Designing Oscillatory Neural Networks by Machine Learning

By:

Tamás Rudner-Halász

Supervisor:

Dr. György Csaba PhD

A dissertation submitted for the degree of
Doctor of Philosophy

Pázmány Péter Catholic University
Faculty of Information Technology and Bionics
Roska Tamás Doctoral School of Sciences and Technology

Budapest, 2025

Acknowledgements

I would first like to express my deepest gratitude to my supervisor, Professor György Csaba, for his invaluable guidance, encouragement, and unwavering support over the past 8 years. His insights and expertise have not only shaped this work but also played a crucial role in the researcher I have become.

I am also grateful to Professor Wolfgang Porod at the University of Notre Dame for the opportunity to collaborate with him on parts of this research. His extensive knowledge and the experiences he shared during our collaboration were inspiring.

My sincere thanks also go to the Intel team, in particular Narayan Srinivasa, Dmitri Nikonov, and Amir Khosrowshahi, for their regular guidance and discussions. They helped me shape my vision of the practical applicability of the proposed solution and provided valuable perspectives on the current state of the art.

I would also like to acknowledge the PHASTRAC consortium for our biweekly meetings and the constructive feedback I received. In particular, I am grateful to the team at TU Eindhoven, led by Professor Aida Todri-Sanial, for their close collaboration on time-dependent signal processing in electric circuits, as well as to the IBM and BMW teams for their insights and comments.

I am very grateful for the sense of community I experienced among my fellow PhD students at the university. In particular, I would like to thank Gábor Dániel Balogh, András Attila Sulyok, Bálint Siklósi, and Mihály András Vághy, who were always ready for a chat—whether about research or about life beyond it.

I owe special thanks to the Faculty of Information Technology and Bionics at Pázmány Péter Catholic University for the opportunity to take part in the doctoral program. I am especially indebted to Professor Gábor Szederkényi and Professor Árpád Csurgay for their guidance and regular feedback during the preparation of this dissertation. Likewise, I am also profoundly grateful to all my colleagues at Pázmány Péter Catholic University, whose support and friendship made these years far more enjoyable. In particular, I would like to thank Dr. Tivadarné Vida Katinka for her kindness and constant willingness to help. Her smile and encouragement were a source of strength, even when the challenges seemed overwhelming.

I am profoundly grateful to my mother, whose love, encouragement, and unwavering support have shaped the person I am today. Finally, and most importantly, I wish to express my deepest thanks to Lenke Rudner-Halász, who has stood by me throughout these years, offering her love, patience, and encouragement. Her presence repeatedly reminded me that I could not have accomplished this without her.

Abstract

Over the last few decades, Moore's law has held for digital computers, stating that the number of transistors on an integrated circuit doubles approximately every 2 years. Although physical limits have nearly been reached (e.g., source-to-drain leakage and limited gate metals), new technologies, primarily involving spintronics, tunnel junctions, and nanowiring, are promising. Additionally, other technologies aim to leverage quantum effects at physical limits, such as quantum computing.

Although these are all promising, the advancement of artificial intelligence has also led to higher training and inference costs for complex models, such as large language models. There is an inherent need today to make intelligent systems as energy-efficient as possible while ensuring they are at least as effective as current standard machine learning models.

To achieve this, a new field has emerged in recent decades: neuromorphic computing. This field is based on the idea that the brain is a large-scale computational system capable of performing extraordinary tasks and complex learning. Inherently, these systems seek to recapture the advantages of analog computing and combine them with structures that resemble the human brain. This is a paradigm shift from traditional von Neumann architecture-based computing, which might enable the creation of more energy-efficient edge AI models or faster, more energy-efficient AI computations. Neuromorphic computing is, in essence, the use of physics for computation.

This dissertation explores the feasibility of using conventional machine learning algorithms to design neuromorphic electronic circuits for solving various AI tasks. By applying proven methods to tune the parameters of a given architecture, it is possible to design electronic circuits that consume less energy during inference than traditional software-based neural networks.

In the first part of this dissertation, I introduce the building blocks of neuromorphic computing, ranging from neurobiological models and analog computing to modern software-based neural networks. I also introduce the Oscillatory Neural Network framework and the electrical-circuit component—ring oscillators—that I have been employing, along with the methods used to design them. Here, I also present brief examples of what is possible with ring oscillators that have not been extensively explored in the literature.

In the second part, I focus on various problems I solved, with varying degrees of success, using ONNs. These problems were both time-independent (static) and time-dependent (dynamic).

In the third part, I briefly discuss an approach for understanding how time-dependent signals can affect circuit-like dynamic systems and outline a method for applying such signals directly to networks without extensive preprocessing. This might lead us to use sensor signals, thereby speeding up calculations and reducing energy costs.

Finally, I conclude the dissertation and summarize the new scientific contributions.

Table of Contents

Glossary

ANN Artificial Neural Network. v

BPTT Back Propagation Through Time. v

EP Equilibrium propagation. v

HNN Hopfield Neural Network. v

ML Machine Learning. v

MLP Multilayered Perceptron. v

MSE Mean Squared Error. v

NN nearest-neighbor. v

ODE Ordinary Differential Equation. v

ONN Oscillatory Neural Network. v

SotA State-of-the-Art. v

SPICE Simulation Program with Integrated Circuit Emphasis. v

WTA Winner-Take-All. v

List of Figures

List of Tables

1 Introduction

Over the last decades, Moore's law has held for digital computers, stating that the number of transistors on an integrated circuit doubles approximately every 2 years [5], [6], [7]. Even though the physical limits have almost been reached (such as source-to-drain leakage, limited gate metals), there are new technologies mostly involving spintronics, tunnel junctions, and nanowiring, which are promising [8], [9], [10].

To address these problems, several proposals have suggested using physics in computing and building architectures that operate in fundamentally different ways than conventional computers [11]. To create such systems, computational problems must be investigated from a fundamentally different perspective. Furthermore, it is necessary to map standard computational problems to physical ones to leverage physics for computing, analogous to quantum computing [12].

Most computing paradigms that leverage physics, using some variation of energy minimization or ground-state finding, are optimization problems. Naturally, all systems tend to converge to their own minimal-energy configuration, which should be the proper solution—or one of the solutions—with appropriate settings [13].

Additionally, by leveraging physics for computation, computational devices are increasingly shifting toward biologically inspired designs, as neural systems and mammalian brains appear to use oscillatory signals to achieve a given capacity.

In this dissertation, I introduce a novel approach to analog neuromorphic computing based on oscillatory networks. In the beginning sections, I highlight two projects involving multiple research groups and companies to emphasize the motivation for reconsidering computers in ways that differ from prior practice.

In later chapters, I lay the foundation for the architectures I have created and explain what led scientists from various fields, namely neuroscience, physics, and computer science, to converge.

1.1 The shared interest of Intel, BMW, and IBM for a novel computing paradigm

In the past couple of years, I have had the opportunity to work with several companies and research groups on various projects. In the following paragraphs, I highlight the ones that are relevant to this dissertation and some of the intended applications of the companies.

1.1.1 Hierarchically Interconnected Multi-layered Oscillator Networks (HIMON)

In this project, Intel Corporation sought to design an energy-efficient, low-cost alternative to conventional computing paradigms to solve computationally hard problems on hardware.

They were motivated to either develop full-scale devices for NP-hard problems or to create an accelerator for computationally expensive algorithms, such as in silico learning.

Several research groups were encompassed. Practical research included on-chip design for NP-hard problems [14], whereas there were more theoretically focused works, such as higher-order Ising machine solutions [15].

1.1.2 Phase Change Materials for Energy Efficient Edge Computing (PHASTRAC)

In this project, the two companies—IBM and BMW—and the research group at TU Eindhoven were interested in two distinct research areas that converged. The primary application was from BMW, which sought a rapid solution to several classification tasks in a car. This is mainly because cars today have numerous sensors, both inside and outside the vehicle, and a large volume of data to process, yet limited capacity to incorporate large AI models.

On the other hand, IBM was motivated to develop a method for using phase-change materials to enhance energy efficiency in edge computing, based on relaxation VO_2 oscillators to emulate neuronal behavior and to couple them with ReRAM. [16]

From these two projects, it is evident to me that interest and need for novel, non-Boolean computing paradigms are on the rise. In the next chapter, I will introduce one of the promising ones while maintaining the dissertation's focus on oscillatory networks.

2 Introduction to Neuromorphic Computing

Neuromorphic computing is a rapidly evolving field inspired by the structure and function of the biological brain. Rather than relying on von Neumann architectures, neuromorphic systems emulate neurons and synapses in hardware, often employing event-driven behavior to achieve remarkable energy efficiency and parallelism [17], [18]. This shift holds promise for low-power edge computing and sensory processing.

Fundamentally, neuromorphic engineering draws on both "bottom-up" approaches—replicating the biophysics of neurons in analog or mixed-signal circuits—and "top-down" approaches—optimizing hardware for AI workloads [19]. Manchester's SpiNNaker is one of the early large-scale digital neuromorphic platforms, using a massively parallel network of ARM cores to simulate spiking neural networks in real time [20].

The previously mentioned companies—Intel and IBM—have also developed their own neuromorphic computing architectures; IBM has TrueNorth [21], and Intel has Loihi [22]. In 2023, Zhejiang University and Alibaba developed Darwin, a recent neuromorphic chip [23].

In the materials and device domain, recent architectures have explored novel substrates, including perovskite composites, nanowires, and organic semiconductors. Perovskite-based devices enhance ion migration and energy alignment, showing strong promise for memristive synapses [24]. Similarly, nanowire-based neuromorphic devices offer high integration density, speed, and low power consumption via advanced materials systems [25]. Organic field-effect transistors (OFETs) further enable flexible, bio-inspired synaptic dynamics suitable for neuromorphic sensory systems [26].

Together, architecture, materials, and device innovations are pushing neuromorphic computing from theoretical promise to practical, energy-efficient, adaptive hardware systems.

In the next couple of sections, I will introduce the pillars of neuromorphic computing and their most important aspects. The need for neuromorphic computing arises because current AI solutions are becoming increasingly infeasible due to high power consumption. The primary goal of neuromorphic computing is to emulate brain-like processes to enable low-power, scalable, energy-efficient AI solutions.

2.1 Brain-inspiration and Brief Neuroscientific Background

The primary goal of neuromorphic computing is to develop architectures that exhibit behavior similar to that of the human brain, which is a large-scale computational system capable of extraordinary tasks and complex learning.

In the last century, research into the brain has been a central focus of neuroscience. From developing increasingly accurate mathematical-physical representations of neurons by simulating them in software environments to creating complete intelligences in the form of artificial

intelligence, researchers have invested enormous effort in this field.

Artificial intelligence nowadays receives significant traction. The performance of models capable of mimicking human intelligence has increased over the last couple of decades, but so has the cost of training and maintaining them. To address this problem, it is paramount to first understand the principles on which artificial intelligence is built and then to circumvent its shortcomings by introducing novel approaches to problems that have already been solved.

To replicate brain function, it was essential to map out the fundamentals, namely, neuronal information processing and propagation. The most complex and widespread model for the modelling of neuron activation is the Hodgkin-Huxley model [27]. Before moving on to simpler and more relevant—particularly for neuromorphic computing—models, I would like to provide a brief overview of the Hodgkin-Huxley model.

2.1.1 The Hodgkin-Huxley Model: Foundations, Equations, and Significance

The Hodgkin-Huxley (HH) model, developed in 1952 through a landmark series of voltage-clamp studies on the squid giant axon, provided the first quantitative description of how ionic conductances generate the action potential [27], [28], [29], [30]. Its success lies in a principled decomposition of the membrane current into capacitive and ionic components, with the latter governed by voltage- and time-dependent "gating" variables. Beyond explaining spike initiation and shape, the HH framework established a canonical conductance-based modeling paradigm that continues to underpin modern cellular electrophysiology and computational neuroscience [31].

At the core of the model is an equivalent electrical circuit for a small patch of membrane: a capacitor (C_m) in parallel with ion-specific conductances (sodium, potassium, and a leak), each with its own reversal potential (see Figure 2.1). When an external current I_{ext} is injected, part of it charges the membrane capacitance, and the other part flows through the voltage-gated ion channels. Hodgkin and Huxley showed that the sodium conductance activates rapidly and inactivates more slowly. In contrast, the potassium conductance activates more slowly and does not inactivate on the timescale of a spike. These qualitative features are captured quantitatively by channel "gates," which open and close stochastically at the molecular level but are modeled deterministically at the macroscopic level by gating variables with first-order kinetics.

Let V denote the transmembrane potential—inside minus outside. The current balance is

$$C_m \frac{dV}{dt} \;=\; I_{\text{ext}} \;-\; \bar{g}_{\text{Na}}\, m^3 h\,(V - E_{\text{Na}}) \;-\; \bar{g}_{\text{K}}\, n^4\,(V - E_{\text{K}}) \;-\; \bar{g}_{\text{L}}\,(V - E_{\text{L}}),$$

where $\bar{g}_{\text{Na}}, \bar{g}_{\text{K}}, \bar{g}_{\text{L}}$ are maximal conductances and $E_{\text{Na}}, E_{\text{K}}, E_{\text{L}}$ the corresponding reversal potentials. The dimensionless gating variables m and h describe sodium activation and inactivation, respectively, and n describes potassium activation. Their dynamics are first-order with voltage-dependent transition rates:

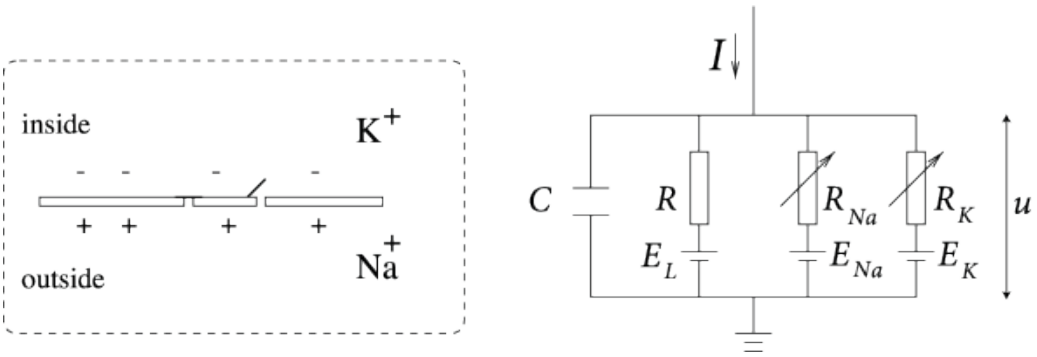


Figure 2.1: **Hodgkin-Huxley equivalent circuit.** The membrane is modeled as a capacitor C_m in parallel with voltage-gated sodium and potassium conductances and a leak pathway, each driving current according to $g_x(V - E_x)$. Schematic adapted from the pedagogical presentation in Neuronal Dynamics (EPFL)—download a version of the figure from: `https://neuronaldynamics.epfl.ch/online/Ch2.S2.html`.

$$\begin{aligned}
\frac{dm}{dt} &= \alpha_m(V)\,[1-m] - \beta_m(V)\,m \;=\; \frac{m_\infty(V)-m}{\tau_m(V)},\\
\frac{dh}{dt} &= \alpha_h(V)\,[1-h] - \beta_h(V)\,h \;=\; \frac{h_\infty(V)-h}{\tau_h(V)},\\
\frac{dn}{dt} &= \alpha_n(V)\,[1-n] - \beta_n(V)\,n \;=\; \frac{n_\infty(V)-n}{\tau_n(V)}.
\end{aligned}$$

In their original parameterization (with V in mV referenced to the resting potential), Hodgkin and Huxley used

$$\alpha_m(V) = \frac{0.1\,(25-V)}{\exp(\frac{25-V}{10})-1}, \qquad \beta_m(V) = 4\,\exp(-\tfrac{V}{18}), \tag{2.1}$$

$$\alpha_h(V) = 0.07\,\exp(-\tfrac{V}{20}), \qquad \beta_h(V) = \frac{1}{\exp(\frac{30-V}{10})+1}, \tag{2.2}$$

$$\alpha_n(V) = \frac{0.01\,(10-V)}{\exp(\frac{10-V}{10})-1}, \qquad \beta_n(V) = 0.125\,\exp(-\tfrac{V}{80}). \tag{2.3}$$

These rates imply steady-state activation/inactivation curves $x_\infty(V) = \alpha_x(V)/[\alpha_x(V) + \beta_x(V)]$ and time constants $\tau_x(V) = 1/[\alpha_x(V) + \beta_x(V)]$ for $x \in \{m, h, n\}$. Figure 2.2 shows representative steady states $m_\infty(V)$, $h_\infty(V)$, and $n_\infty(V)$: sodium activation (m) turns on steeply with depolarization; sodium inactivation (h) turns off with depolarization; and potassium activation (n) turns on over a broader voltage range.

The form m^3h for sodium reflects three independent activation gates that must all be open, and one inactivation gate that must be open, for the channel to conduct; n^4 for potassium posits four activation gates. Although this combinatorial picture predates the molecular characterization of ion channels, the gating exponents 3 and 4 proved sufficient to capture the macroscopic kinetics necessary to explain the action potential's rapid upstroke (Na^+ activation followed by inactivation) and delayed repolarization (K^+ activation) [27], [29], [30], [31].

Equations (2.1.1)–(2.3) describe a space-clamped membrane patch. Propagation along an

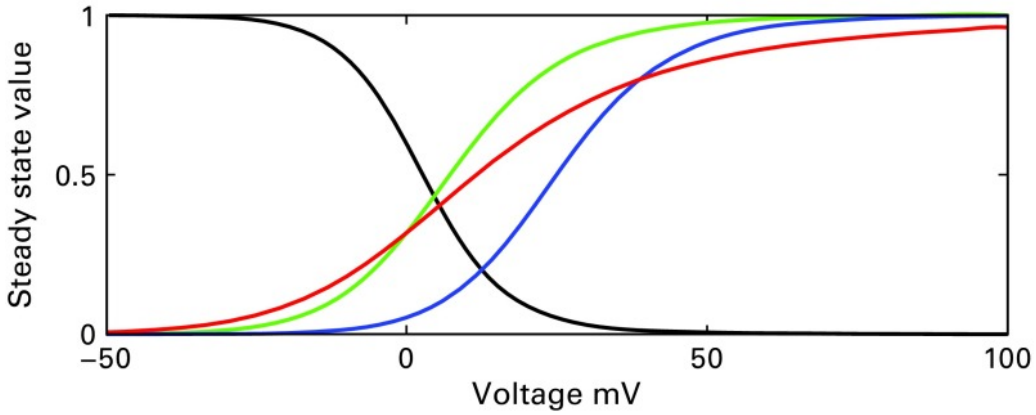


Figure 2.2: **Steady-state gating variables in the HH model.** Typical $m_\infty(V)$ (Na^+ activation), $h_\infty(V)$ (Na^+ inactivation), and $n_\infty(V)$ (K^+ activation) curves. Image source (MIT Press/NCBI Bookshelf): `https://www.ncbi.nlm.nih.gov/books/NBK544602/figure/fig_011_02/`.

unmyelinated axon adds an axial (cable) current term, yielding

$$C_m \frac{\partial V}{\partial t} = \frac{a}{2R_i} \frac{\partial^2 V}{\partial x^2} - \bar{g}_{\mathrm{Na}}\, m^3 h\, (V - E_{\mathrm{Na}}) - \bar{g}_{\mathrm{K}}\, n^4\, (V - E_{\mathrm{K}}) - \bar{g}_{\mathrm{L}}\, (V - E_{\mathrm{L}}) + I_{\mathrm{ext}}(x,t)\,,$$

where a is the axonal radius and R_i the intracellular resistivity. With physiologically measured parameters, the model reproduces finite-velocity action potential propagation and the classical refractory period. Temperature effects enter through Q_{10}-type scaling of the rate functions α_x, β_x and conductances, which Hodgkin and Huxley quantified directly.

The original HH parameter set (at $\sim 6.3°C$ for squid axon) is often recast for mammalian neurons by adjusting $(\bar{g}_x, E_x)$ and rescaling rates to higher temperatures. Because m is the fastest gate (typical $\tau_m \sim 0.4$ ms near rest) while h and n are slower ($\sim$5–8 ms), explicit numerical solvers benefit from adaptive time stepping or semi-implicit discretization to maintain stability. Modern derivations frequently rewrite the gate kinetics in the (x_∞, τ_x) form for clarity and to facilitate parameter fitting against voltage-clamp data.

The HH model accomplished three enduring feats. First, it established that the action potential is an emergent phenomenon arising from the interaction of ionic conductances, rather than a property of a single channel species. Second, it created a quantitative template for conductance-based modeling that generalizes to diverse excitable cells and synaptic mechanisms. Third, by demonstrating how macroscopic currents arise from voltage-dependent gating, it presaged later molecular discoveries of ion channel subunits and their kinetics. Variants and reparameterization continue to be used to capture neuron- and species-specific electrophysiology, optimize channel densities, and study dynamical phenomena such as excitability types, bursting, and bifurcations, all within the HH formal framework [31].

Building upon the initial success of a biologically inspired mathematical model of neurons, researchers laid the groundwork for computational neural networks to imitate human learning and solve complex tasks using artificial neurons, thereby giving rise to the field of neural networks.

Additionally, it is worth noting that the HH model is highly relevant to ONNs, as it can produce intrinsic oscillators—periodic spiking—depending on its parameter values. These oscillations

are essentially limit cycles in the nonlinear dynamic system. The HH model can also exhibit synchronization, as do ONNs. [32], [33]. In this way, the Hodgkin-Huxley model can serve as a biological basis for ONNs.

2.2 Brain-on-a-software: Neural Networks

The history of neural networks spans more than seven decades and reflects a rich interplay between neuroscience, mathematics, and computer science. From early models of cognition to the modern deep learning revolution, neural networks have experienced cycles of enthusiasm, skepticism, and resurgence. This section offers a concise yet comprehensive account of their development, highlighting landmark contributions across eras.

The conceptual origins of neural networks can be traced to McCulloch and Pitts, who, in 1943, proposed a simplified model of the neuron as a binary threshold unit [34]. Their framework established the idea that networks of such artificial neurons could compute logical functions, foreshadowing the digital computer age. Hebb's influential theory of learning, often paraphrased as "cells that fire together, wire together", provided a biologically inspired rule for synaptic modification [35], shaping subsequent models of learning.

In 1958, Rosenblatt introduced the perceptron [36], which was one of the first trainable neural architectures. The perceptron attracted significant attention for its ability to classify input patterns using supervised learning. Minsky and Papert's 1969 book [37], however, exposed the perceptron's limitations, most notably its inability to learn non-linearly separable functions such as XOR. This critique, combined with limited computational resources, ushered in the first "AI winter", during which neural network research saw diminished support.

The late 1970s and early 1980s saw renewed optimism, driven by advances in multilayer networks. Werbos first described the backpropagation algorithm in his 1974 thesis [38], but Rumelhart, Hinton, and Williams popularized it in 1986 [39]. They demonstrated that error backpropagation could efficiently train multi-layer perceptrons. This innovation overcame the XOR limitation by enabling hidden-layer representations.

Parallel contributions included Hopfield networks [40], which provided an energy-based model of associative memory, and Kohonen's self-organizing maps [41], which introduced unsupervised competitive learning for feature mapping. These advances, coupled with advances in hardware, helped neural networks gain traction in pattern recognition, speech recognition, and control systems. LeCun's application of convolutional neural networks (CNNs) for handwritten digit recognition in the late 1980s [42] provided an early glimpse of the power of specialized architectures.

Despite these successes, neural networks again faced skepticism in the 1990s due to difficulties in training deep models, vanishing gradients, and competition from support vector machines. The turning point occurred in the mid-2000s, marked by several breakthroughs. Hinton and colleagues introduced deep belief networks and layer-wise pretraining [43], providing a path toward effective deep learning. Bengio and others further explored deep architectures, highlighting the potential for hierarchical feature learning [44].

The watershed moment arrived in 2012, when Krizhevsky, Sutskever, and Hinton's AlexNet [45] achieved a dramatic improvement in the ImageNet competition using deep CNNs trained on

GPUs. This marked the beginning of deep learning as a mainstream paradigm, sparking rapid adoption across vision, speech, and natural language processing.

Following AlexNet, architectures evolved rapidly. Simonyan and Zisserman's VGGNet emphasized deeper but simpler models [46], while He et al.'s ResNet [47] introduced residual connections that enabled training of networks with hundreds of layers. In parallel, recurrent neural networks and long short-term memory (LSTM) units [48] enabled breakthroughs in sequence modeling, from speech recognition to translation.

Transformers, introduced by Vaswani et al. in 2017 [49], revolutionized natural language processing by replacing recurrence with attention mechanisms. Transformer-based models such as BERT [50] and GPT have since set new state-of-the-art performance across multiple domains, signaling a paradigm shift in how neural networks process sequential data. Transformers are not technically "brain-on-a-chip" because they differ from the brain-like architectures of conventional neural networks. Transformers have connections and structures that humans lack.

From the early perceptron to today's billion-parameter transformers, neural networks have undergone a dramatic evolution. Each phase built on conceptual advances and technological progress, often rediscovering the relevance of ideas that had once been sidelined. Today, neural networks are not only central to artificial intelligence but also to neuroscience, robotics, and cognitive science, reflecting the interdisciplinary vision of their pioneers.

2.3 Brief History on Analog Computing and Physics of Computing

The history of analog computing spans nearly a century, from early mechanical devices to today's neuromorphic and in-memory accelerators. One of the earliest landmarks was Vannevar Bush's description of the Differential Analyzer, a mechanical computer that physically solved ordinary differential equations by interconnecting mechanical integrators [51]. Bush's device not only enabled real-world problem-solving but also laid down the architectural principle of continuous-time analog computation. Building on this foundation, Claude Shannon provided the first rigorous theoretical analysis of such machines in his work on the mathematical theory of the differential analyzer [52]—Shannon's contribution bridged engineering practice with mathematical abstraction, analyzing approximation and error propagation in continuous computation.

Decades later, analog computing was revitalized by insights from neuroscience and physics. Hopfield's seminal work introduced neural networks as physical systems with emergent computational abilities, showing how energy landscapes and collective dynamics could realize content-addressable memory [53]. This interpretation of computation as physical relaxation strongly influenced the development of neuromorphic and analog optimization systems. In parallel, Mead's manifesto on neuromorphic electronic systems proposed that transistor physics itself could serve as the substrate for efficient computation, arguing that analog VLSI circuits operating in biologically inspired regimes could emulate neural processing with extreme energy efficiency [54]. These works shifted the field from mechanical computation to silicon-based analog devices. Carver Mead has been a pioneer in connecting physics and computation. He recognized that many biological computations are inherently analog. Using MOS transistors in the subthreshold regime, he designed analog VLSI circuits that emulate neural functions, including integration, adaptation, and filtering. This work culminated in his influential book [55]. Building on this, he began

developing neuromorphic engineering. In this framework, the hardware is designed to reflect not only neural systems' algorithms but also their physical properties, such as parallelism, low power consumption, graded signaling, and adaptability. He emphasized that computing is inherently a physical process and must be treated accordingly. Energy dissipation, noise, thermal fluctuations, and scalability all derive from physical constraints. He stated that computing architectures should leverage the physical properties of devices rather than being constrained to abstract digital frameworks. Mead's ideas have shaped decades of research in neuromorphic computing. His retrospective accounts highlight both the original motivations and the continued relevance of analog and physics-informed approaches to computation [56].

Theoretical investigations into analog networks culminated in Siegelmann and Sontag's study on the computational power of neural nets, which proved that specific continuous neural architectures can, under idealized assumptions, compute beyond the Turing model [57]. While practical implementations are constrained by precision and noise, this work established the theoretical limit for analog devices and framed the analog–digital distinction in terms of computational complexity.

The 2010s witnessed a resurgence of analog computing, this time motivated by machine learning and signal processing. Appellant et al. introduced delay-based reservoir computing, showing that a single nonlinear dynamical node with delayed feedback could emulate a large recurrent neural network [58]. This minimalist yet powerful approach has inspired a range of physical implementations. Shortly after, Larger et al. demonstrated a photonic reservoir computer, exploiting optical delay systems to achieve extremely high-speed analog information processing [59]. Together, these works anchored photonics as a serious substrate for analog neural computation.

