Summer 8-3-2016

Architectures and Algorithms for Intrinsic Computation with Memristive Devices

Jens Bürger
Portland State University

Architectures and Algorithms for Intrinsic Computation with Memristive Devices

by

Jens Bürger

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Electrical and Computer Engineering

Portland State University
2016

ABSTRACT

Neuromorphic engineering is the research field dedicated to the study and design of brain-inspired hardware and software tools. Recent advances in emerging nanoelectronics promote the implementation of synaptic connections based on memristive devices. Their non-volatile modifiable conductance was shown to exhibit the synaptic properties often used in connecting and training neural layers. With their nanoscale size and non-volatile memory property, they promise a next step in designing more area and energy efficient neuromorphic hardware.

My research deals with the challenges of harnessing memristive device properties that go beyond the behaviors utilized for synaptic weight storage. Based on devices that exhibit non-linear state changes and volatility, I present novel architectures and algorithms that can harness such features for computation.

The crossbar architecture is a dense array of memristive devices placed in-between horizontal and vertical nanowires. The regularity of this structure does not inherently provide the means for nonlinear computation of applied input signals. Introducing a modulation scheme that relies on nonlinear memristive device properties, heterogeneous state patterns of applied spatiotemporal input data can be created within the crossbar. In this setup, the untrained and dynamically changing states of the memristive devices offer a useful platform for information processing. Based on the MNIST data set I'll demonstrate how the temporal aspect of memristive state volatility can be utilized to reduce system size and training complexity for high dimensional input data. With 3 times less neurons and 15 times less synapses to train as compared to other memristor-based implementations, I achieve comparable classification rates of up to 93%. Exploiting dynamic state changes rather than precisely tuned stable states, this approach can tolerate device variation up to 6 times higher than reported levels.

Random assemblies of memristive networks are analyzed as a substrate for intrinsic computation in connection with reservoir computing; a computational framework that harnesses observations of inherent dynamics within complex networks. Architectural and device level considerations lead to new levels of task complexity, which random memristive networks are now able to solve. A hierarchical design composed of independent random networks benefits from a diverse set of topologies and achieves prediction errors (NRMSE) on the time-series prediction task NARMA-10 as low as 0.15 as compared to 0.35 for an echo state network. Physically plausible network modeling is performed to investigate the relationship between network dynamics and energy consumption. Generally, increased network activity comes at the cost of exponentially increasing energy consumption due to nonlinear voltage-current characteristics of memristive devices. A trade-off, that allows linear scaling of energy consumption, is provided by the hierarchical approach. Rather than designing individual memristive networks with high switching activity, a collection of less dynamic, but independent networks can provide more diverse network activity per unit of energy.

My research extends the possibilities of including emerging nanoelectronics into neuromorphic hardware. It establishes memristive devices beyond storage and motivates future research to further embrace memristive device properties that can be linked to different synaptic functions. Pursuing to exploit the functional diversity of memristive devices will lead to novel architectures and algorithms that study rather than dictate the behavior of such devices, with the benefit of creating robust and efficient neuromorphic hardware.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

STATUTORY DECLARATION

I hereby formally declare that I am the sole author of this dissertation and I have not used any source or aid apart from the stated ones. This thesis was neither presented in equal nor in similar form to another examining board at any other university. I cited all used references observing actual academic rules.

In Portland, OR, May 31, 2016 Jens Bürger

1

MOTIVATION

Over the last several decades conventional computing architectures have evolved to powerful systems utilizing billions of transistors on a small footprint, operating at GHz frequencies and allowing large scale parallelism of independent processing units. This trend allowed the for development of ever more powerful electronics that comprise a big part of everyone's present day life. Besides predicted obstacles for future developments of conventional architectures [31], the ever tighter integration of electronics into the human life has made another limitation apparent: whereas human vision, motion control, decision making, etc., demands relatively low attention of its human subjects, the computational demands for conventional computers for these tasks are extremely high. The problem conventional computers face when executing algorithms mimicking neural computation, is the mismatch between the architecture of the underlying hardware and the algorithms themselves. While the separation of processing units and memory in a von-Neumann architecture has given the possibility to compute any problem that can be defined algorithmically, it seems to be this separation that causes neural computation to be inefficient.

The study and design of hard- and software tools providing efficient platforms for neural computation is the aim of neuromorphic engineering. High-level simulation tools allow the study of large neural networks of up to a few million neurons in real-time [6,7,53]. The focus of these tools is the modeling of a diverse set of functionalities observed in biological brains and to develop a deeper appreciation of the possibilities and challenges when computing with such large scale neural networks. These architectures lend themselves much better to neural computation than conventional computing architectures, i.e., as they account for the importance of communication between neurons. However, their ability to model such large networks is, to varying degrees, a product of reusing aspects of conventional architectures which imposes limitations in the ability to achieve levels of energy-efficiency as known from biological brains. More emphasis on energy-efficiency is given with the neuromorphic research on lower-level aspects such as the modeling of multi-compartment neurons [46, 107, 111]. The detailed design of neurons allows mimicking of biological neurons at much less energy cost as compared to the higher-level simulation tools. However, the detailed design of individual neurons shifts the challenges to the scaling of computational performance and network size [39]. With the long-term goal of building computers that are as powerful, and operate as efficiently as biological brains, the mentioned limitations signify the need for architectures that combine network scalability and the efficient utilization of hardware resources.

Emerging nanoelectronics promise to address these existing limitations. In 1971 Leon Chua claimed the existence of a fourth fundamental electrical component [24]. Besides the resistor, capacitor, and inductor, he described a device termed “memristor”. As the name implies, the device exhibits two properties: it provides a resistive relation between voltage and current and a memory of past signals applied to it. The memory is encoded in the device’s resistance and can be altered with the application of corresponding voltage signals. The physical emergence of memristive devices in 2008 [100] and their fabrication in crossbar arrays (dense structures of horizontal and vertical wires) [49] offers new possibilities for the design of neuromorphic hardware platforms. The functional similarities between memristive devices and biological synapses [20], in connection with learning rules such as *spike-time-dependent-plasticity* (STDP), provide the basis for storing and altering the synaptic weights directly in the connections between neurons as opposed to storing them in traditional memory. In addition to the storage aspect, a crossbar connecting layers of neurons allows computation of vector-matrix operations such as the weighted sum of neural inputs directly within the crossbar [117]. Based on these features, hybrid designs of CMOS neurons and memristive synaptic arrays for the implementation of long-term weight storage have been developed as a solution to further improve area and energy efficiency of neuromorphic designs [47, 84, 95].

While memristive devices have been adopted by the neuromorphic community for the efficient implementation of long-term memory, the importance of diverse synaptic dynamics is not yet strongly reflected in memristor-based neuromorphic hardware platforms. As an example, synapses with short-term potentiation have been shown to play crucial roles in activity stabilization of neural circuits [102], memory consolidation [91], and extraction of temporal information from incoming sensory inputs [35]. With memristors exhibiting similar synaptic features [20, 21], my research focuses on the challenges of exploiting memristive device properties for dynamic information processing.

An approach for computing with dynamic elements, where information is extracted from dynamic state transitions is given by the framework of *reservoir computing* (RC) [48, 74]. RC makes the assumption that an untrained dynamical system, the reservoir, inherently performs computation. It is the task of a readout layer to interpret the reservoir and map the computational states to target representations. Besides leading to reduced training complexities [69], the more interesting aspect here is the ability to apply RC to a wide range of physical substrates (i.e., a bucket of water [78]). This motivates the consideration of memristive device properties beyond the stable long-term memory and to build efficient neuromorphic hardware that extract information from the dynamic state changes of memristive devices.

My research establishes a more detailed understanding of how memristive devices can be utilized in neuromorphic hardware. This work:

(a) introduces novel methods to neuromorphic engineering based on dynamic memristive networks,

(b) that can robustly utilize nanoscale memristive devices with nonlinear and volatile properties,

(c) lead to hardware systems that harness different synaptic plasticity from volatile to nonvolatile behavior,

(d) enable a wide range of applications from image and speech recognition to time-series prediction, and

(e) reduce the overall system size and training requirements of neuromorphic architectures by exploiting intrinsic properties of memristive devices.

2

BACKGROUND

This chapter will introduce reservoir computing, memristive devices, and memristive networks. Combinations of those will provide the background for my work.

2.1 RESERVOIR COMPUTING

Reservoir computing (RC) is a novel computing paradigm that harnesses intrinsic dynamics of untrained computational substrates. The universality of the term “untrained computational substrate” was demonstrated in [78] by using a bucket of water for non-linear pattern recognition problems - the application from which the term reservoir was derived. Despite the universality, formalisms that help to study RC mostly rely on implementations of *recurrent neural networks* (RNN) .

In Fig. 2.1 the basic elements of RC are shown - the input layer, the reservoir and the readout layer. The input layer applies the input signal $\mathbf{u}(t)$ via the fixed weight matrix $\mathbf{W}^{in}$ to the reservoir nodes. A reservoir node is the component that performs the computation, i.e., a sigmoidal neuron. The reservoir nodes are connected among each other through the fixed weight matrix $\mathbf{W}^{res}$. Their time-dependent state $\mathbf{x}(t)$ is derived as:

$$\mathbf{x}(t+1) = f\left(\mathbf{W}^{res} \cdot \mathbf{x}(t) + \mathbf{W}^{in} \cdot \mathbf{u}(t)\right), \qquad (2.1)$$

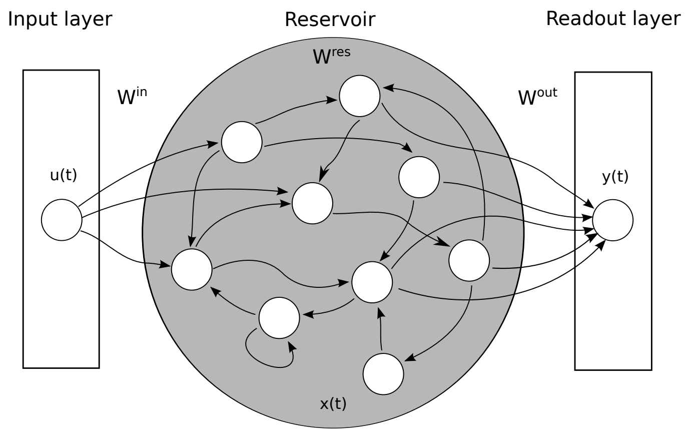


Figure 2.1: Generic reservoir illustration showing an input layer, the randomly structured reservoir performing the computation, and the readout layer that interprets the internal dynamics of the reservoir. Only links from the reservoir to the readout layer will be trained, not the internal links of the reservoir.

where f is the activation function of a reservoir node (i.e., *tanh*). The output $\mathbf{y}(t)$ of the system is produced within the readout layer by the matrix-vector product of the reservoir states $\mathbf{x}(t)$ and the output weight matrix $\mathbf{W}^{out}$:

$$\mathbf{y}(t) = \mathbf{W}^{out} \cdot \mathbf{x}(t). \tag{2.2}$$

Given that $\mathbf{W}^{in}$ and $\mathbf{W}^{res}$ are fixed, learning takes place by adjusting the output weight matrix $\mathbf{W}^{out}$ only. This reduces the training complexity of recurrent neural networks to the output matrix, which is commonly done by using regression techniques.

The definition of the weight matrices $\mathbf{W}^{in}$, $\mathbf{W}^{res}$, and $\mathbf{W}^{out}$ underlies no restrictions and the number of input connections, internal reservoir connections, and reservoir to readout connections are parameters that can be set application specific.

RC simultaneously emerged in two variations, *echo state networks* (ESN) [48] and *liquid state machines* (LSM) [74]. Both approaches rely on the basic principle of using a computational substrate, a reservoir, that creates a higher-dimensional representation from which information can be extracted by a linear readout layer. Founded on the same principles, differences in their implementation have given them slightly different focus areas. The focus of ESN, which was influenced by classical machine learning, was to present an architecture that addresses the complexity of training weights in a recurrent neural network. By assuming that a RNN inherently produces a higher dimensional space of applied inputs, the only task is to train a linear readout layer to interpret a reservoir's response to a given input. Common application areas for ESN are speech recognition [105, 106], motor control [94], and forecasting problems [25] to name just a few. More details on background, training and applications of ESN can be found in [70]. LSM drew more inspiration from neuroscience and argued that biological brains contain neural micro-circuits that are not trained for a single purpose, but provide a computational substrate for various sensory information. The proximity to neuroscience also informed LSM to utilize spiking neurons (ESN typically uses leaky integrator neurons). Besides applications that have also been addressed with ESN (i.e., spoken digit recognition [109]), LSM provides a framework to further study neural computation [14]. In [72] theory of LSM and resulting applications are discussed.

2.1.1 Extreme Learning Machines

Extreme learning machines (ELM) [44] are a class of feed-forward neural networks that are based on untrained random weights between the input and the hidden layer of the network. Under the assumption that the random weight matrix produces uncorrelated vectors, connecting each hidden layer neuron to the inputs, random weight matrices extract independent features from the input and provide good generalization performance. In comparison to fully trained feed-forward neural networks, ELMs typically require more hidden layer neurons to achieve comparable performance [44]. However, this comes with a significantly reduced training complexity (typically linear regression).

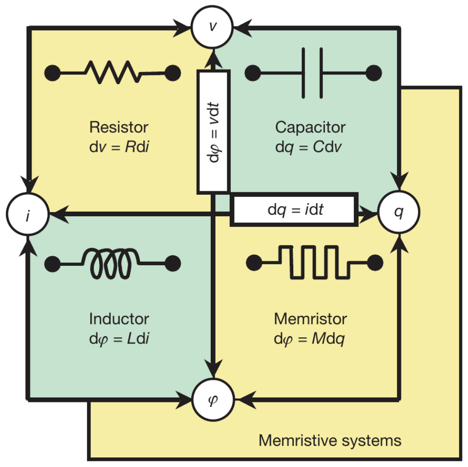
ELM differs from reservoir computing as it does not implement a memory of past inputs. In RC typically achieved through recurrent connections, the feed-forward architecture of the ELM in connection with memory-less hidden nodes does not provide memory. Therefore, ELM is not suited for temporal classification or prediction problems. Implementation of hidden neurons with memory was shown to circumvent such a limitation [104]. The possibility of implementing memory as well as reduced training complexity through random, fixed weights then provides a framework that is similar to RC and a relevant benchmark implementation to the research presented here.

2.2 MEMRISTIVE DEVICES

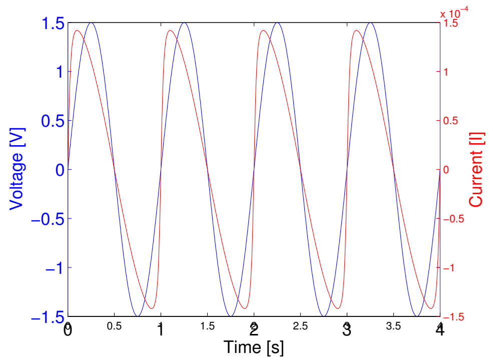
Memristive devices (also called memristor) were first discovered based on symmetry considerations of the fundamental electronic circuit elements resistor, inductor, and capacitor by Leon Chua in 1971 [24]. Each of these devices links two electric phenomena. The resistance is a function of voltage and current, the inductance of voltage and flux, and the capacitor of current and charge. Chua characterized memristance (resistance of a memristor) as linking flux and charge (Fig. 2.2a).

$$M(q) = \frac{d\phi}{dq} \tag{2.3}$$

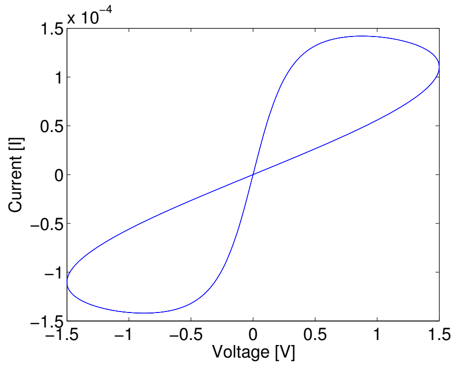
Memristors are based on metal-oxide materials, i.e. TiO_2, and their memristance is defined by the oxygen vacancies distribution within the metal oxide. The distribution of oxygen vacancies is equivalent to the charge of the device. Substituting charge q with its time integral $q = \int_t Idt$ explains how an applied current controls the memristance. The



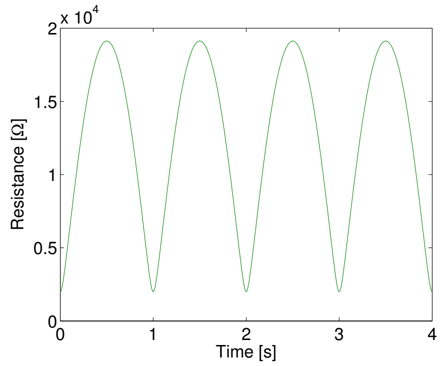
(a) Fundamental circuit elements



(b) Memristor voltage and current



(c) Typical I-V plot



(d) Resistance change

Figure 2.2: Memristor fundamentals. (a) The memristor as the fourth two-terminal circuit element (reprinted from [100]); (b) the nonlinear dependent current through a memristor; (c) the resulting hysteresis loop in the IV plane; (d) the resistance change over time. (b,c,d, created from the model presented in [86]).

whole memristance equation can be rewritten in terms of voltages and currents as:

$$M(q) = \frac{d\phi/dt}{dq/dt} = \frac{V(t)}{I(t)} \tag{2.4}$$

As the memristance is time-dependent, it follows that the resistance of a memristor represents a memory of the history of the applied signals. Nonvolatile memristance is the result of the oxygen vacancies distribution being input signal dependent. In the absence of a signal ($I(t) = 0$) the charge will not change.

In Figures 2.2b-2.2d, I show the fundamental characteristics. An applied input signal (here a sine wave) will result in a nonlinear relationship to the current through the device (linear relationship for a resistor). This reflects the internal state change of the device, which is represented by a varying resistance. The memory property can be better understood when looking at the hysteresis loop, which implies that for a given voltage V, the device current I will depend on the previously applied inputs.

The fabrication of the first physical memristor in 2008 [100] has started a wide variety of research aimed to integrate memristive devices into future architectures. The predicted form factors of 10nm in connection with their nonvolatile memory property, and fast switching times fueled the research on memristors as the basic component of future memory solutions that provide higher memory density at lower energy costs [36]. The same properties make memristors a possible candidate to implement digital logic and it was shown how two memristors suffice to compute any given Boolean logic function [65]. A very different direction of research is motivated by the close link of memristive devices and biological synapses [20, 21]. By utilizing the analog value range of memristors, it is possible to use them for the emulation of synaptic connectivity between neurons. The convincing advantage of this approach over conventional computing architectures is that synaptic weights can be stored directly in the memristive devices, which at the same time is the physical connection between the neurons. Conventional architectures store weights in RAM and weights have to be fetched each time they are used. Considering two layers of neurons, this allows the computation of the weighted sum of inputs for all neurons in a single step. More details on this architecture will be provided in section 2.3.3

2.3 BASICS OF INFORMATION PROCESSING IN MEMRISTIVE NETWORKS

2.3.1 Terminology

Throughout this document I will use words such as “node,” “weight,” and “state.” As some of this terminology is used in both, machine learning and electrical networks, I will first present an overview of how I will use relevant terminology.

Node

In reservoir computing, typically implemented as recurrent neural networks, a neuron is referred to as node. A node performs the processing of its inputs with the attached weights. In addition, a node can apply an activation function to the weighted sum of inputs. When talking about a node in the context of reservoir computing, I will refer to it as reservoir node. For the sake of clarity, in some cases I might simply use neurons to describe reservoir nodes, or nodes of neural networks in general.

In electrical networks the notion of a node is different. A node is a point in the electrical network in which electrical components (i.e., resistor, memristor) connect. A node does not perform any computation. It can be seen as intersections of wires and defines the topology of the network. When talking about nodes in the context of electrical networks I will refer to it as network node.

Weight

Weight is a common term in all neural network architectures. A weight is typically associated with an input signal to a neuron and defines a scaling factor to determine an input's significance. From a graph point of view, a weight is an attribute of an edge.

Often the memristive state, which is a unit-less parameter determining the resistance of a device, is referred to as weight or abbreviated as w due to its functional similarity with biological synapses. In memristive networks, the state of a memristor (weight) can have two functions: (a) in connection with an applied voltage, the resistance is used to scale the current through the device, (b) in a larger network, the resistance of a memristor determines the device's voltage drop in proportion to the total network resistance.

Weight Update

In neural networks a weight update is performed based on algorithmic rules. I.e., backpropagation determines the network error and propagates the error back [92]. Depending on a signal's contribution to the error, the weight will be adjusted to reduce the error for this data sample.

In neuromorphic memristive networks, where the weight is equivalent to a device's resistance, *spike-time dependent plasticity* (STDP) rules are applied to update memristors according to the timing of neuron activities [1, 10, 33] .

Input Signal

In neural networks an input signal (either network or node input) is directly associated with a weight and applied to one or more nodes that perform computation on that signal. It carries information of sensory data or outputs of preceding neurons and is typically unit-less.

In memristive networks an input signal is applied to a network node (wire intersection) and instead of being applied only to the directly connected devices, it spreads throughout the entire network. In my work an input signal to memristive networks will always be a voltage with unit [V]. Every memristive device in the network will experience a voltage drop proportional to its weight (memristive state / resistance).

Network

In a neural network, such as a *liquid state machine* (LSM) and *echo state network* (ESN), the network is a composition of nodes and links. A link is a connection between neurons providing node inputs and corresponding weights, which define the signal flow and the computation within the network.

A memristive network, equivalent to resistive networks, is a composition of memristive devices connected in any possible topology. It differs from neural networks as signal flow and computation are no obvious properties of such a network.

