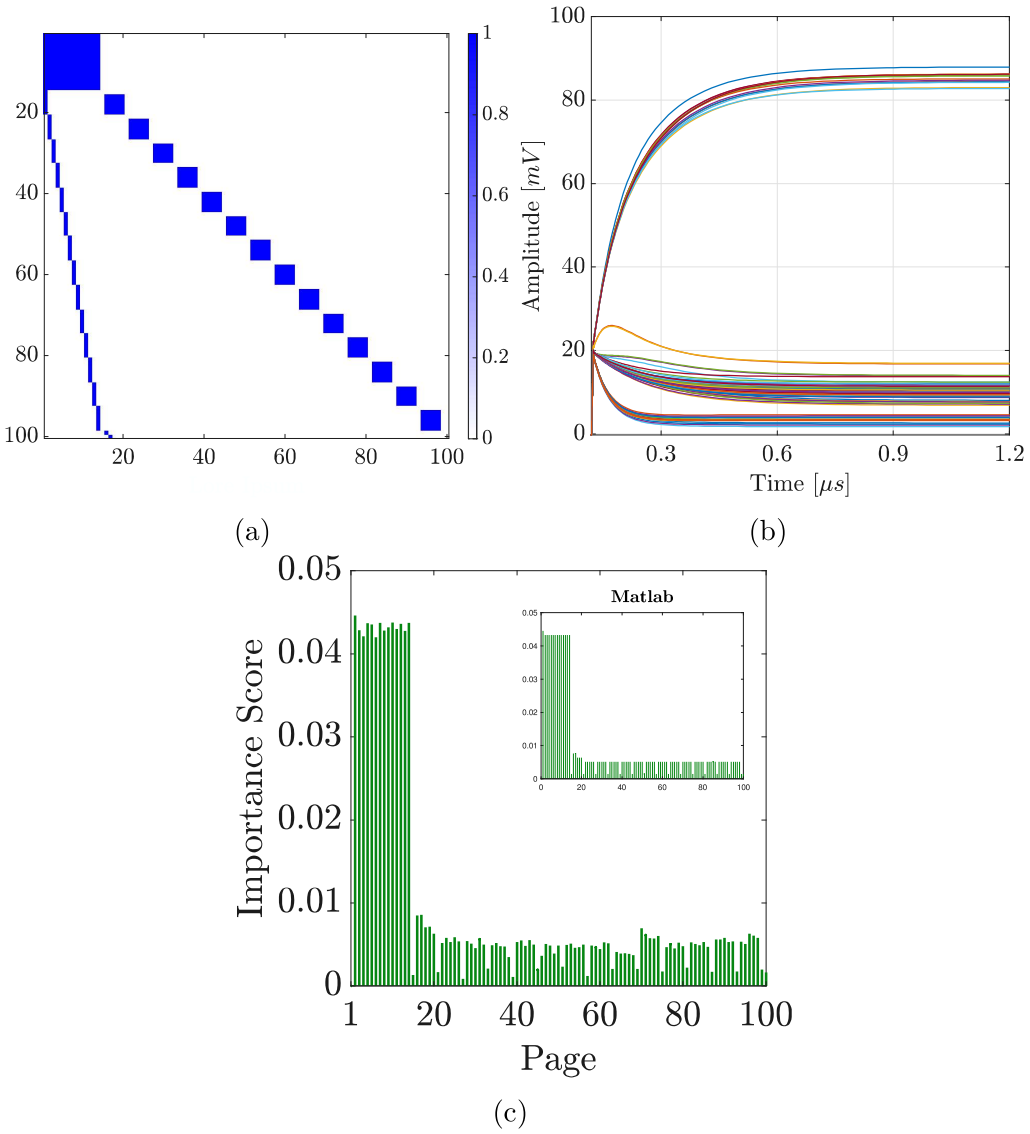


Figure 4.3: (a) Adjacency matrix representing the links between the 100 pages of Mathworks dataset. (b) Transient behavior of the feedback system converging to the limiting distribution of the chain. (c) Simulated pageranks provided by the feedback crossbar. Taken from [67] ©2021 IEEE.

To compute the Pagerank with the open-loop crossbar, the Power method was applied to the Mathworks transition matrix. For a fair comparison with the feedback system, the same op-amp was used. The transistor level of the op-amp given in [67] was used in the simulations with a gain of $\approx 86dB$, gain-bandwidth-product of $1.1GHz$ and a power consumption of $163\mu W$ with a $0.9V$ power supply. Since the open-loop crossbar op-amp does not require the gain or GBW to be as high as

the feedback crossbar, relaxing the requirements of the op-amp can improve energy efficiency. The total power consumption of the open-loop crossbar is:

$$\sum_{j=1,k=1}^{100} (i_{jk}^2 R_{jk} + i_f^2 R_f + P_{amp}) \approx 51\ mW \tag{4.10}$$

where i_{jk} and R_{jk} are the current and resistance values of each memristor, respectively, and i_f is the total current flowing through the feedback resistor. The convergence time is independent of the matrix properties. The SNR is calculated to be $56dB$ for the open-loop configuration. This approximately corresponds to 9 $bits$ output precision per iteration. Percent normwise relative errors are shown in Table 4.1, and in the case of the open-loop crossbar they refer to a single iteration.

The iteration process implemented in the open-loop crossbar can be performed in two ways: transferring output voltages to the digital domain using ADCs [86, 87], performing recovery and normalization steps in a digital computer, and applying them to the crossbar inputs using DACs for the next iteration; or designing analog recovery and normalization circuitry. Registers, which can be either digital or analog sample and hold circuitry, are needed to apply previous iteration outputs as subsequent inputs. In both approaches, there will be a power penalty in addition to the computation time delay. The use of digital systems at each iteration will also introduce quantization noise and further decrease the final output precision. The number of iterations depends on the application and desired output accuracy. The corresponding time and energy penalties to perform k iterations can be found in Table 4.1.