Optimization emerged as another driving application of analog systems. Wang et al. proposed the coherent Ising machine, in which networks of degenerate optical parametric oscillators collectively relaxed to solutions of combinatorial optimization problems [60]. This analog approach to NP-hard problems directly linked Hopfield's vision to modern photonic hardware. Meanwhile, in the domain of memory-based accelerators, Hu et al. presented a memristor crossbar engine capable of performing in-memory matrix–vector multiplications for neural inference, a critical milestone for analog in-memory computing [61].

Most recently, significant advances have focused on improving the practicality and precision of analog accelerators. Song et al. demonstrated a method for programming memristor arrays with arbitrarily high precision, significantly reducing the accuracy gap between analog and digital computation and addressing one of the central challenges of in-memory computing [62]. Going even further, Giuffrida and colleagues showed that phase-change memory devices can support on-chip meta-learning, illustrating that not only inference but also adaptive training loops can be embedded into analog substrates [63]. This represents a decisive step toward fully trainable, resilient analog AI hardware.

From Bush's mechanical integrators to PCM-based meta-learning systems. These works trace a trajectory from solving equations mechanically to exploiting device physics for neural computation to today's efforts toward analog accelerators for AI. They all highlight a continuity of vision that computation is not restricted to symbolic digital models but can emerge from the physical

dynamics of matter itself.

Despite the promise that analog computing held for some time, it consistently fell short for various reasons. The main culprits of the unsuccessful analog hardware have been noise sensitivity, the complexity of scalability and programmability, which mean that digital devices are mostly universal, whereas analog computers were specific to tasks. Additionally, due to industrial factors, as the industry has evolved around digital design, the market has been demanding general-purpose devices.

In this dissertation, however, I explore Oscillatory Neural Networks, which are promising for addressing this shortcoming of analog computing. ONNs harness the same advantages of analog computing, such as parallelism, synchronization, and energy efficiency. However, ONNs are not fighting against noise, variability, or nonlinearity; they actively exploit them. Oscillators naturally synchronize and desynchronize.

Another advantage of ONNs is that they can be used as a hybrid architecture by combining ONNs with digital processors, as I will show in later chapters.

To highlight the integration of analog computing and neural networks, the following section presents a summary of cellular neural/nonlinear networks.

2.4 Cellular Neural Networks: Combining Neural Networks and Analog Computing

Cellular Neural/Nonlinear Networks (CNNs), introduced by Leon Chua and Tamás Roska in the late 1980s and early 1990s, are a class of spatially distributed dynamical systems designed for real-time parallel computation, effectively combining Neural Networks and Analog Computing. A CNN consists of a regular grid of identical dynamical units called cells, each of which is locally connected to its nearest neighbors through feedback and feedforward templates. The interactions are generally nonlinear, enabling CNNs to perform complex signal-processing tasks directly in hardware [64], [65], [66].

A nonlinear differential equation typically describes each cell, and it looks as follows:

$$C\frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + \sum_{(k,l)\in N(i,j)} A_{kl}y_{kl}(t) + \sum_{(k,l)\in N(i,j)} B_{kl}u_{kl} + I.$$

Here x_{ij} is the state of the cell at position (i, j), y_{ij} is its output (usually a piecewise-linear function of x_{ij}), u_{ij} is the input, A_{kl} and B_{kl} are the feedback and control templates defining the local connectivity, C is the cell capacitance, and I is a bias term. The neighborhood $N(i, j)$ typically includes the cell itself and its adjacent neighbors.

CNNs are well-suited to image processing and pattern recognition, in which each pixel in an image can be mapped to a cell. Using carefully designed templates, CNNs can perform tasks such as edge detection, noise removal, motion detection, and template matching directly in hardware at very high speed. The analog VLSI implementation proposed by Roska and Chua demonstrated that CNNs could exploit inherent parallelism, enabling processing rates far exceeding conventional digital approaches of the time [66], [67].

A key innovation of CNNs is their integration of local interconnections from cellular automata

with continuous-time nonlinear dynamics, thereby bridging discrete cellular systems and continuous neural networks. This enables a rich repertoire of spatio-temporal dynamics, including pattern formation, wave propagation, and synchronization phenomena. Roska and Chua's work laid the foundation for both theoretical analyses of CNN dynamics and practical hardware implementations, including analog VLSI CNN chips [65], [66].

Recent research on CNNs explores memristors as coupling elements. [68], [69] This also introduces nonlinearities on top of the inherent nonlinearities of CNNs, and, because of the low energy cost of memristors, CNNs might be feasible for real-time, low-power-consumption edge AI devices for image processing [70], [71]. Another fascinating field for CNN is leveraging its potential chaotic behavior for secure communications [72], [73].

3 Oscillatory Computing Models

In the previous chapter, I introduced the three building blocks of oscillatory computing: modelling neuronal functions, developing brain-inspired architectures—namely, neural networks—and analog computing.

Oscillatory computing is a computing paradigm that leverages the natural dynamics of oscillatory systems to perform computation [74], [75]. Instead of relying solely on conventional digital logic, information is encoded in the phase, frequency, or amplitude of oscillations, and computation arises from the interactions between coupled oscillators [76], [77]. Such systems are inspired by biological neural networks, chemical oscillators, and electronic circuits that naturally exhibit limit-cycle behavior.

A canonical approach uses networks of coupled nonlinear oscillators to implement logical operations, pattern recognition, or optimization tasks. The relative phase of oscillators can represent binary or analog information, and the synchronization phenomenon enables computation without requiring explicit sequential control [78]. For instance, in pattern recognition, clusters of synchronized oscillators can represent different classes, and desynchronization events encode transitions between states.

Oscillatory computing has been explored in a variety of physical substrates, including nano-oscillators, memristive circuits, and photonic devices. Its potential advantages include inherent parallelism, energy efficiency due to low-power oscillatory interactions, and robustness to noise [79]. These properties make oscillatory architectures promising for neuromorphic and unconventional computing applications where standard digital architectures may be less effective.

While still an emerging field, experimental demonstrations have shown that oscillator networks can solve combinatorial optimization problems, perform associative memory tasks, and even emulate certain aspects of neural computation. The interplay between dynamical systems theory and hardware implementation remains a central focus of research, as designing networks with controllable synchronization patterns is key to scaling oscillatory computing to practical applications.

3.1 Mathematical models

In this section, I introduce two widely used mathematical models of coupled oscillators. The first—Kuramoto—model is used to study coupled-oscillator synchronization, whereas the second—FitzHugh-Nagumo—model is used to study simplified oscillatory neuronal activity.

3.1.1 Kuramoto model

The Kuramoto model, introduced by Yoshiki Kuramoto in 1975, is a fundamental mathematical framework for studying the synchronization of large populations of coupled oscillators with

heterogeneous natural frequencies [80]. Each oscillator is represented by its phase θ_i, evolving according to the differential equation:

$$\frac{d\theta_i}{dt} = \omega_i + \frac{K}{N}\sum_{j=1}^{N}\sin(\theta_j - \theta_i),$$

where ω_i is the intrinsic frequency of oscillator i, K is the global coupling strength, and N is the number of oscillators. As the coupling K increases beyond a critical threshold K_c, the oscillators transition from incoherence to collective synchronization.

The original Kuramoto model has been extended in several directions:

- **Complex Networks**: Real-world interactions are rarely all-to-all. Incorporating complex topologies, such as small-world or scale-free networks, significantly affects the synchronization threshold and robustness [81], [82].
- **Inertia and Noise**: Adding inertia and stochastic terms allows the study of systems with delayed responses and random fluctuations, mimicking many physical and biological oscillators.
- **Higher-Order Interactions**: Beyond pairwise coupling, higher-order interactions involving triplets or larger groups capture richer synchronization patterns in complex networks [83].

The Kuramoto model has been widely applied across multiple domains:

- **Neuroscience**: Modeling synchronization of neural oscillations to understand cognitive processes and neurological disorders.
- **Physics**: Studying synchronization in coupled lasers, Josephson junction arrays, and other physical oscillators.
- **Engineering**: Designing robust synchronized systems in power grids, communication networks, and distributed computing.
- **Earth Science and Ecology**: Exploring collective dynamics in climate networks and interconnected ecological systems [83].

The Kuramoto model remains a cornerstone in the study of synchronization phenomena, bridging dynamical systems theory and practical applications in both natural and engineered systems [84].

Its only downside is that it is a purely mathematical model, which limits its applicability to real-world systems. Nevertheless, Kuramoto oscillators can be used for proof-of-concept in synchronization-based computing, and I later used them in Section 6.3 to demonstrate the practical implementability of frequency-synchronizing networks.

3.1.2 The FitzHugh–Nagumo Model: A Minimal Excitable System

The FitzHugh–Nagumo (FHN) model, formulated by FitzHugh (1961) and formalized electrically by Nagumo et al. (1962), is a classic two-dimensional reduction of the Hodgkin–Huxley (HH)

model that retains the essential excitability and spike-generation mechanism while drastically simplifying the mathematics [85], [86], [87], [88]. The FHN model captures the fast activation and slower recovery dynamics through a pair of coupled ordinary differential equations, enabling intuitive phase-plane analysis and analytical tractability.

The FitzHugh-Nagumo model is a "minimal excitable system" because it captures the essential features of neuronal excitability in the simplest mathematical form, thereby reducing the complexity of the Hodgkin-Huxley model. To be a "minimal excitable system," a system must exhibit a resting state, threshold behavior, and action potential generation. It should also have refractory periods, so the system shall not fire again immediately. It should also use the fewest parameters and variables. The FitzHugh-Nagumo model has only two variables—one fast and one slow—along with a nonlinear term for thresholding and spike-like behavior and linear terms that give rise to the refractory period and resting states. Moreover, because it has only two variables, it can be studied in two-dimensional rather than four-dimensional space, as in the Hodgkin-Huxley model.

In its canonical nondimensional form, the model reads:

$$\begin{aligned} \dot{v} &= v - \frac{v^3}{3} - w + I_{\text{ext}}, \\ \tau\, \dot{w} &= v + a - b\, w, \end{aligned}$$

where:

- v mimics the membrane potential (fast variable),
- w is a slower recovery variable (aggregating effects like sodium inactivation or potassium activation),
- I_{ext} is an external stimulus,
- a, b, τ are parameters governing thresholds, feedback strength, and timescale separation.

The cubic v-nullcline (from $\dot{v} = 0$) and linear w-nullcline (from $\dot{w} = 0$) intersect at the resting fixed point. When I_{ext} crosses a threshold, trajectories depart the fixed point, traverse a large excursion (a "spike"), and return via the slower recovery variable dynamics.

The FHN model arises from adiabatically eliminating the fast HH gating variables (e.g., m, h) and retaining only one slow recovery variable, $w \sim n$ [88]. Despite this simplification, it reproduces qualitative HH features, such as excitability classes and refractory responses, and is defined within the same class of excitable systems.

Moreover, the FHN model generalizes the van der Pol oscillator by setting $a = b = 0$, which recovers the van der Pol form — a self-sustained relaxation oscillator. In contrast, the full FHN requires an external drive I_{ext} to generate spikes, emphasizing its role as a model of stimulus-dependent excitability rather than autonomous oscillation [86].

Thanks to its minimal structure, the FHN model serves as a versatile prototype in nonlinear dynamics, excitable media, and pattern formation [89]. It enables the analysis of traveling pulses, wave initiation, and bifurcations in a way that is virtually intractable in full HH dynamics. While

phenomenological, the FHN model continues to underpin theoretical neuroscience, offering a bridge between biophysical realism and mathematical simplicity.

Although FHN is a promising candidate for an oscillatory, brain-inspired computer, its status as a mathematical model limits the ability to build physical devices that exhibit the same behavior.

3.2 Circuit-level Models

Unlike mathematical models, circuit-level models are used to build physical devices that replicate their behavior. Oscillatory Neural Networks have been built using various apparatus, but in this section, I will focus on two dominant ones: VO_2 and ring oscillators. Both have simple circuit descriptions, so their simulation and algorithmic design can be performed in silico.

The only downside of VO_2 oscillators is that their fabrication alongside tunable couplings is currently unfeasible.

3.2.1 VO_2 Oscillators

Vanadium dioxide (VO_2) is a correlated electron material that undergoes a reversible insulator-to-metal transition, or IMT, near 68°C. This transition is accompanied by drastic changes in electrical resistivity and optical properties, enabling VO_2 to act as a natural switchable material. Because of its sharp, hysteretic transition, VO_2 has been exploited to design compact, energy-efficient oscillators that have become key components of unconventional and neuromorphic computing architectures.

The first demonstrations of VO_2-based oscillators date back to the 1960s and 1970s, when the IMT was harnessed in simple relaxation oscillator circuits. However, renewed interest in VO_2 has emerged over the past two decades, driven by advances in thin-film growth, device engineering, and the rise of neuromorphic computing as a promising paradigm. The intrinsic nonlinearity and abrupt switching of VO_2 enable it to emulate neuronal spiking behavior, making it a strong candidate for compact hardware neurons.

Recent research has demonstrated VO_2 oscillators in diverse forms, including two-terminal devices, planar nano-oscillators, and integrated arrays. Shukla et al. (2014) reported scalable two-terminal VO_2 oscillators for neuromorphic architectures [90]. Subsequent works, such as Zhang et al. (2017), explored coupled VO_2 oscillators for implementing associative memory and pattern recognition [91]. More recent studies have examined the synchronization properties of VO_2 oscillator networks and their utility in solving graph coloring and combinatorial optimization tasks [92].

Oscillatory neural networks (ONNs) leverage phase and frequency synchronization between oscillators to encode and process information. VO_2, due to its high nonlinearity and tunability, has emerged as an excellent candidate for building such networks. Shukla et al. (2014) and others have shown that coupled VO_2 oscillators can implement phase-based computing, in which synchronization encodes solutions to computational problems. Munoz-Martin et al. (2019) further demonstrated graph-based problem-solving with coupled VO_2 oscillators [92]. These advances point toward energy-efficient, compact ONNs based on correlated materials.

In neuromorphic computing, VO_2 oscillators serve as artificial neurons. Their spiking behavior,

tunable frequency, and strong nonlinearity allow efficient emulation of biological dynamics. They have been integrated into hybrid CMOS-VO_2 circuits to implement associative memory, reservoir computing, and optimization algorithms [91], [93]. VO_2 oscillators are excellent for scalable hardware implementations due to their nanoscale footprint and compatibility with existing semiconductor fabrication processes.

Also, it has been shown that it is possible to implement FitzHugh-Nagumo (FHN) oscillators with VO_2 devices [94].

A VO_2 oscillator's circuit diagram is shown in Figure 3.1. Such oscillators can be coupled through their V_{out} nodes using resistors. Depending on the value of the coupling, the positive and negative values yield in-phase and out-of-phase pullings, respectively.

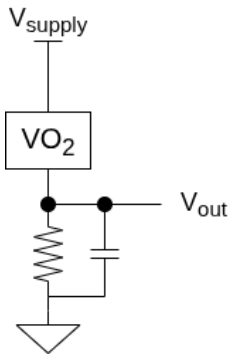


Figure 3.1: **Circuit diagram of a VO_2 oscillator.** Such VO_2 oscillators can be coupled through the V_{out} nodes. Using resistors, coupled VO_2 oscillators can be coupled either positively—in-phase pulling—or negatively—anti-phase pulling.

This device exhibits hysteretic behavior and can be used to create coupled oscillators [95].

In summary, VO_2 oscillators provide a versatile, compact, and energy-efficient platform for neuromorphic computing. From their early demonstrations to modern applications in ONNs, VO_2 devices have become a cornerstone of oscillatory computing architectures.

3.2.2 Ring-oscillator model

Ring oscillators are among the simplest of oscillators. These devices consist only of (an odd number of) inverters, capacitors, and resistances, see in Figure 3.2. From here, the number of inverters in the ring oscillators used in this dissertation will be 7. I've chosen this number arbitrarily; it could have been either 3, 5, or even 9. The number of inverters is directly related to the oscillator's frequency: more inverters introduce more delay, and that lowers the frequency, assuming the remaining device parameters remain constant.

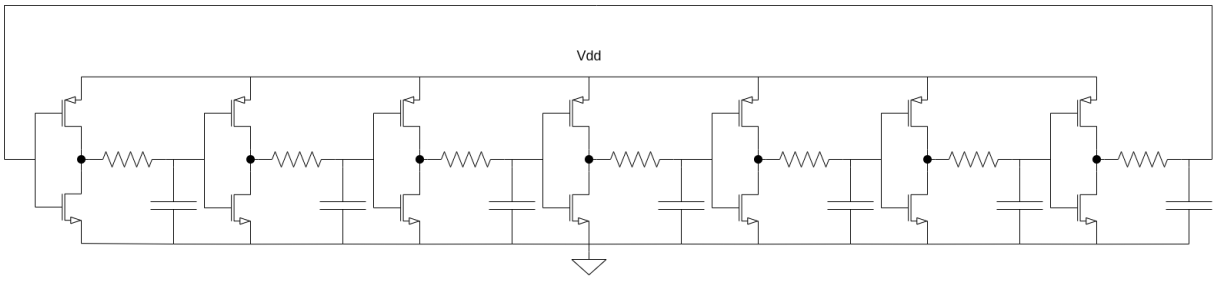
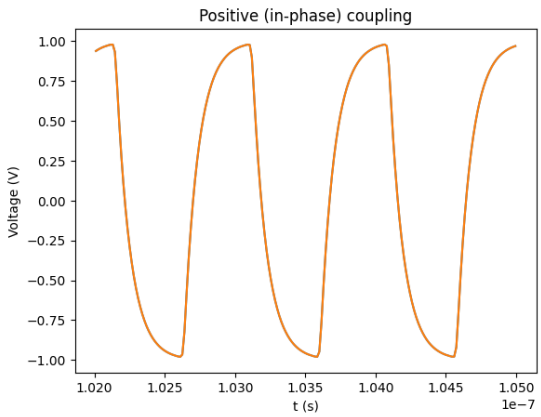
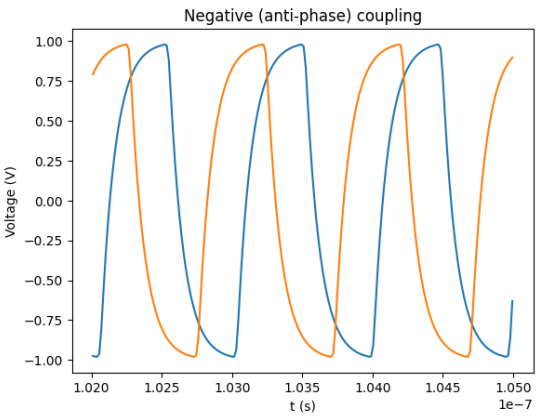


Figure 3.2: **Circuit diagram of a seven-inverter ring oscillator.** Each inverter consists of two transistors—one pMOS and one nMOS —and is connected through a resistor and a parasitic capacitance.

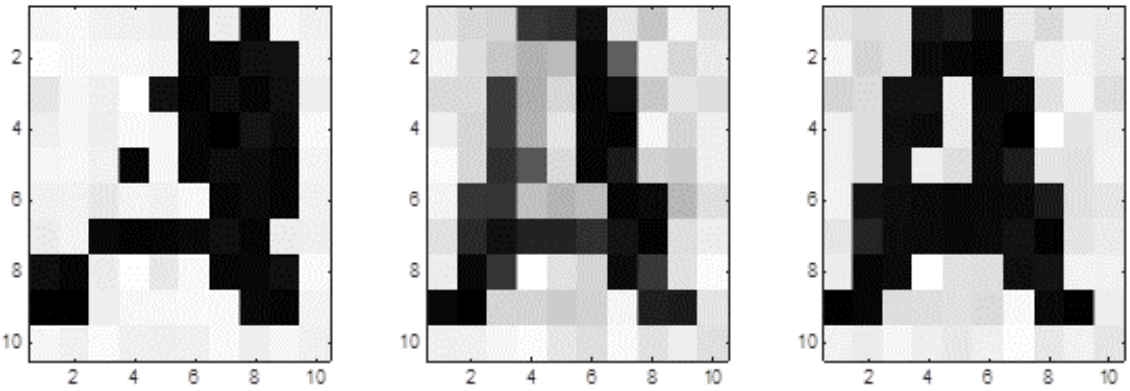
To give a simple example of how ring oscillators compute in phase space, Figure 3.4 shows a two-oscillator system. Nodes that are interconnected by a resistor will synchronize in phase. If identical nodes (say V_3, the 3rd voltage node of the ring oscillators) are connected, the oscillators will run in phase. However, in a 7-inverter oscillator, each node is phase-shifted by an angle of $2\pi/7$ relative to its neighbor. If, say, V_3 of one oscillator is connected to say V_6 of the oscillators, the oscillators will pull toward an anti-phase configuration. The waveforms of these two cases are illustrated in the top part of Figure 3.3.



(a) In-phase oscillators



(b) Anti-phase oscillators



(c) Computing with coupled oscillators to retrieve the written letter ‘A‘

Figure 3.3: **Phase-based computing by two ring oscillators**. On (a) an in-phase configuration is realized, because the two oscillators are connected using their same voltage nodes, i.e., V_3 and V_3, see Fig 3.4. However, on (b) an anti-phase configuration is depicted, where the two oscillators were connected by their different voltage nodes, e.g., V_3 and V_6, see Fig 3.4-it does not have to be necessarily these two nodes. Still, because of the delay in the inverters, it has to be different ones, and the larger the gap between the nodes, the larger the phase difference will be. The oscillators are coupled via resistors, which drive them to the desired phase configurations. If phases correspond to pixels of a grayscale image, the phase dynamics may be used to converge to predefined patterns [96]. The illustrations of the convergence to "A", shown on (c), are taken from [97].

A larger network of oscillators with in-phase or out-of-phase pulling resistors will converge toward an oscillatory ground state configuration, which in fact maps to the solution of the Ising problem [98]. Simply put, the phase of each oscillator will converge to a value that optimally aligns with most of the constraints imposed on the oscillator by other oscillators to which it is coupled. The dynamics of the coupled oscillator network will approximate the solution of a computationally hard optimization task. For an Ising problem, the oscillator-oscillator couplings are part of the problem description; there is no need to calculate them.

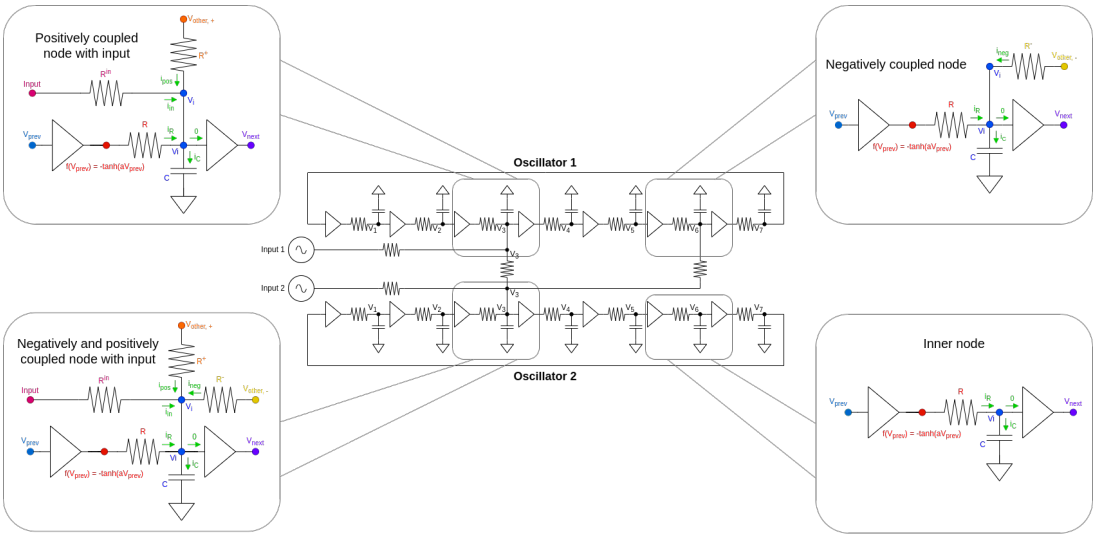


Figure 3.4: **Circuit diagram of a two-oscillator system with both of the coupling schemes.** Here in the middle, a two oscillator system can be seen, coupled together in the two possible ways—positive and negative—with numbered voltage nodes. The oscillators are built using the 7 inverters mentioned. The zoomed-in regions at the four corners represent the different node types that a given system can have. In these four highlights, nodes of the same color are at the same voltage level, and v_{i-1} and v_{i+1} denote the nodes preceding and succeeding v_i, respectively, due to the circular design. The green arrows indicate the agreed-upon directions of the currents. The 0-labeled current denotes 0 A, as by definition the input current to inverters is 0 A. Also, v_- and v_+ denote the voltage node of another oscillator, which is coupled to the particular oscillator negatively and positively, respectively. The *Input*-labelled waveform generators are any sources of voltages that serve as external inputs to the system. Note that these input generators can also serve as current sources; in that case, the connecting resistor is absent.

For the sake of concreteness, let us assume that our circuit consists of n oscillators and each oscillator is composed of the aforementioned 7 inverters. Also, let us assume that the circuit has k input generators. A simple Ordinary Differential Equation (ODE)-based circuit model can be constructed based on the equations derived in [99].

Each inverter is described on a behavioral level by an $f(x) = -\tanh(ax)$ nonlinearity connected to an RC delay element. In this way, a seven-inverter ring oscillator is modeled by seven first-order nonlinear ODEs. The value of a determines the inverter's switching speed.

The mathematical formulation consists of three parts: the internal dynamics of the oscillators (due to the inverters and RC elements), the dynamics due to external signals (input generators connected to the network), and the coupling dynamics (through the resistors between the oscillators).

In Fig 3.4, the basis of the derivation of the ODE of the circuit model can be seen. There are four types of nodes in the system, and for each of them, an ordinary, first-order differential equation can be derived using Kirchhoff's current law as follows:

- Most nodes are inner-nodes (bottom right part in Fig 3.4) in the oscillators (5 in each),

and their equation is rather easy to calculate:

$$C\frac{dv_i}{dt} = \frac{f(v_{i-1}) - v_i}{R}$$

- There can also be negatively coupled nodes (top right part on Fig 3.4), which are a bit more complex than the simple inner nodes. It also has another current component flowing to v_i, which arises from the difference between the voltage at a node of another oscillator and the voltage at the particular oscillator's node, divided by the resistance between the nodes. Here, the requirement for negative coupling is that the two coupling nodes differ in the oscillators' voltage-node numbering. In most of the setups from here on, negative couplings are between nodes 3 and 6. Here, the equation is the following:

$$C\frac{dv_i}{dt} = \frac{f(v_{i-1}) - v_i}{R} + \frac{v_- - v_i}{R^-}.$$

- There can be positively coupled nodes with inputs (top left part on Fig 3.4). Positively coupled nodes can be more complex than the negatively coupled nodes previously described, as they not only have an incoming current from a different oscillator, but they also can have input current from an external input indicated by the waveform generators in Figure 3.4. The setup depicted in the figure uses voltage sources as inputs, but it can also use current sources, as shown below. Note that the requirement for positive coupling between two oscillators is to have an even-even or odd-odd pairing of oscillators. In most of the setups from here on, positive couplings are between nodes 3 and 3. The particular ODE for this kind of arrangement is as follows:

$$C\frac{dv_i}{dt} = \frac{f(v_{i-1}) - v_i}{R} + \frac{v_+ - v_i}{R^+} + \frac{u_{in} - v_i}{R^{in}}.$$

- Assuming that a current generator is used instead of a voltage generator, the equation becomes simpler:

$$C\frac{dv_i}{dt} = \frac{f(v_{i-1} - v_i)}{R} + \frac{v_+ - v_i}{R^+} + u_{in}.$$

  It is worth reiterating that input is not necessarily present at node 3 of an oscillator; therefore, a node may receive additional current from coupled oscillators even in the absence of external input.