Network State

In neural networks, a network state is the result of a neuron's computation (node output). Depending on the network's connectivity, it is either passed onto other neurons as input and/or in case of RC used by a readout layer to determine the network response to an applied input signal.

For memristive networks we have to differentiate between the device and the network state. While a device state is defining a memristors contribution to the network activity, it cannot always be accessed due to physical constraints: (a) if a memristive state functions as current scaler (see above), then the resulting current is a function of the device state and

	Neural Network	**Memristive Network**
Network	composition of nodes and links of neurons	composition of memristive devices
Node	Neuron	Intersection of wires
Weight	Scaling factor	Current scaling / voltage proportional to input
Weight Update	Algorithmically	Dynamic / untrained
Input	Affecting connected nodes	Spreads throughout whole network proportional to device states
State	Node output	Current / network node voltage
Memristive state	N/A	Resistance

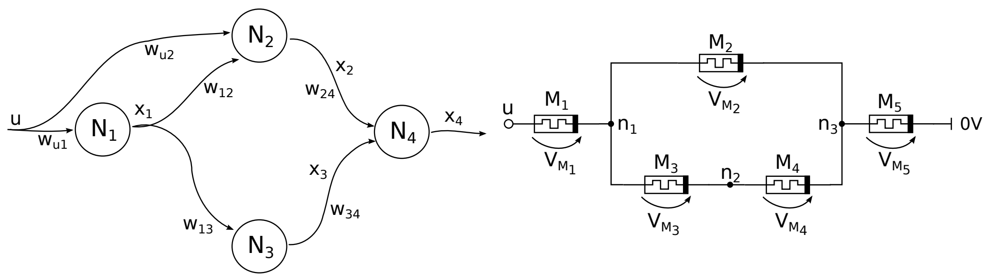
Table 2.1: Memristive network terminology overview

can be used as network state. (b) in case a memristor is connected in a network where it determines a proportional voltage drop, then the exact state of the device cannot be easily inferred and the network node voltage serves as network state.

2.3.2 Random Memristive Networks

To provide a better understanding of the terminology and its application, I present two examples for neural and memristive networks in Fig. 2.3. Fig. 2.3a depicts a neural network with nodes N an input signal u, weights $w_{i,j}$ with i and j denoting the connected node numbers, and network states x_i with i referring to the node that computed the state. For simplicity, the nodes compute only the weighted sum of their inputs and do not apply an activation function. Let's assume $u = 1$ and $w_{i,j} = 0.5$, then the network states will result in $x_1 = 0.5, x_2 = 0.75, x_3 = 0.25, x_4 = 0.5$. Next, I will discuss a memristor network and then highlight the important differences.

Fig. 2.3b shows a memristor network consisting of memristors M, input voltage u, network nodes n, and device voltages V_M. If we assume $u = 3V$ and $M_1 = M_3 = M_4 =$



(a) Simple neural network (b) Simple resistive network

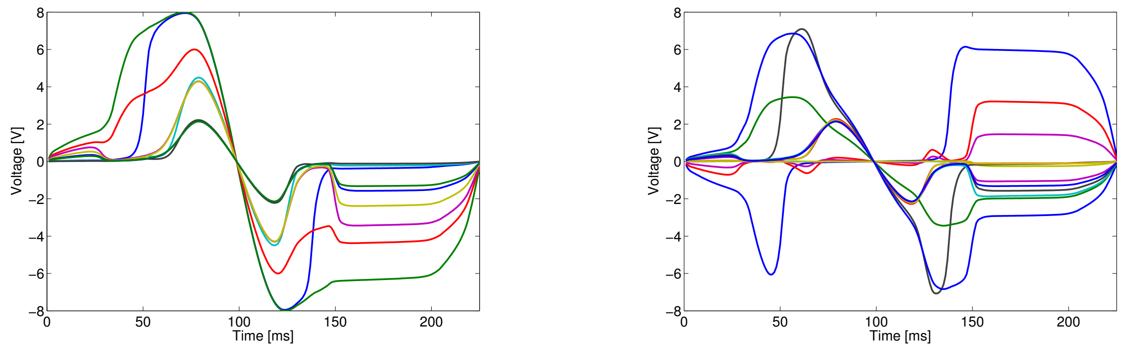
Figure 2.3: Basic elements for computation (a) network of artificial neurons picturing input, weights and network states; (b) resistive network highlighting the relevant network nodes and corresponding voltage potentials for memristive devices.

$M_5 = 1k\Omega$ and $M_2 = 2k\Omega$ we can calculate node voltages n. Following resistive network theory by considering the memristive states to be not changing, the total network resistance $M_{Net} = 3k\Omega$. A node voltage is then computed as portion of the input signal proportional to the ratio of the resistance from node n to ground ($0V$) and M_{Net}. This results in the following equations:

$$n_1 = \frac{(M_2||(M_3 + M_4)) + M_5}{M_{Net}}u, \quad n_3 = \frac{M_5}{M_{Net}}u, \quad n_2 = \frac{M_4}{M_3 + M_4}(n_1 - n_3)$$

and node voltages of $n_1 = 2V, n_2 = 1.5V, n_3 = 1V$.

I will now briefly discuss the fundamental differences that are important to understand how I'll later exploit memristive networks for computation. In a neural network, node signals (input, node outputs) propagate throughout the network and experience scaling by the corresponding weights. In a memristive network, signals spread out instantaneously corresponding to the resistances between nodes. This leads to a few differences. First, it introduces a strong node interdependency as the change of a single memristor will affect all node voltages (states). Second, having shown that node voltages can depend on multiple memristive devices, individual device dynamics might be reflected in the node



(a) Memristive network states when measuring absolute node voltages toward ground. Average correlation of signals in this example is 0.854.

(b) Memristive network states when measuring relative memristor voltages. Average correlation of states in this example is 0.645.

Figure 2.4: Memristive mesh network states measuring either absolute node voltages or relative memristor voltages when exposed to a sine wave. By measuring relative signals rather than with respect to a common ground the individual memristive state changes better reflect the network dynamics.

voltage much alleviated.

These alleviated node dynamics are the result of measuring their absolute voltage with respect to ground (i.e., n_1 reflects the collective states of $M_2 - M_5$ relative to M_1). This method was applied in the first publication on memristive reservoir computing by Kulkarni and Teuscher in 2012 [62]. An improved method that better reflects the device dynamics is by measuring relative voltages V_M [17]. This is done by subtracting the absolute voltages found on both sides of a memristive device (i.e. $V_{M_3} = n_1 - n_2$). An example is given in Fig. 2.4. Here a memristive mesh network is driven with a sine wave and absolute and relative voltages are measured. While there is still strong signal correlation in both setups due to the mentioned reasons, measuring relative voltages V_M can create less correlated signals.

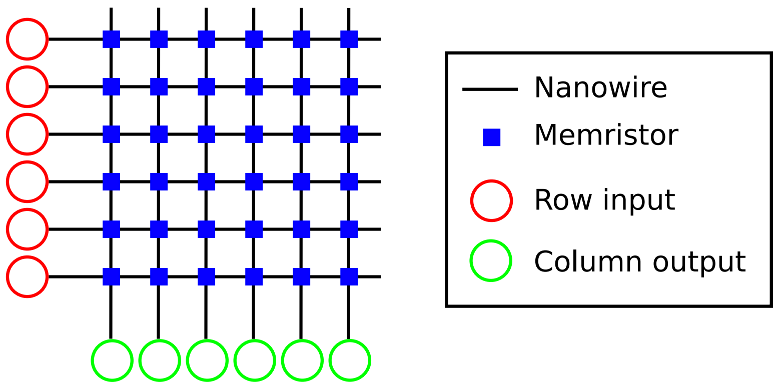


Figure 2.5: Standard crossbar setup with row inputs, memristive devices formed at every intersection of nanowires, and column outputs.

2.3.3 Memristive Crossbars

In Fig. 2.5 a schematic of a memristive crossbar in a neuromorphic architecture is shown. The memristors emerge at intersections of nanowires and enable full connectivity between two layers of neurons, the input neurons (shown in red) and the output neurons (shown in green). The memristive state of each device defines the connection strengths between an input neuron (row) and output neuron (column). In hardware terms, this setup allows the multiplication of an input vector **u**, encoded as voltages, with the synaptic weight matrix **W**, represented as resistances, to result in the output vector **y** that is physically present as currents:

$$\mathbf{y}(t) = \mathbf{W} \cdot \mathbf{u}(t). \tag{2.5}$$

Also implied by the mathematical formulation, it is worthwhile to mention the parallelism within the architecture. A row input applies to all memristive devices along the given row in equal strength. In electrical terms, all columns connected to a specific row represent the same node (see Table 2.1 for the definition of node).

2.4 RELATED WORK

A general, in some ways even philosophical, motivation for my research was well described by Crutchfield *et al.* in their editorial to a focus issue on information processing in dynamical systems [26]. The raised issue of *usefulness* with respect to the output of computation implies questions on the design of a computing architecture. Designed computation is the omnipresent approach in which we build computers to this day, for the reason that the output of the processing is directly *useful* to us. On the contrary, intrinsic computation does not imply the very same understanding of *usefulness*. Instead, it needs to be studied how intrinsic properties of given computational substrates can be harnessed for computation that is *useful*. With this in mind, we then have the possibility to appreciate engineered as well as natural computation and build "faster, less expensive, and more energy efficient" systems.

Before the "rediscovery" of the memristor by Strukov et al. [100] (after the initial formulation in 1971 [24]), the crossbar architecture was already considered for fabricating emerging molecular/nanoscale electronics in dense arrays [23, 40, 66]. Soon after the physical implementation of a memristor [100], first implementations of memristive crossbar arrays were demonstrated, feeding the promises of future dense memory arrays [49, 58, 67]. Subsequent research efforts produced memristive devices that differed in the used materials, their switching characteristics, and their electrical properties [21, 32, 38, 54, 60, 116]. The functional similarities between memristive devices and synapses encouraged neuromorphic applications to consider these emerging devices for more area and energy-efficient architectures [22, 39, 47, 95].

A very recent publication discusses the plasticity of memristive devices for spiking neural networks [93]. The authors argue that the range of existing memristive devices offers a range of different applications, and that researchers should embrace diversity rather than limiting applications to nonvolatile storage.

The possibility of temporal processing with volatile memristive devices was discussed by Berdan *et al.* [8]. Based on a physical device implementation, they have linked the device dynamics to that of biological short-term synaptic plasticity. This short-term plasticity is argued to enhance the discrimination of spatiotemporal patterns. More recently similar propositions were made for another device implementation [71]. However, both articles focused on the device dynamics and did not present an architecture as a functional verification of a system employing short-term synaptic plasticity.

An alternative architecture to crossbar structures was shown to be the random assembly of memristive devices [4, 27, 96, 98]. Due to the simplicity and cost-effectiveness of the fabrication of these random structures, they pose an interesting substrate for computation. The authors highlight the suitability of such architectures for reservoir computing [48, 74]. The complex network structures and device dynamics provide emerging collective network properties from which dynamic state transitions can be harnessed for computation.

Validated by the ability to fabricate random memristive networks, the first demonstration on computing with such networks was published by Kulkarni and Teuscher in 2012 [62]. They have shown the basic suitability of random memristive networks, functioning as reservoir, for simple pattern classification tasks. This work has shown how the frequency dependent state changes of memristive devices can be utilized for nonlinear computation and served as the starting point for my own research presented here.

In [19], a simple connection of memristors in series was shown to utilize inherent device variation to create diverse device responses to an input signal. However, with one current source for each memristive device the architecture is neither scalable nor does it lend itself to utilize complex network dynamics. A traditional echo state network approach with memristive devices as synapses was presented in [76] and a continuation where memristors implement the trainable weights at the readout layer in [61]. A comprehensive system that not only discussed memristors as synapses, but also presented them detailed within the architecture and provided algorithms to train them was shown in [28]. In all three papers memristive devices were used as synapses that get updated during an initialization or training phase. The overall architecture and computation follows more closely other neuromorphic architectures and does not present a memristive reservoir as for example understood by [4, 62].

3

MEMRISTIVE CROSSBARS AS DYNAMICAL SYSTEMS

In neuromorphic hardware systems the most widely discussed memristive architecture is the crossbar, which can provide efficient synaptic weight storage [3, 22, 57, 84, 95]. The regularity of the nanowire placements allows for controllability and scalability of the fabrication process [34]. It is also a scalable architecture in the sense that all devices within a crossbar are directly placed in-between an input and an output neuron. Hence, the lack of device interdependencies, as discussed in section 2.3, does not limit the size of the crossbar and an arbitrary amount of synaptic weights can be implemented by increasing crossbar sizes (other fabrication restrictions such as wire length or resistance might apply).

However, is the simple storage of synaptic weights with occasional, ideally linear, state updates everything that memristive crossbars offer for computation? With respect to Crutchfield *et al.'s* considerations on intrinsic computation [26], is the memristive device behavior complex enough to exhibit nonlinear dynamics that support useful information processing? This question was addressed in publications were memristive devices are considered suitable candidates for nonlinear computation in non-crossbar architectures [4, 16, 17, 30, 62].

We can consider a memristive device as a component that harnesses intrinsic nonlinear device properties. Then, by computing in a higher dimensionality based on nonlinearity [5], the memristive device can potentially pose a more efficient processing unit as compared to the current use as simple synaptic weight storage. While an individual device holds potential for computation, the crossbar, as a collection of many individual devices, is not an obvious candidate for a dynamical system. A dynamical system is characterized by operating in a high dimensional state space with a mathematical function being able to determine the transitioning through such a space [112]. Yet, the only equation relevant for the crossbar is one for the memristive state updates - a single variable equation with a low dimensional state space. In order to evaluate if the crossbar can pose a viable dynamical system that exhibits reservoir computing principles such as fading memory and random structure for heterogeneous computing, we have to consider the broader architecture within which the crossbar operates to overcome limitations, such as the lack of device-interdependencies and the feed-forward flow of data.

The research in this chapter will introduce an interpretation of the crossbar useful for dynamic and heterogeneous computing. I will present the physical and mathematical basics that allow computation within crossbars. The first set of experiments will answer research questions on the general feasibility of the approach as well as give insights on relevant system parameters. In a second set of experiments I will then address relevant components in much more detail and present a comprehensive analysis of computing within memristive crossbars.

At the end of this chapter it will be clear how input data is processed within a crossbar, how to extract relevant information from the memristive devices and how it compares in terms of resources and computational performance (accuracy and speed). The research presented in section 3.4 has been published in [18].

3.1 RESEARCH QUESTIONS

In this section I will briefly outline the concrete research topics that motivated the investigation of memristive crossbars for dynamic computation. Each topic is briefly motivated by pointing to existing work. The formulated questions will then serve as the central themes throughout the chapter.

Research Topic 1. Architecture and Information Retrieval

Neurophysiological experiments highlight the importance of short-term plasticity for computation of spatiotemporal patterns [35]. A software-based neuromorphic implementation for spatiotemporal feature detection mimicking dendritic computation has been presented [104]. Given a set of volatile memristive devices in a crossbar architecture, which is inherently homogeneous in its structure and functionality, the initial work is focused on methods to create within, and extract from the crossbar a heterogeneous representation of the applied input data that is suitable for further processing.

> **Question 1.1.** How can the crossbar structure, where all columns are functionally identical and subject to the same input data, exhibit diverse patterns within it's columns?

> **Question 1.2.** As accessing large amounts of individual memristive devices within a crossbar is impractical, how can one retrieve the state patterns present within the crossbar?

Research Contributions

These research questions lead to an original architectural and algorithmic implementation of memristive crossbars that can be used beyond weight storage. A successful demonstration of creating heterogeneous crossbar states as a function of spatiotemporal input data will: a) demonstrate how state volatility can be harnessed for computation as short-term memory rather than being considered a negative device property, and b) constitute a scalable architecture for nonlinear (pre)-processing of data.

Research Topic 2. Formalizing Dynamic Crossbar Computation

The work presented in [18] relied on exploiting different features in spatiotemporal patterns by applying the same input data in different rotations and combinations to the crossbar rows and columns. For the used data set, this method produced satisfying results. However, as no formalisms were presented, the scalability and viability of this approach for other data sets cannot be predicted reliably. Therefore, the second proposed research contribution aims to answer the following questions:

Question 2.1. Given the mathematical descriptions for an individual device, how can bias matrices for the crossbar columns be designed to best utilize a crossbar of individual memristive devices as a dynamical system?

Question 2.2. How can a bias matrix be designed in order for some input features to have an excitatory and other an inhibitory effect on the resulting individual device states?

Question 2.3. With the fixed connectivity within the crossbar, a single memristive device cannot combine random features from the whole data sample, but is limited to access a subset of it. How strongly does this limit the utility of the crossbar as dynamical system for feature extraction?

Question 2.4. While the crossbar achieves a high dimensional state representation of the applied inputs, accessing these states requires transferring them into an output vector, which causes strong dimensionality reduction. How well can information be preserved in the lower dimensional space?

Research Contributions

As an already established contribution, [18] has been the first example of utilizing a memristive crossbar as a dynamical system for computation. The research questions make contributions by providing formalisms that help to study the computational capacities and limitations of this approach. This work will extend the methods in neuromorphic computation in memristor based architectures, providing the ability to create more area efficient hardware implementations and reduced training complexity by integrating a dynamical system based on short-term synaptic plasticity. While my research focuses on memristive devices, the proposed methods are independent of a specific device and may also be applicable to a wider range of volatile nanoscale devices.

Research Topic 3. Parameter Variations

In hardware implementations, any proposed architecture has to consider device characteristics that might cause divergence from the simulation results. With memristors being produced at the nanoscale, variations in their behavior is an inevitable characteristic [32,77].

Question 3.1. Device variation describes the behavioral differences of individual devices. Based on realistic statistical distributions of device parameters, what extent of device variation can this approach tolerate?

Question 3.2. Given a single memristive device in a fixed state, cycle variation describes the statistical variations in the device's response to an applied input signal. What is the level of cycle variation that can be tolerated?

Research Contributions

I provided an understanding that any hardware implementation at the nanoscale will inherently exhibit variation. This research provides an understanding of: (a) how sensitive memristive crossbars as dynamical systems are to variation in the devices, (b) what devices are suitable based on known statistical distributions in their properties, and (c) how the design of the reference matrix can increase variation tolerance by increasing noise margins.

3.2 VOLATILE MEMRISTIVE DEVICES

In this section, I will discuss intrinsic memristive device properties. While being unfavorable to simple weight storage, these properties have potential in the field of dynamic and temporal processing.

Generally speaking, reservoir computing principles require two properties: a) a dynamic system to map an input signal to a higher dimensional space by means of nonlinear transformations and b) to exhibit fading memory, which provides a short-term memory and ensures that a system's response is only depending on inputs applied over a time frame that corresponds to the temporal nature of the applied input data [48, 74]. Here I will present a volatile memristive device that implements these properties at the device level. We will use this device as the foundation to dynamic computing within crossbars.