The PageRank computed as the solution of the linear system defined in Eq. (4.6) was simulated in the feedback circuit. The transient response of the system's outputs is shown in Fig. 4.3(b). These results were collected and normalized using `MatLAB` software. The importance score of the pages is given in Fig. 4.3(c). To quantify the accuracy, a normwise relative error of 4.2% was obtained from the simulations. Figure 4.3(b) shows that the op-amps have settled within 0.1% error of their final value in $552ns$. The total power consumption of the crossbar, input resistors and op-amps is:

$$\sum_{j=1}^{100} (i_{in}^2 R_{m,j} + i_{in}^2 R_c + P_{amp}) \approx 28\ mW \tag{4.11}$$

where $R_{m,j}$ is the parallel combination of memristor resistances on the $j-$th row and i_{in} is the input current for each row. More than 90% of the power is consumed by the input resistors R_c and op-amps. The power consumption of the op-amp can be decreased with a more advanced technology node. However, the efficiency of the feedback crossbar may be limited because of the high gain and GBW requirements of the system.

The SNR analysis is performed for this application using (3.1). The mean of the signal-to-noise ratio for 100 outputs was calculated as $32dB$, which corresponds to an output precision of 5 *bits*.

The minimum eigenvalue of the transition matrix in the Mathworks application is very small, i.e. $\lambda_{min} = 0.0028$, which results in a longer convergence time. However, for higher values of λ_{min}, the convergence time can decrease significantly [67, 78].

In summary, the feedback crossbar usually offers better energy efficiency because it is a one-step system, limited by the convergence time of the circuit. The use of the same input current for all rows in the feedback structure provides further potential power efficiency through the input peripheral circuitry compared to the open-loop crossbar in a higher level system design. The feedback crossbar can be used in MC applications that only require moderate to high output precision, i.e. 4 to 8 *bits*. If higher precision is demanded (> 9 *bits*), the use of an open-loop crossbar may yield the desired results after a sufficient number of iterations. To dramatically reduce the number of steps required for the Power Method and improve the precision of the feedback crossbar, one can use the stationary distribution estimated by the feedback crossbar as the initial condition for the open-loop crossbar. Both structures outperform their digital counterparts [88] in terms of solutions/s/W (solutions per Joule) as illustrated in Table 4.1.

	Open-Loop Crossbar**	Feedback Crossbar	Google TPU [88]
Time-to-solution	$k \times 6$ ns	552 ns	10.9 ns
Power	51 mW	28 mW	40 W
Energy	$k \times 0.31$ nJ	15.5 nJ	436 nJ
Solutions/s/W	3.2×10^9 / k	6.5×10^7	2.3×10^6
Solutions/s/W***	1.9×10^9 / k	4.4×10^7	2.3×10^6
Accuracy (% Error)	0.69 %	4.19 %	N/A
Precision (% Error)	0.13 %	2.19 %	N/A
Precision (bits)	9-bits	5-bits	8-bits

Table 4.1: Summary of Results and Performance Comparison. †
* Memristor conductance range: $100-1000\mu S$, memristor precision: 8 bits, op-amp gain: $88dB$, op-amp GBW: $1.2GHz$.
** k is the number of iterations.
*** Including ADC power consumptions ([86] for open-loop and [87] for feedback crossbars).
† The individual memristor precision does not affect "time-to-solution", "power" or "energy consumption". However, it influences the overall system precision and accuracy. Results for 100×100 random matrices are shown in Figs. 5 and 8 in [67], respectively. Taken from [67] ©2021 IEEE.

4.2 Gaussian Process

Over the last decade, GPs have become popular in the area of machine learning and data analysis for their flexibility and robustness. Often cited as a competitive alternative to neural networks because of their rich mathematical and statistical underpinnings, GPs provide a probabilistic approach to learning in kernel machines. This gives advantages with respect to the interpretation of model predictions and provides a well founded framework for learning and model selection. For a thorough introduction on GPs see [89].

Despite their attractive formulation, practical use in large-scale problems remains out of reach due to computational complexity. Existing direct computational methods for manipulations involving large-scale $n \times n$ covariance matrices require $O(n^3)$ calculations. Among others, Cholesky decomposition is one of the most used in this setting. To address this persistent challenge, several approximation techniques have been proposed and grouped in two categories: local approximations and global approximations (including low-rank approximations) [90]. In this section, we present the design and evaluation of a simulated computing platform for exact GP inference, that achieves true model parallelism using memristive crossbars.

Let us denote the input vector as $\mathbf{x} \in \mathbb{R}^D$, and the output, or target, as $\mathbf{y}$. The target $\mathbf{y}$ may either be continuous as in the regression case, or discrete as in the classification case. Given a training data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)|i = 1, \ldots, n\}$ of n observations, we wish to make predictions for new inputs $\mathbf{x}_*$ that we have not seen in the training set. To do this we must make assumptions about the characteristics of the underlying model. A GP is a generalization of the Gaussian probability distribution over functions. Whereas a probability distribution describes random variables which are scalars or vectors in case of multivariate distributions, one can loosely think of a function as a very long vector, each entry in the vector specifying the function value $f(\mathbf{x})$ at a particular input $\mathbf{x}$. The function is therefore a sample from a GP with specified mean and kernel functions as shown in Fig. 4.4(a).