- The most complicated node is the one having positive and negative couplings and also some input (bottom left part on Fig 3.4). It is basically the merger of the previous two items, which is manifested in the equation as well:

$$C\frac{dv_i}{dt} = \frac{f(v_{i-1}) - v_i}{R} + \frac{v_- - v_i}{R^-} + \frac{v_+ - v_i}{R^+} + \frac{u_{in} - v_i}{R^{in}}.$$

- Given that the input is not a voltage generator but a current generator, the equation

becomes simpler again:

$$C\frac{dv_i}{dt} = \frac{f(v_{i-1}) - v_i}{R} + \frac{v_- - v_i}{R^-} + u_{in}.$$

Combining this knowledge, it is possible to derive the following ODE for the collection of voltages at all nodes. The following describes all parts given that the differential equation is constructed for every node in the system using Kirchhoff's current law and assuming only resistors as couplings for both types of input generators:

- Voltage generator:

$$C\frac{d\boldsymbol{v}}{dt} = \underbrace{\frac{1}{R}\boldsymbol{A'}(f(\boldsymbol{P_\pi v}) - \boldsymbol{v})}_{\text{Intrinsic dynamics}} + \overbrace{\frac{1}{R_c}\boldsymbol{C'v}}^{\text{Coupling dynamics}} - \underbrace{\frac{1}{R_{in}}\Big(\boldsymbol{B'} \odot (\boldsymbol{v} \ominus \boldsymbol{u}^T)\Big)\mathbf{1}}_{\text{External dynamics}} \tag{3.1}$$

- Current generator:

$$C\frac{d\boldsymbol{v}}{dt} = \underbrace{\frac{1}{R}\boldsymbol{A'}(f(\boldsymbol{P_\pi v}) - \boldsymbol{v})}_{\text{Intrinsic dynamics}} + \overbrace{\frac{1}{R_c}\boldsymbol{C'v}}^{\text{Coupling dynamics}} - \underbrace{\frac{1}{R_{in}}\boldsymbol{B'u}}_{\text{External dynamics}} \tag{3.2}$$

where again, $f(x)$ was used as the behavioral model of the inverters:

$$f(x) = -\tanh(ax), \tag{3.3}$$

Here, $a \in \mathbb{R}^+$. Furthermore, π is a permutation, such that

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}, \tag{3.4}$$

and $\boldsymbol{P} \in \{0,1\}^{(7n)\times(7n)}$ is a block matrix for which in every 7×7 matrix-block in the main diagonal, there is a permutation matrix corresponding to π. This orders the voltage nodes in the ring oscillator to compute the voltage differences across the resistors between the two inverters. $\boldsymbol{B'} \in \mathbb{R}_{\geq 0}^{(7n)\times k}$ is the connector matrix for the inputs, assuming that the input $\boldsymbol{u} \in \mathbb{R}^k$, which is typically a sinusoidal voltage or current generator, but in a later chapter a non-coherent signal will be used as voltage levels for a voltage generator. $\boldsymbol{C'} \in \mathbb{R}_{\geq 0}^{(7n)\times(7n)}$ is the modified coupling matrix which is to be constructed from the real, humanly readable coupling matrix $\boldsymbol{C} \in \mathbb{R}^{n\times n}$. The parameters $R, C \in \mathbb{R}_{\geq 0}$ are fixed for the oscillators. Still, the actual resistance between the inverters might change—either through learning or manual tuning—to set the frequency of the oscillators. This is done by the introduction of the $\boldsymbol{A'} \in \mathbb{R}_{\geq 0}^{(7n)\times(7n)}$ block diagonal matrix. This is directly related to the oscillators' frequency.

In addition, the $R_c \in \mathbb{R}_{\geq 0}$ coupling parameters are one of the remaining two, non-fixed parameters of the system that governs the entire coupling dynamics. The other non-fixed

parameters are the ones gathered in $\boldsymbol{B'}$. The $\boldsymbol{B'}$ matrix is directly related to the resistance value between the oscillators and input generators, if a voltage generator is used, or to the amplitude of the input signal in the case of a current generator.

One way to construct the humanly readable $\boldsymbol{C}$ matrix is as follows:

- Main diagonal entries are 0, as no oscillator is coupled to itself.
- Entries in the upper triangle of the matrix correspond to the positive (in-phase-pulling) couplings.
- Values in the lower triangle of the matrix correspond to the negative (anti-phase-pulling) couplings.

This yields a non-symmetric, non-negative matrix and permits both positive and negative couplings between two oscillators.

The other way to construct $\boldsymbol{C}$ is to restrict the couplings to be either positive or negative. In this way, $\boldsymbol{C}$ will be symmetric, but not necessarily non-negative, because negative numbers must denote negative couplings.

It is important to note that this $\boldsymbol{C}$ matrix has no physical meaning in itself; it just stores the relative coupling strengths between the oscillators. The real coupling resistance value will be set by $\boldsymbol{C'}$ and an appropriate resistance scaling value, which is R_c. This way $\boldsymbol{C'}$ can be interpreted as the—scaled down—admittance between the oscillators.

The construction of $\boldsymbol{C'}$ can be done easily from $\boldsymbol{C}$, algorithmically. As every positive coupling is between $3-3$ nodes of the oscillators and every negative connection is between $3-6$ nodes of the oscillators, the $\boldsymbol{C'}$ matrix is relatively sparse. To put that into perspective, let us assume an all-to-all coupling scheme with both positive and negative couplings is taken, only 2 nodes from 7 will be connected from the oscillators, so the $\boldsymbol{C'}$ matrix will have at most $\frac{2}{7}$ of its elements as nonzero. Similarly, because the inputs are only fed into node 3 of every oscillator, the $\boldsymbol{B'}$ matrix is also sparse. Also, it is possible to use a $\boldsymbol{B} \in \mathbb{R}^{n \times k}$ matrix for the input, as a humanly readable form of a connector matrix, but since the structure of $\boldsymbol{B'}$ is simple, it is not necessary.

The circuit model I described, and used [99], is simpler than a SPICE-level model, as it accounts for transistor characteristics via a behavioral curve. The internals of the MOS transistors are neglected. This simplification is done to facilitate learning.

In this dissertation, most of the introduced architectures are built from ring oscillators. Also, from here on, let us assume that the "input node" of an oscillator is node 3 and the "output node" is node 6. Furthermore, if the resistors are replaced with other devices, such as transistors, between oscillators, unilateral coupling can be created, meaning that one oscillator affects the other but not vice versa. This hasn't been utilized in this dissertation with the electrical circuit model, but it is possible to do so.

Some deviations from these designs occur in Section 6.3 and Chapter 7. The former section also employs Kuramoto oscillators as an initial proof of concept. In contrast, the latter uses the electrical-circuit equivalent of Hopfield neurons, chosen for their robustness and structural simplicity, yet exhibits similarities to ring oscillators.

Note that the equations shown here for the ring oscillators are very similar to Equation 2.4 introduced in Section 2.4, as both have an intrinsic nonlinear dynamics part, coupling, and input

parts. However, the main difference—and advantage—is that, in ring oscillators, the coupling scheme is not limited to nearest-neighbor couplings; long-range couplings are permitted, which can enhance the functionality of the architectures.

3.3 Computationally hard problems

In terms of computational hardness, the previous significant distinction among problems was whether they were in P or NP. [100] Even though it is not the focus of this dissertation to solve these problems, I would like to highlight that neuromorphic computing architectures have been used to tackle NP problems with surprising efficiency [101], [102], [103].

Nowadays, another factor must be considered when discussing computationally hard tasks: the training of large software neural networks, such as deep neural networks and large language models [104], [105].

In Section 1.1.1, the motivation for Intel to address these two problem sets was discussed, and it is paramount that, in the future, we might be able to provide energy-efficient solutions to them.

4 Coupled Oscillator Design Algorithmically

Designing coupled oscillators is complex because the architecture layout and coupling strengths must be appropriately configured to solve a given task. Two other design aspects should be considered. These are, first, determining the oscillator frequency and, second, feeding the input data through the generators into the system. This is a non-trivial task, but fortunately, many machine learning techniques can be leveraged, both gradient-based and gradient-free ones. That said, it also introduces additional complexities: it is paramount to develop methods to extract useful information from the oscillators and to construct appropriate loss or cost functions based on this extracted information for machine learning algorithms. Furthermore, taking this postprocessing into account, it is essential to determine how much of the overall architecture can be implemented on the hardware side when designing electronic circuits, such as ring or VO_2 oscillators. The best-case scenario would be to keep everything from preprocessing through inference to postprocessing, and even learning on the hardware side, but that isn't very easy.

4.1 Gradient-based method

Gradient-based methods extract information from a given architecture after inference, construct a loss function from it, and search for its (hopefully global) minimum by optimizing the loss function's gradient with respect to the optimization parameters. This might be done on raw output data, as in regular ANNs, but for oscillators, it is usually not the most appropriate approach. That said, in Section 4.1.1, I present a simple wave-generation case in which I used the oscillators' outputs directly. In most other cases, I used the frequency or phase of the output oscillators as the system output. I then used those values in a specified loss function. In the literature on gradient-based methods for training ONNs, numerous variations exist; however, Equilibrium Propagation (EP) [106] and Back Propagation through Time (BPTT) [107], [108], [109] are most commonly used. In this dissertation, the primary focus in terms of gradient-based methods is on BPTT.

Backpropagation is the de facto standard algorithm for training neural networks [110]. After each neural network inference, the gradient of the aforementioned properly defined loss function is computed efficiently with respect to the system's trainable parameters. Subsequently, the gradient is used to update the system's trainable parameters, thereby driving them to converge to the—possibly local, but ideally global—minimum of the loss function.

BPTT is backpropagation applied to a dynamic system (i.e., an ODE-based description). The ODE can be solved using any standard time-stepping technique—implicit or explicit—using a discrete time step. This discretized solution can be viewed as a many-layer neural network, with one neural layer corresponding to a temporal snapshot of the system dynamics. The BPTT algorithm computes and stores these layers (snapshots) during the forward pass, then calculates

the derivatives of the loss function with respect to the trainable parameters in the backward pass.

To apply backpropagation through time (BPTT), as mentioned, a loss function—also called an objective or cost function in some literature—must be defined; it takes its minimum value when the system is in the desired computational state. The hardest part of developing these functions is constructing them so that they yield only minimal solutions that are also solutions to the given problem.

The code for the simulation and training of various ONNs has been written in *PyTorch*[111]. The autograd feature of PyTorch makes implementing backpropagation and BPTT straightforward. Additionally, *torchdiffeq*[112] package was used for implementing backward-differentiable ODE solvers in some cases, but sometimes a self-implemented solver was used for controlled voltage extraction. *torchdiffeq* is an external, third-party library built on *PyTorch* that provides various differentiable ODE solvers implemented in *PyTorch*. A useful feature is that it can apply the adjoint method to the backward step [113] and compute gradients with a constant memory cost.

It should be noted that BPTT is computationally demanding for complex dynamic systems such as ONNs. The time-domain solution of an oscillating architecture typically consists of thousands of time steps. As BPTT unwraps the time-domain solution of an ODE into a many-layer neural network, it must handle networks with many thousands of layers, which can lead to memory bottlenecks during training.

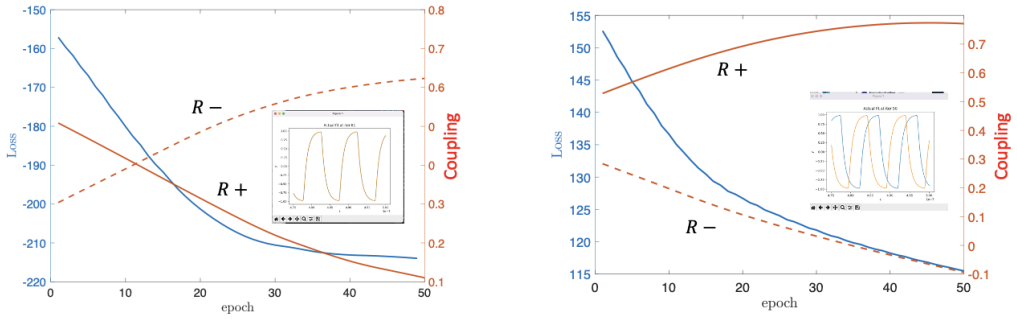
Backpropagation through many layers also inevitably suffers from the vanishing or exploding gradient problem [114]. This arises from the inherent structure of backpropagation, which uses the chain rule of differentiation. When traversing many layers during the backward pass, gradients may become nearly zero, preventing meaningful learning, or they may quickly converge to infinity, rendering learning impossible. I found that the algorithm I used produces functional gradients up to a few thousand time steps (layers). The ONNs introduced later are designed to converge safely within this time frame.

The high computational cost of BPTT is the primary reason for selecting the behavioral representation of inverters over the two-transistor-circuit equivalent for ring oscillators. A typical Level 3 MOS model contains hundreds of parameters, whereas a SPICE-level simulation is straightforward even for larger circuits; however, learning (backpropagation) becomes computationally demanding for such models. Also, it is worth noting that gradient-based learning is not supported in SPICE. Because the gradient function is required for backpropagation, SPICE can be used solely for inference, as the computational steps that enable gradient computation in *PyTorch* are absent.

That said, a theoretical solution to this can be constructed. Because the system is known at training time, it may be possible to compute the gradient analytically in advance and use it to train the system. This is arduous, error-prone work that does not guarantee success. That having been said, it might be a good way in the future to try it out, but I think a better solution to test the network in a better circuit simulation framework is to do the training inside *PyTorch* with BPTT and validate the designed circuit with SPICE. These steps are not present in this dissertation as the focus was on the algorithmic design of different ONNs to different problems and not on their exact electrical behavior, but the models built were done by keeping the physical implementability in mind.

It is also essential that BPTT supports only in silico training. The design of the ONN (i.e. the learning) takes place on different hardware than the inference. Learning is performed using a digital computer model. Once learning is complete, inference is performed on dedicated hardware that uses the learned circuit parameters. Online learning is not feasible in this manner; however, the goal is to achieve efficient, hardware-accelerated inference.

Figure 4.1 exemplifies the learning procedure for the two-oscillator system of Fig 3.4. I selected the system's loss function as the dot product of the oscillator waveforms, a standard choice for this type of problem. The loss should be maximized (minimized) for in-phase (anti-phase) coupling. The machine learning algorithm adjusts the values of the $C_{1,2}$ and $C_{2,1}$ parameters—note that these correspond to the resistances between the oscillators—until the desired phase configuration is achieved. To better understand the role of these resistors in the figure, I named them $R+$ ($C_{1,2}$) and $R-$ ($C_{2,1}$), corresponding to positive and negative coupling, respectively.



(a) In-phase oscillators

(b) Anti-phase oscillators

Figure 4.1: **Learning the in-phase and anti-phase couplings for a two-oscillator system.** On (a), the simulation's result can be seen for the positively coupled oscillators, meanwhile on (b), there is the same for the negatively coupled 2-oscillator system. The loss changed in both cases from high to low. Additionally, the orange curves indicate the learning parameters in $\mathbf{C}$ rather than the actual resistor values. Note that in (b) the parameter value corresponding to $R-$ is going below 0, which would mean a negative resistance because of the connection of the parameters in $\mathbf{C}$ to the physical parameters in this experiment, but this is only the mathematical solution; for a given simulation, the parameters were clamped to be non-negative. If they hit zero, the connection was removed.

This method can be straightforwardly generalized to achieve convergence toward more complex patterns. If the loss function is designed to maximize the dot product of waveforms between like-colored pixels and minimize it between different-colored pixels, then the phase pattern can converge to any prescribed image. If the phase pattern is designed to converge to different patterns for different inputs, the ONN will function as an associative memory.

Since the Machine Learning (ML) technique is used to design a physical circuit, safeguards were implemented to avoid unrealizable circuit parameters, such as negative resistances or extremely large couplings, that would quench oscillations. This was done by clipping the values after each learning step to a given interval.

4.1.1 Wave-generation with coupled oscillators

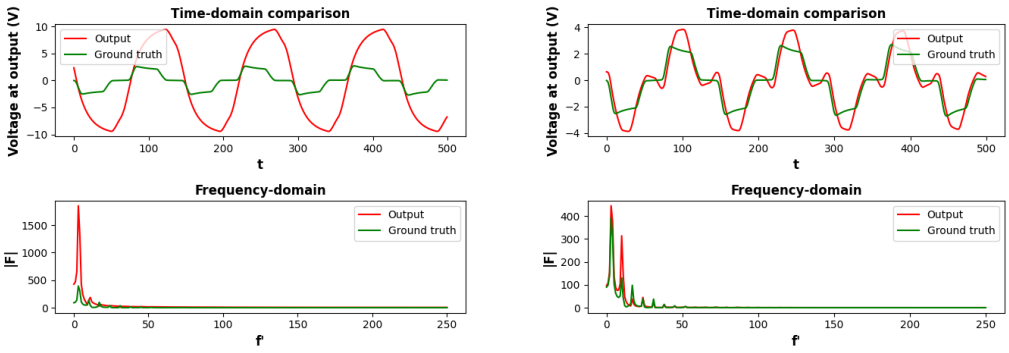
A simple yet interesting problem that coupled oscillators can solve—beyond the previously shown tutorial on in-phase and anti-phase coupled oscillators—is wave generation. Previously, it has not been done using BPTT-trained coupled oscillators. The task is as follows: given an arbitrary periodic wave, the goal is to reproduce it using the sum of some number of coupled oscillators. This is a regression task, so the loss function is the Mean-Squared Error (MSE) of the sum of the oscillators' waveforms and ground truth waveform:

$$l(O, T) = (O - T)^2,$$

where

$$O = \sum_{i=1}^{n} v_i.$$

Here, v_i is the output voltage level of oscillator i. In my setup for this problem, I used $n = 5$ oscillators with an all-to-all coupling scheme. In Figure 4.2, the target waveform—green curve—and the output summed waveform of the oscillators—red—can be seen in the top row. In the bottom row, the frequency-domain representations are shown.



(a) Start of learning

(b) End of learning

Figure 4.2: **Learning a given wave pattern with 5 coupled oscillators.** On (a), the initial output of the sum of the oscillator and the ground truth voltage in the time domain and frequency domain can be seen. In (b), it is evident that convergence occurred, and the algorithm produced a waveform that is qualitatively much closer in both the time and frequency domains than at the start of learning.

The algorithm learned the waveform exceptionally well. The discrepancy arises from the limitation of the phase difference between oscillators; in reality, when other constraints are imposed on them—such as the pulling and pushing of other oscillators—they will never have a complete π phase difference. This means that with 5 fully coupled oscillators, it is impossible to cover the whole range of possible pairwise phase differences.

4.1.2 Ring-oscillators as perceptron-like devices

Now, moving on to the other big topic of ML algorithms, namely classification. To further demonstrate the basic capabilities of coupled ONNs, I created a synthetic dataset consisting of two linearly separable blobs. This is an easy task that a single perceptron can solve. Perceptrons

are foundational in the study of neural networks and modern machine learning [36], [115]. They are linear classifiers, meaning that they are separating a given N-dimensional space with an $N-1$ dimensional hyperplane.

Oscillators are inherently nonlinear, and it is not trivial to map a perceptron's behavior onto oscillators with more complex dynamics. Having said that, it is possible to create a small system of coupled ring oscillators that can learn the same task as a perceptron using BPTT.

I used the two features as input phases—scaled to $[0, \pi]$—and fed them into the system. Two oscillators received the input, and the two oscillators were coupled to each other and to two other oscillators. The architecture is shown in Figure 4.3. I used bilateral couplings, so the oscillators affected each other symmetrically; however, it is easier to visualize the network as a feedforward-like structure. The blue connections correspond to the B', whereas the red connections correspond to the C' matrices. The frequency of the oscillators was set to be equal, so the A' matrices' values were not trained.

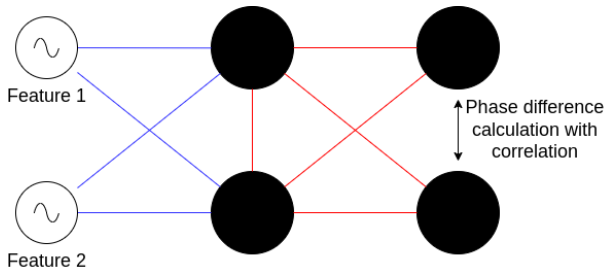


Figure 4.3: **Architecture for binary classification with ring oscillators as perceptrons.** The black-filled circles represent oscillators, while the generators are on the left-hand side, connected to the first two oscillators.

The generated dataset is shown on the left side of Figure 4.4. As shown on the right side of the figure, which presents the training results, the previously mentioned network achieved nearly 100% accuracy. This demonstrates that ring oscillators can be used for this task rather than a perceptron.

4.2 Gradient-free methods

Gradient-free optimization, also known as derivative-free optimization, can refer to two distinct machine learning algorithms: objective- and non-objective-based rules. From the latter, I will consider only those that can be used in Hopfield-like networks as associative memories, as these are the only ones relevant to this dissertation.

First, there is the class of algorithms that optimize objective functions without relying on gradient information; these are objective-based rules. These methods are instrumental when gradients are unavailable, unreliable, or prohibitively expensive to compute, such as in simulations, black-box functions, or highly non-differentiable landscapes. Instead of gradient descent, gradient-free optimization relies on direct function evaluations to iteratively improve candidate solutions.

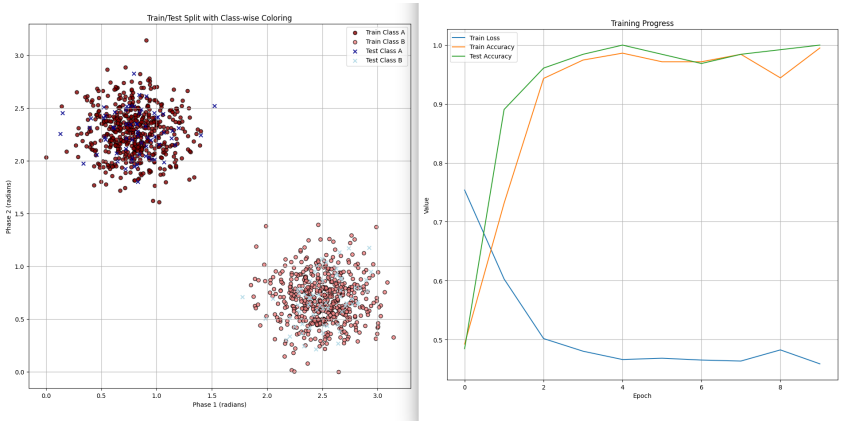


Figure 4.4: **The generated synthetic blobs dataset and the result of training** On the left side, the dataset can be observed with the training and test set having different colors. On the right, the training results across epochs are shown. The network achieved almost 100% accuracy in separating the two blobs, a strong result that demonstrates that ring oscillator networks can simulate a perceptron for this binary classification task.

4.2.1 Objective-based algorithms

Several well-known families of gradient-free, objective-based methods exist. Evolutionary algorithms, such as Genetic Algorithms (GAs) and Differential Evolution (DE), explore solution spaces using biologically inspired operators, including mutation, recombination, and selection [116]. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is another prominent method that adapts the covariance structure of the search distribution to efficiently navigate complex, multimodal landscapes [117]. Particle Swarm Optimization (PSO), inspired by swarm intelligence, models the collective behavior of particles to converge toward promising regions in the search space. Other approaches include Bayesian optimization, pattern search, and random search.

Gradient-free optimization methods are widely applied in machine learning hyperparameter tuning, engineering design, reinforcement learning, and scientific simulations where gradients are either inaccessible or unreliable. For example, hyperparameter optimization of neural networks and black-box optimization problems in physics and chemistry are common application areas.

The Nevergrad Package

Nevergrad is an open-source Python package developed by Facebook AI Research for gradient-free optimization [118]. It provides a wide range of optimization algorithms, including evolutionary strategies, CMA-ES, PSO, and bandit-based methods, under a unified API. Nevergrad is designed to handle continuous, discrete, and mixed-variable optimization problems, making it highly flexible for both academic research and industrial applications. Additionally, it integrates well with machine-learning workflows and provides benchmarking tools for comparing optimizer performance. Its accessibility and versatility have made it a widely adopted tool in black-box optimization tasks.

In Section 6.3.1, I showcase objective-based gradient-free optimization for the design of Oscillatory Neural Networks.

4.2.2 Algorithms for pattern association and Hopfield-like networks

Hopfield networks are classical models of associative memory, in which patterns are stored as attractors of a recurrent neural network. The learning rules used to embed these patterns in the network weights play a crucial role in determining storage capacity, stability, and resistance to interference.

Hebbian Learning Rule

The most fundamental learning rule is the Hebbian rule, often summarized as "neurons that fire together, wire together". In a Hopfield network, for N binary neurons and p stored patterns $\{\xi^\mu\}_{\mu=1}^p$, the weight matrix W is given by:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu, \quad i \neq j \tag{4.1}$$

Hebbian learning is local, simple, and biologically plausible. However, its storage capacity is limited to approximately $0.14N$ patterns, and it is susceptible to crosstalk interference between patterns [35], [40].

Storkey Learning Rule

The Storkey learning rule is an enhancement of the Hebbian rule designed to reduce interference between stored patterns. The update for pattern μ is:

$$\Delta w_{ij} = \frac{1}{N} \left[\xi_i^\mu \xi_j^\mu - \xi_i^\mu h_j^\mu - h_i^\mu \xi_j^\mu \right], \quad i \neq j \tag{4.2}$$

where $h_i^\mu = \sum_{k \neq i,j} w_{ik} \xi_k^\mu$ is the local field. Storkey learning increases storage capacity and reduces spurious states while remaining a local update rule [119].

Pseudo-Inverse Rule

The pseudo-inverse rule enables exact pattern embedding up to the maximal capacity. Given a pattern matrix $\mathbf{\Xi}$, the weights are set as:

$$\mathbf{W} = \mathbf{\Xi} (\mathbf{\Xi}^\top \mathbf{\Xi})^{-1} \mathbf{\Xi}^\top - \mathbf{I} \tag{4.3}$$

While it maximizes capacity (N patterns) and guarantees correct recall, it is nonlocal and less biologically plausible [120].

Other Local Learning Rules

Several other local learning rules have been proposed to improve associative memory performance:

- **Covariance Rule**: Subtracts the mean activity from each pattern to reduce crosstalk [121].
- **Anti-Hebbian / Mixed Hebbian**: Combines Hebbian potentiation and depression to enhance stability.

- **Diederich-Opper Rule**: Designed to store correlated patterns efficiently in spin-glass networks [122]. Additionally, it has been shown that this rule can be used to train Kuramoto oscillators to associate handwritten digits from the MNIST dataset [123].

These rules illustrate the ongoing evolution of Hebbian-inspired learning mechanisms for associative memory, balancing biological plausibility, locality, and network performance.

In the next chapter, I will show how the Hebbian learning rule can be applied to ring-oscillator-based ONNs, and I compare its performance with the gradient-based BPTT method.

Gradient-free optimization provides a robust set of tools for black-box problems in which gradients are neither available nor efficient or straightforward to compute. From evolutionary algorithms to local learning rules, these approaches are central to solving optimization tasks in engineering, AI, and scientific domains.

5 Static Image Processing on MNIST database

MNIST database [124] is the de facto standard for handwritten digit classification tasks. It contains $60,000$ training and $10,000$ test samples of handwritten digits, with 28×28 pixel-sized images. The goal is to train a network to recognize the digits in the database. In this chapter, I will show how oscillators can be used as an associative memory and compare the BPTT-trained network I designed with a long-standing Hebbian-rule-trained network. Additionally, I will introduce a more complex architecture that combines an ONN and a standard ANN—a Multilayer Perceptron, specifically—for high-accuracy classification of these handwritten digits. These networks, which I designed using BPTT, are novel because using BPTT to design electric-circuit-based ONNs is not standard practice; as such, they offer a new way to think about ONNs.

I note that, for training, I did not use the entire MNIST dataset. Since the network should be numerically solved first through thousands of time steps and then backpropagated through the computational graph, a single inference and learning step takes a considerable amount of time—$1 - 2 minutes$—even on GPUs, and the batch size cannot be increased too much as the computational graph is memory-intensive. I used a subset of the MNIST dataset, comprising $1,000$ images for training and 100 for testing. These were randomly selected, and after successful training, I always created new test datasets for validation.

5.1 Pattern Recognition on MNIST as an Associative Memory

The standard MNIST dataset was used to test the associative capabilities of the designed coupled-oscillator system. Because the BPTT algorithm is computationally demanding, I implemented several simplifications. The initial 28×28-pixel MNIST images have been downsampled to 14×14 or 7×7 using average pooling. This allowed me to reduce the dimensionality of the input images while retaining the necessary information via average pooling. Additionally, 14×14 MNIST images remain recognizable to the human eye, allowing me to easily determine whether specific patterns are easier for the algorithm to distinguish from others. On the other hand, 7×7 images are sometimes too small to be easily recognized by humans.