Such a memristive device has been presented in [21] and was described mathematically in more detail in [22]. Equation 3.1 describes the nonlinear integration of applied voltages ($\lambda sinh(\eta V)$) and the gradual state decay over time ($-w/\tau$). The parameters λ, η, and τ are constants derived to match experimental data. Fig. 3.1 demonstrates the nonlinear state changes as function of applied voltage pulses as well as of the inherent state decay.

$$\frac{dw}{dt} = (1 - e^{k})\lambda * sinh(\eta V) - \frac{w}{\tau} \tag{3.1}$$

with k:

$$k = \begin{cases} \frac{-1+w}{0.5} & \text{if } V > 0 \\ \frac{-w}{0.6} & \text{if } V < 0 \end{cases} \tag{3.2}$$

$$I(t) = (1 - w(t))\,\alpha\left[1 - \exp\left(-\beta V(t)\right)\right] + w(t)\gamma\sinh\left(\delta V(t)\right) \tag{3.3}$$

$$R(t) = \frac{V(t)}{I(t)} = \frac{V(t)}{(1 - w(t))\,\alpha\left[1 - \exp\left(-\beta V(t)\right)\right] + w(t)\gamma\sinh\left(\delta V(t)\right)} \tag{3.4}$$

Given the memristive device from equation 3.1, nonlinear integration and inherent state decay allows us to map different temporal sequences to different state representations of the same memristor. The memristive state w is a unit-less variable in the interval $w \in [0, 1]$ that determines the physical device resistance. By applying a read voltage V (a sub-threshold voltage that does not alter the device state) to a memristive device, the state-dependent resistance will determine the device current I (equation 3.3). Again, $\alpha, \beta, \gamma, \delta$ are curve-fitting constants to match experimental data. This ability will be the

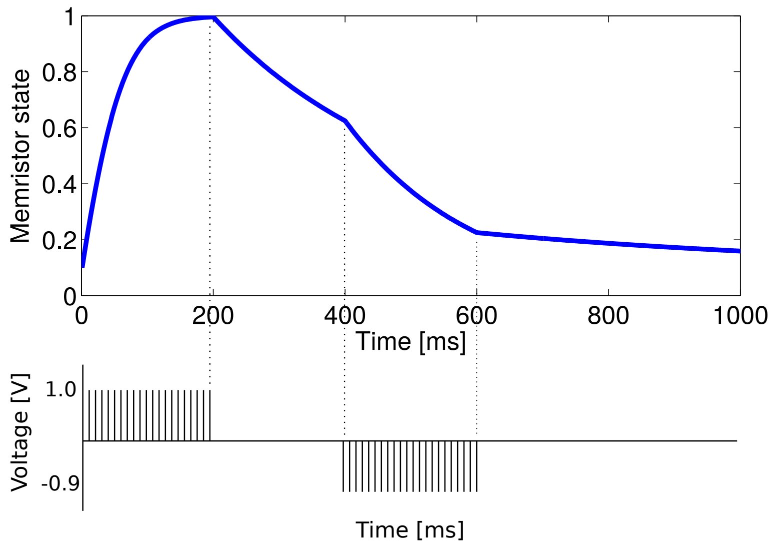
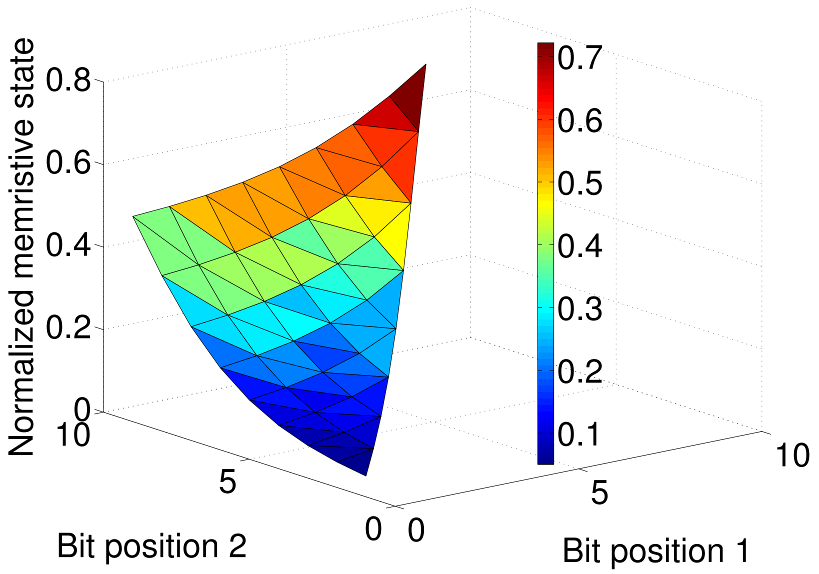


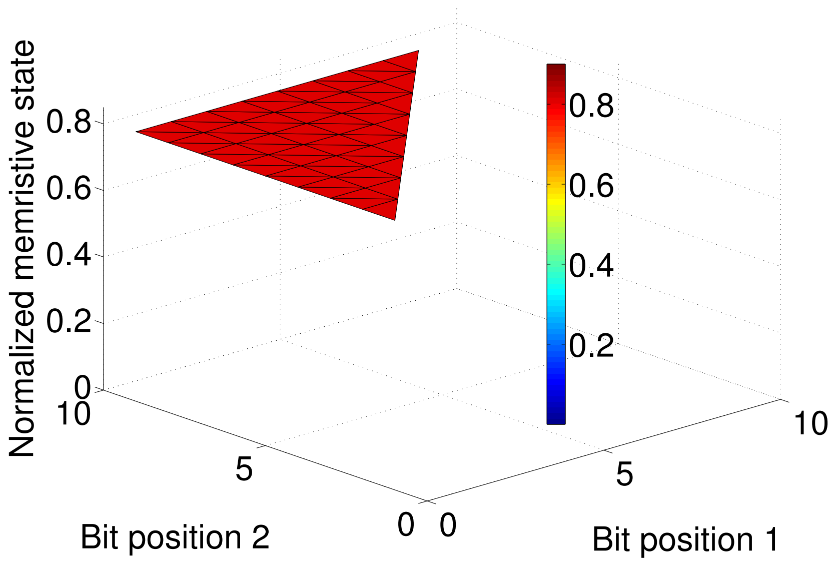
Figure 3.1: Dynamics of a volatile memristive device, as defined by equation 3.1) when applying positive, negative, and no pulses. Positive pulses lead to an increased conductivity, no pulses allow a non-linear state decay, and negative pulses speed up the state decay.

basis for employing individual memristive devices in a crossbar to function as elements of a dynamical system.

To get an intuitive understanding on the relevance of the described properties I demonstrate the mapping of temporal sequences to memristive states in Figure 3.2. Let's assume a set of binary streams S of length ten, with two out of ten bits active (i.e., $s_1 = 0010000010$). With all bit streams containing the same number of active bits, any possible information content has to be encoded in the bit positions. Figure 3.2a is showing the nonlinear state surface that results out of the temporal distances of the active bits. The inherent state decay gives the possibility to intrinsic processing of spatiotemporal patterns, such as speech. The temporal aspect of these devices presents a possibility to implement a short-term memory without the complexity of recurrent connections as typically present in reservoirs with memory-less nodes. In contrast, Fig. 3.2b shows the mapping of the



(a) Memristive state with state decay



(b) Memristive state without state decay

Figure 3.2: Memristive states as a result of various bitstreams. (a) x and y axis define the position of the pulses within the ten steps long bitstream. Due to the nonlinear integration and the state decay the bitstreams are mapped to a nonlinear plane of the memristive state. (b) without state decay, the same setup results in simple integration of two out of ten bits and the bit positions do not affect the result.

same bit streams for a memristive device with no state decay. With integration of input bits only and no state decay, no temporal information can be extracted from input data.

3.3 MATHEMATICAL INTERPRETATION OF CROSSBAR DYNAMICS

In this section we will consider the discussed memristive device within a crossbar and expand our considerations to computational aspects on the architectural level. At first we will have a look at the crossbar architecture and the relevant application of data. In Fig. 3.3 we can see all relevant components to this approach. **X** represents the input data in a spatiotemporal representation. **Y** is called the *reference matrix* and, just as **X**, has a spatiotemporal representation. The function of the reference matrix will be explained in the following paragraphs. **W** is the crossbar matrix in which each element w represent the state of a memristive device.

A requirement for the crossbar being useful as a dynamical system is that it can, similar to conventional reservoir computing approaches, process input data in a higher dimensionality. Reservoir computing, and artificial neural networks in general, typically employ kernel methods at the neuron level to process in a higher dimensionality [5, 13]. A kernel method can be described as:

$$k = f_k(< \mathbf{x}, \mathbf{y} >) \tag{3.5}$$

Here **x** and **y** are input vectors in $\mathbb{R}^m$, f_k is the kernel function, and k is the resulting scalar computed in $\mathbb{R}^n$. A well-known example is a perceptron where **x** would represent the applied input vector and **y** the corresponding weight vector assigned to the inputs [90]. The weighted sum of the inputs is then processed by a kernel function (activation function), i.e., *tanh*. The computation in a higher dimensionality is carried out by the

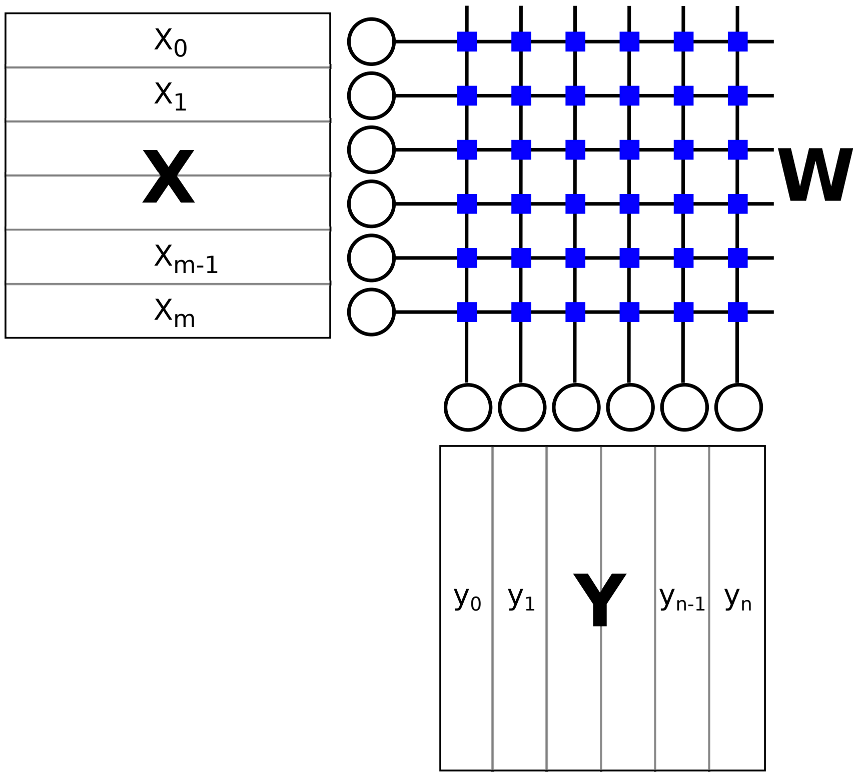


Figure 3.3: Relevant components for the mathematical interpretation of the crossbar as a dynamical system. **X** is the input data, **Y** the reference matrix, and **W** the crossbar matrix composed of memristive devices.

nonlinear kernel function f_k.

Given the nonlinearity of memristive devices we can consider them as providing a kernel function. However, the mathematical definition of a memristive kernel function differs from the general form. It has to be looked at as a two-step process. First, the integration part (Eq. 3.1), followed by an inference step that retrieves the current memristive state (Eq. 3.3). The reason for outlining the intricacies of a memristive kernel function is not to conclude on a specific functional difference, but to give an idea on how the memristive state, which is how we encode information, is depending on the components outlined in Fig. 3.3 and how the memristive states are retrieved for further processing.

In terms of kernel notation the non-linearity is not applied to a weighted sum of vector elements, but the memristive state is the sum of non-linearity applied to each vector element.

$$w_T = \sum_{t=1}^{T} f_m(x_t, y_t, w_{t-1}) \tag{3.6}$$

In this equation, w is the memristive state, x_i is an element of the input data $\mathbf{X}$, and y_i is an element of the reference matrix $\mathbf{Y}$. The memristive kernel function f_m was described by eq. 3.1 and can be rewritten for the kernel method as:

$$w_T = \sum_{t=1}^{T} \lambda sinh(\eta(x_t - y_t)) - \frac{w_{t-1}}{\tau} \tag{3.7}$$

This equation highlights, that the effective input signal to a memristive device is the voltage potential between x and y. Given that at any specific time the crossbar is driven by an input vector $\mathbf{x}$ and a reference vector $\mathbf{y}$, the matrix defining the device specific voltage potentials can then be computed by the outer product of the two vectors ($\mathbf{V} = \mathbf{x} \otimes \mathbf{y}$). Assuming we want a heterogeneous computation of the applied input vector $\mathbf{x}$. With the outer product defining the array of device specific voltages $\mathbf{V}$ in the crossbar, it then becomes apparent that the reference vector $\mathbf{y}$ can be used to create a heterogeneous set of voltages in $\mathbf{V}$.

To retrieve a memristive state for further computation after the application of the input matrix $\mathbf{X}$, a read voltage has to be applied. This read operation causes a current through the memristive device and is a representation of the device state (Eq. 3.3). We now have an understanding of the mathematical background that provides the ability to harness memristive devices as nonlinear computing elements, an understanding of how to use the overall architecture to force heterogeneous nonlinear state changes, as well as to retrieve the states from the devices for further computation.

3.4 VOLATILE MEMRISTIVE DEVICES AS SHORT-TERM MEMORY

The sections on volatile memristive devices and the mathematical interpretation of dynamic crossbars has prepared us to answer research questions 1.1 and 1.2. We have established that the key to heterogeneous processing of input data is within the design of the reference matrix. Furthermore, accessing a memristive state is done by applying a readout voltage. Within this section I will verify these concepts and investigate the internal representations within the crossbar, discuss methods to efficiently access memristive states and show how to interface the crossbar to subsequent classifiers. At the end of this section we will have a better understanding of the overall approach as well as what open questions we have to answer to further develop the concept of computing within crossbars.

3.4.1 Architecture

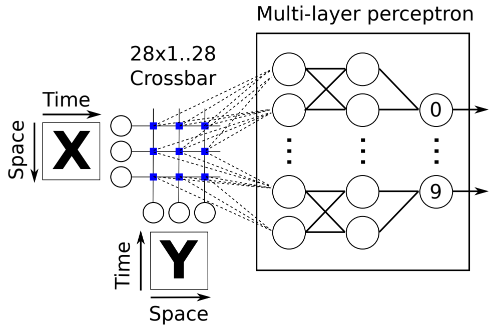
The design of a crossbar architecture useful to computation is informed by the mathematical background given above. However, first I want to create a deeper understanding of an inherent obstacle to computation within the crossbar architecture - parallelism. Given a

$m{\times}n$ crossbar, each row input x_m is connected to n columns that experience the same input signal x_m, and hence would experience the same state transitions. This would result in all n columns to exhibit the same state transitions and resulting state patterns, which creates a redundancy that is not useful to computation. With respect to the mathematical basics presented in section 3.3, the effective input signal to a memristor is not the voltage applied to one of its terminals, but the voltage difference (potential) across both device terminals. This creates the possibility of computing a row input x_m applied to n column memristors in n different ways by modulating the crossbar columns with a reference signal pattern $\mathbf{y}$ of length n.

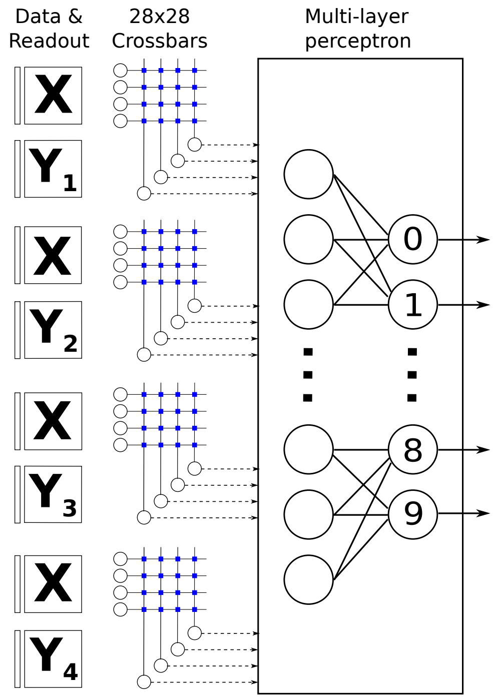
Utilization of a memristive crossbar for computation requires the combination of architectural as well as algorithmic methods. In the following paragraphs I will outline architectural aspects. Some of them will only make complete sense once the algorithmic aspects have been introduced in section 3.4.2.

In Figure 3.4 I present the memristive crossbar architecture in two different versions. Both of them consist of the crossbar itself and different methods of how to interface the crossbar states to a subsequent classifier. In this setup the classifier is a *multi-layer perceptron* (MLP). This was chosen as to compare the crossbar approach, as a pre-processing step, to existing classifier implementations operating on the raw data with similar training complexity. The $\mathbf{X}$ and $\mathbf{Y}$ matrices represent the spatiotemporal input and reference matrices respectively. I will now go into the differences between the two architectures.

Figure 3.4a considers a single crossbar driven by input $\mathbf{X}$ and a reference $\mathbf{Y}$. After the application of an input sample the crossbar exhibits a state pattern made up of the individual memristive states. The information stored as the sum of all memristive states should provide a reference of obtainable classification results. This approach makes the assumption that all memristive states can be accessed and forwarded to the MLP. Under



(a) Memristive crossbar driven by row and column data.



(b) Exemplary crossbar state space after input application.

Figure 3.4: Memristive crossbars functioning as dynamical systems by integrating the applied spatiotemporal pattern. (a) to evaluate the preservation of the input information I directly apply the memristive states to the readout layer (MLP). (b) Under the assumption that memristive device states cannot efficiently be accessed, applying a readout vector transfers the crossbar states into a lower dimensional output vector, which is used by the MLP.

this assumption I can directly use the memristive states, which are encoded as values in the interval $[0, 1]$. However, this approach is only for understanding the computational capacities of the architecture, but not feasible for a hardware implementation. In hardware, we do not have direct access to all memristive states, but would rely on a sequential procedure to read them row-wise. This step is time- and memory-demanding and would defeat the purpose of real-time computation.

A second approach that allows to transfer states to the readout layer in a single step is shown in Fig. 3.4b. Here, after the application of the spatiotemporal data, a readout vector with uniform voltages is applied to the crossbars. This step computes the inner product of the voltage-encoded readout vector with the matrix representing the memristive states in terms of conductances, and results in an output vector of currents ($I = V \times G$). The resulting currents are then converted to linearly scaled voltages in the interval $[0, 1]$ and applied to the MLP. As this produces a much lower dimensionality, this setup utilizes four crossbars [1], all driven by different combinations of the input/reference data.

3.4.2 Methodology

Some aspects of this approach have already been introduced during the discussion of the architecture. Here, I will justify them more from an algorithmic point of view. The input data used in this set of experiments is the MNIST data set [64]. Typically MNIST is not considered to be a spatiotemporal problem. Instead, the digit images are flattened into 1D vectors applied in a single step to the architecture of choice. MNIST was chosen here for two reasons. It presents a compact data set with images of size 28×28 pixels and pixel intensity can be considered binary without a real loss in information. The application of

[1]This initial work relied on the manual selection of input data combinations and four was chosen to be a reasonable trade-off to show the computational principle and competitive performance results. Future work will focus on a more generalized procedure.

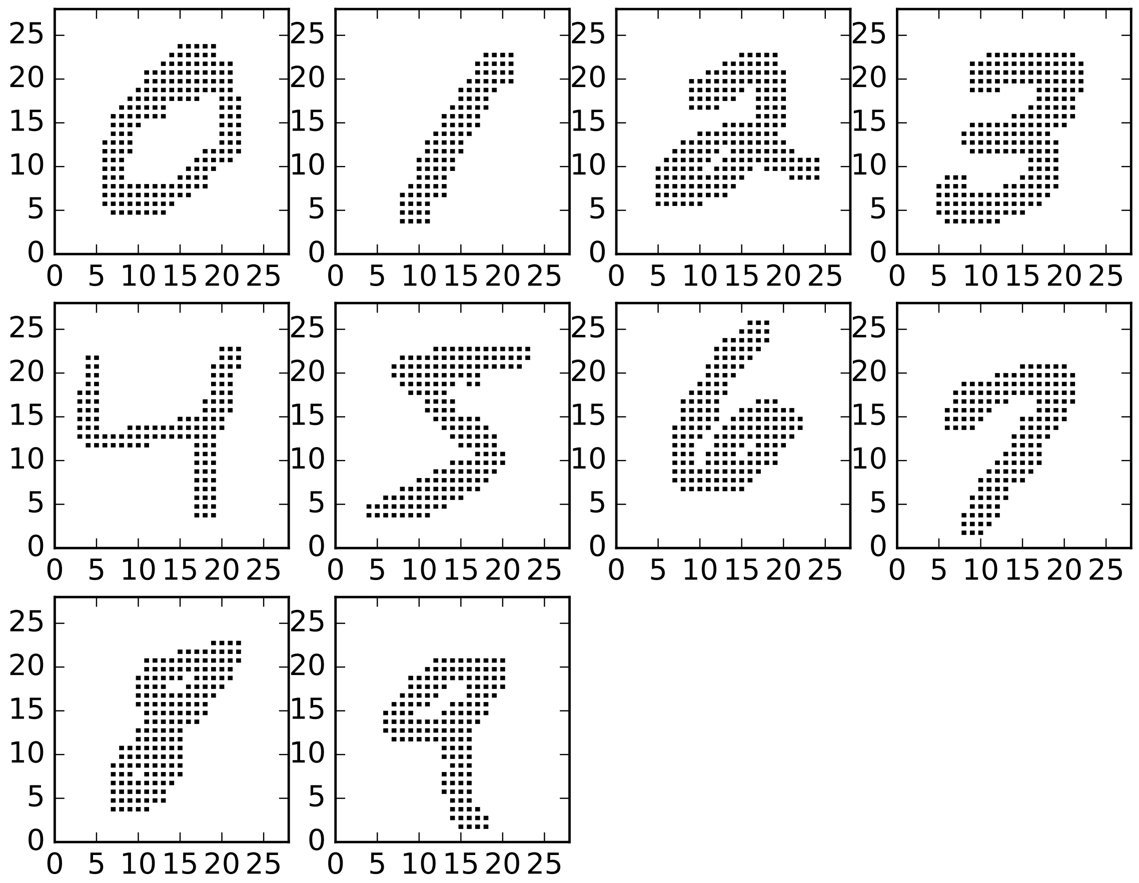
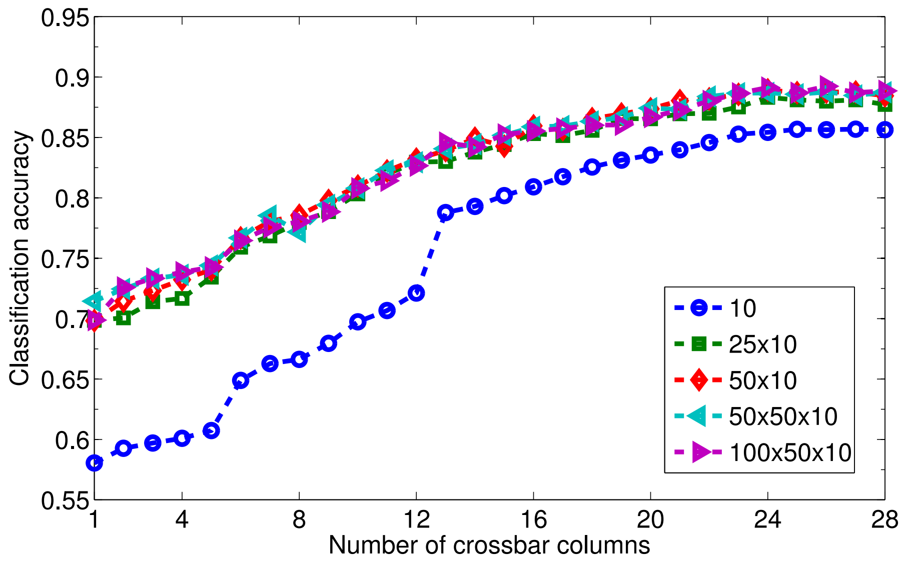


Figure 3.5: Spike representation of MNIST data. The x-axis represents time and the y-axis space. Given the MNIST size of 28 × 28 pixels, each digit is applied column by column to a crossbar with 28 inputs over 28 time steps.