Formally, a GP is a collection of random variables such that the joint distribution of every finite subset of them is multivariate gaussian. A GP is completely specified by:

- a mean function $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$
- a covariance function $Cov(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$

and we will write $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}_i, \mathbf{x}_j))$. Often, Gaussian processes are defined over time, i.e. where the index set of the random variables is time. This is not the case in our use of GPs; here the index set X is the set of possible inputs, which could be more general, e.g. $\mathbb{R}^D$ with $D > 1$. The mean function encodes the central tendency of the function, and is often assumed to be a constant or zero [89].

The covariance function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ defines the entries of a positive definite (symmetric) covariance matrix and encodes information about the shape

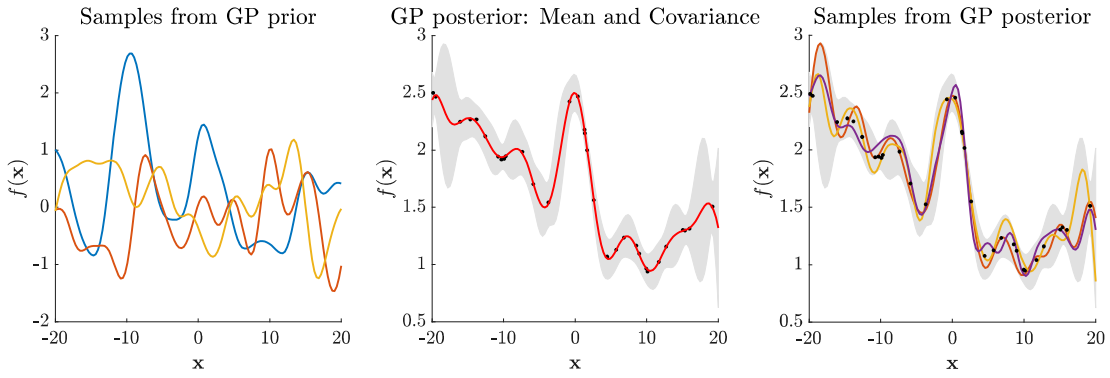


Figure 4.4: (a) Sample from the Gaussian Process prior distribution. (b) Mean (in red) and covariance (in grey shade) functions of the Gaussian Process posterior. Black dots correspond to the training observations. (c) Samples from the Gaussian Process posterior distribution.

and structure the function is expected to have. For each $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^D$, some examples are:

- Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \alpha e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2}}$, $\alpha \in \mathbb{R}$;
- Dot kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \mathbf{x}_j^T$;
- Màtern kernel, constant kernel, etc.

4.2.1 Non-linear Regression

Let $f(\mathbf{x})$ be some unknown smooth function. Suppose to have access to some noisy function values $\mathbf{y}_i = f(\mathbf{x}_i) + \epsilon_i$ with $\epsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2)$. The goal is to derive a probabilistic model $p(\mathbf{y}|\mathcal{D})$ for recovering the systematic component f from noisy data. Since f is a continuous function, thus having infinitely many dimensions, the goal of GP regression is to avoid parametric assumptions and set a distribution over functions to define the nonlinear model. Suppose we have selected a GP prior $\mathcal{GP}(m(\cdot), k(\cdot,\cdot))$ for the function f. Let $\mathcal{D}$ be the set of training points and $X_* = \{\mathbf{x}_1^*, \ldots, \mathbf{x}_m^*\}$ the set of test points. The joint distribution of the training outputs, $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)^T$, and the test outputs $\mathbf{y}_* = (\mathbf{y}_1^*, \ldots, \mathbf{y}_m^*)^T$ according to the prior is:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \mathcal{N}\left(\mu, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix} + \sigma^2 \mathbb{I}_{nm \times nm}\right) \tag{4.12}$$

where $\sigma^2 \mathbb{I}_{nm \times nm}$ is a diagonal matrix representing the contribution of the noisy data. If there are n training points and m test points then:

- $\mathbf{K}$ denotes the $n \times n$ matrix of the covariances evaluated at all pairs of training points;
- $\mathbf{K}_{**}$ denotes the $m \times m$ matrix of the covariances evaluated at all pairs of test points;
- $\mathbf{K}_{*}$ denotes the $n \times m$ matrix of the covariances evaluated at all pairs of training and test points.

To get the posterior predictive distribution of the GP evaluated at the m test points, the joint prior distribution needs to be restricted to contain only those functions which agree with the observed data points. Figure 4.4(b) shows the mean and covariance functions of the posterior predictive distribution and Fig. 4.4(c) depicts some sample from it. This corresponds to conditioning the joint Gaussian prior distribution on the observations to give

$$\mathbf{y}_*|X_*, \mathcal{D} \sim \mathcal{N}\left(\mathbf{K}_*^T\hat{\mathbf{K}}^{-1}\mathbf{y},\ \ \hat{\mathbf{K}}_{**} - \mathbf{K}_*^T\hat{\mathbf{K}}^{-1}\mathbf{K}_*\right) \tag{4.13}$$

where $\hat{\mathbf{K}} = \mathbf{K} + \sigma^2\mathbb{I}$ and $\hat{\mathbf{K}}_{**} = \mathbf{K}_{**} + \sigma^2\mathbb{I}$. The case where $\sigma = 0$ provides the noise-free posterior predictive distribution.

For the case of a single test point $\mathbf{x}_*$, the matrix $\mathbf{K}_*$ reduces to the vector $\mathbf{k}_*$ and the scalar $\hat{k}_{**}$ represents the variance at the test point. Using this compact notation, Eq. (4.13) reduces to

$$\mathbf{y}_*|\mathbf{x}_*, \mathcal{D} \sim \mathcal{N}\left(\mathbf{k}_*^T\hat{\mathbf{K}}^{-1}\mathbf{y},\ \ \hat{k}_{**} - \mathbf{k}_*^T\hat{\mathbf{K}}^{-1}\mathbf{k}_*\right). \tag{4.14}$$

4.2.2 Case study

Let us consider a regression problem with a training set X of $n = 32$ noisy points evaluated at the unknown function $f(x)$. Given the training set, the objective is to use a GP to model and predict the behavior of the function $f(x)$ on a test set X_* of $m = 128$ points depicted in Fig. 4.5. The GP prior mean is assumed to be zero. The prior's covariance is specified as a Radial Basis Functions kernel and the hyperparameters of the kernel are optimized during fitting by maximizing the log-marginal-likelihood.