The architecture used as an associative memory is shown in Figure 5.1. Note that in the figure, the generators are assumed to be current generators. Still, the scheme works with voltage generators as well, given that a resistor is between the generator and the connected oscillator, as shown in Section 3.2.2. The main idea is as follows:

- **Input encoding:** the input image is encoded into the phases of the input generators using the grayscale $[0, 1]$ values from the picture pixel-by-pixel. Each pixel has its own generator that feeds its value into the connecting oscillator.

- **Computing architecture:** the architecture is positioned into a grid to match the dimensions of the input image, and it can have any couplings—in my case, I used all-to-all and nearest neighbor couplings; the network then goes through a transition to settle into a stable state, and then the output can be read out.
- **Output decoding:** the output of the oscillators is read out, and the phase difference of the oscillators to a reference oscillator is calculated and then mapped to the $[0, 1]$ range to have proper grayscale values again, thus creating a phase pattern.

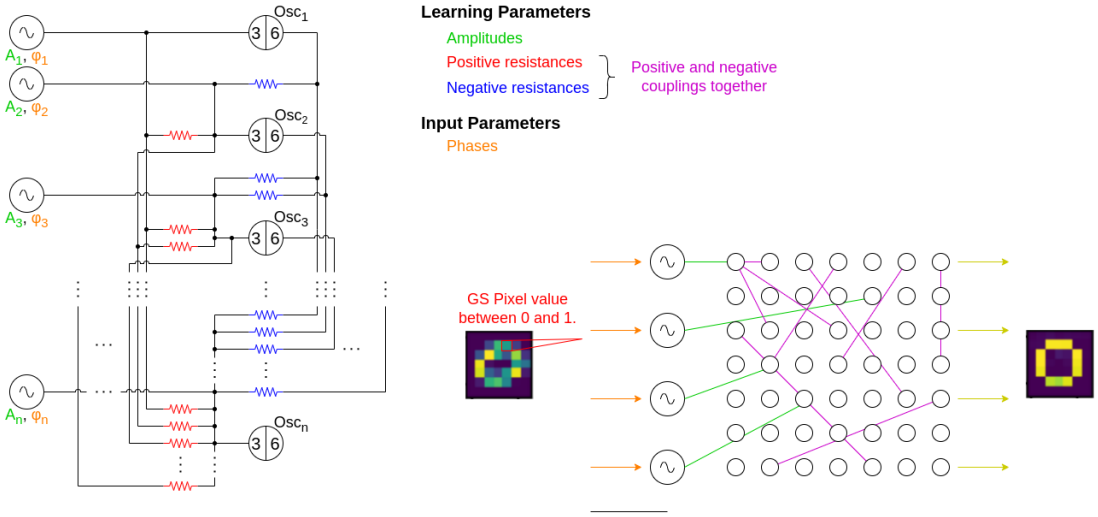


Figure 5.1: **The circuit diagram and logic of the entire computational layer with the input generators**. Input signal generators generate sinusoidal signals whose phases correspond to an input pattern, such as the pixels of an image. These generators are connected to the computing oscillators, whose phase pattern provides the solution to the problem. The $3-6$ marks on the oscillators indicate the input and output nodes in the ring oscillators' circuit. The values of the green, red, and blue-colored circuit elements are learned during the learning process, and the phases indicated in orange are the inputs. In the schematic figure, the purple connections indicate both positive (red) and negative (blue) couplings, as both can occur for optimal convergence. The grayscale pixel value is read from the image, converted into phase information, and then the sinusoidal current generators are connected to the oscillators in a one-to-one mapping. The yellow arrows show that the output is read from the oscillators and an image is formed.

5.1.1 Comparison of BPTT-trained and Hebbian Rule-trained networks

Because ONNs with all-to-all couplings resemble Hopfield networks, Hebbian learning can also be used to train ONNs [95]. If the goal is for the phase pattern to converge toward ξ or η for inputs resembling ξ or η, then the weights that realize this associative memory are as follows:

$$C_{ij}^{cpl} = \frac{1}{2}(\xi_i \xi_j + \eta_i \eta_j), \tag{5.1}$$

where ξ_i, ξ_j is the i-th, j-th element of the pattern ξ, and η_i, η_j is the i-th, j-th element of the pattern η, respectively. Note that these are not grayscale, but binary patterns. It would

have been possible to use grayscale patterns, but binary patterns were more straightforward and improved the model's performance; therefore, I chose them.

This rule assumes all-to-all couplings, which makes large-scale network implementation difficult, if not impossible, in practice.

In my Hebbian learning scheme, the weights were determined initially in a single-shot formula, and in my test case, I applied the learning to optimize the value of base coupling resistances, R_c, and the parameters in $\mathbf{B}'$, which are the amplitudes of input current generators. This is essentially an augmentation of the Hebbian rule: the specified parameters should be set appropriately for the circuit to function, but it does not detract from the Hebbian rule; instead, it may even strengthen it.

The inner RC time constant of the ring-oscillators was $2.0 \cdot 10^{-10}$ sec, which translates into a 500 MHz oscillation frequency (time period $T = 2$ ns). The total simulation time for the network was 500 ns. The phase pattern is calculated from the last 300 ns window, so convergence is achieved after less than 100 oscillation cycles or 200 ns.

The same functionality as Hebbian learning can be achieved using the BPTT method. With BPTT, any coupling scheme can be employed; here, I used an all-to-all and a nearest-neighbor coupling scheme. Note that the nearest neighbor connection is defined on the $N \times N$ grid that matches the dimensions of the input image. The loss function that was selected for BPPT is the following:

$$L = \frac{1}{n} \sum_{k=0}^{n} (O_n - T_n)^2, \tag{5.2}$$

where O_k is the pattern calculated from the output of the oscillators for the k-th input in the batch and T_k is the ground truth for the same, which were ideal patterns of '0' and '1'. In the above formula, n denotes the batch size used for learning.

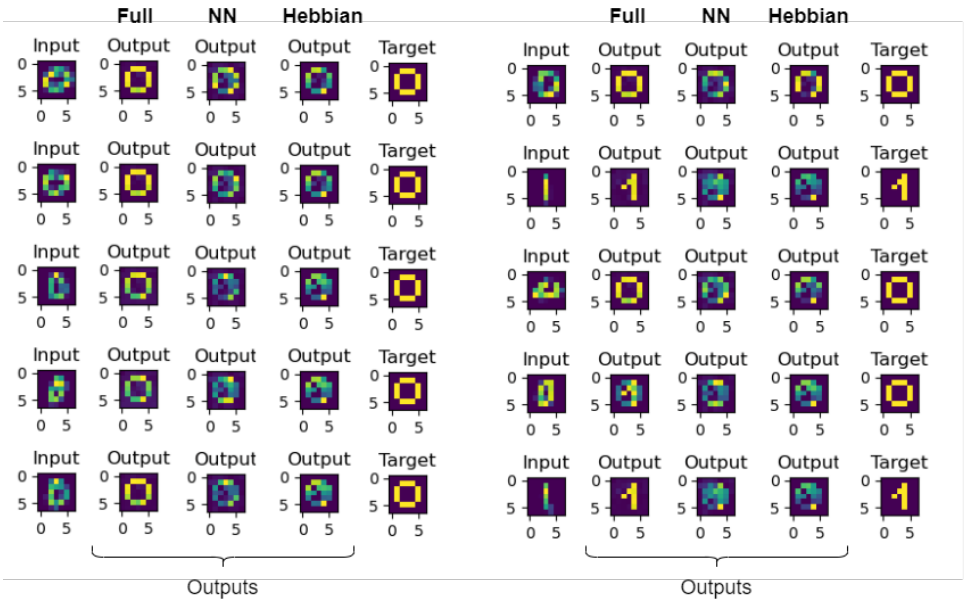


Figure 5.2: **Output association comparison of the differently trained ONNs.** Here, the comparison of the results of the fully connected, nearest neighbor connected, and the Hebbian-rule-based networks can be seen. There are two blocks of 5 inputs, shown side by side. The first column in each block corresponds to the input digit; the next three columns are the outputs of the systems—ordered from left to right: fully coupled, nearest-neighbor coupled, and Hebbian-based. The last column in both blocks shows the target digit. It is apparent that the fully connected system performed best, but even our other proposed nearest-neighbor connected topology outperformed the Hebbian-based architecture.

Figure 5.2 compares the results from the Hebbian and BPTT-based designs qualitatively. It is visually apparent that the BPTT-based, fully connected network associates the correct pattern with highly distorted patterns. For the experiments shown in Figure 5.2, the images were downsampled from 28×28 to 7×7, which distorted many of the inputs. It accelerated computations because an all-to-all-coupled 728 oscillator system would require nearly 620000 resistors. This is also practically impossible to realize.

Most importantly, the BPTT-based design enables sparsely interconnected circuit topologies. I used it to design the C_{ij} matrix of an associative memory, assuming only nearest-neighbor interconnections. The nearest-neighbor interconnected, BPTT-designed network outperforms the fully interconnected Hebbian network—even if the number of trainable parameters in the system—$\approx 8n$ vs. $\frac{1}{2}n^2$—is significantly less.

The finding that a nearest-neighbor, BPTT-designed network outperforms a Hebbian-designed, fully connected network is essential. In a fully connected ONN, the number of connections grows quadratically with the number of oscillators, making large, fully connected circuits unrealizable. Only locally connected architectures yield to scalable, physically realizable ONN circuits.

Method	Hebbian	Proposed fully connected	Proposed NN-connected
#Params	1176	2352	312
MSE	0.068	0.020	0.047

Table 5.1: **Parameter count and MSE performance comparison for the different training methods on ONNs.** The MSEs of all the elements from the set and their respective ground truths for the different methods in the case of associative learning. It is apparent that the fully connected network performed the best, but even the nearest neighbor connected layer is good enough to beat the Hebbian learning in terms of quantitative association.

Quantitatively, the results of the different approaches on the whole dataset $S = \{0, 1\}$ are presented in Table 5.1. From this, it is evident that even the nearest-neighbor connected network outperformed the Hebbian rule network in terms of MSE.

5.2 Pattern classification on MNIST

Single-layer associative memories are not particularly efficient for classifying all ten MNIST classes, as there are strong correlations among the digits. The BPTT method does not require the network oscillators to converge to a prescribed phase pattern, so there is no need to use associative memory for classification. For this reason, I investigated a simple multilayer ONN in which the second layer comprises a single oscillator connected to all oscillators in the input layer, as illustrated on the left side of Figure 5.3.

I used the architecture in Figure 5.3 in various ways: for binary classification, a single hidden layer and a single output yielded decent results, as described in Section 5.2.1. To classify all ten digits, I trained 10 blocks (see Figure 5.5), each responsible for recognizing a particular digit, and evaluated them with a winner-take-all decision (see Section 5.2.3). Finally, I replaced the winner-take-all method with a small MLP comprising only a few neurons. This architecture is shown in Figure 5.5 and discussed in Section 5.2.4.

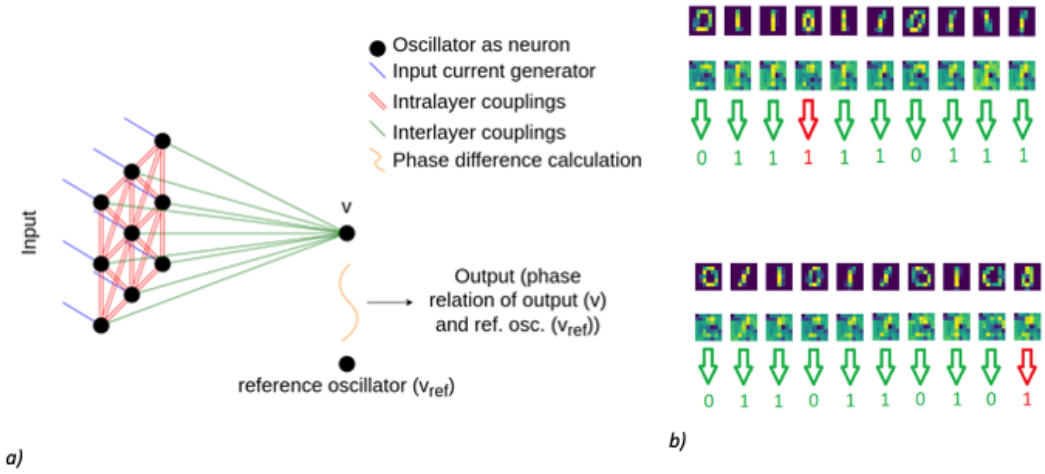


Figure 5.3: **A simple two-layer classifier showing also the patterns forming in the hidden layer.** On the left, the architecture can be seen that I used for binary classification. It might appear to be in a Feedforward-like arrangement, but this is merely a clearer visual representation. In reality, all the oscillators were connected bilaterally. On the right side, samples from the binary classification of 0 and 1 are shown, along with input, hidden-layer pattern, and output-prediction triplets. The two red numbers indicate the network's prediction errors. I want to highlight that, in some cases, there is an apparent structure in the hidden layer, as some phase patterns there resemble either 0 or 1, but this is not universal; at other times, there is no apparent, humanly visible structure at this stage of the architecture.

5.2.1 Two-layered, binary classifier with a single output

The two-layer, binary classifier is shown in Figure 5.3. The phase of the output oscillator encodes the classification result: the output oscillator's phase is compared with a reference oscillator's phase, and the phase difference is maximized (minimized) for one (or the other) pattern. The cost function I used was the following binary cross-entropy function:

$$l(y_{pred}, y_{true}) = -y_{true} \log y_{pred} - (1 - y_{true}) \log (1 - y_{pred}),$$

where y_{true} is the real label of the classes, so either 0 or 1 and y_{pred} is calculated as follows:

- First, the phase difference between the output oscillator and the reference oscillator is computed as their correlation, yielding a value between $[-1, 1]$.
- Then this value is scaled to the $[0, 1]$ range to match the loss function's expected range.

If the output oscillator is in phase with the reference oscillator, then y_{pred} is 1; if the output oscillator is out of phase with the reference oscillator, then y_{pred} is 0. Note that this phase calculation yields a continuous number; therefore, the meaning of this phase difference differs when it is close to 0, 1, or 0.5. The goal of binary cross-entropy is to minimize the difference between the predicted and the actual label. In other words, the phase difference quantifies the

likelihood that the input image belongs to class 1. The subsequent decision is straightforward, as y_{pred} can be rounded to 0 or 1, yielding the predicted class for the input image.

Because the BPTT algorithm determines the optimal oscillator couplings, this device does not necessarily function as an associative memory. The phase patterns in the hidden layer are nonintuitive, although they occasionally vaguely resemble the images to be recognized.

That said, without any apparent, clearly visible structure in the hidden layer, the network achieved a worst-case success rate of 98% for the two classes, whereas the average-case success rate was 99%. The predictions made on some images are present on the right-hand side of Figure 5.3.

	All-to-All	**NN**
#Param	38416	1600
Perf. (%)	98	98

Table 5.2: **Quantitative comparison of binary classifiers.** In the column 'Perf. (%)' The worst-case scenario is reported. Note that the reported parameter count for the networks is for the 14×14 images, so the architecture had a 14×14 sized hidden layer. It is encouraging that the nearest-neighbor connected hidden layer performed similarly to the fully connected one.

In Table 5.2, the quantitative comparison of the all-to-all and the nearest-neighbor connected binary classifiers is shown. The nearest-neighbor connected binary classifier performed almost identically to the fully connected one.

5.2.2 10-digit classifier with regular ANN-like structure

A straightforward extension of the binary architecture to a multiclass classifier is to introduce 10 output neurons rather than 1. This way, each oscillator in the output layer corresponds to a single class, and it is straightforward to use standard multiclass cross-entropy as the loss function for training. The schematic of this network can be seen in Figure 5.4. This network was tested with both 7×7 and 14×14 pictures. In Figure 5.4, the 7×7 version is depicted, but the 14×14 version is similar, with 196 oscillators in the hidden layer, instead of just 49.

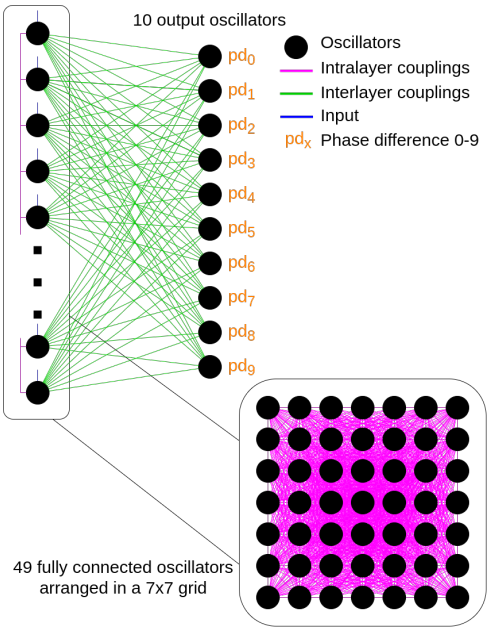


Figure 5.4: **Multilayer network for multiclass classification of MNIST.** The structure is similar to the one shown in Figure 5.3, but the output layer has 10 oscillators instead of just one. Additionally, the hidden layer is flattened here to improve overall network clarity, while its gridlike structure is depicted in the bottom-right corner. The pd_x values represent the phase difference of the output oscillators and a reference oscillator. These pd_x values will be the input of the cross-entropy loss function alongside the real class values.

However, after extensive training, the results were unsatisfactory, with the best-case scenario achieving only 70% accuracy on 10 classes using 14×14 images. To put this number in context, random guessing would be 10%, but the state of the art on MNIST digits exceeds 99% using hundreds of thousands of parameters [125], [126]. This may be because I could not use the entire MNIST dataset for training, as doing so would have required substantial time, but I did not investigate this further. The 7×7 version performed significantly worse, with the best-case scenario at 40%. From this point forward, I worked only with the 14×14 dataset.

As these were not the satisfactory results I hoped for, I proceeded to the different approaches mentioned previously.

5.2.3 10-digit classifier with subnetworks using a winner-take-all aggregator

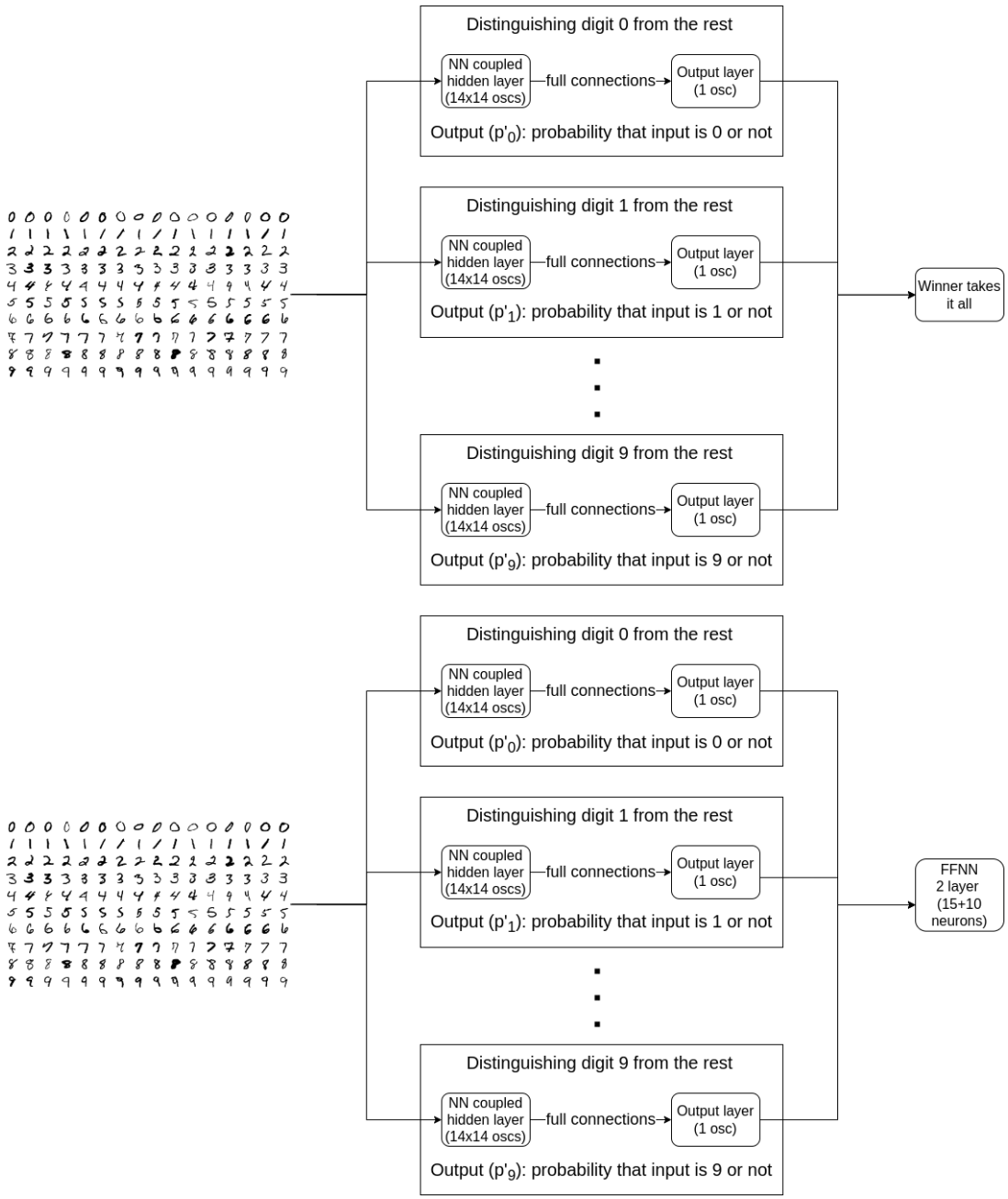


Figure 5.5: **Two of the three tested architectures for the time-independent MNIST classification.** Both consist of individually trained, nearest-neighbor-connected subnetworks, which were designed to distinguish between a single class and the rest of the classes using a binary cross-entropy loss function. The top block diagram illustrates the algorithm: to select the prediction, I take the maximum of the individual network outputs. The more sophisticated version is shown in the bottom block diagram. Here, I used the individual classifiers' output probabilities as inputs to a small, regular FFNN and trained it as a 10-class classification problem using cross-entropy.

Classifying all ten digits is significantly more difficult than the basic binary classification. It requires many more oscillators than the baseline straightforward extension of the binary classifier, as discussed in the previous section. Training numerous oscillators simultaneously is prohibitively

difficult with my method. Instead of training everything at once, I trained ten separate blocks (subnetworks), each responsible for recognizing a particular digit, as shown in Figure 5.5. The blocks themselves are nearest-neighbor connected. The outputs of the individually trained networks—the phase-difference values converted to probabilities—are connected to a winner-take-all function that determines the classification result for the 10-class classification.

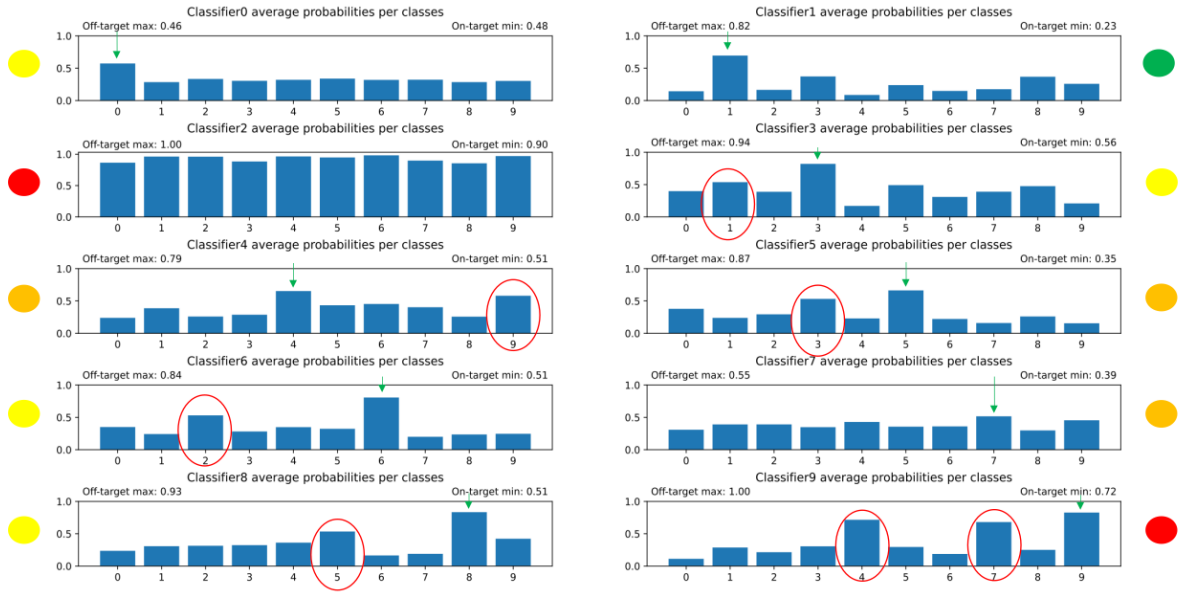


Figure 5.6: **The distribution of predicted average probabilities for the individual, competitive networks in the winner-take-all model.** The red-circled bars are those that, on average, were too high as probabilities because the given subnetwork should not have high values for that specific digit. The green arrows indicate which bar should be the highest. The yellow, orange, and red dots near the plots indicate the extent to which the subnetwork solved its task.

The distribution of the average predictions for each individual in the competitive network is shown in Figure 5.6 after the first round of training. For certain digits, some subnetworks predict a high likelihood of incorrect classes. Using the winner-take-all algorithm (i.e. the decision is made by the ONNs using the 10 output likelihoods from the architectures and the highest one is the winner), I achieved an accuracy around 63%.

After extensive training, I improved the per-class distributions, as shown in Figure 5.7. Using these networks, the accuracy increased to a best-case scenario 70%, which was still not what I wanted to achieve, so I proceeded with the modification outlined in the next section.

All of these accuracies are reported on the test set; the training set was a few percent higher, but it is still not competitive with state-of-the-art solutions.

5.2.4 10-digit classifier with subnetworks using a trained second layer

Instead of a winner-take-all decision, I used a simple multilayer perceptron at the end to improve classification accuracy; see Figure 5.8. It consisted of 2 layers: a hidden layer and an output layer. The hidden layer has 15, and the output layer has 10 neurons. This means that only 325 additional parameters are introduced, which is negligible relative to the roughly 16000 parameters

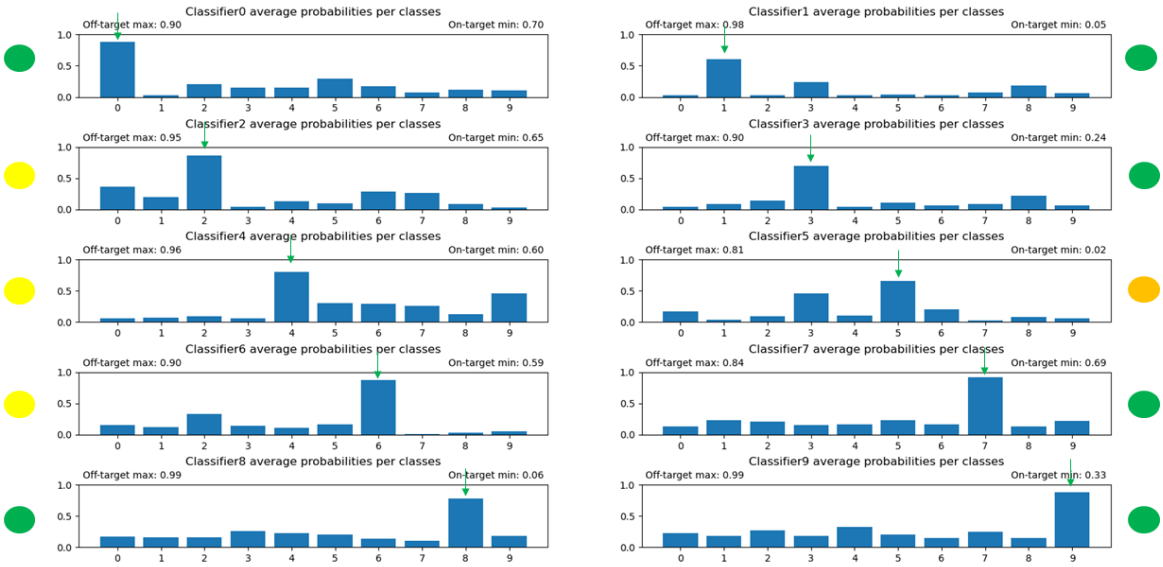


Figure 5.7: **The distribution of predicted average probabilities after intensive training.** It is evident that the in-class distributions are much better now. Apart from some ambiguities, such as in classes 2, 4, and 6, and primarily in class 5, the algorithm appears more robust than the previous version.

of the ONN layers.