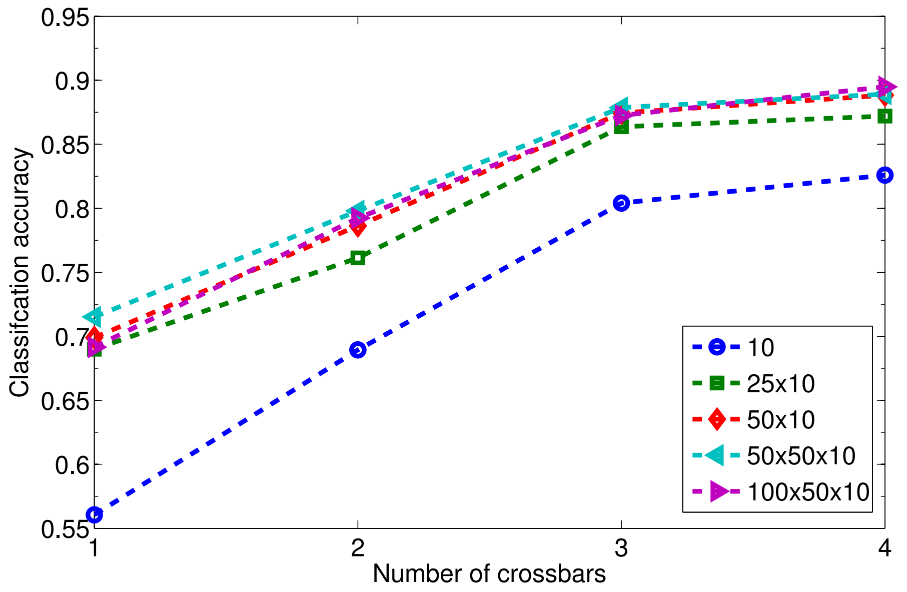
MNIST data is done as spikes and allows the adjustment of the spike amplitude and width to optimally utilize the memristive state range. A different interpretation of the MNIST data set, one that intuitively makes it easier to recognize the data as a spatiotemporal problem is shown in Fig. 3.5. Columns are applied one at a time over 28 time steps (in case of MNIST) and each dot represents a spike. From a hardware point of view, this approach was also motivated by a significantly reduced number of input neurons, which would lead to area savings.

After applying an MNIST image over space and time, a readout operation is performed to transfer the crossbar states to a lower dimensional output vector. Electrically, the readout vector is a set of uniform sub-threshold voltages that results in memristive currents determined by the individual device resistances (modeled by their internal state w). The individual device currents are summed up along the crossbar columns and provide the output of the readout operation. Under the assumption of constant and uniform readout voltages device current (equation 3.3) and device resistance (Eq. 3.3) become linearly correlated to the device state w. Computationally this step is then performed as the inner product of the readout vector and the state matrix $\mathbf{W}$.

Contrary to most RC implementations using a linear readout layer, here I use a multi-layer perceptron, allowing for nonlinear transformations (*tanh*) within the MLP. In the results section, the implications of the crossbar architecture on the MLP will become more clear. Training of the MLP is done with back-propagation and is based on the MNIST training set containing $60,000$ samples. Testing is based on a separate set of $10,000$ samples.



(a) Direct access to memristive states



(b) Incresing number of crossbars

Figure 3.6: MNIST classification performance: (a) 28×28 images applied to 28×28 crossbar. Resulting states are applied to MLP. (b) 28×28 images applied to multiple independent crossbars. Crossbar states are transferred to 28 element output vector per crossbar for classification by MLP. For 4 crossbars the 784 dimensional images are reduced to 112 dimensions without performance loss comapred to (a).

3.4.3 Results

As mentioned, the architecture shown in Fig. 3.4a was chosen to investigate how well the integration of signals over time into memristive devices preserves information. As input data as well as reference matrix we are using the MNIST data with 784 pixels (28×28). The used crossbar is of size $m \times n$ with $m = 28$ representing the spatial domain of the input and n representing the number of columns in the crossbar. To study the impact of the crossbar size n will be scaled from 1..28. In its 28×28 setup, the crossbar contains 784 devices which is the same dimensionality as the MNIST samples. Figure 3.6a shows the classification results as a function of crossbar columns and size of the MLP. For the setup with 784 devices ($n = 28$), a single readout layer reaches a classification performance of 85% and different MLP implementations of up to roughly 90%. The benchmark published in [64] reached about 95.3% with a MLP of size $784 \times 300 \times 10$. A memristor based hardware proposal of the same system size could achieve a classification rate of 93.5% [84]. Using larger MLPs, the classification performance of this approach could not be significantly improved anymore. Hence, I conclude that the integration by memristors over time did not preserve the information perfectly. A likely reason for that can be found in Fig. 3.2a. While the temporal patterns can be projected on a concave trajectory, this mapping is not unique for all temporal patterns.

The following paragraphs will answer research questions 1.1 on creating diverse state patterns and 1.2 on accessing those patterns and forward them for further processing. The second architecture shown in Figure 3.4b uses a realistic approach to access the memristor states by transferring them into a lower dimensional output vector. The first research question is concerned with the creation of diverse state patterns in the crossbar architecture where all columns are exposed to the same input data. This was attempted

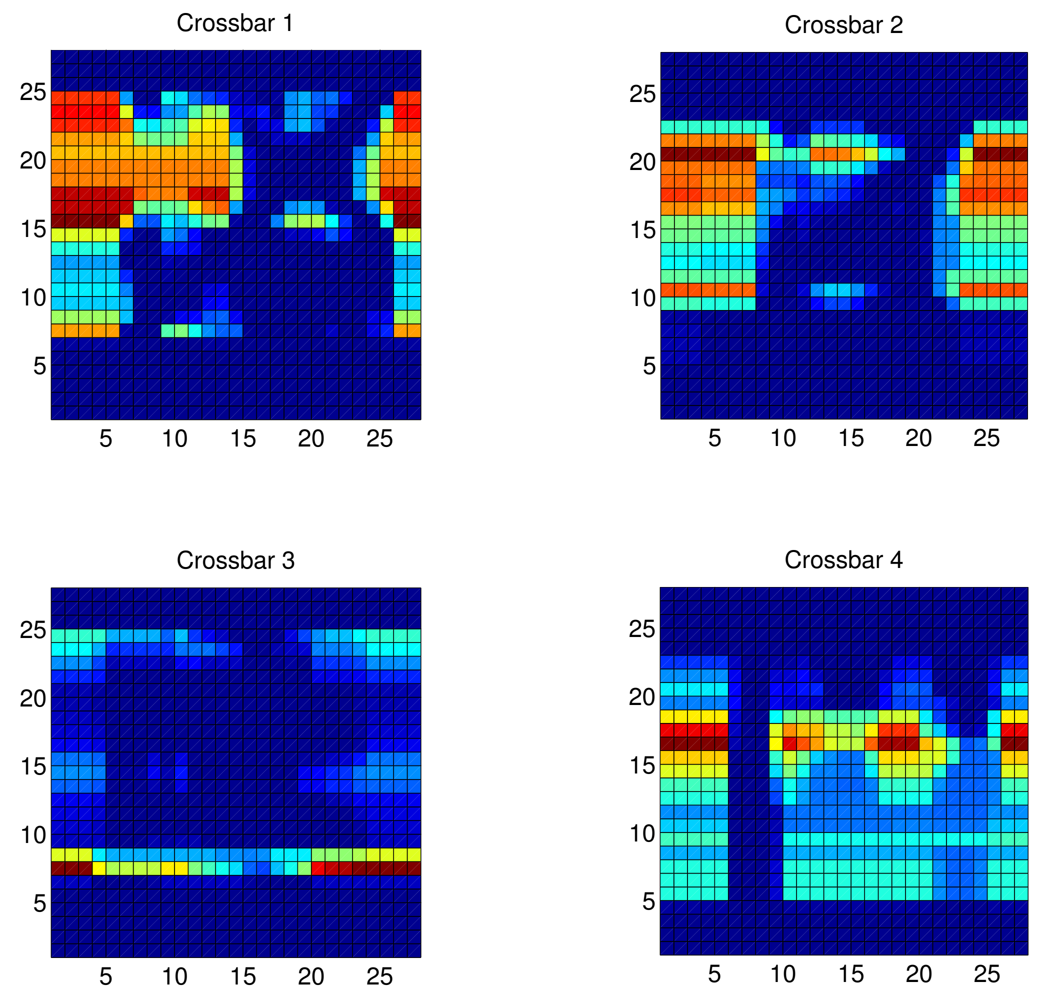


Figure 3.7: Example of a rotation- and direction-dependent representation of a digit “3” within the memristive crossbars. Different rotations and directions allow the individual crossbars to extract different temporal and spatial information and to improve classification accuracy.

by modulating the crossbar columns so that different columns experience different voltage potentials. An example of how different modulation causes diverse state patterns is shown in Figure 3.7. In this figure four crossbars were driven with a sample of a digit 3 from the MNIST data set. The difference in the state patterns were achieved by applying the digit in different rotations to the different crossbars. There are two aspects to highlight. First, if we look at only one crossbar, we can see varying state patterns along the columns (the coloring represents memristive states in the range $w \in [0, 1]$). This is the outcome of how the input matrix and the reference matrix coincided. The variety in state patterns gives rise to the assumption that we can exploit this in terms of information processing. The second aspect considers the variety of patterns across all 4 crossbars. By changing the order in which the spatiotemporal patterns are applied we create different combinations of input and reference signals ($\mathbf{V} = \mathbf{X} \otimes \mathbf{Y}$) and hence different state patterns.

The classification results are shown in Figure 3.6b and are almost identical to the ones from the first architecture. However, in its largest setup, only 112 neurons were interfaced to the MLP, which is a significantly lower dimension compared to 784. Being able to achieve comparable results with a lower dimensional vector implies that the state correlation in the lower dimensionality is smaller. This is the result of applying the data samples in different rotations to the crossbars which allows to extract different spatiotemporal patterns by each of the four crossbars. Training the MLP over 10 iterations, this approach achieved the same performance (93.6%) as the mentioned memristor-based feed-forward neural network implementation [84]. However, operating the crossbar as a dynamical system, this classification performance is achieved at a much reduced hardware cost, which is expressed by the number of neurons and synapses required. Table 3.1 gives an overview of performances based on different system setups.

The crux of this approach is the inherent state decay of volatile memristive devices. Out of this, another question that arises is: what is the optimal rate of state decay? In order to understand how sensitive our approach is to the value of the time constant, we ran a set of simulations evaluating the classification performance as a function of the time constant. Fig. 3.8 shows the results of running a 4 crossbar setup with a $100 \times 50 \times 10$

Table 3.1: Complexity and performance comparison for the presented architectures. Bold numbers refer to MLP sizes.

Setup	Neurons Input + Network	Weights	Epochs	Accuracy
[64]	784 + 300	235,200	20	95.3%
[83]	784 + 300	235,200	3	93.5%
10	112 + 122	1,120	3	82.5%
25x10	112 + 147	3,050	3	87.2%
50x10	112 + 172	5,700	3	88.8%
50x50x10	112 + 222	8,200	3	88.9%
100x50x10	112 + 272	15,900	3	89.5%
100x50x10	112 + 272	15,900	10	93.6%

MLP and a variety of time constants. In the figure the value of the time constant is normalized to the time step a single data set is applied for. $\tau = 1$ refers to τ being as long as applying a single column. With increasing τ the decay slows down and the memristive state represents a longer range of input signals.

The figure highlights three important issues. First, the memristive devices have to satisfy a minimum time constant that allows the memristive state to represent more than just the last (or the last few) signals applied to them. Second, the classification performance goes slightly down again due to the memristive devices functioning only as non-linear adders without relevant state diffusion. This means that the temporal information inherent in the pixel positions cannot be exploited. Third, the most important message: if we allow for slightly lower than maximum classification rate, we can observe a plateau of decay rates that are suitable. This is important for dealing with device variation as well as to relax the fabrication constraints for those devices.

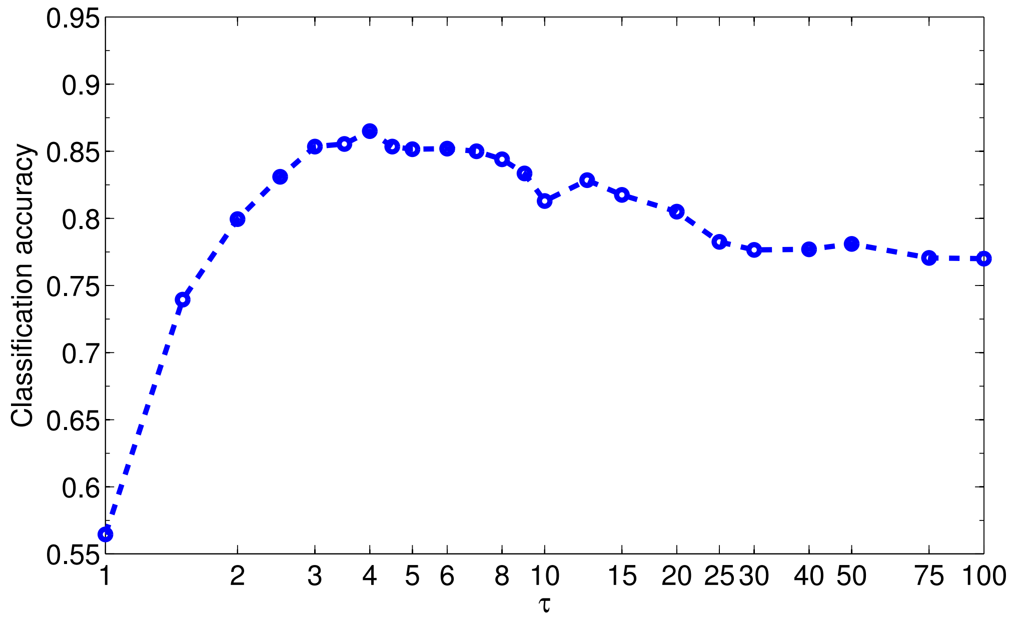


Figure 3.8: Diffusion time constant vs. classification accuracy. Strong state decay (small values) does not allow devices to combine features over the full temporal range and hence does not provide a base for good classification. Too slow state decay (large values) practically implements a device that cannot extract temporal information, but turns the system into a spike-rate encoding where devices only count the number of applied spikes.

3.4.4 Discussion

In this section I have introduced an approach that utilizes memristive crossbars composed of volatile devices to process spatiotemporal input patterns. The first research question to be addressed was on the ability to create diverse representations in a homogeneous architecture. This was answered by expanding the view to the overall architectural setup and creating diversity by modulating the homogeneous architecture with a reference signal that created heterogeneity. The second research question dealt with a feasible method of interfacing the crossbar to subsequent elements in the architecture. I've outlined that direct access to the states is not practical and shown that by performing a readout operation following the matrix-vector product, one can retrieve a lower dimensional representation within a single time step. In combination with diverse state patterns this dimensionality reduction still offered a good base for training a classifier.

3.5 OPTIMIZATION OF CROSSBAR UTILIZATION

The novel approach outlined in the previous section has demonstrated the ability to utilize intrinsic features of memristive devices for nonlinear computation and harnessing state decay to extract temporal features from applied input data. However, a few conclusions on the limitations can be drawn that lead to the methods introduced throughout this section.

When it comes to generalizing this approach, the major limitation is that we relied on using the input data as reference signals to extract features. Moreover, we did rely on manually selected rotations and combinations of the input images. This approach does not lend itself to develop a generalized method of creating the reference signals applicable to a wider range of data sets. Another aspect to the manual selection of reference signals is the lack of scalability. In case one would need more crossbars to improve, i.e., classification rates, what other combinations of input signals could be realistically created and provide a blueprint for other data sets. In this section I will show different methods to create efficient reference signals that are both, aware of the actual input data distributions and scalable. Such a reference matrix should lead to more efficient utilization of the crossbar.

This section will address research questions 2.1 on the design of the reference matrix, 2.2 discussing the implementation of excitatory and inhibitory features, and 2.4 provide methods to balance dimensionality reduction and the information content used for classification.

3.5.1 Characteristic Features of Input Data

In the next paragraphs I will outline how to derive a reference matrix based on input data statistics. Demonstrated on the base of MNIST it provides a general approach that is not limited to this data set. Three methods will be presented: a method that aims to find class specific mean representations, a *principal component analysis* (PCA) based method, and a random generation of the reference matrix. I will discuss the full approach based on the class mean representations as it offers a more intuitive visual representation along the way as compared to the PCA. The described methods will, as already mentioned, discuss how to derive a Y matrix as well as how to implement features acting excitatory or inhibitory. Hence I will address research questions 2.1 and 2.2.

For the class mean representations as well as the PCA approach I use the MNIST training set of $60,000$ samples to obtain the reference matrix. The same reference matrices are then used during the testing phase based on a separate $10,000$ sample test set.

Method 1: Class Mean Representations

The first method to create a reference matrix **Y** aims to extract characteristic features from the input data and translate them into a reference matrix that allows extraction of said features. At the core of this approach is to find a method that can produce a set of characteristic features based on available training data.

In a first method I'll extract characteristic features based on class differences. The question I ask here is: what, on average, separates a given digit best from other digits? The first step is to compute the average digit for each class.

$$m_i = \frac{1}{N} \sum_{x \in C_i} x \tag{3.8}$$

The mean values m_i of each class C_i appear to be some blurry example of the corresponding class samples. The next step creates characteristic features by subtracting the mean across all classes, but the target class from the target class.

$$D_i = m_i - \frac{1}{C-1} \sum \forall m \neq m_i \tag{3.9}$$

The result of this subtraction can be seen in Fig. 3.9. Red represents areas in which a given class is statistically more present than the average of other classes. Blue represents areas in which other classes statistically show more activity. In other words, if we can control the crossbar to integrate the red and blue areas we gain statistical information on the target class. Here, red areas are considered excitatory and blue areas inhibitory signals with respect to determining a class label.

Given a set of representations that indicate dominant features of each digit, in the next step I reduce the dimensionality in order for the Y matrix to become selective to the

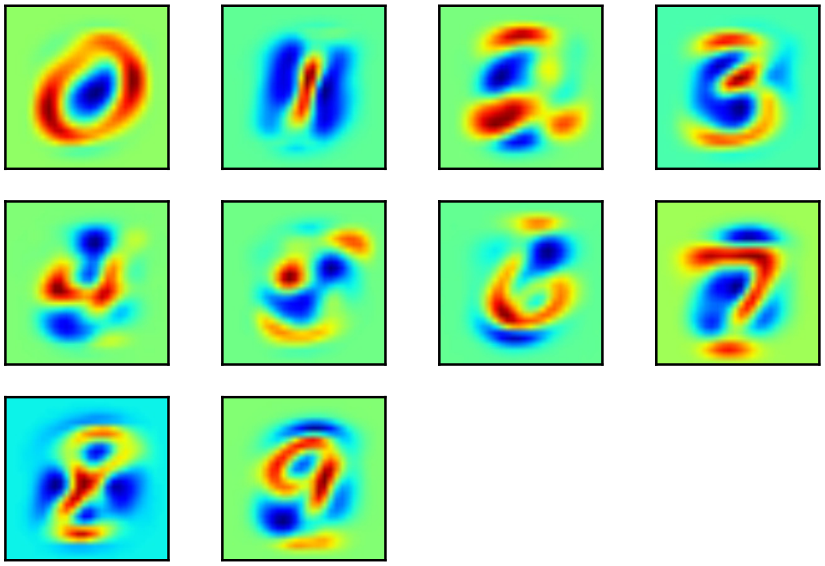


Figure 3.9: Characteristic features of digits as computed by the class mean representation. Red features are characteristic (excitatory) for a given class. Blue features are discouraging (inhibitory) the detection of a given class. Features were obtained from the MNIST training set.

dominant features. Thresholding is applied to features in order to keep only the dominant ones. A result for a threshold value of 0.85 of the maximum can be seen in Fig. 3.10. The subplot in the lower right corner is a plot of all features added in one image to illustrate the different shapes and locations of the features.

The next question is how to translate the features selected by thresholding into a reference matrix **Y**. Spatial features of the input data appear within a specific column, while rows contain temporal features. In contrast, the reference matrix' temporal domain is encoded in its columns, while rows represent the spatial distribution of features.

Assuming that the computed features are descriptive of the target classes, they can be translated to form the reference matrix. In Figure 3.11 I show the feature matrices from Figure 3.10 concatenated into a single vector. To address the different orientation of spatial and temporal features between the input data and the reference matrix the features

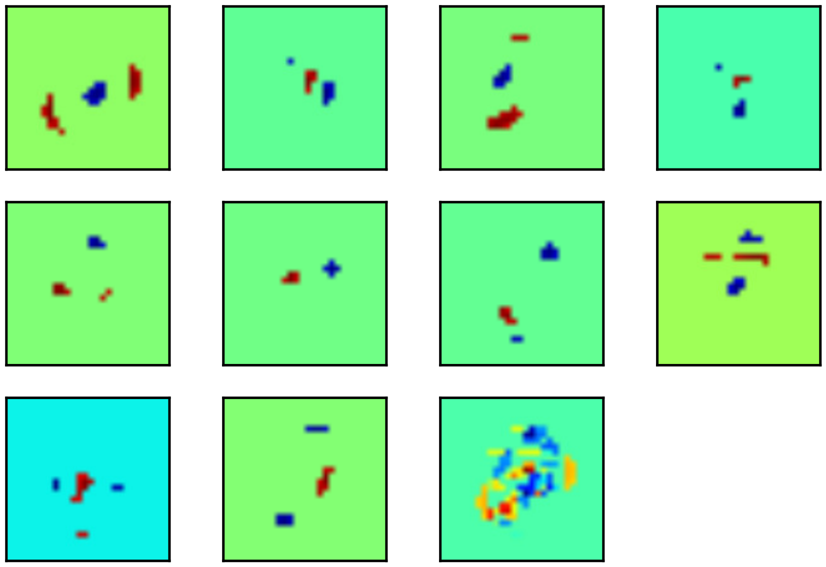


Figure 3.10: Thresholded features of digits in the order from 0 to 9. Red coloring represent excitatory features toward a specific class label, blue coloring represent inhibitory features. The lower right plot shows all features within a single plot.