In GP inference, the main expensive computation relies on finding the posterior predictive mean and covariance defined in Eq. (4.13). This operation can be performed in the circuit defined in Section 3.4.2 and shown in Fig. 3.7. Computing the predictive posterior covariance matrix can be recast into the computation $\mathbf{Z} = \mathbf{B}\mathbf{A}^{-1}\mathbf{C}$ where $\mathbf{C} \in \mathbb{R}^{n \times m}$. Each column $[\mathbf{Z}]_j\ \forall j = 1, \ldots, m$ can be computed by solving m times:

$$[\mathbf{Z}]_j = \mathbf{B}\mathbf{A}^{-1}[\mathbf{C}]_j$$

for each j-th column of matrix $\mathbf{C}$. In the context of crossbar implementations, this means that for each application, matrices $\mathbf{A}$ and $\mathbf{B}$ are fixed and the calibration step can be computed only once. The only changing factor in the system is the input vector to the linear system. Thus, the mean and covariance of the Gaussian Process can be computed in $m+1$ operations in addition to a single calibration step. This dramatically reduces the number of operations and avoids the computation of the covariance matrix's inverse.

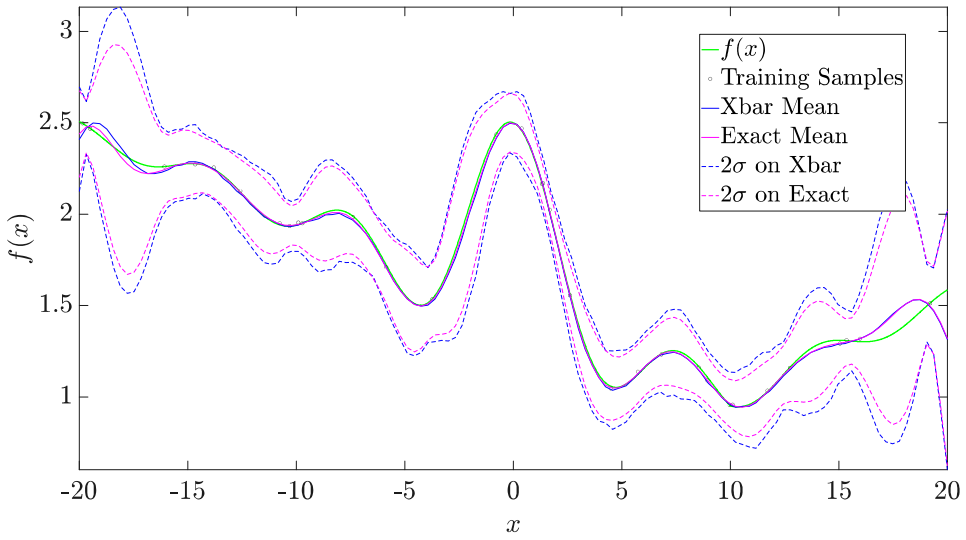


Figure 4.5: Gaussian Process used to solve a nonlinear regression problem with a training set of 32 noisy points evaluated at an unknown function $f(x)$.

To prevent parasitic resistances caused by high conductance memristors, low conductance memristors (Ag-Si) with a conductance range $[1-10]\mu S$ are used [91]. It is possible to program these devices up to 97 different conductance states which provides a precision of 6/7 bits. Stochastic noises were carefully analyzed for the case of MC applications and further results on the influence on the matrix size can be found in [67]. For this reason, this section focuses on the systematic error contribution provided by a finite gain, wire resistance and programming error.

The open-loop structure is inherently stable and does not require a very high performance op-amp. The parallel combination of the memristors in a single column and feedback resistor of the TIA sets the closed-loop gain of the open-loop crossbar and thus the bandwidth. To decrease the input resistance of the TIA seen by the crossbar, a moderate to high gain op-amp is required. To ensure high accuracy and high speed operation, the same op-amp was used for both open-loop and feedback crossbars. Since the designed op-amp has a low output resistance, buffers

between the two structures are not necessary. However, whenever high conductance memristors are chosen, then buffers should be used.

Figure 4.5 shows the mean vector of the GP posterior defined in Eq. (4.13) computed by `MatLAB` (in blue) and Cadence (in pink). The green line is the target function to infer and the black dots represent the training set. As shown in the plot, the two simulated curves perfectly overlap with a normwise relative error of 0.66%. The uncertainty region (gray shade area in figure) boundaries are defined with a distance of $\pm 2\sigma$ from the mean vector. The standard deviation vector σ was computed as the square root of the covariance matrix's diagonal defined in (4.13). Each entry $\boldsymbol{\sigma}_i^2$ of the variance vector $\boldsymbol{\sigma}^2$ corresponds to the model uncertainty at the test points $\mathbf{x}_i^*$ $\forall i = 1 \ldots, m$. Note, that the uncertainty does not fall to zero at the training points since the model takes into account the possibility of noisy data. Moreover, regions of high uncertainty correspond to interval where no training points are provided. The normwise relative error between the two simulated covariance matrices is 2.05%.