I have chosen a traditional Feedforward Neural Network (FFNN) layer to improve accuracy for purely practical reasons. Such an FFNN is easy to train, and I could train it straightforwardly after all the ONN blocks were designed. I emphasize that this conventional MLP layer does not alter our conclusions; the vast majority of the computation remains performed by the ONN network. It is worth noting that there are very few multi-layered ONNs in the literature (a few examples are [127],[128], or [129]).

Additionally, I note that I moved away from a fully connected hidden layer in these competitive networks, and even with nearest-neighbor connections, the network achieved high accuracy.

Using the outputs of the competitive networks as inputs to this small neural network, I achieved 96.7% predictive accuracy on the test set in the best-case scenario. This is excellent accuracy for a network of this size. I implemented feedforward (perceptron) neural networks with the same number of parameters, and these networks typically achieve $93-95$ percent accuracy. While the MNIST problem is solved with relatively trivial networks that achieve accuracy approaching 100%, these networks use hundreds of thousands of parameters. In contrast, the ONN-based network had only approximately 20000 parameters in my training scheme.

I emphasize again that, in terms of computational workload, the heavy lifting in this architecture is performed by the ONN-based preprocessing layer; the output layer contains only a few parameters and is a very small-scale neural network by any standard. The output layer is included because it is easily trainable, enabling it to maximize network performance at low training cost. The ONN accounts for most of the network's power consumption; thus, the entire architecture benefits from the ONN's energy-efficient operation. This result suggests that ONNs

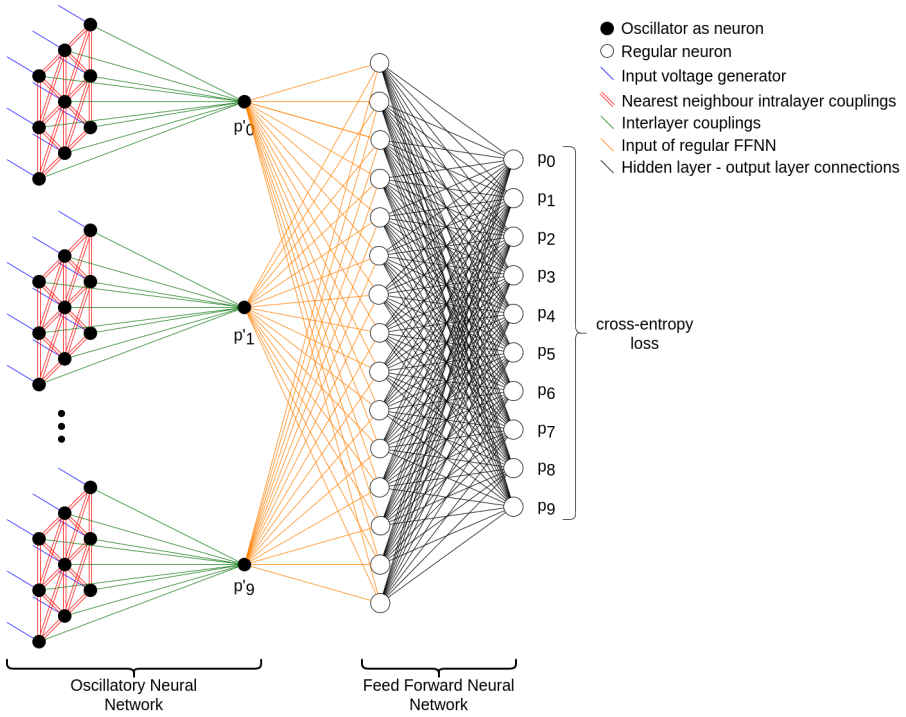


Figure 5.8: **A network with ONN layers as pre-processors and a traditional neural network working tandem**. The easy-to-train output layer significantly improves classification accuracy while keeping the network simpler than state-of-the-art methods. Moreover, the bulk of the work is still performed by the ONNs as pre-processors rather than by the small neural network at the end.

excel as first layers—preprocessing layers—in an AI pipeline.

Integrating Oscillatory Neural Networks (ONNs) with compact traditional neural networks, such as perceptrons, offers a promising avenue to leverage their combined strengths. ONNs can perform complex, dynamic computation, but they are challenging to train. Perceptrons (as described in this dissertation) can be easily trained for specific tasks, but they have limited computational capacity. Placing ONNs close to sensory inputs, where most input data is handled (and where most power is consumed), and refining the computational function to a higher level with an easily trainable layer could harness the best of both worlds and yield the best overall power efficiency for the network.

A similar idea is introduced in Section 6.4.1 where I combine ONN with an ANN.

5.2.5 Comparison of ONN classifier architectures

Table 5.3 summarizes key quantitative findings from my work on multiclass classifiers. Most importantly, the ONN-based network—augmented by the small MLP at the output—outperforms a standard FFNN with the same amount of parameters. This is not entirely surprising for two reasons: first, ONNs are recurrent neural networks that exhibit complex dynamics, unlike FFNNs. The other reason is that ONNs carry information in the phase, frequency, and amplitude of their signals, while a standard neuron outputs only one value (which is usually a static voltage in a hardware realization). One may expect that an ONN, if adequately trained, can perform more complex functions with the same number of neurons (processing units).

	ONN	WTIA	Augmented	MLP
#Param	40180	16000	16325	16363
Perf. (%)	72.3 (70–75)	66.7 (65–70)	95.1 (93–97)	94.4 (93–95)

Table 5.3: **Quantitative comparison of multi-class classifiers.** 'Perf. (%)' is in the format of 'mean (range)'. The Augmented network uses a two-layer MLP as the final decision function. Note that the reported parameter count for the pure ONN version refers to the version with a 14×14-sized hidden layer, because the 7×7 layer performed extremely poorly.

As a back-of-envelope calculation if a hardware similar to [98] is assumed, a single ring oscillator in the ONN circuits I designed would consume about a pJ of power per inference—so the net power consumption of the competitive multi-layered device—with 2000 oscillators assuming that all 10 architectures are running simultaneously—is estimated to be $2 \times 10^{-8} J$ per inference. Highly optimized lightweight hardware neural networks achieve $1\mu J$ per inference for a similar problem [130]. GPU-based networks are usually designed to achieve higher accuracy at much higher power consumption—even if state-of-the-art GPU chips are manufactured using a much more advanced technology than the work of [98]. Overall, these numbers suggest that building the ONNs I studied here via simulation would yield orders-of-magnitude improvements in power efficiency compared with state-of-the-art solutions.

In conclusion, the ONN is not only more economical in terms of parameters but also achieves significantly higher power efficiency than equivalent digital or software implementations.

5.2.6 Overview of the networks from a Machine Learning point of view

One of the most critical aspects of machine learning is model selection for a given task. In my case, model selection for MNIST prediction was an incremental process. As I began with association learning, my entire network was organized into a grid aligned with the input images' dimensions. This enabled me to generate an output image from the network, allowing me to compare the proposed architecture's output with an ideal output. That said, this representation was purely for visualization, as transforming the two-dimensional images into row vectors and using them on a flat layer would yield the same result.

When I moved on to the classification task, I thought it would be natural to retain the same architecture in the hidden layer and add a prediction layer. This architecture somewhat mimics that of standard feedforward neural networks with a single hidden layer and an output layer. This worked well for binary classification, but the approach's continuity for multi-class prediction with 10 output neurons, rather than one, did not generalize. Because I managed to make the binary classification work, I returned to that approach and created 10 competing binary networks, each responsible for distinguishing one specific digit from the rest. From this, I had to compare the networks' outputs. Unfortunately, the classical winner-take-all approach was still insufficient, so I decided to augment the network with a small FFNN, which has proven effective. This idea stems from Convolutional Neural Networks, in which a small FFNN typically sits at the end of the convolutional layers to perform classification after the previous layers have extracted features.

For this, my hyperparameters were the batch and epoch counts, as well as the physical

parameters of the couplings. As I only trained the values of **C'**—related to couplings—and **B'**—related to input amplitudes—, and those parameters were generally between 0 and 2, I had to scale them accordingly to a physical value which is a fixed hyperparameter in the system.

The learning was somewhat limited by the constraints of the training dataset, as BPTT is computationally expensive; therefore, I pruned the dataset to reduce training time. This reduced the architecture's ability to generalize because it had seen only a small number of samples from the training set. I used 1000 from the 60000 available training samples in the MNIST dataset to train the model. This is also an advantage, because the ONNs, thanks to their inherent nonlinearities and dynamics, solved the recognition task with high efficiency using far fewer training samples than a traditional network would require.

6 Static and Dynamic Vowel Recognition

Automatic speech recognition (ASR) has undergone a significant transformation with the advent of deep neural networks and, more recently, self-supervised learning (SSL). Early systems combined handcrafted features (e.g., Mel-frequency cepstral coefficients) with hidden Markov models, but modern approaches increasingly rely on end-to-end neural architectures that learn robust acoustic representations directly from raw or minimally processed speech [131], [132].

Among the most influential architectures is *wav2vec 2.0* [131], which employs a convolutional encoder followed by a Transformer to learn contextualized representations via a contrastive predictive task. With pretraining on large unlabeled corpora and subsequent fine-tuning, *wav2vec 2.0* achieves word error rates (WER) as low as 1.8% on LibriSpeech test-clean. The HuBERT model [132] builds on this paradigm by predicting masked cluster assignments, improving phonetic structure encoding, and leading to strong downstream phoneme recognition performance.

The Conformer architecture [133] combines convolutional modules with self-attention to capture both local and global dependencies, balancing efficiency and accuracy. More recently, Whisper [134] demonstrated the effectiveness of large-scale, weakly supervised pretraining across hundreds of thousands of hours, yielding robust multilingual recognition with models scaling up to $1.55B$ parameters.

Vowels are central to intelligibility, characterized by steady-state spectral formants (F1–F3). In controlled conditions (e.g., sustained vowel phonation), classification accuracy can exceed 90% with convolutional neural networks [135]. However, vowel recognition remains challenging in continuous speech due to coarticulation, speaker variability, and accent effects. Neural models trained for second-language (L2) vowel identification, for example, still show systematic confusions, achieving accuracies around $80 - -85\%$ [136].

At the phoneme level, self-supervised models have significantly reduced error rates on benchmark datasets. On TIMIT, HuBERT has achieved a phoneme error rate (PER) of approximately 10.2%, compared with 13.4% for *wav2vec 2.0* and 22.3% for contrastive predictive coding (CPC) [137]. Moreover, unsupervised approaches such as *wav2vec-U* achieve PER values near 11.3% without labeled data [138].

Model complexity varies widely: *wav2vec 2.0* base contains approximately 95M parameters, the large version over 300M, Conformer models typically range from 10M to 30+M, and Whisper spans from 38M to 1.55B parameters [131], [133], [134]. While increased capacity generally improves robustness, it comes at the cost of memory and inference latency. Efficiency-oriented designs such as [139] explore compact multilingual phoneme recognition suitable for on-device deployment.

Neural networks, particularly SSL-based encoders, have achieved state-of-the-art performance in speech and vowel recognition, reducing PER on TIMIT below 12% and yielding high vowel classification accuracy in controlled tasks. However, real-world complexity—such as domain shift,

coarticulation, and L2 perception—continues to challenge even the most advanced systems.

Keeping all these in mind, I gained some insights into what to expect from my model. The ultimate goal is to use oscillators for the vowel recognition task, with minimal preprocessing of the vowels, to preserve the audio signals as much as possible.

6.1 Vowel Database and Preprocessing

To test my ONN systems on the vowel recognition task, I used a relatively simple, well-tested set of English spoken vowels, as described in [140].

The database I used comprises 12 vowels: “ae”, “ah”, “aw”, “eh”, “ei”, “er”, “ih”, “iy”, “oa”, “oo”, “uh”, “uw”. Additionally, this dataset contained signals with a maximum duration of 1 second for spoken vowels, sampled at 16 kHz. First, I padded all vowel sequences with zeros at the end to ensure an exact duration of 1 second. The dataset includes recordings from both men and women.

These vowels have two to four dominant frequency components. Still, in some cases, their two most dominant components are enough to be extracted and used as pre-processed, synthetic sinusoidal signals as inputs to an oscillator-based architecture. The scatter plot of all the vowels' first two formants can be seen in Figure 6.1. The two distinct regions in which the data points are scattered—the two parallel lines—arise from gender differences among the speakers.

For my simulations, I utilized both the raw waveforms and the formants extracted from them. In each case, the time axis was scaled so that vowel frequencies, initially in the 100–1100 Hz range, were shifted to the 100–1100 MHz range. This scaling was necessary to ensure realistic parameters for the oscillator network, placing its frequencies in the MHz-GHz range.

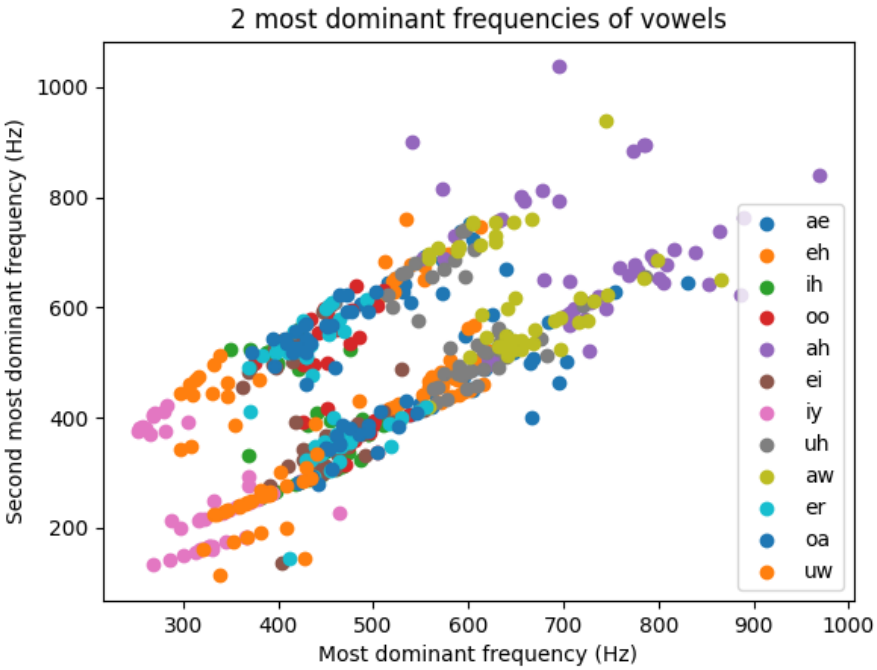


Figure 6.1: **Scatter plot of all the first and second most dominant frequencies of all the vowels in the dataset.** The two different lines the data points are gathered around are possibly arising from the fact that either male or female speakers recorded these vowels, and female speakers tend to have a higher pitch.

The only drawback of this dataset is its relatively small size: it contains only 45 samples per vowel per gender. Therefore, when evaluating the performance of my architecture, this should be kept in mind.

6.2 Frequency-Based Computing

The frequency-based computing scheme I introduce in this section is based on the observation that two oscillators synchronize in frequency when connected to a standard driving signal whose frequency is close to their own. A phenomenon well known in the mathematical literature of synchronization [141], [142].

Specific frequency components in the driving signal can be detected by observing the mutual synchronization of otherwise independent oscillators.

It can be observed that if a sinusoidal signal arrives at one of the nodes of an oscillator, then the oscillator is capable of synchronizing its frequency to the frequency of the input signal, given that the frequency of said input signal is sufficiently close to the principal frequency of the oscillator to which it is connected. This interaction with several oscillators can be seen in Figure 6.2. The oscillator frequencies are grouped in the image, with four groups of six oscillators: frequencies are close to one another within each group but relatively far apart between groups. This figure also depicts the same setup as the hidden layer presented in the next section.

The frequency of an input sinusoidal voltage generator is swept over a frequency range shown on the x-axis. It can be seen that when the frequency approaches that of a group of oscillators, all

the oscillators in that group begin to jump together— in other words, synchronize—in frequency, one by one, until all the oscillators are synchronized to the input signal's frequency.

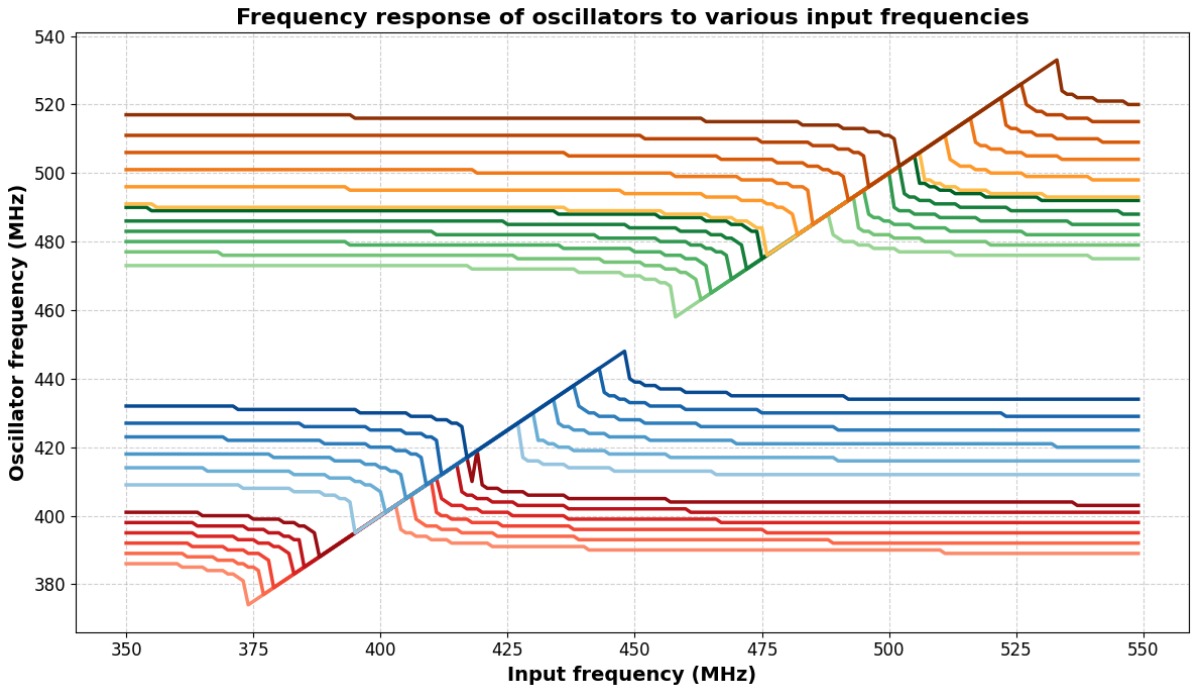


Figure 6.2: **Frequency synchronization of oscillators to input sinusoid with certain frequencies.** A frequency sweep over a range of frequencies for a single input sinusoid voltage generator connected to a 24-uncoupled oscillator network can be seen. The four oscillator groups either synchronize to the input signal or do not, depending on the input frequency. In each group, the oscillators synchronize to the input signal's frequency one-by-one, until all of them are synchronized, then they run together for a specific frequency range, and then they depart in frequency from the rest of the group one-by-one again.

This simple frequency-synchronization phenomenon can be used to construct a two-layered network of oscillators to solve a formant recognition problem.

Note that this synchronization occurs in all types of oscillators, but the range of frequencies to which an oscillator can lock under external stimuli varies across oscillator types. Notably, mathematical models of oscillators, such as Kuramoto oscillators, have a much wider range than electrical-based ones, such as ring oscillators.

6.3 Formant Recognition as a Static Problem

The key to the scheme introduced here is that oscillator groups detect frequency components in the input waveform by synchronizing to the input signal's frequency. This phenomenon can be used to detect audio signals composed of a few dominant frequency components, such as the previously mentioned English vowels.

A key requirement for the operation of this scheme is the proper choice of oscillator frequencies, which match typical vowel frequencies. The frequencies are learnable parameters that can be

determined either by gradient-based or gradient-free optimization techniques introduced in the earlier chapter, or by hand for a simple network.

The first architecture, which I propose to distinguish two vowels based on their two most dominant frequency components, consists of two layers and is shown in Figure 6.3.

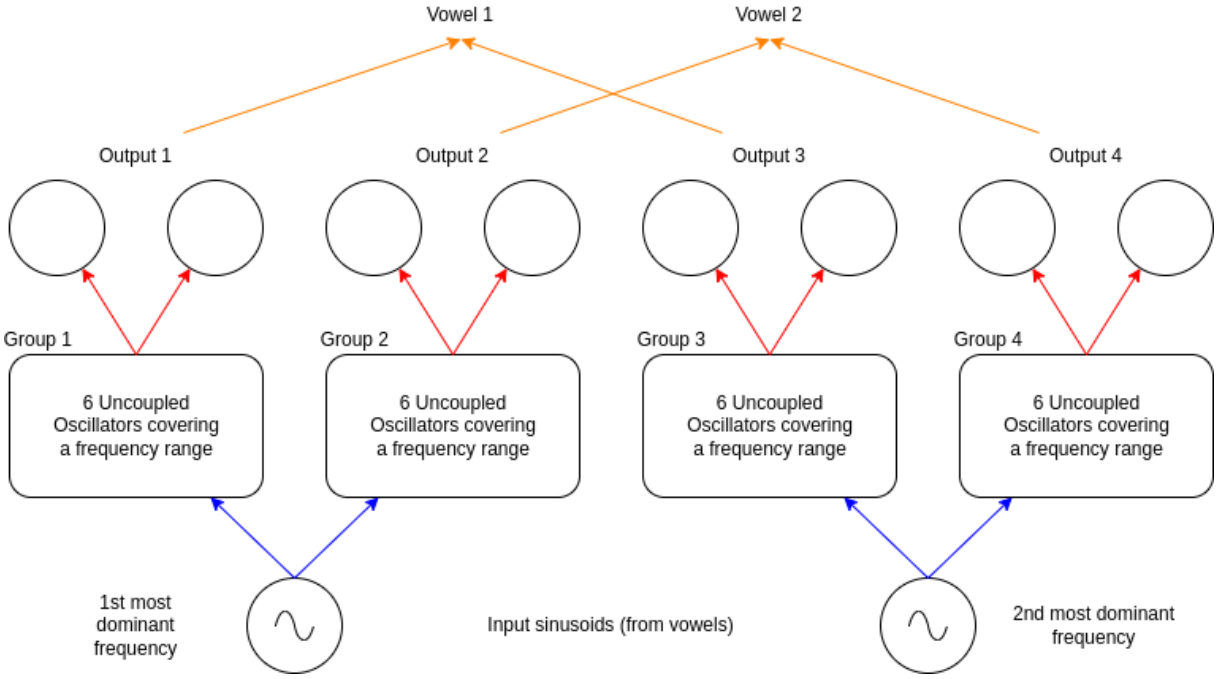


Figure 6.3: **My proposed architecture.** The input signals are represented as sinusoid voltage generators. The frequency of these generators is set by the first and second most dominant frequency components of a given input vowel. These are connected to a single layer of unconnected oscillators, grouped into 4 distinct groups to highlight their collective responsibility for detecting a specific frequency in the input signal. The groups are synchronized in frequency if the input sinusoid lies within the group's frequency range. These groups are then connected to a pair of oscillators covering the same frequency range. If the 6 oscillators in either group are synchronized in frequency, then the pair of oscillators in the output layer will be synchronized in frequency as well. This way, certain frequencies can be signalled by different groups found in the input sinusoids.

The two oscillator layers do not have any intralayer couplings, so no oscillator in a given layer is connected to any other oscillator from the same layer. This prevents spurious synchronization between oscillators within the same layer when the input is insufficient.

The two layers serve two distinct purposes. The first layer extracts frequency components from the input signal, and the frequency synchronization of the six oscillators indicates the presence of a particular frequency in the input. The second layer performs the actual classification step by detecting the pair of formants that are characteristic of the vowel.

First, the two frequency values are converted into two distinct sinusoidal signals at the specified frequencies. These two signals are then fed into the oscillators in the first layer.

The oscillator groups in the first layer of Figure 6.3 are used for detecting the presence of frequency components. The frequencies of these oscillators were evenly distributed within the expected range. Because I used two frequency components and performed binary classification,

there are four possible frequency combinations. The first layer's task is to indicate whether the four frequency ranges are present in the input sinusoids.

This information is then fed "forward" to the second layer. It is not a forward propagation of information, again, as the connections are bilateral. Still, it is easier to understand this way. Each of the four groups in the first layer is connected to two oscillators from the output layer. The mechanism in the second layer is simple. If the six oscillators in the first layer are synchronized in frequency, then the two oscillators in the second layer to which they are connected will also synchronize. In this way, by measuring frequency in the second layer, it is possible to determine which frequency ranges were present in the input sinusoids.

This is a straightforward architecture with a low number of connections. The number of couplings and oscillators is summarized in Table 6.1.

Additionally, I created simulations of this architecture using both the Kuramoto and ring oscillator models to compare the two. As I mentioned previously, Kuramoto oscillators—being purely mathematical models—exhibit a wider range of synchronization than ring oscillators, and I predicted that the operation of Kuramoto oscillators should be smoother.

Also, for the Kuramoto oscillators, I used a modified approach than discussed in Section 3.1.1 as I used the following equation:

$$\frac{1}{2\pi}\frac{d\phi_i}{dt} = \underbrace{f_i}_{\text{Intrinsic frequency}} + \overbrace{\sum_{j=1}^{N} C_{i,j}\sin\left(\phi_j - \phi_i\right)}^{\text{Coupling dynamics}} + \underbrace{\sum_{k=1}^{N} B_{i,k}\sin\left(u_k - \phi_i\right)}_{\text{External dynamics}}$$

The only difference is the way external dynamics are handled. Here, I treated the external signals as "fixed" oscillators, meaning that their connections to the computing oscillators are one-sided: the computing oscillators cannot affect the phases of the external oscillators, but they can freely affect the phases of the oscillators to which they are connected.

	Input	***Layer #1***	***Layer #2***	**All**
# of neurons	2	24	8	34
# of frequencies	0	24	8	32
# of couplings to next	24	48	0	72
# of parameters	24	72	8	104

Table 6.1: **Neuron, coupling, and parameter counts for the proposed multilayer oscillatory network.** The network is lightweight with only 72 couplings—including the inputs' couplings to the oscillators. Note that the network's real parameter count is the sum of the number of oscillators and couplings, as the former sets the oscillators' frequencies and the latter sets the coupling strengths between connected oscillators.

To evaluate the network's performance, I extracted the dominant frequencies of the previously mentioned vowels. The scatter plot of these data points can be seen in Figure 6.4. Both the baseline and scaled frequencies are shown in the figure.

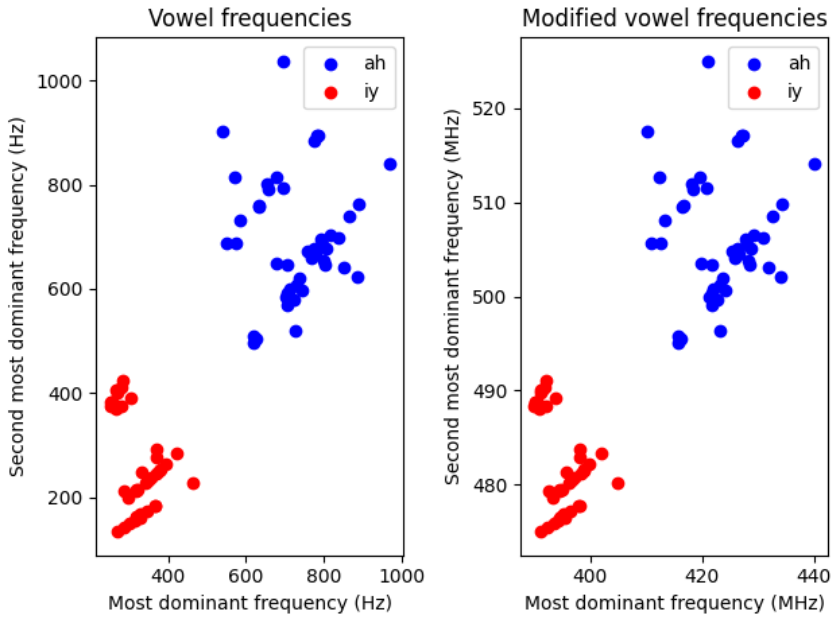


Figure 6.4: **The dataset of vowels "ah" and "iy".** I used this subdataset to test the proposed architecture. This is a linearly separable problem, so the architecture should have no difficulty solving it.