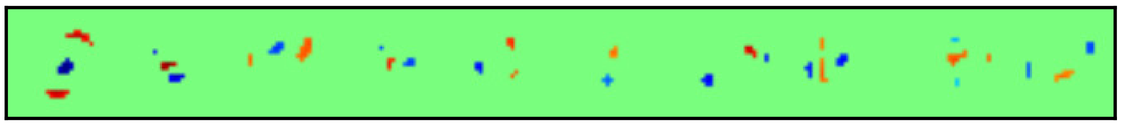
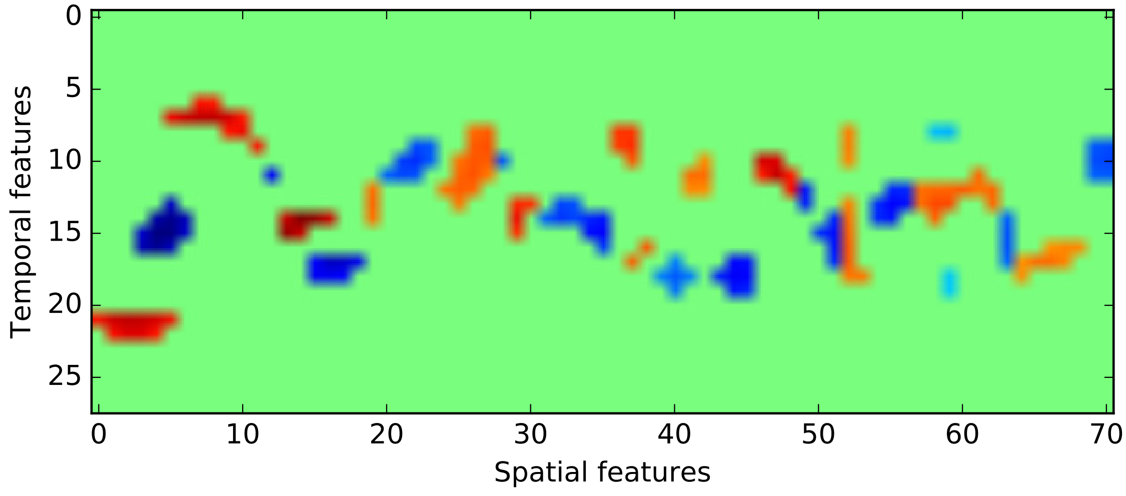
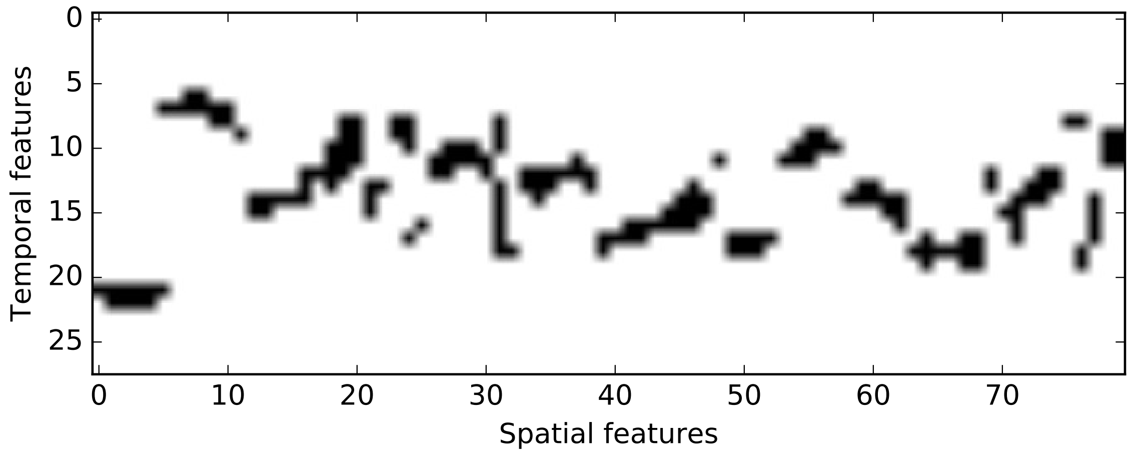


Figure 3.11: Single **Y** matrix comprised of transposed image features.



(a) Thresholded excitatory and inhibitory features



(b) Resulting reference matrix with separated excitatory and inhibitory features

Figure 3.12: Reference matrix features. (a) Reference matrix containing all features without empty columns. (b) final reference matrix that spatially separates excitatory and inhibitory features to allow separate recognition of such features. The resulting black features represent a spatiotemporal code for when and where the crossbar integrates the input signals applied to it.

within **Y** are the transpose of the originals.

In a last step to create the reference matrix **Y**, I'll address how to make the best use of the excitatory and inhibitory signals present in the characteristic features of each digit. As the output signal of a crossbar is derived by the inner product of a uniform row vector with the weight matrix **W**, specific weight distributions across a crossbar column can get lost within the sum over all elements. Same is true for a single memristive device integrating excitatory and inhibitory signals. Based on the resulting state, one might not be able to distinguish between the presence of both, excitatory and inhibitory signals, or an absence of both. Hence, I separate excitatory and inhibitory features into independent reference matrix columns to allow independent integration of both signals. This can be seen in Figure 3.12.

Method 2: Principal Component Analysis

In a second approach, and maybe even a bit more intuitive one, I am using principal component analysis (PCA) to extract the most descriptive features [81]. This approach differs to the first one as it is agnostic of specific classes in the sense that it doesn't require any knowledge about class labels. To do so, I flatten the MNIST data into a two dimensional array with the rows containing the different samples and the columns the 784 variables (pixels). The PCA results in a 784×784 output matrix with each row containing a principal component. With the principal components giving a representation of the MNIST data that are as much as possible linearly uncorrelated these components promise to provide features that are useful to extract relevant information from the MNIST data.

The components are ordered in order of most descriptive to least descriptive and in Figure 3.13 I'll show the 16 most descriptive principal components. After deriving the

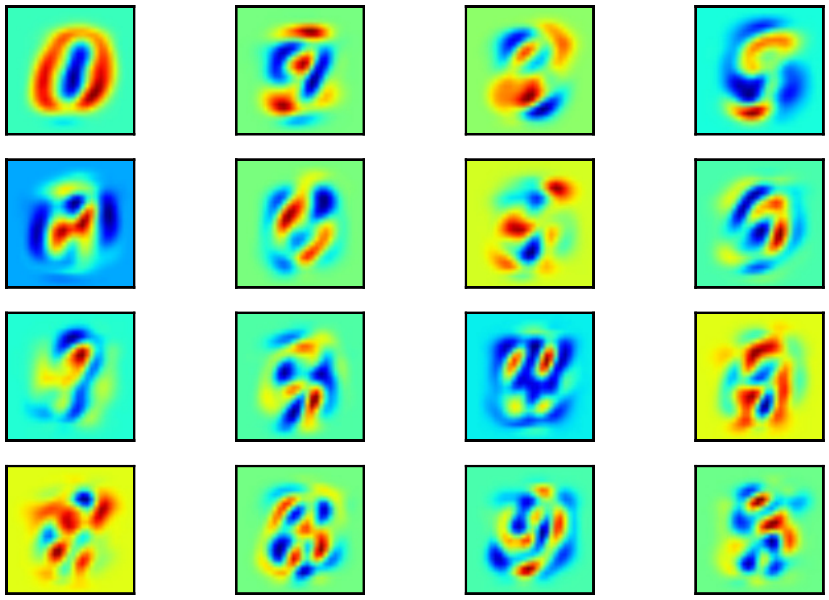


Figure 3.13: Set of principal components of MNIST digits. While the PCA approach is agnostic of the class labels, it does provide sets of excitatory and inhibitory features characteristic of the given data set similar to the class mean approach. Features were obtained from the MNIST training set.

principal components thresholding is applied to extract the dominant features and the **Y** matrix is created just as in the previous example for the class mean representation. The features itself appear, visually, to be similar to the mean features. Hence, for the sake of brevity no further figures are shown for the PCA approach.

Method 3: Random Feature Selection

In a third approach I will utilize randomness to create the **Y** matrix. In [44] randomness was described as providing good generalization properties in the context of extracting features by means of a random, fixed weight matrix. In Fig. 3.14 such random matrix is shown. The design of the matrix is controlled by a sparsity term that determines the amount of matrix elements that would allow integration of the applied input signal **X**. The shown matrix has a sparsity of 0.1 which means that 10% of the matrix elements (black

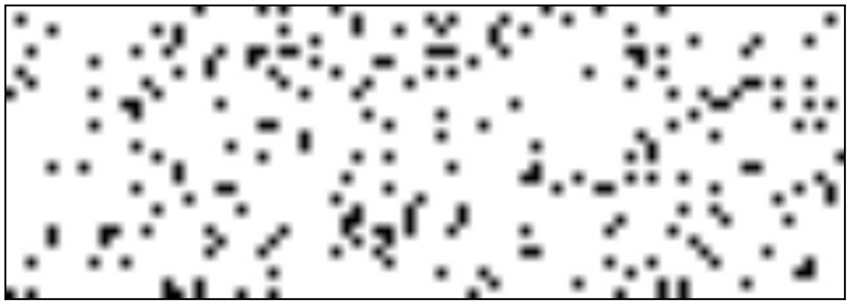


Figure 3.14: Random **Y** matrix with black dots indicating the spatiotemporal code for when a crossbar can integrate an input signal.

dots) allow integration of the input data. To reiterate, the position of a black dot along a row defines what crossbar column will integrate an input signal. The position of a black dot along a specific column in **Y** defines at what time step the corresponding crossbar column is receptive to an input signal.

3.5.2 Crossbar Architecture

The architecture used here follows the same principles as in section 3.3. At the center of the approach is a crossbar with volatile memristive devices. The devices are driven by spatiotemporal input data and a reference matrix is applied to the crossbar columns to achieve heterogeneous integration of the input data across the crossbar. The focus on optimizations of the crossbar utilization also has a few implications on the architecture. I will briefly mention them here and explain them more detailed in the results sections as to maintain concise descriptions of the different experiments conducted.

The creation of a generic reference matrix that does not rely on manual choices of the design allows to work with a single memristive crossbar whose number of columns is defined by the amount of columns within the reference matrix. In addition, contrary to the architecture presented in section 3.4.1 I will use a ridge regression classifier, which allows to better attribute any potential increase in classification rate to the data representation obtained from the crossbar, rather than from possible further nonlinear mappings within a multi-layer perceptron.

3.5.3 Experiments

Throughout this section I will present different experiments and their impact on the classification rate based on the MNIST data set. Each subsection is more or less self-contained and the results presented do not directly compare as the individual experiments explored different setups (crossbar sizes, number of readouts, reference matrix design, etc.). Each subsection will outline one aspect of optimizing the computation with memristive crossbars. At the end I will present results that combine the lessons learned from all the subsections.

Crossbar Readout

So far the readout of the crossbar matrix $\mathbf{W}$ was performed by the inner product of a row vector $\mathbf{r}$ and $\mathbf{W}$ itself ($\mathbf{o} = \langle \mathbf{r}, \mathbf{W} \rangle$). Given $\mathbf{W}$ of dimension $R^{m \times n}$ with m rows and n columns this operation results in a dimensionality reduction from $\mathbb{R}^{m \times n}$ to $\mathbb{R}^{n}$. This leads to the question if one can derive a mathematical approach to estimate the impact of dimensionality reduction and in turn trying to optimize it based on a given objective function?

There exist mathematical theorems on dimensionality reduction, i.e., compressive sensing, Johnson-Lindenstrauss lemma (JLL), random down-projections [11, 13, 29, 52]. The specific set of constraints imposed due to maintaining hardware plausibility of this architecture imply limits to the degrees of freedom with which one can perform dimensionality reduction. In the following paragraphs I'll briefly relate these constraints to the mentioned general methods.

The basic concept motivating the JLL is to preserve relative distances when reducing dimensionality of a vector. If we would treat the crossbar as a single vector in $\mathbb{R}^W$ ($W = m \times n$), then the JLL could give an estimate about the introduced error when reducing the dimensionality to $\mathbb{R}^n$. However, the given crossbar approach reduces dimensionality not by (randomly) selecting individual elements, but by summing whole columns into a scalar value. This does not allow to directly apply the JLL to study the error introduced by the readout operation of the crossbar as we cannot preserve the relative distances when compressing the crossbar columns into scalars.

With respect to compressive sensing, while the sampling faces the same issue as just discussed, the assumption of sparsity in the higher dimensional representation (crossbar states) is dependent on the input data distribution and cannot be safely assumed. In the case of the MNIST data and a random reference matrix with 10% active elements (sparse), 70% of crossbar elements were in average non-zero. Hence, the crossbar approach does not implement the sparsity constraint of compressive sensing required to correctly reconstruct a signal in an underdetermined system.

Similarly to the JLL, random down-projections also assume individual access to elements in the higher dimensionality to be directly mapped to the lower dimensionality. Based on the raw MNIST data applied to a classifier this works (also thanks to the amount of redundancy in the data). Trained on all 784 pixels, a ridge regression classifier achieved 86% classification rate. A random down-projection (random subset) of 300 elements lead to an average of 85%. In order to employ random down projections, one needs to read out one column/row at a time and repeat that step until a sufficient number of devices are read. However, with the readout being column/row-wise the random selection of elements needs to be performed after most of the crossbar has been read. Reading all/most devices of the crossbar only to afterwards narrowing it down, contradicts the goal of a time and

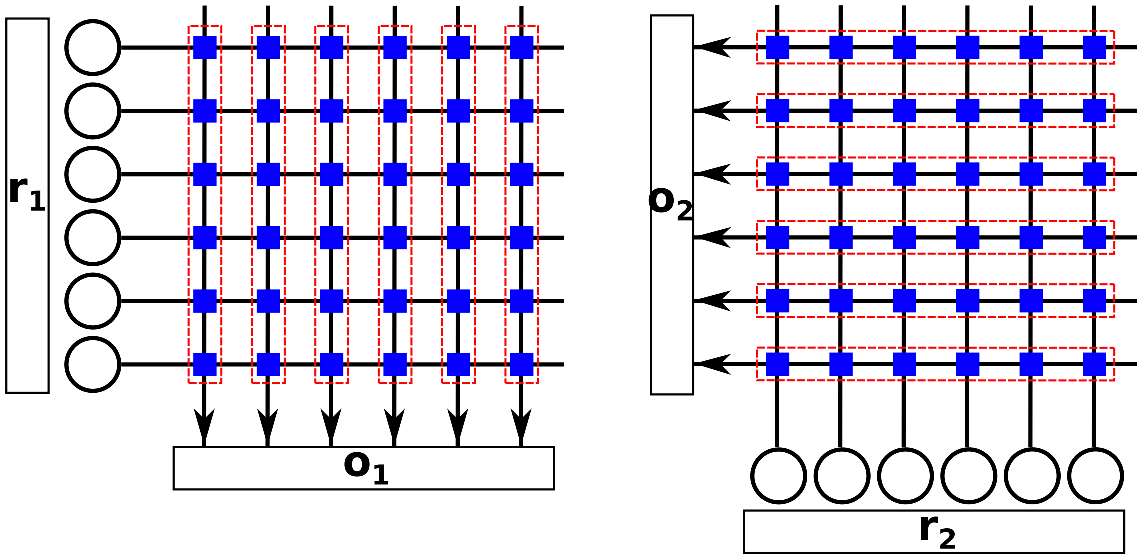


Figure 3.15: Two step readout process of crossbar. The first step applies a uniform readout vector $\mathbf{r_1}$ to the crossbar which produces $\mathbf{o_1}$. This step practically sums up the memristive states along the crossbar columns. In a second step readout vector $\mathbf{r_2}$ is applied to the crossbar columns. This step sums up the memristive state along the crossbar rows and produces output vector $\mathbf{o_2}$.

resource efficient operation of the crossbar.

Given the limited applicability of existing mathematical approaches, I'll not aim to find a mathematical solution to the problem of how much information we can preserve from one dimensionality to the other, but rephrase the problem at hand to how we can increase the amount of information obtainable from the crossbar.

To approach this question I reinterpret the functionality of the crossbar array. The crossbar dynamically integrates input data governed by the reference matrix. This implies that the input data is stored in a different representation within the crossbar. The two-dimensional distribution of the individual memristive states is then what encodes the applied input and which we want to get a lower-dimensional image from. A very similar problem to that is that of error checking and correcting codes (ECC) in dense memory arrays [50]. ECCs aim at detecting precise positions of bit flips that would cause mem-

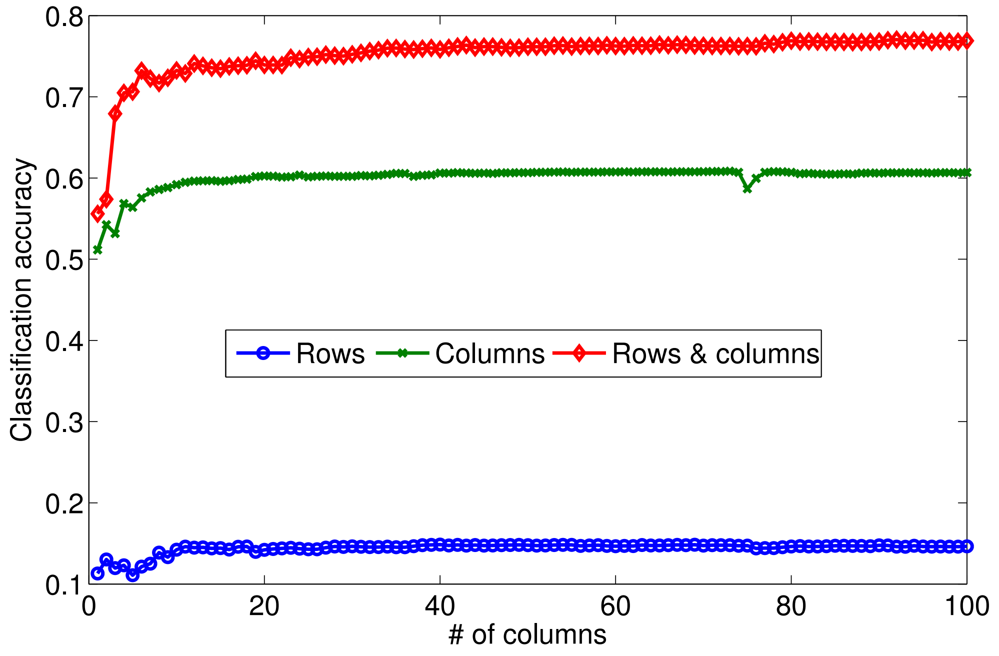


Figure 3.16: Classification rate as a function of the number of columns within a crossbar for different readout methods. Reading out the crossbar along its row inputs results in very poor classification. Reading out along the columns shows a significant improvement. Most importantly, though, is that the combined classification rate is more than the sum of the two readout methods and allows to further improve classification rate as the number of crossbar columns increase.

ory corruption. In the general sense detecting a bit flip and detecting a pattern of stored data are very similar applications. While many ECC techniques are designed to recover single bit failures, detecting and correcting multi-bit failures typically infers large overhead [55, 56]. A 2D ECC approach was shown to be able to recover multi-bit errors over large memory arrays with only slight overhead increases as compared to single-bit recovery schemes.

In this work I will introduce a 2D crossbar readout that, similarly to the ECC, can give a much better coverage of the distributions of device states within the crossbar array. The 2D readout is a two-step process in which first all memristive states along columns are summed up into an output vector and following we sum up all memristive states along the rows of the crossbar. Mathematically the two output vectors $\mathbf{o_1}, \mathbf{o_2}$ are obtained by $\mathbf{o_1} = \langle \mathbf{r_1}, \mathbf{W} \rangle$ and $\mathbf{o_2} = \langle \mathbf{r_2}, \mathbf{W^T} \rangle$. In hardware this can be easily realized by applying the readout vector to the crossbar rows and reading at the columns to obtain $\mathbf{o_1}$, and by applying the readout vector to the columns and reading along the rows to obtain $\mathbf{o_2}$. An illustration of the process is given in Fig. 3.15.

The results of investigating the 2D readout operation are shown in Fig. 3.16. The experiment assumed a crossbar with 28 rows, according to the dimensions of the MNIST data set and an increasing number of columns. The reference matrix was chosen to be random with 10% active elements. In a first step a 1D readout was performed along the columns ($\mathbf{o_1} = \langle \mathbf{r_1}, \mathbf{W} \rangle$). We can observe that the maximum classification rate of around 60% was achieved rather quickly after a crossbar size with 20 columns. Any additional increase in the number of columns didn't further increase the classification rate. This can be explained by the lack of access to different distribution along the crossbar columns. Summing up along the columns eliminates any information on which elements within a column was most informative and reduces the information to be encoded in the amplitude of the output signal. The same experiment was performed for summing along the crossbar rows ($\mathbf{o_2} = \langle \mathbf{r_2}, \mathbf{W^T} \rangle$). We can see that the classification rate with that approach is just slightly above chance. First of all, an increasing number of columns has less of an effect here as the total number of outputs is determined by the crossbar rows, not the columns. Second, if we look back at Fig. 3.7 it becomes clear that there is typically less variation in the memristive states along a specific row than there is for the columns. However, if we combine the two approaches we can significantly increase the classification rate. Two things are noteworthy here. First, the resulting classification rate is more than simply the sum of the two 1D approaches. Second, the classification rate increases, slowly, with additional crossbar columns. This supports the approach of the ECC methods where a 2D readout can achieve a much better image of a 2D array with little overhead as compared to a 1D approach.