4.3 Conclusion

The solution to MC and GP problems using memristor crossbars was described. For the case of discrete MC, the open-loop crossbar can be used to implement iterative computations and find the $k-$th distribution in a sequence, while the feedback crossbar directly finds the stationary distribution problem and avoids the possible convergence issues arising in iterative algorithms. A detailed circuit analysis was performed and a variety of circuit trade-offs were discussed. The effect of memristor precision on the output accuracy and precision was demonstrated using SNR, output bit precision and normwise relative error metrics. The Mathworks dataset was used to demonstrate the effectiveness of the modeled architecture for ill-conditioned matrices.

Results were extended to the case of nonlinear regression using GPs by interconnecting the open-loop and feedback configurations into one single circuit. Simulations show that analog crossbars are promising computing platforms that can have advantages over conventional digital computing in terms of energy efficiency while maintaining a competitive output precision. This is particularly attractive from the algorithmic point of view since GPs represent an ambitious alternative to neural networks and currently large-scale problems remain out of reach due to the computational complexity.

Chapter 5

Conclusions

5.1 Summary

Learning algorithms are based on the minimization of a cost function. Most optimization problems are solved numerically by iterative methods that can be considered as the discrete-time realizations of continuous-time dynamical systems. Even though the capabilities of current AI models constantly increase, the von Neumann architecture adopted by the current computers is extremely power-hungry. The advantage of using systems that can be described by continuous-time differential equations results in:

- massively parallel operations;
- avoiding finite precision arithmetic issues;
- real-time signal processing.

One of the most popular method used in optimization and deep neural networks, namely gradient descent algorithm, makes use of the discrete-version of a gradient system. Besides its extraordinary ability, the learning process is computationally demanding and energy consuming. This motivated the exploration of local learning rules that could make use of analog implementations for accelerating the training and inference process. The content of the thesis are summarized as follows.

Chapter 1 gives a mathematical treatment of gradient descent learning algorithms for neural networks using the general framework of nonlinear dynamical systems. Whenever the network is a gradient-like (or gradient) system, the convergence to one of the possibly many minima of the objective function is guaranteed. These stationary points are stable equilibria of the dynamical system. In case the system does not admit a similar gradient representation, other assumptions can be made to let the system converge and avoid oscillatory or chaotic behaviors. For

instance, the jacobian matrix of the neural network must verify all the hypotheses defined in the Corollary 1 of the Gershgorin Circles Theorem to admit stable equilibria.

This thesis focuses on equilibrium point learning, i.e. the system's parameters update depends only on the output of the system evaluated at equilibrium. Since the system parameters are functions of the fixed points for fixed external inputs, the aim is to find the most suitable parameters so that the output approaches a desired target. Three different learning rules, Recurrent Back-Propagation, Contrastive Learning and Equilibrium Propagation are analyzed and rewritten in a unified formulation. Depending on the characteristics of the weights matrix, the learning rules are divided into symmetric and asymmetric updates. Due to the novelty of the Equilbrium Propagation algorithm, a particular attention has been paid to the generalization of this learning rule to systems that can be easily mapped into hardware platform. Proposition 1 shows that the asymmetric version of the weight change is justified from a simple geometric manipulation of the optimization problem. Theorem 5 generalizes Equilibrium Propagation from gradient systems to the case of gradient-like dynamics by considering a modified version of the energy used in [12]. This choice was dictated by the need of removing the dependence on the activation function's derivative characterizing the dynamics provided in the original formulation of the algorithm. A Hopfield-like neural network can be easily implemented by an electric circuit and has been deeply analyzed in the literature.

Chapter 2 aims at investigating dynamic neural networks in solving pattern recognition tasks and validate the theoretical results found in Chapter 1. Ideal resistive switching components are used to model synapses and sequentially update and adjust the synaptic weights. Simulations of continuous-time recurrent neural networks trained with Equilibrium Propagation showed convincing results that the two learning rules have significant capabilities in solving image reconstruction and classification tasks motivating further research in this direction. Moreover, the algorithm is even able to improve the current poor retrieval capabilities of the Kuramoto model to solve the same problem of pattern restoration. From a first analysis, the training process seems to be slower compared to the conventional neural architecture but this is probably due to the multi-modal characteristic of the associated potential function.

Despite the fascinating potential for neuromorphic applications of memristive devices, their intrinsic characteristics reveal undesirable properties when it comes to programming their conductance. These effects combined with the physical properties of the circuit components can be mitigated by either performing complex tuning protocols or offline training on conventional computers. The symmetric version of Equilibrium Propagation could be detrimentally limited due to this intrinsic stochastic asymmetry. On the other hand, the asymmetric version is potentially more suitable for VLSI implementation even though the system stability may need a careful analysis to avoid undesirable behaviors. Since the fundamental building

block of most neural network is a MVM operation, a thorough analysis of both random and systematic errors arising from a circuital implementation of memristive crossbars is therefore necessary to establish system's performance. Chapter 3 introduces the mathematical tools to compute the SNR and determine the output precision of the system in computing linear algebra operations. This method is non-application specific and allows the users to have a valid comparison between crossbars and their analog and digital counterparts for any practical use. This chapter provides the formulation of the output noise distribution produced by the sum of programming, thermal and shot noise contributions and gives an exact formula to evaluate the wire resistance influence when solving a MVM operation.

Chapter 4 evaluates the system in performing the inference step of probabilistic models that rely on linear algebra computation: Markov Chains and Gaussian Processes. For the case of discrete MC, the open-loop crossbar is used to implement iterative computations and find the $k-$th distribution in a sequence, while the feedback crossbar directly finds the stationary distribution of the chain solving an associated linear system. Results were extended to the case of nonlinear regression using GPs by interconnecting the open-loop and feedback configurations into one single circuit. Simulations show that analog crossbars are promising computing platforms that can have advantages over conventional digital computing in terms of energy efficiency while maintaining a competitive output precision.