It is worthwhile to note that this network is expected to correctly classify only linearly separable problems, which is the case for the vowels shown in Figure 6.4 above. Also, from the Figure 6.1, there are a lot of pairs that could have been chosen for this task, so the choice of "ah" and "iy" is an arbitrary one.

6.3.1 Choosing the Network Parameters for High-Accuracy Classification

To optimize the network in Figure 6.3, the oscillator frequencies and the coupling strengths between the input voltage generators and the hidden layer, and between the hidden and output layers, must be specified. The coupling strength translates into resistance values in the case of ring oscillators as introduced in Section 3.2.2. In this section, I present the results of this approach using simple considerations.

Since the problem involves four distinct frequency ranges—where vowels can be differentiated based on their primary and secondary dominant frequencies—configuring the frequencies for the hidden and output layers is somewhat easier. I only needed to determine the frequencies of the grouped oscillators that could interact with the input sinusoids. Additionally, I had to select a coupling strength between the oscillators that would avoid signal distortion and prevent overlapping synchronization regions.

In the left subfigure of Figure 6.5, the synchronization regions of the two-oscillator groups in the output layer appear as dips in the corresponding curves. On the x-axis, there is a sweep of frequencies over an extended range of frequencies, incorporating the frequency ranges of the vowel's two most dominant frequency components. It can be observed that the $O1$ and $O2$ groups are responsible for recognizing lower frequencies, whereas the $O3$ and $O4$ groups are responsible for higher frequencies. The synchronization regions do not overlap, which is a direct consequence of the problem being linearly separable.

To measure and algorithmically decide whether any group of oscillators is synchronized or not, I used the following formula:

$$S = \sum_{i,j} (f_i - f_j)^2,$$

where f_i are the frequencies of the oscillators in the given group. This S value is then scaled and used as a dimensionless number, because the magnitude of this difference is irrelevant; only whether it is low for synchronizing cases or high for non-synchronizing cases matters. Note that this criterion was also used later on in Section 6.3.1 for the training algorithm.

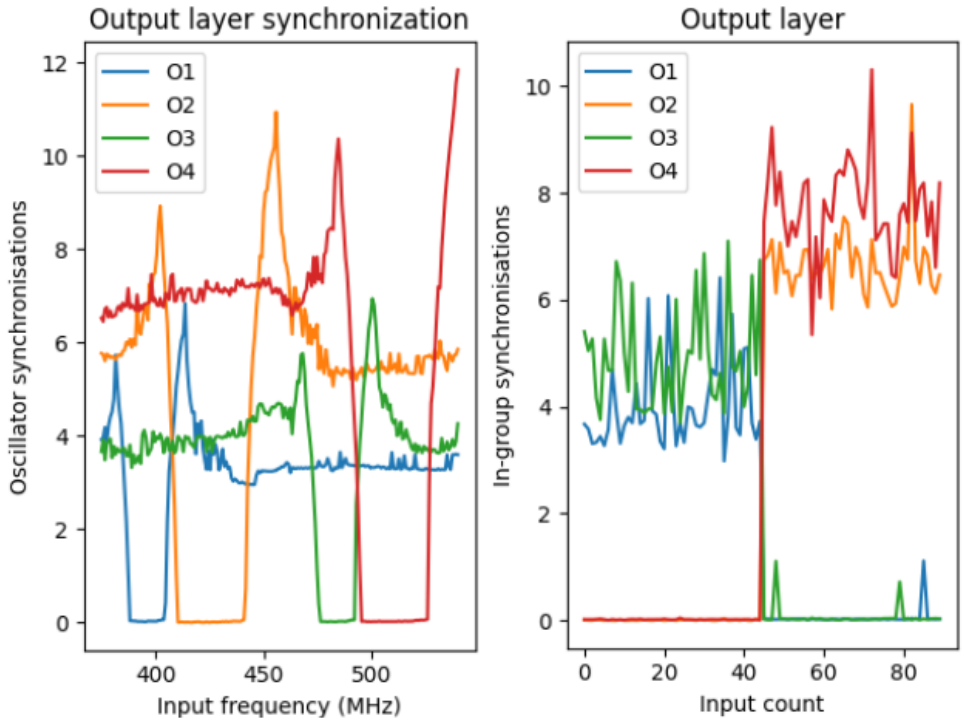


Figure 6.5: **Synchronization values for groups of the ring-oscillator architecture after parameter tuning.** The left subfigure shows the synchronization value of the 4 pairs of output oscillators. The input sinusoids' frequency is swept over a given frequency range, and the synchronization appears as a constant part of the well for the curves. On the right side, the same synchronization value is observed. Still, it is computed after feeding the two vowel types into the system and measuring the synchronization values of the pairs of output oscillators. For the first 45 vowels, being "ah", $O2$ and $O4$ pairs are synchronized, and for the other 45 vowels, being "iy", the $O1$ and $O3$ pairs are synchronized.

In the right subfigure of Figure 6.5, the synchronization values of the four oscillator groups in the output layer are shown as a function of the input indices on the x-axis, using the ring oscillator architecture. The first 45 sample was from the "iy" vowel and the other 45 sample was from the other, "ah" vowel. It is clear that, using this synchronization metric, the decision is straightforward. If the sum of the synchronization value of $O1 + O3$ is lower than that of $O2 + O4$, it can be said that it is an "ah" vowel; otherwise, it is an "iy" vowel. It can be seen that for the first batch of samples, the sum of $O1$ and $O3$ is close to zero, but the sum of the

other two is high; meanwhile, for the other part of the set with the other vowel, it is the other way around.

Additionally, this indicates that the architecture is capable of solving this task with 100% accuracy.

The physical parameters of the ring oscillator network are summarized in Table 6.2.

Parameter	**Physical Value**
C	$1.0 \cdot 10^{-13}$ F
R	$2.0 - 2.7 \cdot 10^{3}$ Ω
R_{in}	$1.0 \cdot 10^{4}$ Ω
R_c	$1.2 - 2.2 \cdot 10^{6}$ Ω
V_{in}	1 V

Table 6.2: **Physical parameters of the ring oscillator network for formant recognition**. All these parameters were set by trial-and-error and used for the simulations.

In my ring oscillator setup, the frequency difference of the oscillators should be around $20 - 40$ MHz. In this case, a coherent, sinusoidal injected signal can synchronize them. The precise value of the frequency difference depends on the input strength; however, the input cannot be too strong, as this would disrupt the oscillatory behavior due to the inherent electrical circuit configuration.

This problem was absent in the Kuramoto oscillators; I could have created much broader synchronization regions, but it was unnecessary, as the ones I made were perfectly capable of solving the task.

Gradient-Free Optimization of Network Parameters

To eliminate the somewhat heuristic procedure above, I used the Nevergrad package [118], version 1.0.8, written in Python, to let the algorithm figure out the parameters of the network, such as the optimal coupling strengths between the Kuramoto oscillators and also their frequencies, the ring oscillator frequencies, and also the value of the resistors between the ring oscillators.

Now, in Figure 6.6, an output layer synchronization—similar to the one in Figure 6.5—can be seen at different points in the learning procedure. The leftmost figure shows the initial state, the middle one an intermediate step during learning, and the last one the state in which the network achieved 100% separation. Note that this learning was conducted only on the ring oscillator network, as that is the primary focus of this dissertation; the Kuramoto model was a proof-of-concept that this methodology can work not only in principle but also in practice. It was never my intention to proceed with Kuramoto oscillators, as I want to propose circuit-based solutions rather than mathematical ones.

For this gradient-free optimization, I used a genetic algorithm to determine the frequencies and couplings.

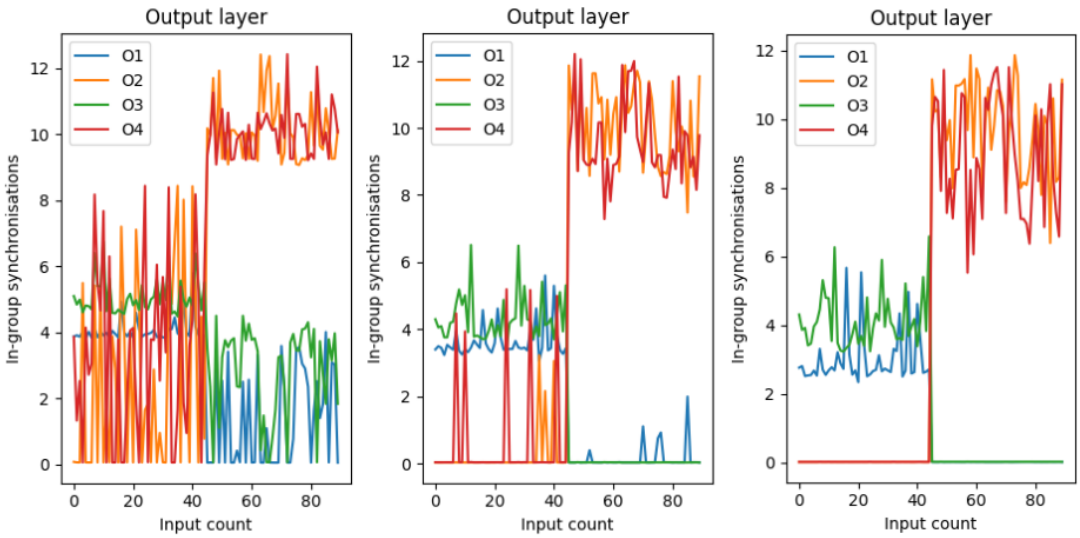


Figure 6.6: **The synchronization value of the output pairs for the test dataset throughout different stages of learning.** The initial state of the left figure is random, then during learning, the synchronization values start to settle to the desired values, and lastly, when the learning is finished, the synchronization values indicate a 100% separation.

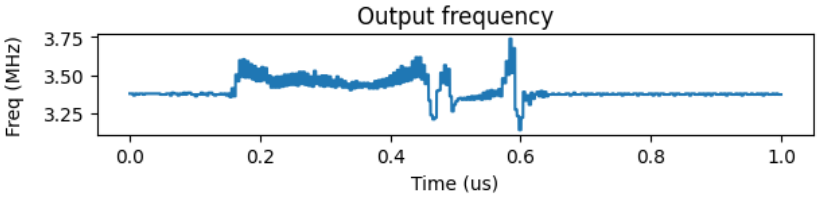
The result is very similar to trial-and-error "by hands" learning, as indicated by the final state of the learning process.

6.4 Time-dependent Vowel Recognition

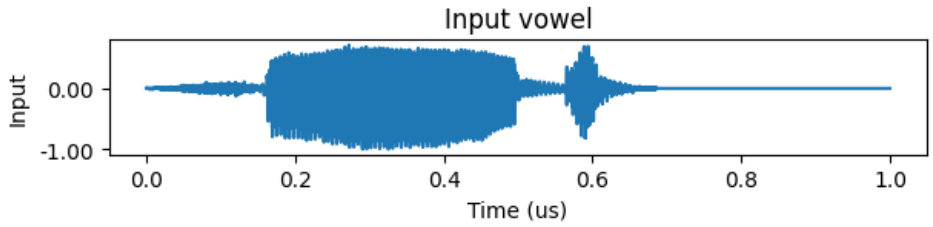
In the previous scenario of Figure 6.3, the oscillators are subjected to a coherent, sinusoidal injection signal. Waveforms, or to be precise, vowels, to be classified in a real-world scenario, are typically a far cry from this: most time series are wideband signals with continuously and abruptly shifting frequencies, and typically, many different frequencies are acting simultaneously on a particular oscillator.

In the circuit model, an arbitrary waveform—such as a vowel waveform—can be straightforwardly applied to the oscillators. I have achieved this by connecting an additional voltage source, via a resistor, to a ring oscillator node.

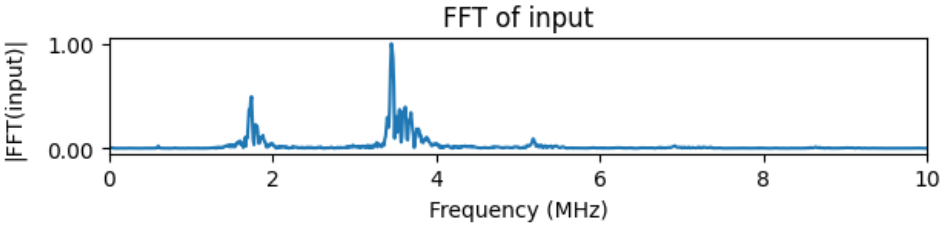
The effect of a vowel waveform on an oscillator is exemplified in Figure 6.7a. The actual waveform is given in Figure 6.7b, and its Fourier spectrum is in Figure 6.7c. As described above, the sampling frequency was scaled to match the oscillator's natural frequency by speeding up the signal from 1s to 100μs.



(**a**)



(**b**)



(**c**)

Figure 6.7: **Effect of a vowel waveform on a ring oscillator's frequency. (a)** The effect of a time-domain waveform on a single oscillator's frequency. **(b)** depicts a single vowel in the time domain. **(c)** shows the frequency domain of the vowel. The most dominant frequency component of this particular vowel is around 3.50 MHz. Note that this vowel has been increased in speed, hence in frequency, by applying it over 100μs instead of $1s$. On **(a)**, the frequency response of a single oscillator can be seen, to which the input vowel has been fed through a voltage generator. The oscillator frequency changes when the input vowel is spoken on the recording, is synchronized to approximately 3.50 MHz, and then returns to its intrinsic frequency of approximately 3.30 MHz. This shows that ring oscillators can respond to external stimuli by modulating their frequency, even when the input is incoherent.

In Figure 6.7a, the instantaneous frequency of the oscillator is shown, which was determined from the time period of the oscillation. The time-dependent waveform of the oscillator is continuously pulled to the dominant frequency of the waveform as long as the vowel waveform is acting on the oscillator.

It is evident that when the vowel is actually spoken in the recording, the frequency of the oscillator is changed and synchronized to the principal frequency of the input vowel for the time being. When the speaker stops saying the vowel, it goes back to the natural frequency of the oscillator.

Clearly, the effect of the oscillator may be viewed as a "filter"—its frequency is pulled when a close-by frequency component appears in the input. Importantly, this effect persists even for

highly incoherent input.

To use this effect for vowel recognition, the given waveform can be applied simultaneously to many oscillators, whose frequencies span the range of frequencies in the vowel. The behavior of an oscillator filter bank for one particular input is shown in Figure 6.8, where again the instantaneous frequencies are plotted as a function of time. It can be seen that groups of oscillators—with closely lying frequencies—jump together when the vowel waveform is acting. Oscillators with farther-away frequencies are affected by the vowel, but they do not sync in frequency. It is important to note that oscillators that frequency-synchronize in this scenario are typically not phase-synchronized, and as a result, their waveforms do not overlap.

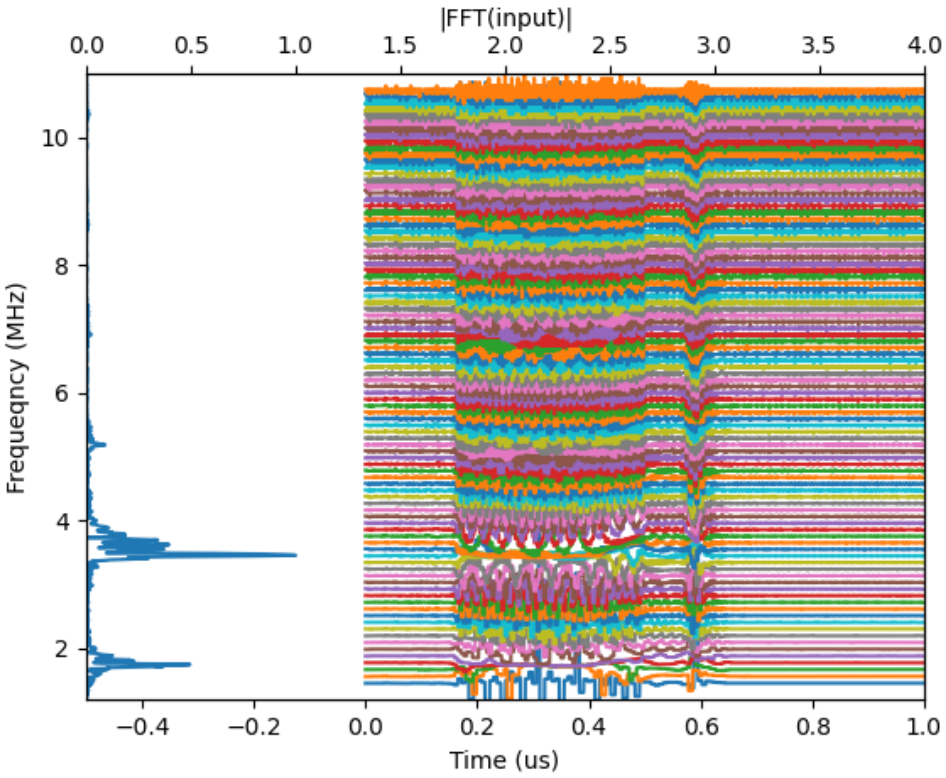


Figure 6.8: **Layer of oscillators frequency response to a single, time-domain input vowel.** The oscillators with intrinsic frequencies close to the frequency peaks are synchronized in frequency to each other for the time the vowel is actually spoken and then return to their original frequencies. Colors are used only for better visibility; they bear no significance otherwise.

Qualitatively, the oscillator layer may be viewed as a filter bank, in which the presence of a particular frequency component is signaled by the synchronization of oscillators around that frequency.

6.4.1 Integrating ONNs with ANNs for Recognition

For some of the results below, I used the ONN layer introduced in the previous section, in conjunction with traditional neural network layers, to improve performance on the oscillator layers.

To train the conventional multi-layer perceptron introduced in the next section, I used standard training methods, including the Adam optimizer and cross-entropy loss. These were implemented

in *PyTorch* [111]. Due to the small size of this added network, the learning process was fast and lightweight.

In Figure 6.7 and in a previous section, it was demonstrated that applying the vowel waveform directly to the oscillator has a strong effect on its frequency, even if this effect cannot be described as a clean phase and frequency synchronization to an injected signal. However, the frequency correlations between neighboring oscillators can be detected using appropriate circuitry.

To process raw waveforms, I used a single layer of oscillators whose frequencies spanned the vowel's spectral range, as described earlier. Unlike in the network of Section 6.3, I used a small traditional neural network as a second layer for classification. Oscillators act only as preprocessing layers. The schematic of this architecture is shown in Figure 6.9. This is reminiscent of Deep Convolutional Neural Networks again, as was the case for the architecture introduced in Section 5.2.4.

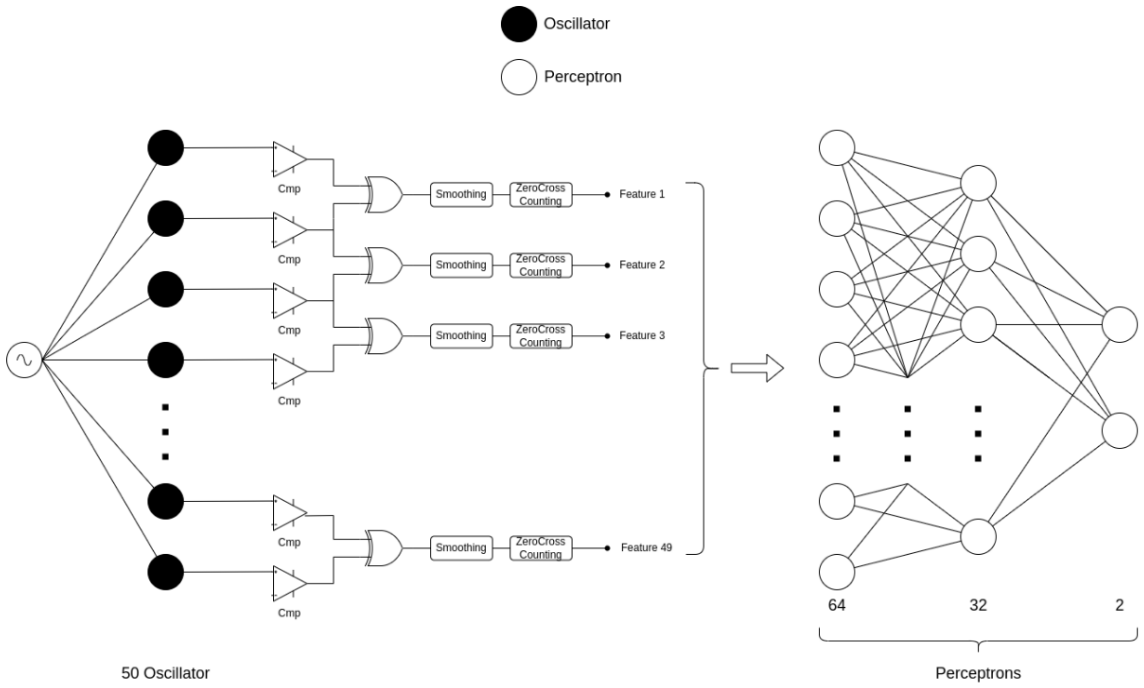


Figure 6.9: **Mixed architecture for time-domain binary vowel recognition.** The first part of the architecture extracts the frequency features of the time-domain vowels by measuring the frequency synchronization strength of neighboring oscillators. The second part uses this feature set as input data to perform the actual binary classification with a simple multi-layer perceptron architecture.

The first part of the architecture—shown on the left side of Figure 6.9—is entirely oscillator-based. Its primary function is to extract frequency components from the time-domain vowel input by measuring the synchronization strength between neighboring oscillators. Each oscillator's output voltage is first converted into a binary signal using comparators. Then, the outputs of adjacent oscillators are passed through XOR gates, which operate as follows:

- If two oscillators do not have synchronized frequencies, then the output of their XOR's duty signal will be constantly changing.

- If two oscillators are synchronized in frequency, then the output of their XOR's duty signal will be constant over time

The output of the XOR gates passes through a low-pass filter. A circuit can perform this straightforwardly; in my simulations, I used a moving-average filter to emulate its effect.

To create a single feature vector from the time-dependent filtered XOR signal, I counted its zero crossings. Here, if the duty signal is constant—indicating synchronized oscillators —, fewer zero crossings will be observable. Otherwise, if the duty signal is constantly changing for two non-synchronizing oscillators, then the number of zero crossings will be high. The detailed changes in the signals coming from the oscillators as output voltages passing through this apparatus are depicted in Figure 6.10.

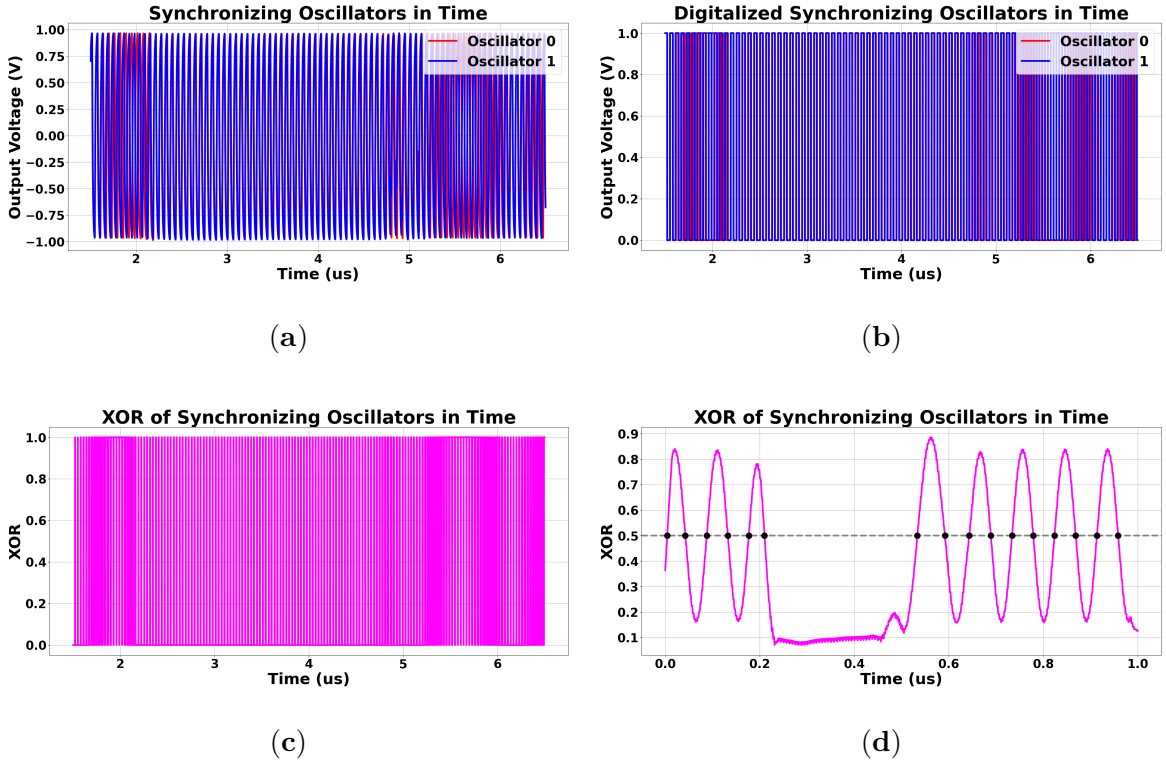


Figure 6.10: **Transformations on the output signals of two neighboring synchronizing oscillators.** The transformations conducted on the two oscillator signals via the small circuit after the oscillators can be seen on (**a**) to (**d**): (**a**) output voltages of synchronizing oscillators near the synchronization time frame, and it can be observed that in the synchronizing region the two oscillators frequencies are not changing relative to other, but before and after that oscillator 1 is constantly moving over oscillator 0 as the higher frequency oscillator; (**b**) after digitizing the output signals, two square signals can be observed; (**c**) the XOR of the synchronizing signals shows similar behavior than (**a**) as the XOR is changing with constant duty signal in the synchronization region but outside of it the duty signal's length is constantly changing; (**d**) after applying a mean filter for the whole XOR signal, the synchronization region can be seen clearly and if zero-crossing are counted on this signal it will result in a lower value than if this synchronization had not been present as it can be seen that outside of this region the smoothed XOR signal is sinusoid-like. Note that I inverted this value and transformed it to the $[0, 1]$ range to obtain the relative signal strength. In this way, it is more intuitive to refer to it as synchronization strength.

An example illustrating how oscillators respond to a time-dependent input is shown in Figure 6.11. It is clear from the figure that the oscillators with intrinsic frequencies close to the frequency components in the vowel are synchronized, and the rest are not. On the right, however, the relative frequency synchronization strength between the oscillators is evident. For clarity, the zero-crossing values have been normalized to the [0, 1] interval and reversed to indicate better the locations of the frequency peaks in the vowel.

I first used the oscillatory architecture to extract the mentioned features from the two vowel sets. After that, I used the resulting dataset as input to the aforementioned low-parameter-count multi-layer perceptron. It used 49 inputs and 2 outputs, with 64 neurons in the first hidden layer

and 32 in the second.

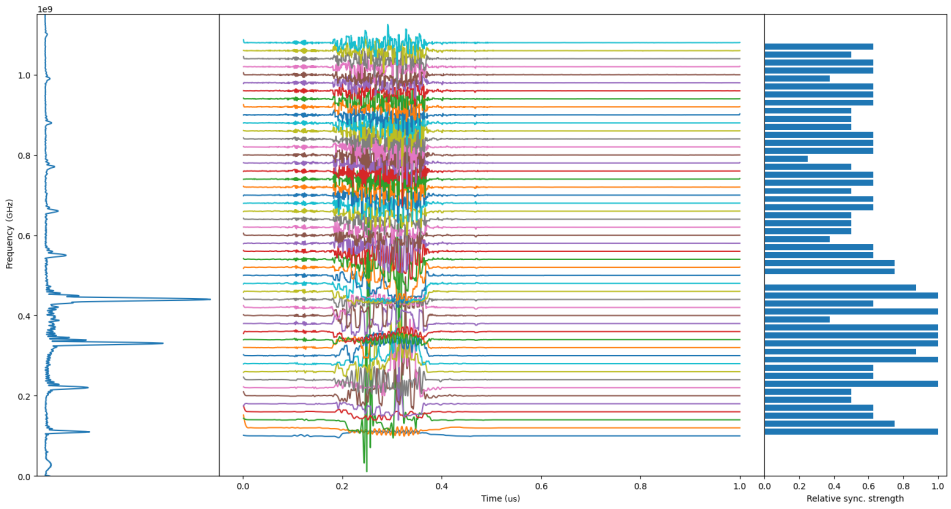


Figure 6.11: **FFT feature extraction with oscillatory layer**. The feature vector element values are high when synchronization is present between two oscillators and low when it is absent, as desired. The frequency spectra of the vowel are on the left, rotated 90 degrees clockwise, for clarity. The colored curves are the instantaneous frequencies for each oscillator.