Multiple Independent Readout Operations

As we have just seen the effect of scaling the number of crossbar columns quickly stagnates and does not lead to a significant continuous increase in classification accuracy. While the crossbar itself might hold more information as the number of columns increase, the "inner-product style" dimensionality reduction along rows/columns cannot necessarily capture additional information. An explanation to that is given with the concentration of measure theory. In [103] it is described as "a random variable that depends (in a "smooth" way) on many independent variables (but not too much on any of them) is essentially constant." The random variable here is our crossbar output, the independent variables are the crossbar elements along a column/row (depending on the readout direction), and the fact that the output does not depend more on one element than on others is given by summing all values with equal significance. While the reference matrix of the crossbar ensures that different elements of the crossbar are integrating different input data, the more of these elements get "lumped together" to a scalar value, the less informative this scalar is with respect to recognizing, the different classes of the applied input data.

In other words, increasing the number of elements that get compressed to a scalar will lead to an output vector in which the scalars that comprise that vector have less variance and are therefore less informative about the more detailed distributions within the crossbar. To better capture the state distributions within the crossbar we need to break up the vector in smaller sections to capture distribution differences for different input data. Here I will present two approaches that are realizable computationally as well as in

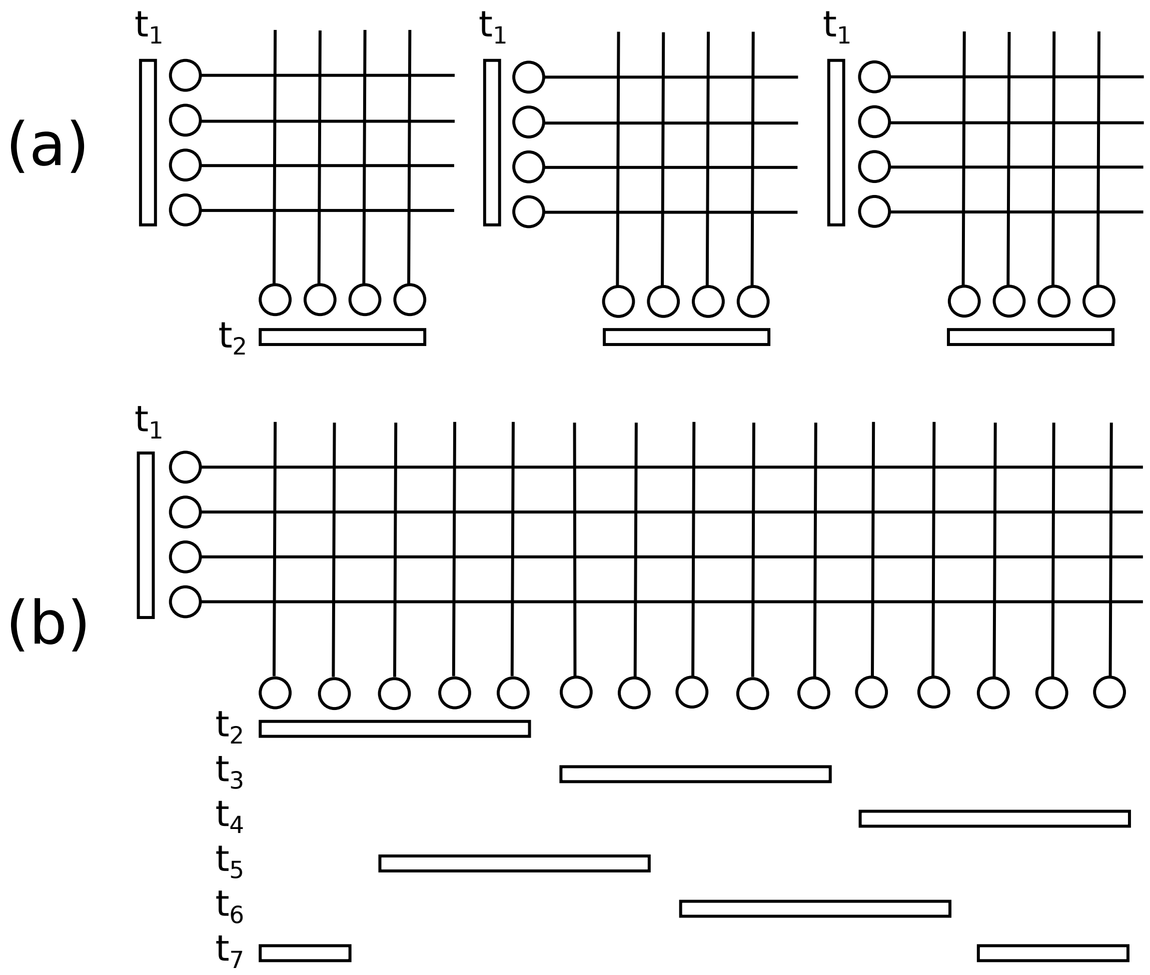


Figure 3.17: Crossbar readout. a) independent crossbars with two-step readout. b) single crossbar with multi-step readout to extract different signal distributions. The trade-off lies withing hardware resources and time steps to obtain data. Independent crossbars require more hardware but can be read out in two steps. Single crossbar can reuse the row inputs, but requires multiple time steps to get similar fine-grained readout as independent crossbars.

hardware. Fig 3.17a shows a set of independent crossbars of a fixed size along it's rows and columns. In such a setup each crossbar has it's own set of input neurons and output neurons and can be read out in two time steps. One step to compute $\mathbf{o_1} = \langle \mathbf{r_1}, \mathbf{W} \rangle$ and another one for $\mathbf{o_2} = \langle \mathbf{r_2}, \mathbf{W^T} \rangle$ were $\mathbf{o_1}, \mathbf{o_2}$ are the outputs of a particular crossbar. The advantage of this approach is the reduction of vector lengths that allows to address the limitation formulated within the concentration of measure concept as well as the two-step readout to obtain all crossbar outputs. These advantages come at the cost of multiple sets of input neurons to the independent crossbars. A second approach harnesses a readout vector length reduction just as well, but saves input neurons by taking more time steps to readout the crossbar. In 3.17b a single large crossbar is depicted, which could be scaled arbitrarily. To obtain the distributions of the information stored in the crossbar readout vectors of length r with $r < n$ are applied (here n is the total number of columns). Different subsets of the crossbar are obtained by choosing the length and location of the readout vectors. This approach allows combining arbitrary sections of the crossbar and the amount of information obtainable will be determined, among others, by how many readout steps the system design allows.

An experiment was setup as follows. The first approach of defining physically independent crossbars was implemented as four crossbars of size 28×25 each. Each crossbar is exposed to the same input data, but has it's own random reference matrix. The crossbars are then read out in two steps as explained in the previous paragraphs. Each crossbar will result in 53 output signals $(28 + 25)$ which are used to train the ridge regression classifier. Based on the whole MNIST data set this led to a classification rate of around 75%. The second implementation used a single crossbar of size 28×100. So in terms of memristive states it is identical to the first setup. In Fig. 3.18 I then show the classification rate as a function of the number of readout operations performed and used for classification. The

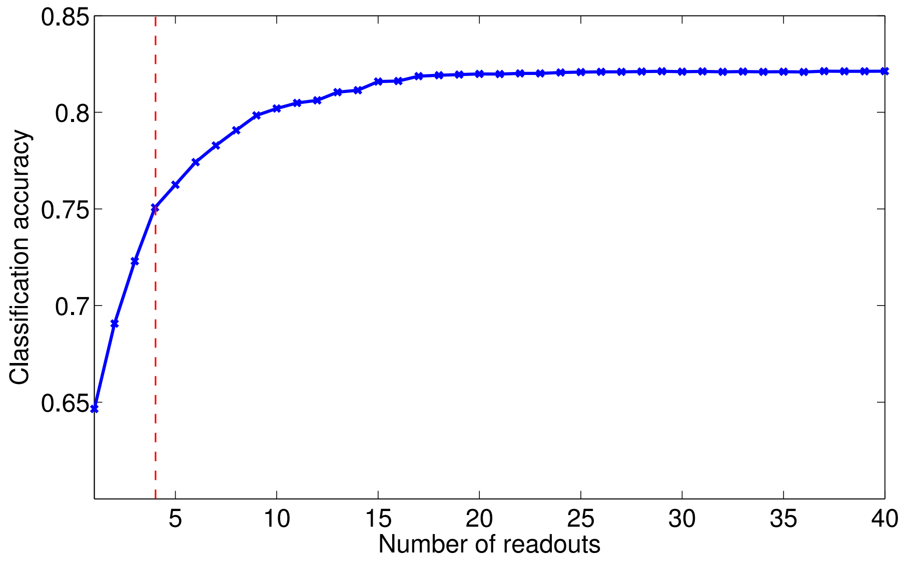


Figure 3.18: Classification performance as a function of independent readout operations. The crossbar had 100 columns and was read in chunks of 25. The dashed red line indicates four readout operations which covered all columns. Performing further readouts by combining random columns into chunks of 25 further increases the performance until it converges after around 18 readout operations (550 data points).

first four readouts were identical to the first setup, as they were non-overlapping readouts of size 25 as to cover all columns (similar to time steps t_2-t_4 in Fig. 3.17b). Further readouts were than performed by random combinations of 25 out of the 100 total columns. This was done in order to combine various features across the crossbar. The dashed red line indicates four readout operations for which the first and second setup were identical in terms of signals extracted from the crossbar(s) and also showed identical classification rates. However, the important message is that we can further increase classification rate as we introduce more readout operations that implement different combinations of crossbar columns. This is an important finding as it allows to better balance future implementations with respect to the inherent trade-offs between classification rate, hardware resources, and computation time.

Y Matrix Features

In section 3.5.1 I have discussed different methods to create the reference matrix $\mathbf{Y}$. Two methods based on the input data statistics and a random approach. I'll compare the different methods from two different angles. The first one is to determine, based on the same size of the reference matrix, which method performs best. The second aspect is to understand the scalability of the different methods.

The control of the reference matrix size varies among all three approaches which I will outline now. In the simplest case, the random reference matrix, the size is determined straight forward by defining the target amount of columns. For the two approaches that derive features from the training data the reference matrix size can only be defined indirectly. The class mean method creates one set of features per class (Fig. 3.9). For each class a thresholding parameter θ is applied in order to extract only the most significant features with respect to what separates a class from others (Fig. 3.10). Therefore,

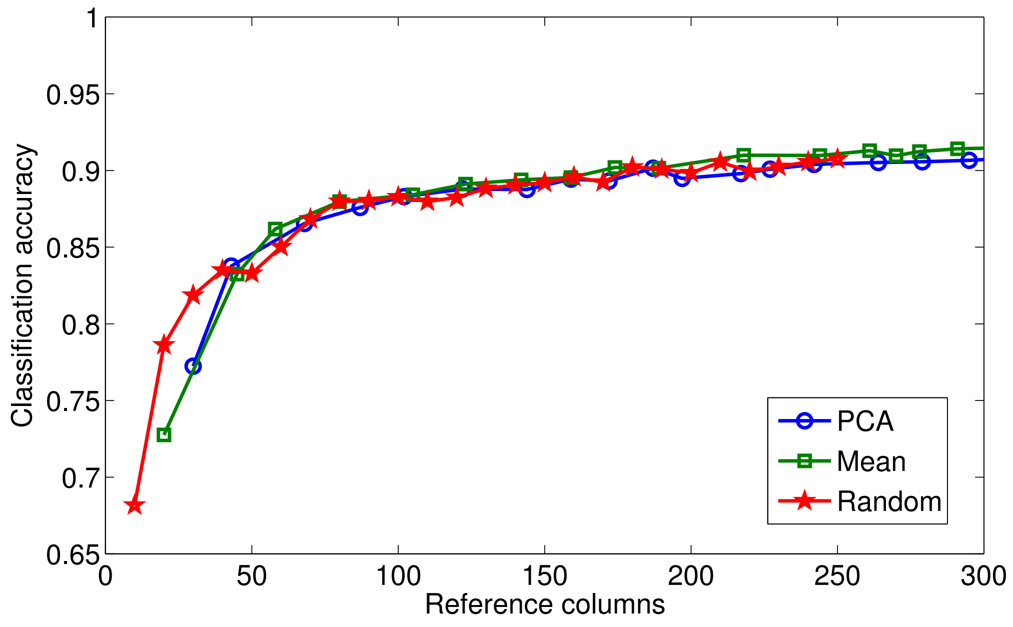


Figure 3.19: Classification accuracy for different methods to create the reference matrix as a function of matrix columns. The class mean approach extracting class specific features performs slightly better than the PCA approach that extracts features without class awareness. With a sparsity of 0.1 the random reference matrix achieves competitive classification accuracy.

as this method only creates one set of features per class, the parameter to control the size of the reference matrix (the amount of features represented) is the threshold factor θ. A high threshold will lead to few significant features and smaller reference matrices. Lower thresholds lead to a wider set of features, including less descriptive ones, but to a larger reference matrix. The PCA method creates a principal component per input dimension. Hence, in case of the MNIST data set this leads to 784 principal components. While a threshold factor also determines the feature extraction of a single principal component, this parameter can be fixed and the reference matrix can be set by choosing the amount of principal components to use.

While varying the size of the reference matrix, the experiments also incorporated the methods established above. A 2D readout is performed to obtain a better coverage of the crossbar state patterns. To avoid the concentration of measure limitation for large readout vectors, the crossbar is read out in sections of 10 columns. I.e., for a crossbar 100 columns this would result 10 readout operations along the columns.

Based on the results in Figure 3.19 the first conclusion to draw is that the overall concept is rather agnostic to how the reference matrix is designed. All three methods to create a reference matrix lead to results of within 1% classification rate on the MNIST data. The random matrix, created without any knowledge of the input data achieved up to 91%. In [44] it was argued based on an extreme learning machine implementation that randomness has good generalization features and can lead to competitive classification rates. This assumption also holds true for the crossbar approach. However, the ability to extract signals efficiently with the random reference signals depends on the sparsity of the random matrix. This can be seen in Fig. 3.20. Sparse reference signals lead to the highest classification rates, while a loss of sparsity causes the classification rate to drop. The second conclusion addresses scalability. All three methods continue to see modest

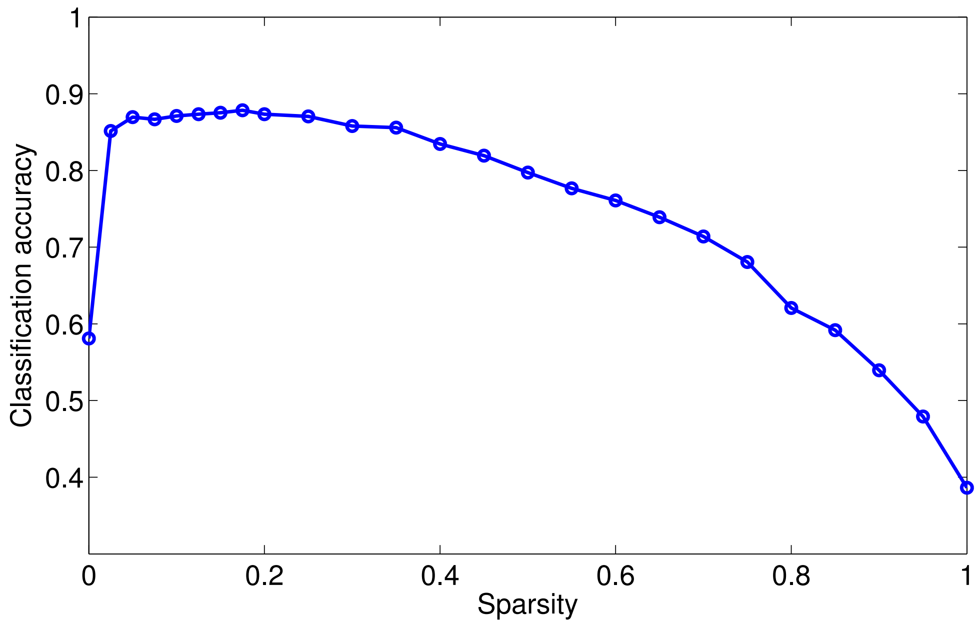


Figure 3.20: Classification rate with differently sparse random reference signals. Contrary to the PCA and class mean methods for creating the reference signals, the classification with random references is sensitive to sparsity. Loosing sparsity will not allow the crossbar to extract descriptive features and the crossbar columns exhibit more linear dependency.

classification rate increases as the size of the reference matrix grows.

Neuron Activation Function

So far all results have assumed that the crossbar output is simply the inner product of a readout vector and the crossbar matrix **W**. However, neural networks typically employ an activation function to the weighted sum of the neuron inputs. As was outlined in section 3.3 the nonlinear memristive kernel function is operating element wise on the applied inputs. The actual state of an output neuron is the linear sum of the the element wise nonlinear operations. Based on the same experimental setup as used for evaluating the reference matrix features, I have applied a *tanh* activation function to the linear sum of the crossbar outputs. This additional function helped to increase the classification rate in average by 1% and in its maximum setup from 92% without activation function to 93% with activation function.

3.5.4 Discussion

Throughout this section I have demonstrated various ways in which to better utilize memristive crossbars for computation. Research question 2.1 was answered based on utilizing either a class mean approach or principal component analysis. By extracting discriminative features from the training data, a reference matrix could be designed that allowed the crossbar to capture discriminative features as they appeared over time. Also part of the design method for the reference matrix was the separation of excitatory and inhibitory features which was the topic of research question 2.2. By creating the reference matrix columns in way that they contained either excitatory or inhibitory features it was possible to capture such features separately. Research question 2.3 dealt with the issue of the combining random features from the input data when the crossbar architecture does not provide a random structure. It was shown that by creating random readout operations one can combine random sections of the crossbar and further increase the classification rate. All methods combined (2D readout, short readout vectors and multiple crossbars/readout operations, a reference matrix derived based on the training data, and an activation function) a classification rate of above 91% could be achieved with an untrained crossbar and a single linear output layer. In terms of hardware a hardware implementation that does not need to simulate memristive devices, this would provide an extremely compact implementation with minimum training complexity.

3.6 DEVICE AND CYCLE VARIATION

In this section I will investigate the impacts of device variation (research question 3.1) and cycle variation (3.2) and to which extent they affect the results presented in section 3.5.3 when assuming ideal devices.

Device variation [2, 59], will be an inherent part of memristive devices. On a physical level these devices will experience *spatial variation*, which is a result of thickness irregularities in the different physical layers [32]. First simulations on device variation in connection with mesh networks have shown the general reservoir computing immunity to a significant level of variation [17]. The crossbar approach relies on the nonlinear integration of states and the state decay. It is likely that the crossbar approach is very robust to device variation, because it does not rely on a uniformity of the devices, but on the dynamical state changes instead. In this context device variation can be considered a manifestation of heterogeneity among devices and variation might be useful to the dynamic computation [51, 72, 87, 88].

However, a second type of variation that memristors are subject to is cycle variation or *temporal variation*, which describes stochastic switching behavior [32]. Given a single memristor, an initial device state, and a fixed pulse applied, the resulting device response can vary to a certain degree. In other words, the device behavior is not completely deterministic. Cycle variation is not only a phenomenon found in nanoelectronics, but also in biology. In [41] the stochasticity of synapses is described as an effect caused by random variations in the release of neurotransmitters responsible for the magnitude of a post-synaptic current. As the crossbar approach relies on mapping the same input samples to the same state representations, it is necessary to investigate the impact of cycle variation on the stability of the classification performance.

3.6.1 Modeling

Device variation is modeled by modifying the internal parameters of the device models. All memristive devices were modeled individually with their own set of parameters. The variation of those parameters is based on experimental data of the *relative standard deviation* (RSD). To investigate the impact of device variation on the classification performance we use the RSD and multiply it with a range of factors to model varying degrees of device variation.

Cycle variation [32, 77, 113], has not been presented with specific numbers based on an existing device. Typically it is just mentioned that cycle variation should be as low as possible. However, the degree of realistic device variation can be estimated from results presented in [2]. They presented a training algorithm that allowed the tuning of a memristive device state up to 1% accuracy. The achieved tuning accuracy did not derive from algorithmic limitations, but reflects cycle variation. The 1% accuracy reflects the variation in device response with respect to otherwise constant parameters. In [77] it is suggested to model cycle variation following a Gaussian distribution.