5.2 Future Works

Further studies are needed to analyze the behavior of the physical implementation of continuous-time dynamical system exposed to noise perturbations. It is well known that conventional neural networks are prone to fail to generalize well from the training data to the test data. This is because traditional neural network models are not able to provide estimates with uncertainty information. Nowadays, it is becoming increasingly evident that organisms acting in uncertain dynamical environments often employ exact or approximate Bayesian statistical calculations in order to continuously estimate the environmental state, integrating information from multiple sensory and noisy inputs. For this reason, the investigation to link neural networks to probabilistic models has become a broad area of research [92, 93] and the exploitation of non-idealities and intrinsic stochasticity of memristive devices have started to catch neuromorphic researchers' interest [94, 95, 96, 97, 98]. Future works aim to provide computing platform that incorporates nonlinear dynamics and bayesian integration from the very beginning directly into hardware with the aim of performing complex tasks in a large variety of applications: medical treatment follow-up, human-robot interaction, etc.

Bibliography

[1] Stephen W Keckler et al. "GPUs and the future of parallel computing". In: *IEEE Micro* 31.5 (2011), pp. 7–17.

[2] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[3] Doo Seok Jeong et al. "Memristors for energy-efficient new computing paradigms". In: *Advanced Electronic Materials* 2.9 (2016), p. 1600090.

[4] Catherine D Schuman et al. "A survey of neuromorphic computing and neural networks in hardware". In: *arXiv preprint arXiv:1705.06963* (2017).

[5] Leon Chua. "Memristor-the missing circuit element". In: *IEEE Transactions on circuit theory* 18.5 (1971), pp. 507–519.

[6] D. Strukov, G. Snider, and D. Stewart R. S. Williams. "The missing memristor found". In: *Nature* 453 (2008), pp. 80–83.

[7] Sungho Kim et al. "Experimental demonstration of a second-order memristor and its ability to biorealistically implement synaptic plasticity". In: *Nano letters* 15.3 (2015), pp. 2203–2211.

[8] Zhongrui Wang et al. "Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing". In: *Nature materials* 16.1 (2017), pp. 101–108.

[9] Geoffrey W Burr et al. "Neuromorphic computing using non-volatile memory". In: *Advances in Physics: X* 2.1 (2017), pp. 89–124.

[10] Yang Zhang et al. "Brain-inspired computing with memristors: Challenges in devices, circuits, and systems". In: *Applied Physics Reviews* 7.1 (2020), p. 011308.

[11] James C.R. Whittington and Rafal Bogacz. "Theories of Error Back-Propagation in the Brain". In: *Trends in Cognitive Sciences* 23.03 (2019), pp. 235–250. DOI: 10.1016/j.tics.2018.12.005.

[12] Benjamin Scellier and Yoshua Bengio. "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation". In: *Frontiers in computational neuroscience* 11 (2017), p. 24.

[13] Gianluca Zoppo, Francesco Marrone, and Fernando Corinto. "Equilibrium propagation for memristor-based recurrent neural networks". In: *Frontiers in neuroscience* 14 (2020), p. 240.

[14] Maxence Ernoult et al. "Equilibrium propagation with continual weight updates". In: *arXiv preprint arXiv:2005.04168* (2020).

[15] Jérémie Laydevant et al. "Training Dynamical Binary Neural Networks with Equilibrium Propagation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 4640–4649.

[16] Peter O'Connor, Efstratios Gavves, and Max Welling. "Training a spiking neural network with equilibrium propagation". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 1516–1523.

[17] Armin Najarpour Foroushani et al. "Analog circuits to accelerate the relaxation process in the equilibrium propagation algorithm". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2020, pp. 1–5.

[18] Gianluca Zoppo, Francesco Marrone, and Fernando Corinto. "Local learning in Memristive Neural Networks for Pattern Reconstruction". In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2021, pp. 1–5.

[19] Jack Kendall et al. "Training end-to-end analog neural networks with equilibrium propagation". In: *arXiv preprint arXiv:2006.01981* (2020).

[20] Benjamin Scellier et al. *Generalization of Equilibrium Propagation to Vector Field Dynamics*. 2018. arXiv: 1808.04873 [cs.LG].

[21] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. Science Editions, 1962.

[22] Paul J Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), p. 436.

[24] Yoshua Bengio et al. "Towards biologically plausible deep learning". In: *arXiv preprint arXiv:1502.04156* (2015).

[25] Adam H Marblestone, Greg Wayne, and Konrad P Kording. "Toward an integration of deep learning and neuroscience". In: *Frontiers in computational neuroscience* 10 (2016), p. 94.

[26] Jianshi Tang et al. “Bridging biological and artificial neural networks with emerging neuromorphic devices: fundamentals, progress, and challenges”. In: *Advanced Materials* 31.49 (2019), p. 1902761.

[27] Simone Carlo Surace et al. “On the choice of metric in gradient-based theories of brain function”. In: *PLoS computational biology* 16.4 (2020), e1007640.

[28] Chang-Yuan Cheng, Kuang-Hui Lin, and Chih-Wen Shih. “Multistability in recurrent neural networks”. In: *SIAM Journal on Applied Mathematics* 66.4 (2006), pp. 1301–1320.

[29] Peng Liu, Jun Wang, and Zhigang Zeng. “An overview of the stability analysis of recurrent neural networks with multiple equilibria”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[30] Pierre Baldi. “Gradient descent learning algorithm overview: A general dynamical systems perspective”. In: *IEEE Transactions on neural networks* 6.1 (1995), pp. 182–195.