Performance on Binary Datasets

Due to the small dataset and the simple architecture, training and evaluation are fast.

The confusion matrix for the test set evaluation is shown in Figure 6.12.

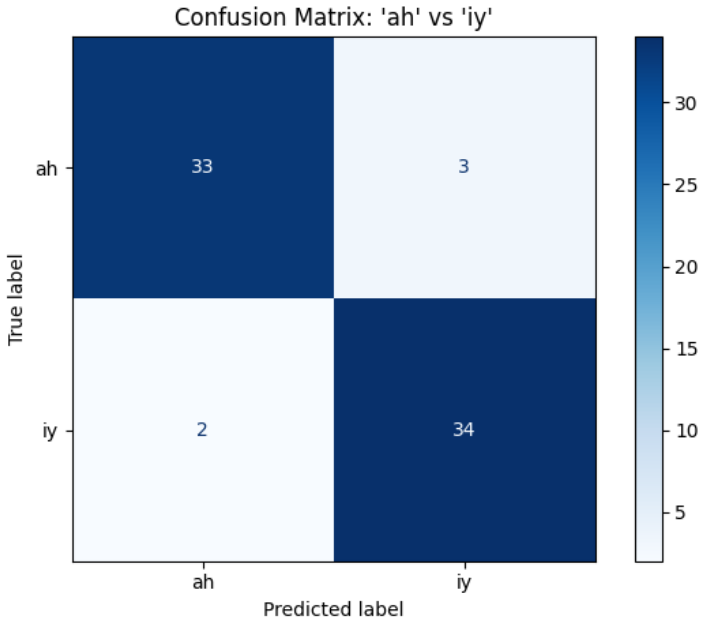


Figure 6.12: **Confusion matrix of "ah" and "iy" vowels with the mixed architecture.** The architecture made some errors, but overall its accuracy is 93%.

Overall, in terms of accuracy, it performed slightly worse than the multi-layered oscillatory network because the input signals were more heavily pre-processed for that network. In contrast, here, preprocessing is performed by the oscillatory layer. Note that the multilayer perceptron is a simple network but is prone to overfitting because the dataset I used was small, as discussed earlier.

Regarding the physical parameters, all oscillators had the same input resistor value of 2.5 $k\Omega$. Here, the remark from the end of Section 6.3.1 is valid with the added restriction that the difference between the frequencies of neighboring oscillators should be at most $20 - 25$ MHz, so that $2 - 4$ oscillators could synchronize in frequency to the incoherent input signal for the duration of the speech.

7 Controlling Continuous-Time Hopfield Networks with Time-Dependent Data

In the literature, time-domain signals, such as audio signals, are either treated as 1-D images[143], [144], [145] or converted to other kinds of static representations, like Mel spectrograms[146], [147], [148]. Although studies employing these approaches report promising results, they have some drawbacks. The former discards time evolution from its methodology. Meanwhile, the latter employs extensive signal preprocessing to create a high-dimensional image, thereby enabling static image recognition. One of the main arguments for using oscillators is that their inherent dynamics can process dynamic data, thereby allowing preprocessing to be more lightweight or omitted altogether. Additionally, because oscillators can be built using circuits, as shown in earlier chapters, it may be possible to feed sensor data directly into the oscillatory circuit. This would further simplify the system, as the need for analog-to-digital conversion would not be critical.

The most challenging part of transitioning from static data processing with dynamic models to using time-dependent signals is determining how to effectively incorporate time-domain data into the dynamic system for the task at hand, such as classification.

To make this jump, I first had to analyze the effects of such signals on small, simple systems, such as Hopfield neural networks. To do that, I devised a simple experiment as follows. I constructed a two-neuron Hopfield network as a baseline and fed it various time-dependent signals to better understand their effects.

Our network for this task is shown in Figure 7.1. This network has two stable states that can be used for binary classification. I have chosen this architecture because its phase portrait is easy to visualize, being two-dimensional. This enabled me to observe and track the system's trajectory as I simulated it over time under various input conditions.

Mathematically, a continuous-time Hopfield network's electric circuit equivalent can be described by the following ODE [149]:

$$C\frac{du}{dt} = TV - \frac{u}{R} + BI$$
$$u = g^{-1}(V),$$

where $g(u)$ is in the following form:

$$g(u) = \frac{2}{\pi}\arctan\frac{\pi u}{2}.$$

The dynamics of this network are governed by the following energy function [149]:

$$E = -\frac{1}{2}\sum_i \sum_j T_{ij} V_i V_j + \sum_i \frac{1}{R_i} \int_0^{V_i} g_i^{-1}(V) dV + \sum_i (BI)_i V_i.$$

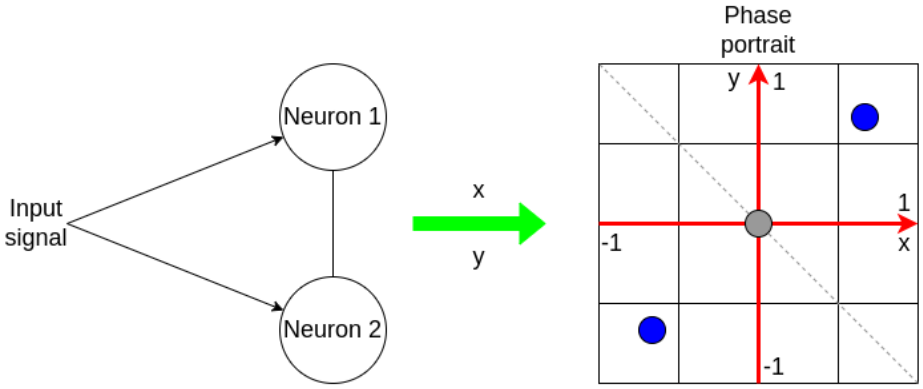


Figure 7.1: **Hopfield network with 2 positively coupled neurons**. This network has two stable equilibria, denoted by blue dots. The gray dot corresponds to an unstable equilibrium at the origin. The dashed gray line separates the two convergence regions. If the initial state lies above this line, the system converges to $(+1, +1)$; if it lies below this line, the system converges in the opposite direction.

The system depicted in Figure 7.1 has two stable equilibrium points, onetowardthe point $(+1, +1)$ and onetowardthe point $(-1, -1)$, colored blue in the figure. The unstable equilibrium point is gray and is at the origin. The separator between the two stable regions is the $y = -x$ line, depicted as a dashed gray line. Note that for the system to have these equilibrium points, the two Hopfield neurons should be positively coupled.

If there is no input to the system, the initial state determines the final state, which depends on its proximity to the stable points. However, if a constant input is present, the energy landscape is distorted and may differ; this is not the focus of our work, as it corresponds to a basic continuous-time Hopfield network.

I was particularly interested in time-dependent inputs. This means that I cannot precisely formulate an energy function that governs the system. However, it is possible to create pseudo-initial states by allowing the input to affect the system for part of the simulation, then turning it off entirely and allowing the autonomous system to evolve independently. In this way, the region of attraction of the two stable states can be slightly modified or even completely changed.

7.1 Static input with different application time

First, in Figure 7.2, I introduce the effect of a constant input of $I = (-1, -1)$ to the system, but for different amounts of time. The initial state of both simulations was the same, as indicated by the top-left corner, from which the red trajectories were started. This image was taken at the same simulation time point; however, for the right trajectory, the input signals were turned off earlier. This resulted in the system not crossing the gray line; when the input was turned off, the system continued evolving and converged toward the top-right corner at $(+1, +1)$. In the right

figure, the input had been turned off later, and the input managed to push the system through the gray line, so when the input got turned off, the system continued going down toward the bottom left corner, toward the point $(-1, -1)$.

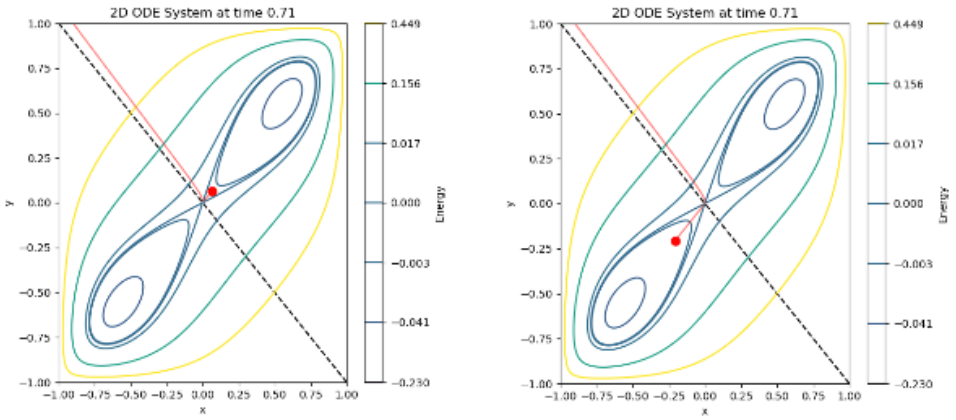


Figure 7.2: **Different convergence points of the system with the same input applied for different amounts of time.** Both systems started from the same initial state, but ended up in different stable points due to the difference in the time the input signal was applied to the two systems. The energy contours are also present in the system, corresponding to the autonomous system, even in the absence of input.

7.2 Learning dynamically drawn circle directions

Now, using this knowledge, I decided to create synthetic circles as inputs with the following equations:

$$x(t) = 0.5\cos(\pm 2\pi f t + \varphi)$$
$$y(t) = 0.5\sin(\pm 2\pi f t + \varphi),$$

and I fed it to the system as $I(t) = (x(t), y(t))$.

The different parameters in these circles were their direction $(\pm)$ and initial phase (φ). My goal with this dataset was to determine appropriate parameters for the system —specifically, the coupling parameter $T_{ij} = T_{ji}$, the coupling strength B, and the system's RC time constant. I did this using Backpropagation Through Time (BPTT), which I also used in the earlier works presented in this dissertation.

Due to the small size of the network, it was easy to train and performed surprisingly well, as shown in Figure 7.3. It reached almost 100% in determining whether the circle is moving clockwise or counterclockwise.

Interestingly, the system was only able to reach this high level of accuracy if the initial states were somewhat restricted to a maximum π *rad* long interval, irrespective of it being $[0, \pi]$, $[\pi/2, 3\pi/2]$, or others, but only restricted to this size. It is possible that adding more neurons to this network would resolve this issue, but it would defeat the purpose of using a small system to learn how time-dependent data affects dynamic systems.

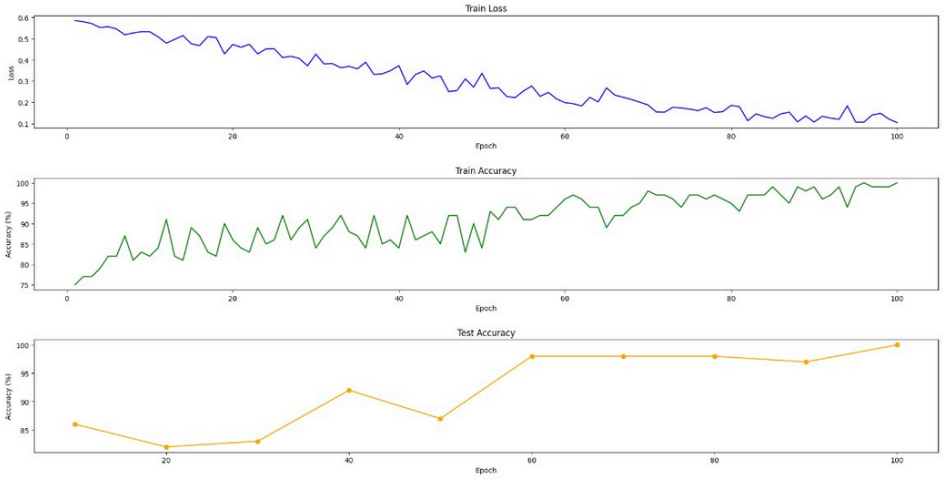


Figure 7.3: **Learning curve and accuracy on test-training sets throughout the epochs for the circle direction classification**. The system learned with very high accuracy which way the circle's drawing is oriented.

This is a crucial step in determining how to control dynamic systems with time-dependent data, but it still requires numerous experiments to be fully resolved.

8 Summary

In this chapter, I briefly summarize the dissertation and dedicate a section to the new scientific results and thesis points.

Computations have been continually shifting since their introduction, as the need for better, faster, and more efficient computers has been high and will remain so. The long-standing standard of Boolean-logic-based computers built on the von Neumann architecture is approaching its physical limits. Hence, researchers either had to circumvent those limitations or even exploit them—in the case of quantum computers. Another area actively researched today is analog computing. The rise of these machines coincided with the emergence of brain-inspired computers that exhibit oscillatory behavior and can be designed using physics for computation, making them inherently analog. This way it is expected to lower the enormous computational—and in tandem, electrical—cost of current trainable networks to create low-power computers for edge AI.

Throughout this dissertation, I introduced a method for designing Oscillatory Neural Networks based on ring oscillators using conventional machine learning algorithms. I presented networks based on different synchronization criteria—frequency and phase—and outlined the next step toward using neural networks as circuits connected directly to sensors, serving as time-dependent processing units.

I want to emphasize that ONNs—and, by extension, neuromorphic computing as a whole—can be key to an energy-efficient solution for edge AI applications.

8.1 New Scientific Results

Thesis I.

I have developed a computational framework for designing oscillatory neural networks composed of electrical-circuit components and for training them using Backpropagation Through Time (BPTT). I have created two simple architectures: one for wave generation and one mimicking a perceptron, to demonstrate that Oscillatory Neural Networks (ONNs) can implement such devices.

This part of the work was implemented in *PyTorch* to simulate the dynamics of the oscillatory neural network circuit and to train the machine learning algorithm to infer the physical values of the circuit components. Due to the limitations of BPTT—namely that it represents the incremental, numeric solution of the dynamics described by the set of Ordinatry Differential Equation (ODE) of the network as a deep neural network having hundreds or thousands of layers—I simplified the implementation of inverters to a mathematical function, $f(v) = -\tanh(a \cdot v)$, to facilitate learning. The Figures 3.3,3.4,4.1 show how this procedure works in

practice.

Using this framework, I formulated a wave-generation task for 5 oscillators, employing MSE as the loss function. This resulted in near-perfect convergence, and although the problem was simple, this has never been achieved before. This is presented in Figure 4.2.

Additionally, to test whether conventional neural network components can be reproduced using Oscillatory Neural Networks, I created a simple two-dimensional problem in which I had to separate two blobs of data. To that end, I used 4 oscillators to mimic the behavior of a standard software-based perceptron. The 4-oscillator system I created performed well, achieving nearly 100% accuracy on both the training and test sets. The results are shown on Figures 4.3-4.4.

The framework can simulate and train not only ring-oscillator-based neural networks but also other oscillator types, such as the Kuramoto oscillator.

Publications related to this thesis are: [J1], [C1].

Thesis II.

I have designed an Oscillatory Neural Network-based associative memory trained using the Backpropagation Through Time method with both nearest neighbor and fully connected coupling schemes. Both achieved lower Mean Squared Error (MSE) on the given set than a network trained by Hebbian Rule.

This part of the work was conducted to compare ONNs trained with BPTT to ONNs trained with the Hebbian learning rule. One of the most limiting factors for new neuromorphic chips is connectivity. Most such architectures use an all-to-all coupling scheme, which is infeasible to implement physically beyond a certain number of oscillators. To circumvent this limitation, I created not only a fully connected ONN but also a nearest-neighbor connected one by arranging the oscillators on an $N \times N$ grid matching the size of the MNIST images and connecting each oscillator only to its nearest neighbors. In Figure 5.1, this particular setup is visualized.

For this problem, I used a subset of the handwritten digits from the MNIST database and scaled them to either 14×14 or 7×7 images, as the BPTT method can have memory issues with large networks.

The results were promising: both the fully connected and nearest-neighbor connected BPTT-trained ONNs outperformed the Hebbian-rule-trained ONN, demonstrating that it is not necessary to use an all-to-all coupled system. Not only did the BPTT-trained networks outperform in terms of quantitative accuracy, meaning that the resulting patterns from the BPTT-trained networks were closer to the real patterns in terms of MSE, but also in terms of parameter count, as the nearest neighbor connected ONN used much fewer parameters than the fully connected, Hebbian-rule-based network. This is mainly because the Hebbian rule assumes all-to-all connectivity. Quantitative comparisons of the results are presented in Table 5.1, and qualitative comparisons are shown in Figure 5.2.

Publications related to this thesis are: [J1], [C1]

Thesis III.

I have designed an ONN-based architecture for binary and multiclass classification on the MNIST dataset. I have created a two-layer ONN for binary classification of the 0 *and* 1 *classes, with all-to-all and nearest-neighbor couplings in the hidden layer, and compared the results. Managed to reach, on average,* 99% *accuracy and observed no consistent patterns in the hidden layer, confirming that the algorithm learned structures that are not necessarily recognizable to the human eye. I have also developed several purely ONN architectures for multiclass classification on the MNIST dataset and compared their performance with a standard neural network. In terms of accuracy, it is behind State-of-the-Art (SOTA) solutions, but in terms of energy efficiency, ONN-based approaches pull ahead.*

Although state-of-the-art solutions for MNIST classification achieve over 99% accuracy, they often have hundreds of thousands, or even millions, of parameters. To reduce this number, we can leverage ONN dynamics.

Classification is a standard task in artificial intelligence. Dynamic systems, such as oscillatory neural networks, may not be the best choice for classifying static data. This is mainly because, in that case, the oscillators' inherent dynamics are not fully utilized. That said, classification remains possible using ONNs.

In Figure 5.3, it can be seen that a network that resembles regular artificial neural networks but assumes bilateral flow of information can be used to distinguish between binary classes of MNIST. On the figure, I also showcased the ability of the network to classify 0 and 1. It was not always true, but sometimes a humanly recognizable structure existed within the hidden layer. The quantitative performance of the net was excellent, achieving 99% prediction accuracy on the test set.

For multiclass prediction, I tried two approaches: a regular ANN-like architecture, as shown in Figure 5.4, and a distributed, binary-classifier-based solution with ten competing networks. The logic of the latter can be seen in Figure 5.5.

Unfortunately, the performance of purely ONN-based networks did not reach the desired levels, so I moved on to a combination of ONN and an Artificial Neural Network. The comparison of such networks can be seen in Table 5.3.

Publications related to this thesis are: [J1], [C2], [C3]

Thesis IV.

I have created a frequency-based oscillatory neural network using both the Kuramoto and ring-oscillator-based models. Managed to tune the parameters of the networks either manually or with the utilization of a gradient-free optimization method. My network was capable of solving the task at hand with 100% *accuracy using both models and parameter tuning schemes.*

In literature, oscillators are mainly used as phase-based computers. That said, they can be used as frequency-based architectures that exploit their frequency-synchronization property, as depicted in Figure 6.2.

Building upon this principle, it is possible to design an architecture, depicted in Figure 6.3, with a limited number of parameters, as can be seen in Table 6.1, and it can be used to distinguish

between two vowels represented by their two most dominant formant frequencies.

I successfully distinguished between the vowels "ah" and "it" with 100% accuracy using this network, further demonstrating that a frequency-based architecture is feasible.

Additionally, it provided a useful parallel to the phase-based perceptron I introduced earlier, as this network is somewhat equivalent to that architecture.

The tuning of the parameters, as shown in Table 6.2, is performed using both manual tuning and a gradient-free optimization method to find the optimal parameters of the oscillatory neural network. This method was employed to test its capabilities because the BPTT method can pose significant challenges when the architecture reaches a point at which gradient-based methods would consume an enormous amount of memory.

Publications related to this thesis are: [J2], [C4].

Thesis group V.

I have successfully demonstrated, in two tasks—MNIST and vowel recognition—and with both phase- and frequency-based computing, that ONNs can be used effectively as preprocessing layers when combined with standard neural networks. This emphasizes the power of ONNs in terms of energy efficiency and the ability to handle large-scale problems.

Unfortunately, pure oscillator-based networks did not perform well enough. However, combining the ONN with a standard ANN can achieve performance comparable to standard neural networks.

Thesis V.1: *The best ONN network I designed for the MNIST classification task was combined with a standard ANN, yielding a hybrid ONN-ANN architecture that achieved an average accuracy of* 95.1% *on a 10-class classification task. Comparing this to a regular, an FFNN with the same parameter size achieved an average* 94.4% *accuracy on the same problem.*

Due to the purely oscillator-based network's inability to reach a high percentage on the MNIST dataset, even with the distributed, competing networks and the winner-take-all algorithm, I decided to move forward with an approach borrowed from Convolutional Neural Networks, meaning that after a couple of layers, there is a fully-connected multi-layered perceptron to take care of the classification.

The architecture I used is shown in Figure 5.8, and a comparison of all multiclass classification networks is presented in Table 5.3.

I was able to reach at most 96.7% classification accuracy with this combined network. In comparison, a similarly sized standard neural network achieved at most 95% accuracy, which is a strong result given that the network was trained for only a few epochs due to computational constraints and was very small in terms of parameter count.

Publications related to this thesis are: [J1], [C2], [C3]

Thesis V.2: *I have created a mixed network using ONNs as preprocessors and a Multi-layered Perceptron (MLP) as the output layer to recognize spoken vowels from their original time-dependent waveforms. On the problem set, the algorithm achieved* 95% *accuracy, and the main advantage of this solution is that the network is low-complexity and computationally*

lightweight. This is because the ONN layer and the transforming layer that converted the ONN's output signal to static data were built using only simple electrical components.

Building on the success of the combined ONN and ANN, I created a network to classify vowels. For this task, I used time-dependent vowel waveforms rather than the dominant formants. The first, oscillatory layer was responsible for extracting frequency information from the vowels by creating a filter-like layer.

This frequency representation is a downsampled version of the signal's FFT, and the features associated with a single vowel have been extracted using only simple electrical circuit components, as shown in Figure 6.9. The feature extraction process is shown in Figure 6.10, and the resulting features are demonstrated in Figure 6.11.

I tested the same vowels, "ah" and "it", and achieved 95% classification accuracy. This is a good result, as most of the literature treats time-dependent audio signals as either one-dimensional sequential data or converts them into representations such as Mel spectrograms and then uses image recognition networks.

The approach I presented offers a new way to reduce the complexity of such classifiers and paves the way for architectures that can receive data directly from sensors.

Publications related to this thesis are : [J2], [C4]

List of author publications

List of Journal Publications

[J1] T. Rudner, W. Porod, and G. Csaba, "Design of oscillatory neural networks by machine learning," *Frontiers in Neuroscience*, vol. 18, p. 1 307 525, Mar. 2024. DOI: 10.3389/fnins.2024.1307525 (cit. on pp. 68–70).

[J2] T. Rudner-Halász, W. Porod, and G. Csaba, "Oscillator-based processing unit for formant recognition," *Information*, vol. 16, no. 7, 2025, ISSN: 2078-2489. DOI: 10.3390/info16070611. [Online]. Available: https://www.mdpi.com/2078-2489/16/7/611 (cit. on pp. 70, 71).

List of Conference Posters and Presentations

[C1] T. Rudner, G. Csaba, and W. Porod, "Design of oscillatory neural networks by machine learning algorithms," in *International Workshop on Computational Nanotechnology*, Conference presentation, 2023 (cit. on p. 68).

[C2] T. Rudner, W. Porod, and G. Csaba, *Design of oscillatory neural networks by machine learning techniques*, Poster presented at the Workshop "Frontiers of Neuromorphic Computing", 2023 (cit. on pp. 69, 70).

[C3] T. Rudner, W. Porod, and G. Csaba, *Design of oscillatory neural networks by machine learning*, Poster presented at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, 2023 (cit. on pp. 69, 70).

[C4] T. Rudner, W. Porod, and G. Csaba, *Training oscillator networks for neuromorphic computing*, Poster presented at the International Conference on Neuromorphic Computing and Engineering, 2024 (cit. on pp. 70, 71).

References

[5] G. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998. DOI: 10.1109/jproc.1998.658762 (cit. on p. 1).

[6] G. E. Moore, "Progress in digital integrated electronics [technical literature, copyright 1975 ieee. reprinted, with permission. technical digest. international electron devices meeting, ieee, 1975, pp. 11-13.]," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 36–37, Sep. 2006, ISSN: 1098-4232. DOI: 10.1109/N-SSC.2006.4804410 (cit. on p. 1).

[7] C. A. Mack, "Fifty years of moore's law," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, no. 2, pp. 202–207, May 2011, ISSN: 1558-2345. DOI: 10.1109/TSM.2010.2096437 (cit. on p. 1).

[8] D. E. Nikonov and I. A. Young, "Overview of beyond-cmos devices and a uniform methodology for their benchmarking," *Proceedings of the IEEE*, vol. 101, no. 12, pp. 2498–2533, Dec. 2013, ISSN: 1558-2256. DOI: 10.1109/JPROC.2013.2252317 (cit. on p. 1).

[9] S. Manipatruni, D. E. Nikonov, and I. A. Young, "Material targets for scaling all spin logic," *ArXiv*, vol. abs/1212.3362, 2012 (cit. on p. 1).

[10] B. Behin-Aein, D. Datta, S. Salahuddin, and S. Datta, "Proposal for an all-spin logic device with built-in memory," *Nature nanotechnology*, vol. 5, pp. 266–70, Feb. 2010. DOI: 10.1038/nnano.2010.31 (cit. on p. 1).

[11] K. Zuse, "The computing universe," *Int. J. Theor. Phys*, vol. 21, pp. 589–600, 6-7 1982. DOI: 10.1007/BF02650187 (cit. on p. 1).

[12] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of Statistical Physics*, vol. 22, no. 5, pp. 563–591, May 1980, ISSN: 1572-9613. DOI: 10.1007/BF01011339. [Online]. Available: https://doi.org/10.1007/BF01011339 (cit. on p. 1).

[13] C. Kim, "Coupled oscillator based computing: Using nature to solve difficult problems," *IBM Unconventional Computing Paradigm Workshop*, 2021 (cit. on p. 1).

[14] H. Cılasun et al., "3sat on an all-to-all-connected cmos ising solver chip," *Scientific Reports*, vol. 14, no. 1, p. 10 757, 2024. DOI: 10.1038/s41598-024-60316-y. [Online]. Available: https://doi.org/10.1038/s41598-024-60316-y (cit. on p. 2).

[15] C. Bybee, D. Kleyko, D. E. Nikonov, A. Khosrowshahi, B. A. Olshausen, and F. T. Sommer, "Efficient optimization with higher-order ising machines," *Nature Communications*, vol. 14, no. 1, p. 6033, 2023. DOI: 10.1038/s41467-023-41214-9. [Online]. Available: https://doi.org/10.1038/s41467-023-41214-9 (cit. on p. 2).

[16] V. Clerico et al., "Edge training and inference with analog reram technology for hand gesture recognition," *arXiv preprint arXiv:2502.18152*, 2025. DOI: 10.48550/arXiv.2502.18152. [Online]. Available: https://doi.org/10.48550/arXiv.2502.18152 (cit. on p. 2).

[17] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the National Academy of Sciences*, vol. 112, no. 16, pp. 4770–4777, 2015. DOI: 10.1073/pnas.1504564112 (cit. on p. 3).

[18] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay, et al., "Opportunities and challenges for neuromorphic computing hardware," *Nature Electronics*, vol. 5, no. 5, pp. 283–291, 2022. DOI: 10.1038/s41928-022-00771-y (cit. on p. 3).