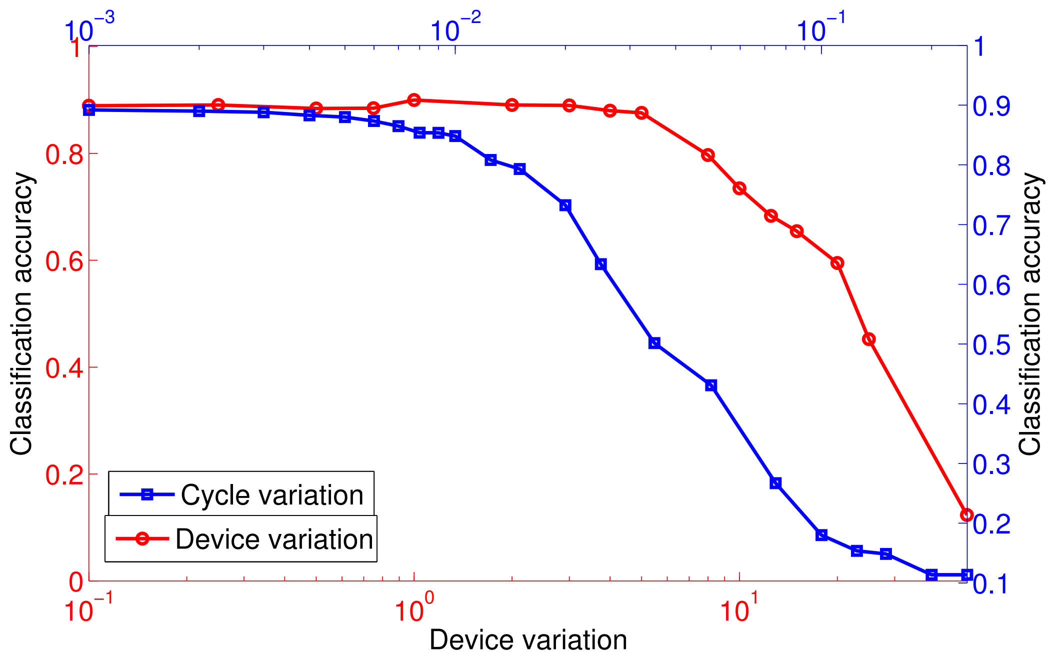


Figure 3.21: Classification performance as a function of device and cycle variation. Result based on physical data is shown at a value of one. Only when exceeding reported device variation by more than 6 times will the performance start degrading. Cycle variation causes small, gradual performance losses for variation less then 1.5% and more drastic losses for cycle variation larger then that.

3.6.2 Results

The results for modeling device variation are shown in Fig. 3.21. The data implies that the presented architecture is highly tolerant to device variation. The performance is stable for a wide range of variation and only starts slowly dipping after a variation factor of 6 (6 times higher than experimental data indicates). As was argued in [17], where memristive networks were used for simple pattern classification, device variation was beneficial as long as the devices offered non-linear transitions in their state. For the same reasons, our approach is device-variation-tolerant as we do not rely on precisely trained weights, but on the dynamic transitions of the synaptic efficacy. The drop in classification rate after 6 times higher variation than experimentally reported coincides with the narrowing dynamic range for which the memristive devices exhibit nonlinear state changes and state decay.

Cycle variation has been reported in [2] to be between 0% and 1%. To account for other devices that might exhibit a larger range, we have simulated cycle variation between 0% and 25%. The results in Fig. 3.21 indicate a gradual performance loss from 89.36% to 84.82% for devices with up to 1% cycle variation. Beyond 1%, performance starts to drop at a quicker rate with falling below 80% performance for more than 1.5% cycle variation. If physical devices have a normally distributed cycle variation with more values closer to the mean, performance loss will be slightly alleviated. Overall, the presented results on device and cycle variation make a strong case for the presented architecture. Whereas reported levels of device variation have no negative impact on performance, cycle variation based on experimental data causes only a small loss of performance.

3.6.3 Discussion

Device variation, as reported in [22], is not a problem as the presented approach relies on dynamic state changes and not on precise values across all devices. This allowed to show that device variation can be dealt with. It also provided the background for showing at what levels device variation would become a problem. Once variation is large enough, devices with a significantly lower resistive range will dominate the output vector results, hence not allowing the equal extraction of data.

Cycle variation, however, poses a problem as it introduces non-deterministic behavior. The important question is at what level cycle variation becomes a problem. This level will be closely related to the efficient use of the memristive crossbar and the separation property [74] that defines how easy data sets can be separated from one another.

Variation can also be found in synaptic functions of biological brains [82]. As an artifact of synaptic variation, Rigotti *et al.* have shown the importance of diversity in neural responses [88]. For reservoir computing, Maass has highlighted the benefits of heterogeneous reservoirs providing the theoretical context for increased computational power [72]. Based on these references, one can believe that memristive device variation also holds the potential to be beneficial to computation. However, as Maass put it, it only provides the theoretical context for being beneficial to computation. To harness variation its extent as well as its integration into the larger system has to be studied. Without variation increasing the feature representation or without the long-term memory being able to harness the diversity, variation will not be able to improve computational capabilities.

By answering research questions 3.1 and 3.2, I provided an understanding of the sensitivity of this approach to device parameters. As was mentioned in [59], device variation is a trade-off with memristive switching characteristics. An improved controllability of the state switching will decrease cycle variation of a single device, but will lead to increased variation across devices. In the bigger picture this trade-off will guide the selection of devices for hardware implementations of memristive short-term memory and hence, enable more robust hardware.

3.7 BENCHMARKING

In this section I will put the presented methods and results into a bigger context. First, I will compare the presented results on the MNIST data set to an ELM implementation which compares well with respect to the training complexity of the network. In the second part of this section I will introduce another data set and, based on the very same methods, demonstrate the general feasibility of utilizing dynamic crossbars for spatiotemporal pattern recognition.

3.7.1 Dynamic Crossbar vs. Extreme Learning Machines

So far, all introduced methods were evaluated based on the same architecture and same data set. To get a better feeling how the presented methods compare to other approaches, I will take an extreme learning machines setup as reference. For the MNIST data set there exist a myriad of published classification results. Here I will compare the presented approach to an ELM implementation as this is a good reference with respect to having similar network and training complexity. ELM is a feed-forward network with a fixed weight matrix between the input and hidden layers and training happening at the output layer only. While similar in complexity to the crossbar approach, it was also shown that ELM is comparable to state-of the art methods with respect to classification rates on the MNIST data set [75].

For this experiment I have trained a series of ELM with varying sizes of the hidden layer. Starting with N=100 hidden layer neurons, I increased N in steps of 100 up to 2,500. Increasing the number of hidden layer neurons directly reflects into increasing classification accuracy. The number of hidden layer neurons is equivalent to the number of inputs to the output layer and determines the amount of trainable weights. To create a comparison with the crossbar approach, I implemented varying sizes of the reference matrix. As the reference matrix is created based on features from the input data set, I varied the number of principal components to be used for the creation of the reference matrix. Increasing numbers of principal components increase the amount of features and hence the number of columns in the reference matrix. The crossbar was read out in chunks of 10 columns to avoid the concentration of measure limitation (see section 3.5.3).

The results in Figure 3.22 compare the size/classification rate relationship of the two

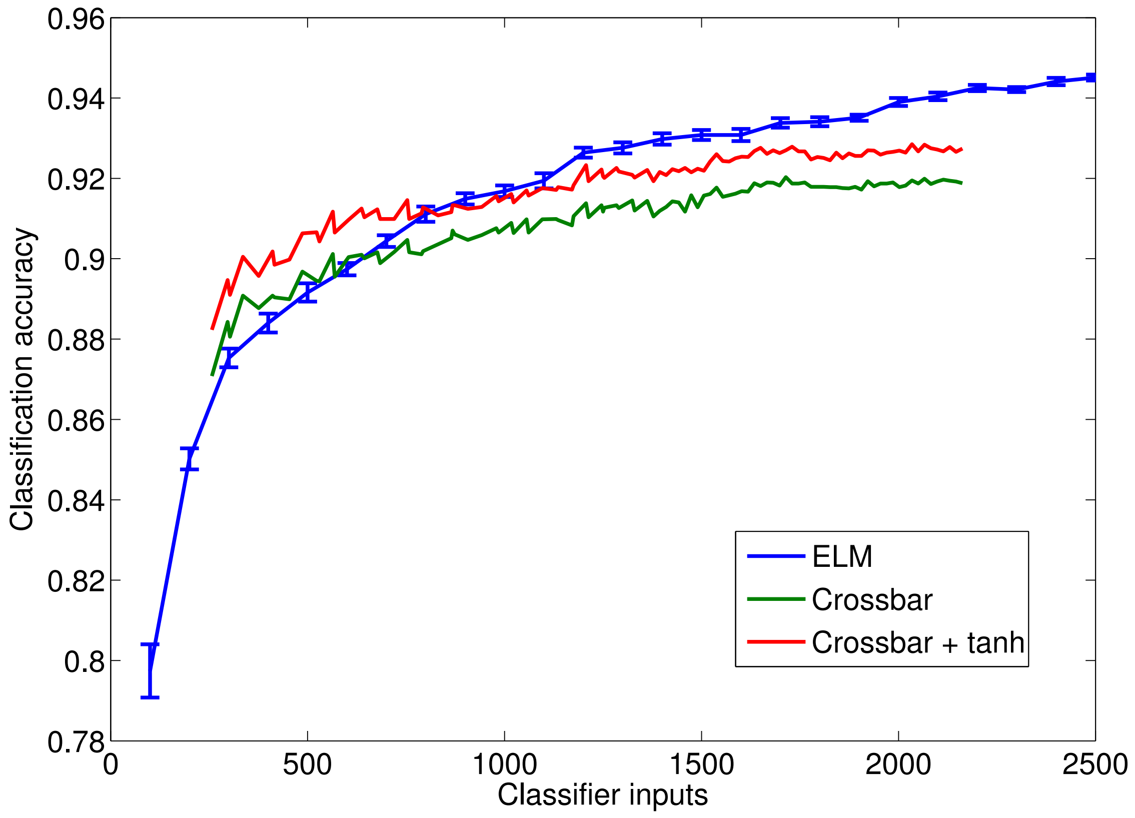


Figure 3.22: Classification rate as function of readout layer inputs. For the ELM approach the number of inputs to the readout layer is given by the number of hidden nodes. For the crossbar we increased the amount of principal components used to create the reference matrix. With increasing number of principal components the reference matrix grows, and hence the number of inputs to the classifier. The ELM results are plotted along the standard deviation due to random weight matrices. The crossbar approach does not show error bars as it is deterministic and does not contain random components.

approaches. For a smaller number of signals passed to the output layer, the crossbar achieves better performances. However, the ELM approach scales better with respect to further performance increases. The reason is to be found in the different dimensionality mappings happening for both approaches. For an ELM with few hidden nodes (i.e., $N = 100$) a down-projection from 784 dimensions to 100 dimensions is performed. Without a trained weight matrix between the input and hidden layer this strong dimensionality reduction does not capture enough information from the input data for the classifier to reach good results. Contrary, the crossbar approach, while also passing only a small number of crossbar outputs to the classifier, does integrate all 784 dimensions of the input data and the data that is passed to the classifier contains that information in a compressed form. While for a few neurons, the crossbar approach has advantages by integrating the full dimensionality of the input and compressing them to a smaller dimensionality, for larger N this compression becomes a disadvantage. The ELM scales better with N as each neuron can capture a different set of input combinations and add to a heterogeneous representation in a higher dimensionality. While outperforming the ELM for small N, the compression of memristive states into row and column vectors creates an underdetermined representation from which different crossbar state patterns cannot be precisely distinguished. However, in defense of the crossbar approach it is worth remembering that we are first integrating over time (one MNIST column at a time), while the ELM approach did not underlie such a constraint.

3.7.2 Jittered Spike Trains

In this section I will apply the presented methodology of dynamically computing within a crossbar to a spatiotemporal data set of jittered spike patterns. The benchmark used here has been introduced by Maass *et al.* [37, 73] as a general method to verify neural

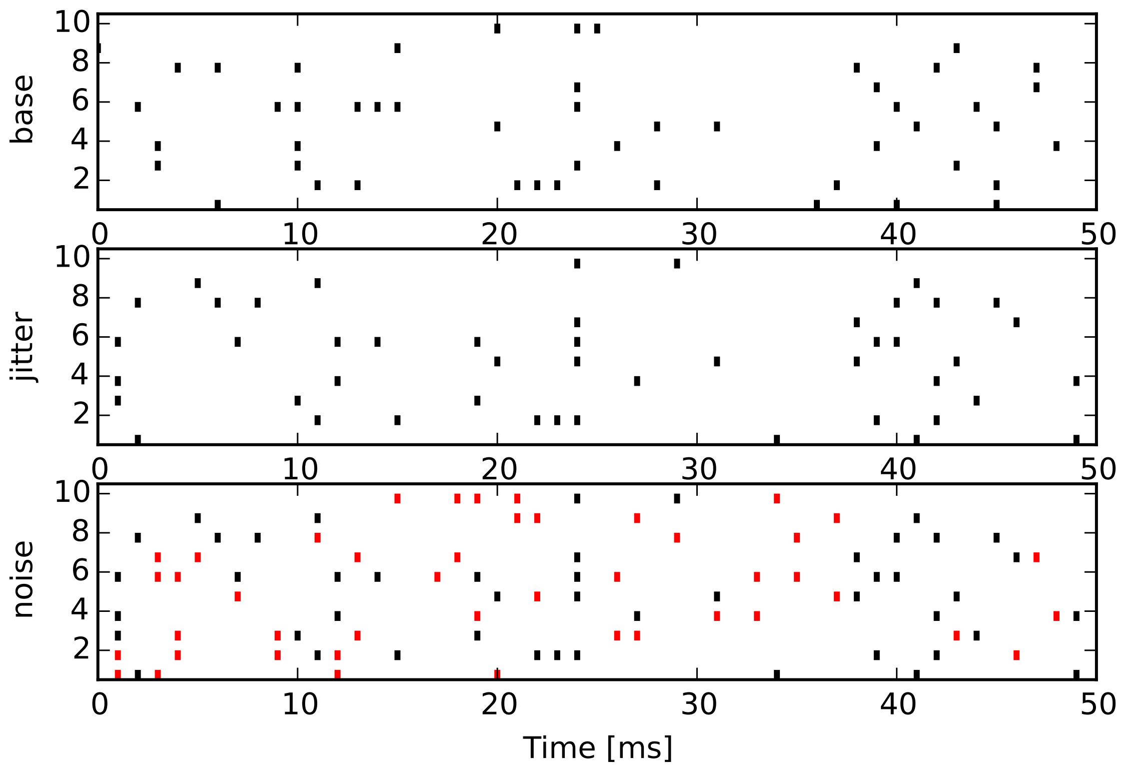


Figure 3.23: Example Poisson spike trains with 10 channels and a length of 50 time steps. (a) Base pattern for one class. (b) Jittered version of base pattern. For each class 2000 different jittered versions were created. (c) Jittered version with added noise in red. This example contains as much noise as signal spikes.

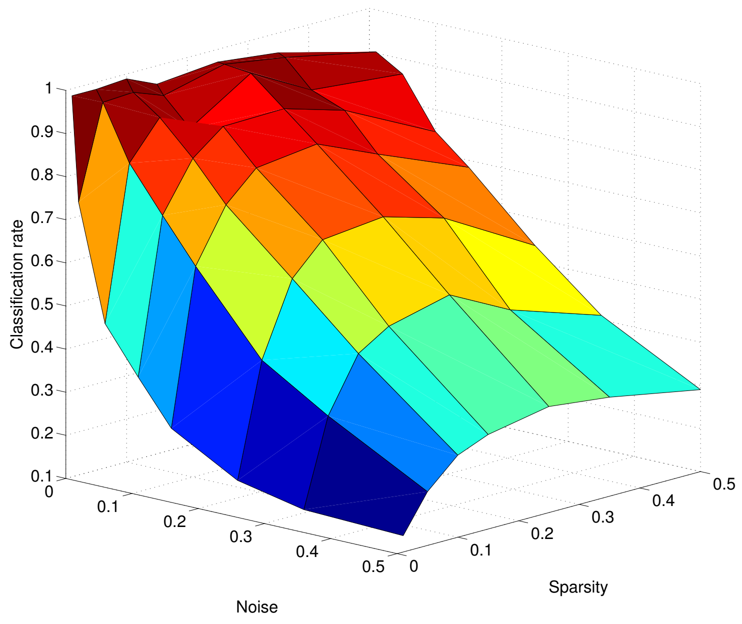
microcircuits which serves as the biological reference for the liquid state machine [74]. Neural firing patterns are commonly modeled by Poisson distributions, i.e., [45, 110]. Based on Poisson spike trains an artificial spatiotemporal data set was created as described by algorithm 1.

Algorithm 1 Jittered Spike Patterns

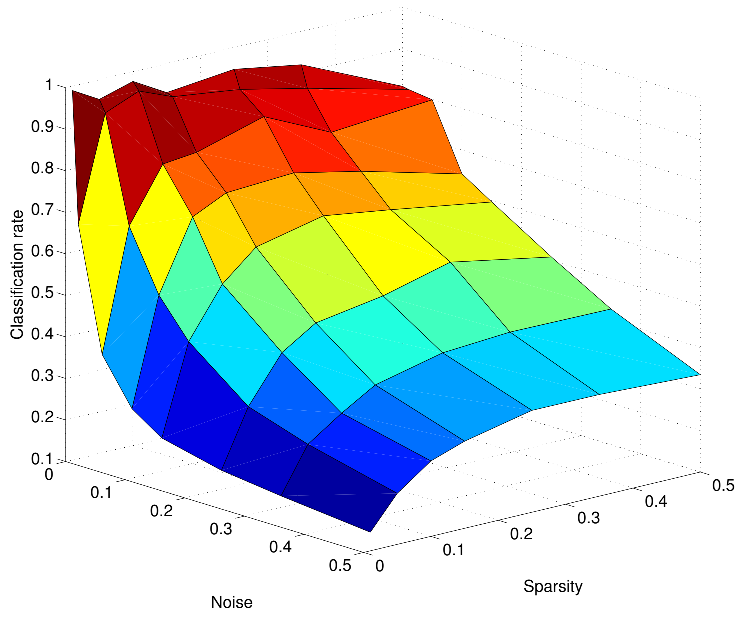
```
for $i$ = 1 to $n\_classes$ do
    create base pattern
    for $s$ = 1 to $n\_samples$ do
        create jittered sample
        add noise
    end for
end for
```

The created data set contained ten different classes with 2,000 samples each. Each sample had ten channels and the length of the sample was 50 time steps (which relates to 50ms in the simulation). The creation of the base patterns was controlled by a sparsity parameter that determines the spike rate per channel. The jitter is added by moving each spike with a distance drawn from a Gaussian distribution with zero-mean and standard deviation $\sigma = 4$. Noise is added to each sample by adding a randomly created sparse matrix of the same dimensions as the jittered sample. A noise factor determines the amount of noise created. Example spike patterns are shown in Figure 3.23.

For comparison, an existing echo state network implementation for classification of spatiotemporal signals was used. As part of the Oger framework [108], an echo state network reservoir consisting of leaky integrator neurons was provided for the classification of analog speech signals. Spatiotemporal speech signals are applied to a reservoir with N nodes and a ridge regression classifier is trained based on the states of the N reservoir nodes. The same setup is used for the classification of the jittered spike trains. To compare the crossbar and the ESN classification performances, I design the ESN size (number



(a) Crossbar



(b) ESN

Figure 3.24: Jittered spike pattern classification for different sparsity and noise levels: (a) dynamic crossbar (b) echo state network. The crossbar approach has a slower decay of classification rate as noise increases.

of reservoir nodes with fixed connectivity) as to match the number of inputs to the ridge regression classifier in both approaches.

The results of using a crossbar and an ESN for the classification of jittered spike patterns are shown in Figure 3.24. The presented data was obtained based on the methodology described above, with a $\sigma = 4$. This level of jitter was chosen as to achieve near perfect classification results for sparse patterns with no noise added. With that as a reference point, the impact of added noise and decreasing sparsity (more spikes) is evaluated. The first, and most encouraging outcome of the classification is the ability of the crossbar approach to achieve competitive classification results in comparison to established reservoir computing architectures. Both plots exhibit similar classification rates for the extreme cases (i.e., no noise / 50% noise). However, the intermediate parameter space is where the crossbar approach outperforms the ESN. This is shown in Figure 3.25. This plot shows the difference in classification rates, with positive values indicating the difference in classification rate in favor of the crossbar approach. While in the absence of noise (left part of the figure), the difference between the two approaches is minimal for most sparsity levels, with added noise the crossbar approach achieves significantly better classification rates. In the extreme case of 50% noise, both approaches start to fail and hence exhibit similar classification rates gain.