[31] Barak A Pearlmutter. “Gradient calculations for dynamic recurrent neural networks: A survey”. In: *IEEE Transactions on Neural networks* 6.5 (1995), pp. 1212–1228.

[32] Luis B Almeida. “A learning rule for asynchronous perceptrons with feedback in a combinatorial environment.” In: *Proceedings, 1st First International Conference on Neural Networks*. Vol. 2. IEEE. 1987, pp. 609–618.

[33] Fernando J Pineda. “Generalization of back propagation to recurrent and higher order neural networks”. In: *Neural information processing systems*. 1988, pp. 602–611.

[34] Conrad C. Galland and Geoffrey E. Hinton. “Deterministic Boltzmann Learning in Networks with Asymmetric Connectivity”. In: 1991.

[35] Pierre Baldi and Fernando Pineda. “Contrastive learning and neural oscillations”. In: *Neural Computation* 3.4 (1991), pp. 526–545.

[36] Xiaohui Xie and H Sebastian Seung. “Equivalence of backpropagation and contrastive Hebbian learning in a layered network”. In: *Neural computation* 15.2 (2003), pp. 441–454.

[37] Javier R Movellan. “Contrastive Hebbian learning in the continuous Hopfield model”. In: *Connectionist models*. Elsevier, 1991, pp. 10–17.

[38] Thomas Mesnard, Wulfram Gerstner, and Johanni Brea. “Towards deep learning with spiking neurons in energy based models with contrastive hebbian plasticity”. In: *arXiv preprint arXiv:1612.03214* (2016).

[39] Danijela Marković et al. “Physics for neuromorphic computing”. In: *Nature Reviews Physics* 2.9 (2020), pp. 499–510.

[40] Andrea Costamagna. "Equilibrium Propagation for Recurrent Neural Networks based on Resistive Switching devices: from circuit implementation to supervised machine learning". MA thesis. 2021.

[41] Gianluca Zoppo, Francesco Marrone, and Fernando Corinto. "Local Learning in Memristive Neural Networks for Pattern Reconstruction". In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2021, pp. 1–5. DOI: 10.1109/ISCAS51556.2021.9401709.

[42] Fernando Corinto, Pier Paolo Civalleri, and Leon O Chua. "A theoretical approach to memristor devices". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 5.2 (2015), pp. 123–132.

[43] Leon Chua. "Resistance switching memories are memristors". In: *Handbook of memristor networks*. Springer, 2019, pp. 197–230.

[44] Simone Raoux. "Phase change materials". In: *Annual Review of Materials Research* 39 (2009), pp. 25–48.

[45] Manuel Le Gallo and Abu Sebastian. "An overview of phase-change memory device physics". In: *Journal of Physics D: Applied Physics* 53.21 (2020), p. 213002.

[46] Ilia Valov et al. "Electrochemical metallization memories—fundamentals, applications, prospects". In: *Nanotechnology* 22.25 (2011), p. 254003.

[47] Yangyin Chen. "ReRAM: History, status, and future". In: *IEEE Transactions on Electron Devices* 67.4 (2020), pp. 1420–1433.

[48] Can Li et al. "Analogue signal and image processing with large memristor crossbars". In: *Nature Electronics* 1.1 (2018), p. 52.

[49] Miao Hu et al. "Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine". In: *Advanced Materials* 30.9 (2018), p. 1705914.

[50] M. Hu et al. "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication". In: *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2016, pp. 1–6.

[51] M. Hu et al. "Dot-product engine as computing memory to accelerate machine learning algorithms". In: *2016 17th International Symposium on Quality Electronic Design (ISQED)*. 2016, pp. 374–379.

[52] A. Shafiee et al. "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars". In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, pp. 14–26.

[53] Aayush Ankit et al. "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference". In: *CoRR* abs/1901.10351 (2019).

[54] Andrzej Cichocki and Rolf Unbehauen. "Neural networks for optimization and signal processing". In: 1993.

[55] E. J. Merced-Grafals et al. "Repeatable, accurate, and high speed multi-level programming of memristor 1T1R arrays for power efficient analog computing applications". In: *Nanotechnology* 27.36 (2016).

[56] R. Liu et al. "Investigation of Single-Bit and Multiple-Bit Upsets in Oxide RRAM-Based 1T1R and Crossbar Memory Arrays". In: *IEEE Transactions on Nuclear Science* 62.5 (2015), pp. 2294–2301. ISSN: 1558-1578. DOI: 10.1109/TNS.2015.2465164.

[57] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[58] Amos Storkey. "Increasing the capacity of a Hopfield network without sacrificing functionality". In: *International Conference on Artificial Neural Networks*. Springer. 1997, pp. 451–456.

[59] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *Science* 220.4598 (1983), pp. 671–680.

[60] Michele Bonnin. "Amplitude and phase dynamics of noisy oscillators". In: *International Journal of Circuit Theory and Applications* 45.5 (2017), pp. 636–659.

[61] Michele Bonnin. "Phase oscillator model for noisy oscillators". In: *The European Physical Journal Special Topics* 226.15 (2017), pp. 3227–3237.

[62] Michele Bonnin, Fabio Lorenzo Traversa, and Fabrizio Bonani. "Coupled Oscillator Networks for von Neumann and non-von Neumann Computing". In: *Advances in Artificial Intelligence-based Technologies*. Springer, 2022, pp. 179–207.

[63] Fan Zhang and Miao Hu. "Memristor-based Deep Convolution Neural Network: A Case Study". In: *CoRR* abs/1810.02225 (2018).

[64] Z. Sun et al. "In-Memory PageRank Accelerator With a Cross-Point Array of Resistive Memories". In: *IEEE Transactions on Electron Devices* 67.4 (2020), pp. 1466–1470.