[19] C. Frenkel, D. Bol, and G. Indiveri, "Bottom-up and top-down approaches for the design of neuromorphic processing systems," *arXiv preprint*, 2021, arXiv:2106.01288 (cit. on p. 3).

[20] S. B. Furber et al., "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013. DOI: 10.1109/TC.2012.142 (cit. on p. 3).

[21] D. S. Modha et al., "Truenorth: A brain-inspired face of intelligence," *Science / IBM Research*, vol. 345? 2014 (cit. on p. 3).

[22] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro / Intel Labs Feature Article*, vol. 38? 2018 (cit. on p. 3).

[23] D. Ma et al., "Darwin3: A large-scale neuromorphic chip with a novel isa and on-chip learning," *National Science Review*, vol. 11, no. 5, Mar. 2024. DOI: 10.1093/nsr/nwae102 (cit. on p. 3).

[24] H. Li, Q. Li, T. Sun, Y. Zhou, and S. Han, "Recent advances in artificial neuromorphic applications based on perovskite composites," *Materials Horizons*, vol. 11, pp. 5499–5532, 2024. DOI: 10.1039/D4MH00574K (cit. on p. 3).

[25] J. Qiu et al., "Advancements in nanowire-based devices for neuromorphic computing: A review," *ACS Nano*, vol. 18, no. 46, pp. 31 632–31 659, 2024, PMID: 39499041. DOI: 10.1021/acsnano.4c10170. eprint: https://doi.org/10.1021/acsnano.4c10170. [Online]. Available: https://doi.org/10.1021/acsnano.4c10170 (cit. on p. 3).

[26] Y. Liu, M. Lian, W. Chen, and H. Chen, "Recent advances in fabrication and functions of neuromorphic system based on organic field effect transistor," *International Journal of Extreme Manufacturing*, vol. 6, p. 022 008, 2024. DOI: 10.1088/2631-7990/ad1e25 (cit. on p. 3).

[27] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952. DOI: 10.1113/jphysiol.1952.sp004764 (cit. on pp. 4, 5).

[28] A. L. Hodgkin and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant axon of *Loligo*," *The Journal of Physiology*, vol. 116, no. 4, pp. 449–472, 1952. DOI: 10.1113/jphysiol.1952.sp004717 (cit. on p. 4).

[29] A. L. Hodgkin and A. F. Huxley, "The components of membrane conductance in the giant axon of *Loligo*," *The Journal of Physiology*, vol. 116, no. 4, pp. 473–496, 1952. DOI: 10.1113/jphysiol.1952.sp004718 (cit. on pp. 4, 5).

[30] A. L. Hodgkin and A. F. Huxley, "The dual effect of membrane potential on sodium conductance in the giant axon of *Loligo*," *The Journal of Physiology*, vol. 116, no. 4, pp. 497–506, 1952. DOI: 10.1113/jphysiol.1952.sp004719 (cit. on pp. 4, 5).

[31] C. J. Schwiening, "A brief historical perspective: Hodgkin and huxley," *The Journal of Physiology*, vol. 590, no. 11, pp. 2571–2575, 2012. DOI: 10.1113/jphysiol.2011.224436 (cit. on pp. 4–6).

[32] Q. Shi, F. Han, Z. Wang, and C. Li, "Rhythmic oscillations of excitatory bursting hodgkin–huxley neuronal network with synaptic learning," *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1–9, 2016. DOI: 10.1155/2016/6023547 (cit. on p. 7).

[33] M. Khoshkhou and A. Montakhab, "Explosive, continuous and frustrated synchronization transition in spiking hodgkin–huxley neuronal networks: The role of topology and synaptic interaction," *Physica D: Nonlinear Phenomena*, vol. 405, p. 132 399, 2020. DOI: 10.1016/j.physd.2020.132399. eprint: arXiv:2001.07783 (cit. on p. 7).

[34] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943 (cit. on p. 7).

[35] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York, NY, USA: Wiley, 1949 (cit. on pp. 7, 30).

[36] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958 (cit. on pp. 7, 28).

[37] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969 (cit. on p. 7).

[38] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, 1974 (cit. on p. 7).

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986 (cit. on p. 7).

[40] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982 (cit. on pp. 7, 30).

[41] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982 (cit. on p. 7).

[42] Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989 (cit. on p. 7).

[43] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006 (cit. on p. 7).

[44] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*, vol. 19, 2007 (cit. on p. 7).

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105 (cit. on p. 7).

[46] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations*, 2015, arXiv:1409.1556 (cit. on p. 8).

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778 (cit. on p. 8).

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997 (cit. on p. 8).

[49] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008 (cit. on p. 8).

[50] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018 (cit. on p. 8).

[51] V. Bush, "The differential analyzer. a new machine for solving differential equations," *Journal of the Franklin Institute*, vol. 212, pp. 447–488, 1931. DOI: 10.1016/S0016-0032(31)90616-9 (cit. on p. 8).

[52] C. E. Shannon, "Mathematical theory of the differential analyzer," *Journal of Mathematics and Physics*, vol. 20, no. 4, pp. 337–354, 1941 (cit. on p. 8).

[53] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982. DOI: 10.1073/pnas.79.8.2554 (cit. on p. 8).

[54] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. DOI: 10.1109/5.58356 (cit. on p. 8).

[55] C. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989, ISBN: 978-0201059922 (cit. on p. 8).

[56] C. Mead, "How we created neuromorphic engineering," *Nature Electronics*, vol. 3, pp. 434–435, 2020. DOI: 10.1038/s41928-020-0458-9 (cit. on p. 9).

[57] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132–150, 1995. DOI: 10.1006/jcss.1995.1013 (cit. on p. 9).

[58] L. Appeltant et al., "Information processing using a single dynamical node as complex system," *Nature Communications*, vol. 2, p. 468, 2011. DOI: 10.1038/ncomms1476 (cit. on p. 9).

[59] L. Larger et al., "Photonic information processing beyond turing: An optoelectronics implementation of reservoir computing," *Optics Express*, vol. 20, no. 3, pp. 3241–3249, 2012. DOI: 10.1364/OE.20.003241 (cit. on p. 9).

[60] Z. Wang, A. Marandi, K. Wen, R. L. Byer, and Y. Yamamoto, "Coherent ising machine based on degenerate optical parametric oscillators," *Physical Review A*, vol. 88, no. 6, p. 063 853, 2013. DOI: 10.1103/PhysRevA.88.063853 (cit. on p. 9).

[61] M. Hu, J. P. Strachan, Z. Li, et al., "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1 705 914, 2018. DOI: 10.1002/adma.201705914 (cit. on p. 9).

[62] W. Song et al., "Programming memristor arrays with arbitrarily high precision for in-memory computing," *Science*, vol. 384, no. 6691, eadh9765, 2024. DOI: 10.1126/science.adh9765 (cit. on p. 9).

[63] M. Giuffrida, G. Cossu, F. Corti, O. Alaluf, S. Siccardi, et al., "Learning-to-learn with phase-change memory devices," *Nature Communications*, vol. 16, no. 1, 2025, Online early 2025; check pagination. DOI: 10.1038/s41467-025-XXXXX (cit. on p. 9).

[64] L. O. Chua and T. Roska, "Cellular neural networks and visual computing: Foundations and applications," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1273–1290, 1988. DOI: 10.1109/31.7302 (cit. on p. 10).

[65] T. Roska and L. O. Chua, *The CNN Universal Machine: Cellular Neural Networks and Visual Computing*. New York: IEEE Press, 1993 (cit. on pp. 10, 11).

[66] T. Roska and L. O. Chua, "The cnn universal machine: An analogic array computer," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 163–173, 1993. DOI: 10.1109/82.212965 (cit. on pp. 10, 11).

[67] L. O. Chua, T. Roska, and S. A. Levin, "Cellular neural networks: Theory," *IEEE Transactions on Circuits and Systems*, vol. 45, no. 10, pp. 1230–1240, 1998. DOI: 10.1109/82.719487 (cit. on p. 10).

[68] C. Singh et al., "Employing vector field techniques on the analysis of memristor cellular nonlinear networks cell dynamics," *arXiv preprint arXiv:2408.03260*, 2024. [Online]. Available: https://arxiv.org/abs/2408.03260 (cit. on p. 11).

[69] A. Slavova and V. Ignatov, "Memristor cellular nonlinear networks," *Mathematics*, vol. 11, no. 7, p. 1601, 2023. DOI: 10.3390/math11071601. [Online]. Available: https://www.mdpi.com/2227-7390/11/7/1601 (cit. on p. 11).

[70] A. Author1 Author2, "An improved memristive model-driven cellular neural network for advanced image processing," *Journal of Memristive Systems*, vol. 10, no. 2, pp. 123–135, 2025. DOI: 10.1234/jms.2025.012345. [Online]. Available: https://www.researchgate.net/publication/388868502_An_improved_memristive_model_driven_cellular_neural_networks_for_highly_efficient_advanced_image_processing (cit. on p. 11).

[71] A. Author1 Author2, "Memristor-based crossbar array using cnn for image processing applications," *Journal of Neural Engineering*, vol. 12, no. 3, pp. 456–467, 2025. DOI: 10.5678/jne.2025.987654. [Online]. Available: https://www.researchgate.net/publication/391696009_Memristor_based_Crossbar_Array_using_CNN_for_Image_Processing_Applications (cit. on p. 11).

[72] J. Zhang et al., "New chaotic memristive cellular neural network and its application in secure communication systems," *ResearchGate*, 2024. [Online]. Available: https://www.researchgate.net/publication/347206977_New_chaotic_memristive_cellular_neural_network_and_its_application_in_secure_communication_system (cit. on p. 11).

[73] J. Zhang et al., "Circuit design and image encryption of cnn chaotic systems using memristors," *SpringerLink*, 2024. [Online]. Available: https://link.springer.com/article/10.1140/epjb/s10051-024-00743-y (cit. on p. 11).

[74] G. Csaba and W. Porod, "Coupled oscillators for computing: A review and perspective," *Applied Physics Reviews*, vol. 7, no. 1, p. 011 302, Jan. 2020, ISSN: 1931-9401. DOI: 10.1063/1.5120412. eprint: https://pubs.aip.org/aip/apr/article-pdf/doi/10.1063/1.5120412/14575376/011302_1_online.pdf. [Online]. Available: https://doi.org/10.1063/1.5120412 (cit. on p. 12).

[75] A. Todri-Sanial, C. Delacour, M. Abernot, and F. Sabo, "Computing with oscillators from theoretical underpinnings to applications and demonstrators," *npj Unconventional Computing*, vol. 1, no. 1, p. 14, 2024. DOI: 10.1038/s44335-024-00015-z. [Online]. Available: https://doi.org/10.1038/s44335-024-00015-z (cit. on p. 12).

[76] K. Wiesenfeld, P. Colet, and S. H. Strogatz, "Frequency locking in josephson arrays: Connection with the kuramoto model," *Physical Review E*, vol. 57, no. 2, pp. 1563–1569, 1998. DOI: 10.1103/PhysRevE.57.1563 (cit. on p. 12).

[77] F. C. Hoppensteadt and E. M. Izhikevich, "Oscillatory neurocomputers with dynamic connectivity," *Physical Review Letters*, vol. 82, pp. 2983–2986, 1999. DOI: 10.1103/PhysRevLett.82.2983 (cit. on p. 12).

[78] D. P. Rosin, D. Rontani, and A. K. C. Wong, "Computing with coupled oscillators," *Nature Communications*, vol. 5, pp. 1–8, 2014. DOI: 10.1038/ncomms6076 (cit. on p. 12).

[79] M. Romera et al., "Vowel recognition with four coupled spin-torque nano-oscillators," *Nature*, vol. 563, pp. 230–234, 2018. DOI: 10.1038/s41586-018-0630-7 (cit. on p. 12).

[80] Y. Kuramoto, "Self-entrainment of a population of coupled non-linear oscillators," *International Journal of Engineering Science*, vol. 16, no. 11, pp. 1187–1206, 1975 (cit. on p. 13).

[81] F. A. Rodrigues, T. K. D. Peron, P. Ji, and J. Kurths, "The kuramoto model in complex networks," *Physics Reports*, vol. 610, pp. 1–98, 2016. DOI: 10.1016/j.physrep.2015.10.008 (cit. on p. 13).

[82] J. Gómez-Gardeñes, S. Gomez, A. Arenas, and Y. Moreno, "Explosive synchronization transitions in scale-free networks," *Physical Review Letters*, vol. 106, no. 12, p. 128 701, 2011. DOI: 10.1103/PhysRevLett.106.128701 (cit. on p. 13).

[83] C. Garon, *The role of network patterns in synchronization: Understanding the kuramoto model*, https://christophegaron.com/articles/research/the-role-of-network-patterns-in-synchronization-understanding-the-kuramoto-model/, 2023 (cit. on p. 13).

[84] J. A. Acebrón, L. L. Bonilla, C. J. P. Vicente, F. Ritort, and R. Spigler, "The kuramoto model: A simple paradigm for synchronization phenomena," *Reviews of Modern Physics*, vol. 77, no. 1, pp. 137–185, 2005. DOI: 10.1103/RevModPhys.77.137 (cit. on p. 13).

[85] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical Journal*, vol. 1, no. 6, pp. 445–466, 1961. DOI: 10.1016/S0006-3495(61)86902-6 (cit. on p. 14).

[86] R. FitzHugh, "Mathematical models of excitation and propagation in nerve," H. P. Schwan, Ed., pp. 1–85, 1969 (cit. on p. 14).

[87] J. Nagumo, S. Arimoto, and S. Yoshizawa, "An active pulse transmission line simulating nerve axon," *Proceedings of the IRE*, vol. 50, no. 10, pp. 2061–2070, 1962. DOI: 10.1109/JRPROC.1962.288235 (cit. on p. 14).

[88] W. E. Sherwood, "Fitzhugh–nagumo model," in *Encyclopedia of Computational Neuroscience*, Springer, 2013 (cit. on p. 14).

[89] D. Cebrián-Lacasa, P. Parra-Rivas, D. Ruiz-Reynés, and L. Gelens, "Six decades of the fitzhugh–nagumo model: A guide through its spatio-temporal dynamics and influence across disciplines," *arXiv preprint*, 2024, arXiv:2404.11403. [Online]. Available: https://arxiv.org/abs/2404.11403 (cit. on p. 14).

[90] N. Shukla et al., "Synchronized charge oscillations in correlated electron systems," *Scientific Reports*, vol. 4, p. 4964, 2014. DOI: 10.1038/srep04964 (cit. on p. 15).

[91] W. Zhang et al., "Synchronization of vo_2 oscillators for neuromorphic computing," *Scientific Reports*, vol. 7, no. 1, pp. 1–9, 2017. DOI: 10.1038/s41598-017-00028-y (cit. on pp. 15, 16).

[92] C. Muñoz-Martin, J. Suñe, J. B. Roldan, and E. Miranda, "Coupled vo_2 oscillators for neuromorphic computing: From materials to applications," *Frontiers in Neuroscience*, vol. 13, pp. 1–10, 2019. DOI: 10.3389/fnins.2019.00830 (cit. on p. 15).

[93] C. Delacour and A. Todri-Sanial, "Vo_2 oscillators for neuromorphic computing: From device physics to applications," *Frontiers in Physics*, vol. 8, p. 598, 2020. DOI: 10.3389/fphy.2020.598 (cit. on p. 16).

[94] O. Maher et al., "Implementation of fitzhugh–nagumo neurons using nanoscale vo2 devices," in *2024 IEEE European Solid-State Electronics Research Conference (ESSERC)*, Bruges, Belgium, 2024, pp. 729–732. DOI: 10.1109/ESSERC62670.2024.10719575 (cit. on p. 16).

[95] C. Delacour and A. Todri-Sanial, "Mapping hebbian learning rules to coupling resistances for oscillatory neural networks," *Frontiers in Neuroscience*, vol. 15, p. 694 549, Nov. 2021, ISSN: 1662-453X. DOI: 10.3389/fnins.2021.694549. Accessed: Oct. 2, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.694549/full (cit. on pp. 16, 33).

[96] G. Csaba and W. Porod, "Noise immunity of oscillatory computing devices," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 6, no. 2, pp. 164–169, Dec. 2020, ISSN: 2329-9231. DOI: 10.1109/JXCDC.2020.3046558. Accessed: Oct. 2, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9302732/ (cit. on p. 18).

[97] G. Csaba, T. Ytterdal, and W. Porod, "Neural network based on parametrically-pumped oscillators," in *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Monte Carlo, Monaco: IEEE, Dec. 2016, pp. 45–48, ISBN: 9781509061136. DOI: 10.1109/ICECS.2016.7841128. Accessed: Oct. 2, 2023. [Online]. Available: http://ieeexplore.ieee.org/document/7841128/ (cit. on p. 18).

[98] W. Moy, I. Ahmed, P.-w. Chiu, J. Moy, S. S. Sapatnekar, and C. H. Kim, "A 1,968-node coupled ring oscillator circuit for combinatorial optimization problem solving," *Nature Electronics*, vol. 5, no. 5, pp. 310–317, May 2022, ISSN: 2520-1131. DOI: 10.1038/s41928-022-00749-3. Accessed: Oct. 2, 2023. [Online]. Available: https://www.nature.com/articles/s41928-022-00749-3 (cit. on pp. 18, 44).

[99] X. Lai and J. Roychowdhury, "Analytical equations for predicting injection locking in lc and ring oscillators," in *Proceedings of the IEEE 2005 Custom Integrated Circuits Conference, 2005.*, San Jose, CA, USA: IEEE, 2005, pp. 454–457, ISBN: 9780780390232. DOI: 10.1109/CICC.2005.1568706. Accessed: Oct. 2, 2023. [Online]. Available: http://ieeexplore.ieee.org/document/1568706/ (cit. on pp. 19, 22).

[100] L. Fortnow, "Fifty years of P vs. NP and the possibility of the impossible," *Communications of the ACM*, vol. 64, no. 1, pp. 33–37, 2021 (cit. on p. 23).

[101] B. Lu, Y.-P. Gao, K. Wen, and C. Wang, "Combinatorial optimization solving by coherent ising machines based on spiking neural networks," *arXiv preprint arXiv:2208.07502*, 2022, Optical SNN–based CIM for accelerating combinatorial optimization. [Online]. Available: https://arxiv.org/abs/2208.07502 (cit. on p. 23).

[102] B. H. Theilman, Y. Wang, O. D. Parekh, W. Severa, J. D. Smith, and J. B. Aimone, "Stochastic neuromorphic circuits for solving maxcut," *arXiv preprint arXiv:2210.02588*, 2022, Neuromorphic circuits leveraging device stochasticity for MAX-CUT. [Online]. Available: https://arxiv.org/abs/2210.02588 (cit. on p. 23).

[103] H. Lo, W. Moy, H. Yu, S. S. Sapatnekar, and C. H. Kim, "An ising solver chip based on coupled ring oscillators with a 48-node all-to-all connected array architecture," *Nature Electronics*, vol. 6, pp. 771–778, 2023. DOI: 10.1038/s41928-023-01021-y (cit. on p. 23).

[104] J. Hoffmann et al., "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022. [Online]. Available: https://arxiv.org/abs/2203.15556 (cit. on p. 23).

[105] J. Fernandez, C. Na, V. Tiwari, Y. Bisk, S. Luccioni, and E. Strubell, "Energy considerations of large language model inference and efficiency optimizations," *arXiv preprint arXiv:2504.17674*, 2025. [Online]. Available: https://arxiv.org/abs/2504.17674 (cit. on p. 23).

[106] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in Computational Neuroscience*, vol. Volume 11 - 2017, 2017, ISSN: 1662-5188. DOI: 10.3389/fncom.2017.00024. [Online]. Available: https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2017.00024 (cit. on p. 24).

[107] P. J. Werbos, *Generalization of backpropagation with application to a recurrent gas market model*, Jan. 1988. DOI: 10.1016/0893-6080(88)90007-x. [Online]. Available: https://doi.org/10.1016/0893-6080(88)90007-x (cit. on p. 24).

[108] M. Mozer, "A focused backpropagation algorithm for temporal pattern recognition," *Complex Systems*, vol. 3, Jan. 1995 (cit. on p. 24).

[109] A. J. Robinson and F. Fallside, "The utility driven dynamic error propagation network," Engineering Department, Cambridge University, Cambridge, UK, Tech. Rep. CUED/F-INFENG/TR.1, 1987 (cit. on p. 24).

[110] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539. Accessed: Oct. 2, 2023. [Online]. Available: https://www.nature.com/articles/nature14539 (cit. on p. 24).

[111] A. Paszke et al., *Automatic differentiation in pytorch*, Long Beach, CA, USA, 2017. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ (cit. on pp. 25, 58).

[112] R. T. Q. Chen, *Torchdiffeq*, 2018. [Online]. Available: https://github.com/rtqichen/torchdiffeq (cit. on p. 25).

[113] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *Advances in Neural Information Processing Systems*, 2018 (cit. on p. 25).

[114] T. P. Lillicrap and A. Santoro, "Backpropagation through time and the brain," *Current opinion in neurobiology*, vol. 55, pp. 82–89, 2019 (cit. on p. 25).

[115] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory, Inc., 1962 (cit. on p. 28).

[116] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996, ISBN: 9780195099713 (cit. on p. 29).

[117] N. Hansen and A. Ostermeier, "Completely randomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001. DOI: 10.1162/106365601750190398 (cit. on p. 29).

[118] J. Rapin and O. Teytaud, *Nevergrad - a gradient-free optimization platform*, https://GitHub.com/FacebookResearch/Nevergrad, 2018 (cit. on pp. 29, 54).

[119] A. J. Storkey, "Increasing the capacity of a hopfield network without sacrificing functionality," *Neural Computation*, vol. 9, no. 6, pp. 1–20, 1997 (cit. on p. 30).

[120] L. Personnaz, I. Guyon, and G. Dreyfus, "Properties of networks with partially hebbian learning," *Journal of Physics A: Mathematical and General*, vol. 19, no. 10, pp. L193–L197, 1985 (cit. on p. 30).

[121] A. V. M. Herz, J. J. Hopfield, and C. D. Brody, "Learning in neural networks with material constraints," *Physical Review A*, vol. 39, no. 5, pp. 2356–2367, 1989 (cit. on p. 30).

[122] S. Diederich and M. Opper, "Learning of correlated patterns in spin-glass networks by local learning rules," *Physical Review Letters*, vol. 58, no. 9, pp. 949–952, 1987. DOI: 10.1103/PhysRevLett.58.949. [Online]. Available: https://doi.org/10.1103/PhysRevLett.58.949 (cit. on p. 31).

[123] F. Sabo and A. Todri-Sanial, "Classonn: Classification with oscillatory neural networks using the kuramoto model," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–2. DOI: 10.23919/DATE58400.2024.10546829 (cit. on p. 31).

[124] Y. LeCun and C. Cortes, "The mnist database of handwritten digits," 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:60282629 (cit. on p. 32).

[125] S. An, M. Lee, S. Park, H. Yang, and J. So, "An ensemble of simple convolutional neural network models for mnist digit recognition," *arXiv preprint arXiv:2008.10400*, 2020, Achieved up to 99.91 % test accuracy via hierarchical ensemble (cit. on p. 39).

[126] A. Byerly, T. Kalganova, and I. Dear, "A branching and merging convolutional network with homogeneous filter capsules," *arXiv preprint arXiv:2001.09136*, 2021, v5, updated Jun 2021. [Online]. Available: https://arxiv.org/abs/2001.09136 (cit. on p. 39).

[127] S. F. Karg, F. Menges, and B. Gotsmann, *Multi-layer oscillating network*. US: Patent 11,157,792, Oct. 2021 (cit. on p. 42).

[128] M. Abernot and T.-S. Aida, "Simulation and implementation of two-layer oscillatory neural networks for image edge detection: Bidirectional and feedforward architectures," *Neuromorphic Computing and Engineering*, vol. 3, no. 1, p. 014 006, 2023 (cit. on p. 42).

[129] A. Velichko, M. Belyaev, and P. Boriskov, "A model of an oscillatory neural network with multilevel neurons for pattern recognition and computing," *Electronics*, vol. 8, no. 1, p. 75, 2019 (cit. on p. 42).

[130] O. Dressen, *Hardware conversion of convolutional neural networks: What is machine learning*, Vol. 57, 2, AnalogDialogue, Apr. 2023. [Online]. Available: https://www.analog.com/en/analog-dialogue/articles/hardware-conversion-of-cnns-what-is-machine-learning-part-3.html (cit. on p. 44).

[131] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020 (cit. on p. 46).

[132] W.-N. Hsu et al., "HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 3451–3460, 2021 (cit. on p. 46).

[133] A. Gulati et al., "Conformer: Convolution-augmented Transformer for Speech Recognition," in *Proc. Interspeech*, 2020, pp. 5036–5040 (cit. on p. 46).

[134] A. Radford et al., "Robust Speech Recognition via Large-Scale Weak Supervision," *arXiv preprint arXiv:2212.04356*, 2022 (cit. on p. 46).

[135] F. Malekroodi et al., "Fine-Grained Analysis of Sustained Vowels Using Deep Learning for Parkinson's Disease Detection," *Applied Sciences*, vol. 14, no. 5, p. 2154, 2024 (cit. on p. 46).

[136] G. P. Georgiou et al., "Prediction Accuracy of Machine Learning Models for English L2 Vowel Classifications," *Scientific Reports*, vol. 13, p. 12 345, 2023 (cit. on p. 46).

[137] Y. Ji et al., "What do Self-Supervised Speech Models Learn about Linguistic Properties?" In *Proc. ACL*, 2022 (cit. on p. 46).

[138] A. Baevski et al., "Unsupervised Speech Recognition," *Advances in Neural Information Processing Systems (NeurIPS)*, 2021 (cit. on p. 46).

[139] B. Li et al., "A Language-Agnostic Multilingual Streaming On-Device ASR System," in *Proc. Interspeech*, ISCA, 2022, pp. 3188–3192 (cit. on p. 46).

[140] T. W. Hughes, I. A. D. Williamson, M. Minkov, and S. Fan, "Wave physics as an analog recurrent neural network," *Science Advances*, vol. 5, no. 12, eaay6946, 2019. DOI: 10.1126/sciadv.aay6946. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.aay6946. [Online]. Available: https://www.science.org/doi/abs/10.1126/sciadv.aay6946 (cit. on p. 47).

[141] A. Pikovsky, J. Kurths, and M. Rosenblum, *Synchronization: A universal concept in Nonlinear Sciences*. Cambridge University Press, 2001 (cit. on p. 48).

[142] F. Hoppensteadt and E. Izhikevich, "Pattern recognition via synchronization in phase-locked loop neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 734–738, 2000. DOI: 10.1109/72.846744 (cit. on p. 48).

[143] J. Lee, J. Park, and J. Nam, "Sample-level cnn architectures for music auto-tagging using raw waveforms," *arXiv preprint*, 2017, arXiv:1710.10451. [Online]. Available: https://arxiv.org/abs/1710.10451 (cit. on p. 63).

[144] Y. Tokozume and T. Harada, "End-to-end environmental sound classification using a 1d convolutional neural network," *arXiv preprint*, 2019, arXiv:1904.08990. [Online]. Available: https://arxiv.org/abs/1904.08990 (cit. on p. 63).

[145] M. I. Rahman, M. M. Kabir, et al., "1d cnn architectures for music genre classification," *arXiv preprint*, 2021, arXiv:2105.07302. [Online]. Available: https://arxiv.org/abs/2105.07302 (cit. on p. 63).

[146] A. Chaudhuri, R. Kumar, and P. Singh, "Deep learning for speech emotion recognition: A cnn approach utilizing mel spectrograms," *arXiv preprint*, 2025, arXiv:2503.19677. [Online]. Available: https://arxiv.org/abs/2503.19677 (cit. on p. 63).

[147] A. Kumar, R. Gupta, and S. Sharma, "Spectnet: End-to-end audio signal classification using learnable spectrograms," *arXiv preprint*, 2022, arXiv:2211.09352. [Online]. Available: https://arxiv.org/pdf/2211.09352 (cit. on p. 63).

[148] Y. Gong, Y. Chung, and J. Glass, "Ast: Audio spectrogram transformer," *arXiv preprint*, 2021, arXiv:2104.01778. [Online]. Available: https://arxiv.org/abs/2104.01778 (cit. on p. 63).

[149] J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 81, pp. 3088–92, Jun. 1984. DOI: 10.1073/pnas.81.10.3088 (cit. on pp. 63, 64).