[65] Jiyong Woo, Xiaochen Peng, and Shimeng Yu. "Design Considerations of Selector Device in Cross-Point RRAM Array for Neuromorphic Computing". In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2018, pp. 1–4. DOI: 10.1109/ISCAS.2018.8351735.

[66] B. Razavi. *Principles of data conversion system design*. IEEE Press, 1995.

[67] Gianluca Zoppo et al. "Analog Solutions of Discrete Markov Chains via Memristor Crossbars". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.12 (2021), pp. 4910–4923. DOI: 10.1109/TCSI.2021.3126477.

[68] P. Yao, H. Wu, B. Gao, et al. "Fully hardware-implemented memristor convolutional neural network". In: *Nature* 577 (2020), pp. 641–646.

[69] F. Alibart et al. "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm". In: *Nanotechnology* 23.7 (2012).

[70] George Casella and Roger L Berger. *Statistical inference*. Vol. 2. Duxbury Pacific Grove, CA, 2002.

[71] B. Razavi. *Design of Analog CMOS Integrated Circuits*. McGraw-Hill Education, 2000.

[72] G. Vasilescu. *Electronic Noise and Interfering Signals: Principles and Applications*. Signals and Communication Technology. Springer Berlin Heidelberg, 2006.

[73] Harold V Henderson and Shayle R Searle. "On deriving the inverse of a sum of matrices". In: *SIAM Review* 23.1 (1981), pp. 53–60.

[74] Alexander Dozortsev, Israel Goldshtein, and Shahar Kvatinsky. "Analysis of the row grounding technique in a memristor-based crossbar array". In: *International Journal of Circuit Theory and Applications* 46.1 (2018), pp. 122–137.

[75] Zhong Sun et al. "Solving matrix equations in one step with cross-point resistive arrays". In: *Proceedings of the National Academy of Sciences* 116.10 (2019), pp. 4123–4128. DOI: 10.1073/pnas.1815682116.

[76] Zhong Sun et al. "One-step regression and classification with cross-point resistive memory arrays". In: *Science advances* 6.5 (2020), eaay2378.

[77] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*. Vol. 37. Springer Science & Business Media, 2010.

[78] Z. Sun and R. Huang. "Time Complexity of In-Memory Matrix-Vector Multiplication". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.8 (2021), pp. 2785–2789. DOI: 10.1109/TCSII.2021.3068764.

[79] Bruno Sericola. *Markov chains: theory and applications*. John Wiley & Sons, 2013.

[80] Wai-Ki Ching and Michael K Ng. "Markov chains". In: *Models, algorithms and applications* (2006).

[81] Jeffrey J Hunter. "Generalized inverses of Markovian kernels in terms of properties of the Markov chain". In: *Linear Algebra and its Applications* 447 (2014), pp. 38–55.

[82] Carl D Meyer Jr. "The role of the group generalized inverse in the theory of finite Markov chains". In: *SIAM Review* 17.3 (1975), pp. 443–464.

[83] Amy N Langville and Carl D Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton university press, 2011.

[84] Tie-Yan Liu. *Learning to rank for information retrieval*. Springer Science & Business Media, 2011.

[85] C. Moler. *Experiments with MATLAB*. MATLAB Central File Exchange, 2020.

[86] C. Liu, M. Huang, and Y. Tu. "A 12 bit 100 MS/s SAR-Assisted Digital-Slope ADC". In: *IEEE Journal of Solid-State Circuits* 51.12 (2016), pp. 2941–2950.

[87] G. Van der Plas and B. Verbruggen. "A 150MS/s 133uW 7b ADC in 90nm digital CMOS Using a Comparator-Based Asynchronous Binary-Search sub-ADC". In: *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*. 2008, pp. 242–610.

[88] N. P. Jouppi et al. "In-datacenter performance analysis of a tensor processing unit". In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 2017, pp. 1–12.

[89] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, Jan. 2006, p. 248.

[90] Haitao Liu et al. "When Gaussian process meets big data: A review of scalable GPs". In: *IEEE transactions on neural networks and learning systems* 31.11 (2020), pp. 4405–4423.

[91] Bing Wu et al. "ReRAM Crossbar-Based Analog Computing Architecture for Naive Bayesian Engine". In: *2019 IEEE 37th International Conference on Computer Design (ICCD)*. 2019, pp. 147–155. DOI: 10.1109/ICCD46524.2019.00026.

[92] Vincent Fortuin et al. "Bayesian neural network priors revisited". In: *arXiv preprint arXiv:2102.06571* (2021).

[93] Bernhard Nessler et al. "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity". In: *PLoS computational biology* 9.4 (2013), e1003037.

[94] Thomas Dalgaty et al. "Hybrid neuromorphic circuits exploiting non-conventional properties of RRAM for massively parallel local plasticity mechanisms". In: *APL Materials* 7.8 (2019), p. 081125.

[95] Akul Malhotra et al. "Exploiting Oxide Based Resistive RAM Variability for Bayesian Neural Network Hardware Design". In: *IEEE Transactions on Nanotechnology* 19 (2020), pp. 328–331.

[96] Johannes Bill and Robert Legenstein. "A compound memristive synapse model for statistical learning through STDP in spiking neural networks". In: *Frontiers in neuroscience* 8 (2014), p. 412.

[97] MR Mahmoodi, M Prezioso, and DB Strukov. "Versatile stochastic dot product circuits based on nonvolatile memories for high performance neurocomputing and neurooptimization". In: *Nature communications* 10.1 (2019), pp. 1–10.

[98] Thomas Dalgaty et al. "In situ learning using intrinsic memristor variability via Markov chain Monte Carlo sampling". In: *Nature Electronics* 4.2 (2021), pp. 151–161.