An explanation to the performance difference can be found in the utilization of the reference matrix. Both presented methods to create this matrix (PCA, class mean) act as a filter as they are looking for dominant features that separate classes. Noise added from a uniform distribution is not exhibiting any specific local concentration of spikes. In contrast, the distribution of data spikes, inherent in the definition of what constitutes data samples belonging to a certain class, will result in a set of features that are spatially and temporally more pronounced than uniformly distributed noise. The thresholding of these

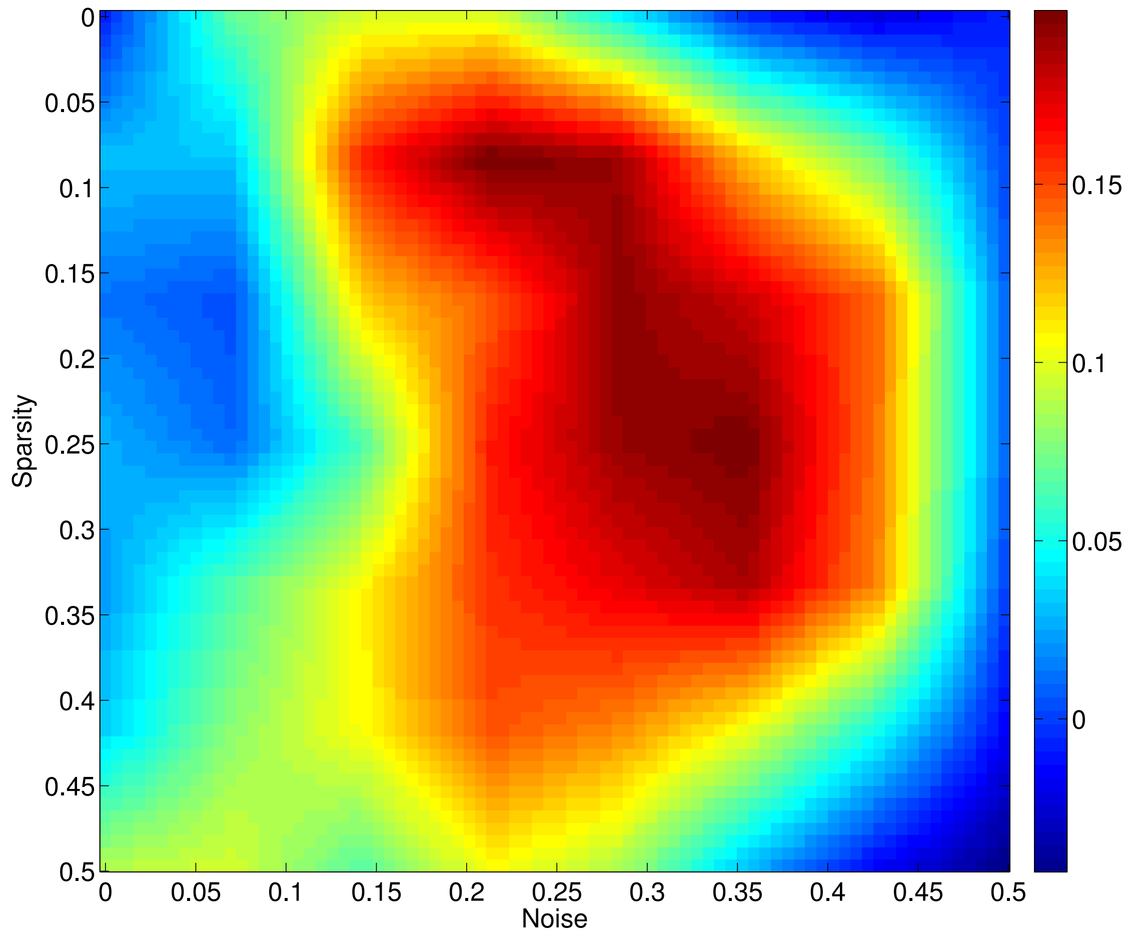


Figure 3.25: Difference map of classification rates of crossbar and ESN (warmer colors indicate better classification accuracy of the crossbar approach). In the absence of noise (near perfect classification) or with close to 50% of possible spikes being noise (low classification accuracy) both systems perform equal. For noise levels in between the extreme cases the crossbar approach performs better as the reference matrix allows targeted integration of data signals and to some extend prevents integration of noise signals.

features than assures that the reference matrix will contain mostly features present in the input data and prevent the crossbar from integrating noise (see sections 3.3 and 3.5.1 for details on when the crossbar integrates and how the reference matrix is created based on the input data).

Another aspect determining computational performance in such systems is memory capacity. In [43] the memory capacity of memristive devices was discussed and concluded that inherent state decay offers high-level biological behaviors. For traditional reservoir computing approaches a detailed analysis of memory capacity as a function of reservoir time constants and size was shown in [42]. Increasing the number of reservoir nodes can counteract increased noise levels. This could also be verified for the jittered spike patterns used here. Hence, the difference in classification rate between the crossbar and the ESN can partly be attributed to insufficient memory capacity in case noise is present.

3.8 DISCUSSION

In this chapter I have presented my research on utilizing an existing and well studied architecture, the memristive crossbar, for dynamic computation. The use of MNIST as the main data set throughout the chapter was mostly of historical nature. It was an easily available data set with a myriad of architectures to compare to. The initial experiments using different the MNIST data as inputs as well as as reference signals has shown the general ability to utilize a memristive devices for dynamic computation. The optimization techniques presented have introduced a variety of options to generalize the design of the reference matrix and to retrieve information from the crossbar. All these methods function as parameters to define a desired trade-off between the computational capabilities as well as the attached hardware costs. In addition to the presented methodology, the investigation on variation has demonstrated that the approach can is robust for the published results on device and cycle variation.

The benchmarking has then expanded the view and allowed to gain a better understanding of how this approach compares to other implementations. By utilizing temporal features of the memristive devices, the crossbar approach can pose a very compact implementation for spatiotemporal pattern recognition. This is indicated by outperforming other approaches for small system sizes. The fact that scaling up system sizes does not reflect in computational capacities, brings up a few considerations. Because of the compact system size the crossbar could serve as a complementary implementation and promote systems that harness heterogeneous methods to achieve a common objective function (i.e. classification).

Beyond any conclusions based on the MNIST data set, the jittered spike patterns have strengthened the case for computing in crossbars. The introduced methods to derive reference matrices have proven functional. More importantly though, were the results on the general noise immunity of this approach. Deriving the reference matrix from non-noisy training data and creating a reference matrix based on that, provides a spatiotemporal code that prevents the crossbar from integrating large portions of the noise. This relaxes the constraints for a subsequent classifier to distinguish between signals and noise.

Based on all presented results, I consider memristive crossbars as a viable platform to go beyond weight storage in neural networks, but to actually utilize the intrinsic device properties for computation. I believe while having had very limited access to spatiotemporal data sets, that the next step is to examine this approach on a set of diverse applications. Especially interesting would be an extension that does not require a fixed temporal frame size (number of time steps before reading out the crossbar). Larger reference matrices with repeated, but temporally shifted features and a scheme to continuously transfer crossbar patterns into output nodes, should be the two design aspects to achieve this.

4

COMPOSITION AND MORPHOLOGIES OF RANDOM MEMRISTIVE NETWORKS

In this chapter I will outline my research on random memristive networks. The ability to randomly grow nanowires, and in turn assemble memristive networks, has been demonstrated by [4,96,98]. With respect to neuromorphic engineering, these random assemblies are claimed to present structures with highly complex connectivity and network dynamics similar to neuron assemblies in biological brains [96].

We investigated random memristive networks for the first time in [17, 62]. In these publications the memristive network's ability to perform simple pattern classification was shown. While these publications have presented a case for the possibility of harnessing intrinsic computation, they also made limitations apparent. In section 2.3 I have discussed the basic processing capabilities of memristive networks and concluded that due to the passive nature of these electrical networks, they suffer from limited computational scalability and strong correlation across the nodes as was described in section 2.3.

The publications on physical implementations of random memristive networks focused on the manufacturing of individual networks and contained experiments that indicated fundamental computational capabilities relevant for reservoir computing [96]. The details given on the manufacturing process also implied the ability to fabricate such networks in a modular way.

In this chapter, with reference to the physical assemblies and to the demonstrated basic computational capabilities of random memristive networks, I will present a comprehensive analysis on how to further increase the computational capabilities of this novel approach. Based on a simple creation of random structures, I will demonstrate a hierarchical approach that treats individual memristive networks as reservoir nodes in an underlying reservoir computing architecture [16]. The modularization of memristive networks allows the creation of independent networks with different random connectivity. This in turn provides a way to overcome scalability limitations and strong signal correlation in favour of computational capabilities. In a second step, I investigate the role of the network morphology. Instead of the simple network creation that does not follow specific hardware constraints, I will present a network modeling approach that allows a detailed analysis of network morphologies and their influence on the computational capabilities of random memristive networks [15]. A third aspect to this research is the switching characteristics of memristive devices. I will highlight the important characteristics necessary to harness memristive devices for reservoir computing. This work was partly done in collaboration with Alireza Goudarzi, PhD student at UNM, and was published in [15, 16].

4.1 RESEARCH QUESTIONS

Random memristive networks are another type of memristive networks that have been published [4, 96, 98]. The networks were shown to exhibit nonlinear dynamics and memory, which are fundamental characteristics of reservoir computing. With the possibility to randomly assemble networks of memristive devices, the following questions will address issues around scalability, network topologies and switching dynamics.

Research Topic 4. Physical and Computational Scalability

Given random memristive networks [4], the analysis of similar networks [17, 62] has shown limitations in their computational capacities due to strong internal signal correlation. These limitations are related to the dynamics in purely passive resistive networks. Therefore scalability is not simply achieved by increasing the amount of devices used within a random network (in contrast to artificial neural networks, where increasing the amount of neurons within a layer often improves computational performance).

> **Question 4.1.** What are physical possibilities to create less correlated network states that improve computational capacities and allow for scalability?

Research Contributions

While so far all published results of random memristive networks have emphasized their overall computational potential based on device and emerging network dynamics, application to a complex task has yet to be shown. Demonstrating an architecture that, by increasing its physical resources, scales in computational performance is fundamental to the mastering of more complex tasks.

Research Topic 5. Memristive Network Topology

Network topology is defined by the number and physical location of network nodes and their connections established by memristive devices. With all memristive networks being created randomly [16], we can assume that from a set of different networks not all networks will be computationally useful to the same degree. First investigations of utilizing exact copies of the same memristive network in a hierarchical structure have shown a link between reservoir performance (measured as NRMSE on the NARMA-10 task) and network topology. However, it is not yet clear what constitutes a computationally useful network topology.

> **Question 5.1.** Network topology is defined by the device density, the locality of connections, and the placement of the interface nodes. How do these parameters have to be designed to achieve a computationally useful memristive network topology?

Research Contributions

Evidence of what defines good random networks will guide the fabrication of these networks. Having some control over the network properties of random memristive networks [96], fabrication processes can be improved to create networks that are more likely to exhibit target properties. This will utilize the available chip area more efficiently, produce more deterministic results, and improve average performance by increasing the number of computationally useful memristive networks.

Research Topic 6. Memristive Switching Characteristics

In [16] it is shown that by adjusting system parameters that directly affect the switching dynamics within the memristive networks, it is possible to optimize the system's output with respect to match a target output. Tasks that required the generation of higher frequency components performed best under frequent state switching. Memory-intensive tasks performed better with slower dynamics in the memristive networks.

> **Question 6.1.** How do the task dependent switching characteristics of memristive networks relate to the intrinsic device properties and what device types should be chosen for a given task?

Research Contributions

The research will provide an understanding of the correlation between intrinsic device properties, switching characteristics of the memristive networks, and the reservoir's utility for varying tasks. These results will serve as a design guide for selecting memristive devices, defining aspects of the memristive networks itself, as well as for the larger reservoir system with input scaling and weight matrix design. Such generalized relations will open the system design to a broader range of researchers without in-depth knowledge of the memristive networks switching dynamics.

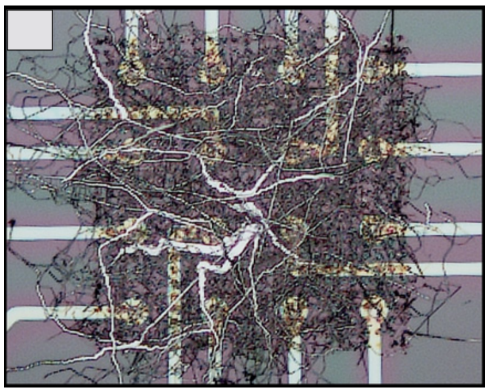
4.2 HIERARCHICAL COMPOSITION OF MEMRISTIVE NETWORKS

4.2.1 Architecture

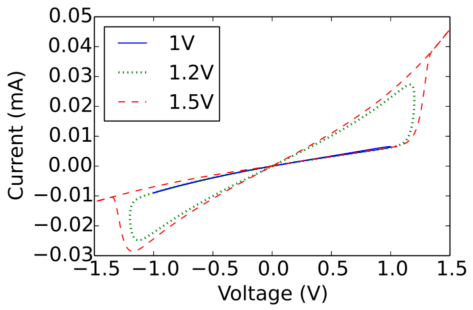
The random assembly of memristive devices was physically realized and presented in [4,96,98]. Figure 4.1a shows an image of such an assembly where the nanowires forming memristive devices evolve around underlying wires that interface the random assembly to CMOS circuitry providing signal input and output. The physical extent (network size) and density (number of devices) is controlled by the underlying wires. This is the basis for the presented architecture.

I will first discuss the memristive devices, which are the basic elements. Section 2.3 discussed the memristive state transitions to be the key for nonlinear mapping. Equations 4.1 and 4.2 describe the nonlinear memristive state change.

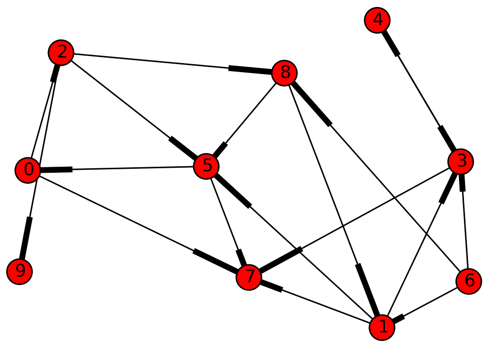
$$\frac{dw}{dt} = (1 - e^{k})\lambda(\eta_1 sinh(\eta_2 V) - \frac{w}{\tau}), \tag{4.1}$$



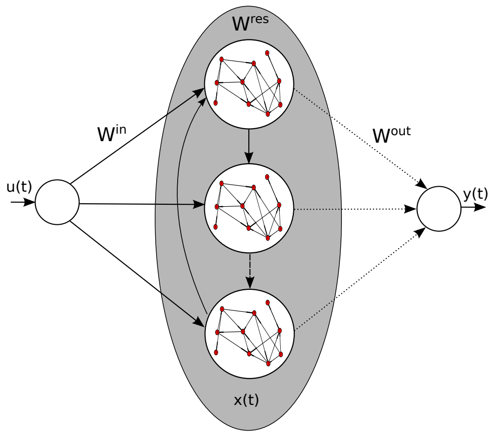
(a) Random assembly of memristive devices (reprinted from [96]).



(b) Memristive switching



(c) Random network topology



(d) Memristive SCR

Figure 4.1: Hierarchical reservoir of memristive networks. (a) physical assembly of nanowires forming memristive devices. The network size and density is guided by the underlying larger wires and pads. (b) memristive switching characteristics as used in this work. (c) example graph of a random memristive network. The red circles are network nodes and the links represent memristive devices. (d) simple cycle reservoir structure with memristive networks places inside of reservoir nodes.

with k:

$$k = \begin{cases} \frac{-1+w}{0.5} & \text{if } V > 0 \\ \frac{-w}{0.6} & \text{if } V < 0 \end{cases} \tag{4.2}$$

The state change dw is a function of the device state $w(t)$ and the applied voltage $V(t)$. α, β, γ, δ, λ, and η are constants that were determined to match experimental data. τ describes the speed of the state decay of the device. Figure 4.1b shows the device current plotted against sine waves of different amplitudes. It highlights the differences in device switching based on the applied input signal. Given that the range of device voltages (the voltage experienced by a single device) is diverse throughout a network, meaning that devices experience different voltages, this also implies a variety of switching dynamics that can be harnessed for nonlinear computations.

In Figure 4.1c, an example graph of a random assembly is shown where circles represent network nodes (voltage nodes for input and output) and the links represent the memristive devices. Given that such networks can be assembled around a set of wires, this provides the possibility of treating them as individual units. These units are physically and electrically isolated from each other and hence, do not experience the strong signal correlation like single memristive networks, as described in section 2.3. Therefore scalability is achieved by creating multiple of such networks, physically separated from each other. Electrical separation is achieved by CMOS circuitry that only probes the network activity, without influencing it. This leads to the proposed architecture shown in Fig. 4.1d. Here I present a classical neuron-based reservoir approach with a few modifications. First, instead of a traditional activation function, the memristive networks perform the computation and replace an activation function. The second modification is motivated by an investigation on the required network complexity of an ESN [89]. It was shown that instead of random connectivity in the reservoir, a *simple cycle reservoir* (SCR) can achieve similar performance (correlation between a target and the computed output) on a variety of tasks. I relate to that work as the simplicity in the connectivity of reservoir nodes lends itself well to hardware implementations. Reducing the wiring costs by relying on a deterministic and minimal connectivity of in-degree $K = 1$ (number of connections per node) allows simple wire routing.

4.2.2 Methodology

In accordance to the *Simple Cycle Reservoir* (SCR) shown in [89], the input signal is applied to all reservoir nodes with a weight v and a sign following a Bernoulli distribution. The reservoir nodes are connected in a ring structure with uniform weights w equal to the spectral radius of the reservoir weight matrix W^{res}. Each reservoir node produces one output state that is the voltage difference between two memristive network nodes and forwarded to the readout layer for training. Measuring the output signal as a differential voltage follows the established knowledge of device voltages producing less correlated results as compared with node voltages (chapter 2.3).

This architecture is numerically simulated in software, but is based on the physical realization of random assembly as shown in [96]. Therefore, I validate this architecture on a first set of three examples of *Higher harmonic generation* (HHG) that were shown for the physical assembly. HHG is a nonlinear process in which a dynamical system is excited by a signal with frequency f and in turn generates signals with other frequencies not present in the input. The first example is a sine wave generation. For this task, the reservoir is driven with a sine wave at frequency f and the output is trained to produce a sine wave at frequency $2f$. The second example is a triangle wave generation. For this task, the reservoir is driven with a sine wave and the output is trained to produce a triangle wave given by:

$$x(t) = \frac{8}{\pi^2}\sum_{k=1}^{\infty}(-1)^k \frac{\sin(2\pi(2k+1)ft)}{(2k+1)^2} \tag{4.3}$$

The third example is a square wave generation where the reservoir is driven with a sine wave and the output is trained to produce a square wave given by:

$$x(t) = \frac{4}{\pi}\sum_{k=1}^{\infty} \frac{\sin(2\pi(2k-1)ft)}{(2k-1)} \tag{4.4}$$

For these tasks, I compare the results from [96] with a single memristive network as well as a SCR using multiple memristive networks. A visualization of the tasks is given in the following results section.

After verifying the ability to solve the HHG task that requires nonlinear transformations but no memory, I will evaluate this architecture based on the NARMA-10 problem. *Nonlinear autoregressive moving average* 10 (NARMA 10) is a discrete-time temporal task with 10^{th}-order time lag. To simplify the notation we use y_t to denote $y(t)$. The NARMA 10 time series is given by:

$$y_t = \alpha y_{t-1} + \beta y_{t-1}\sum_{i=1}^{n} y_{t-1} + \gamma u_{t-n}u_{t-1} + \delta, \tag{4.5}$$

where $n = 10$, $\alpha = 0.3$, $\beta = 0.05$, $\gamma = 1.5$, $\delta = 0.1$. The input u_t is drawn from a uniform distribution in the interval $[0, 0.5]$. This task presents a challenging problem to any computational system because of its nonlinearity and dependence on long time lags. Calculating the task is trivial if one has access to a device capable of algorithmic programming and perfect memory of both the input and the outputs of up to 10 previous time steps. This task is often used to evaluate the memory capacity and computational power of ESN and other recurrent neural networks.

Architecture	Task	MSE (stdev)	v
Single network	20Hz sine	0.0617(0.0191)	2
	triangle	0.0015(2.45e^{-4})	2
	square	0.0521(0.0029)	12.5
SCR	20Hz sine	0.0111(0.0064)	2
	triangle	8.14e^{-4}(1.62e^{-4})	2
	square	0.0307(0.0037)	12.5
[96]	20Hz sine	0.02	2
	triangle	0.04	10
	square	0.125	7

Table 4.1: The best observed MSE for HHG tasks and corresponding parameters for systems passing 16 values to the readout layer.

4.2.3 Results

We first compared single memristive networks with the memristive SCR architecture. Both setups are compared based on the number of signals extracted from them and forwarded to the readout layer. Table 4.1 shows the best results from setups comparable to [96]. The single memristive network had 16 output nodes and contained 120 memristive devices. The SCR was made up of 16 nodes with each node providing one signal and utilizing memristive networks of approximately 50 devices. Figure 4.5a shows a more detailed representation of the waveform generation performance of a single memristive network as a function of increasing input biases. Similar to [96], we observe the best generation of the sine wave for $v = 2V$ and the square signal for larger voltages. Besides some similarities we can also observe clear discrepancies, such as the absolute MSE values and the minimal error for generating the triangle wave. Without having absolute certainty, we suspect differences in the memristive devices, constraints on the network topology, and the application/reading of input/output signals to be likely causes of these differences.

Figure 4.5b compares the combined MSE (sine, triangle, square) of single memristive networks and of the memristive SCR. The x-axis shows the number of signals read from

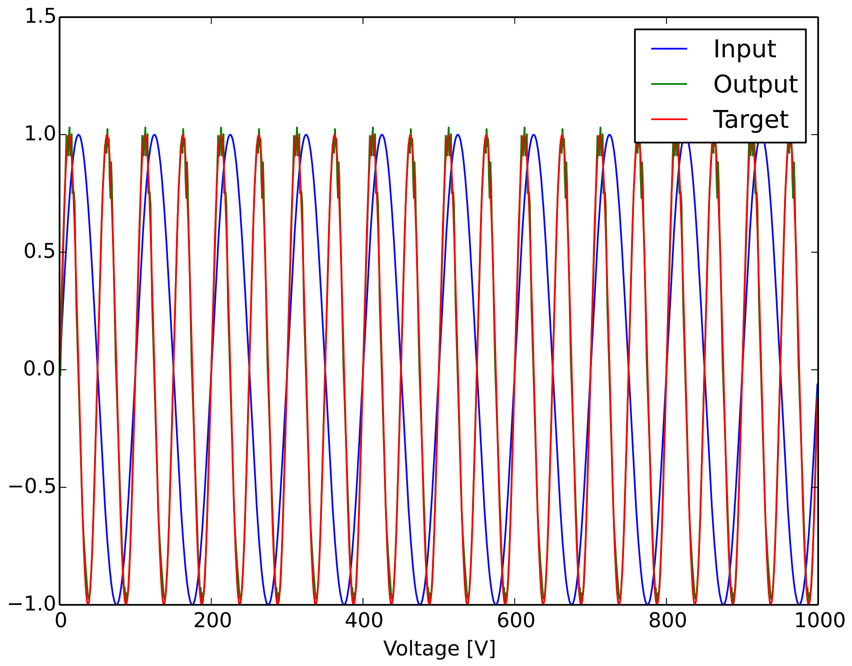


Figure 4.2: Higher harmonics generation of a sine wave with double the frequency of the input signal.

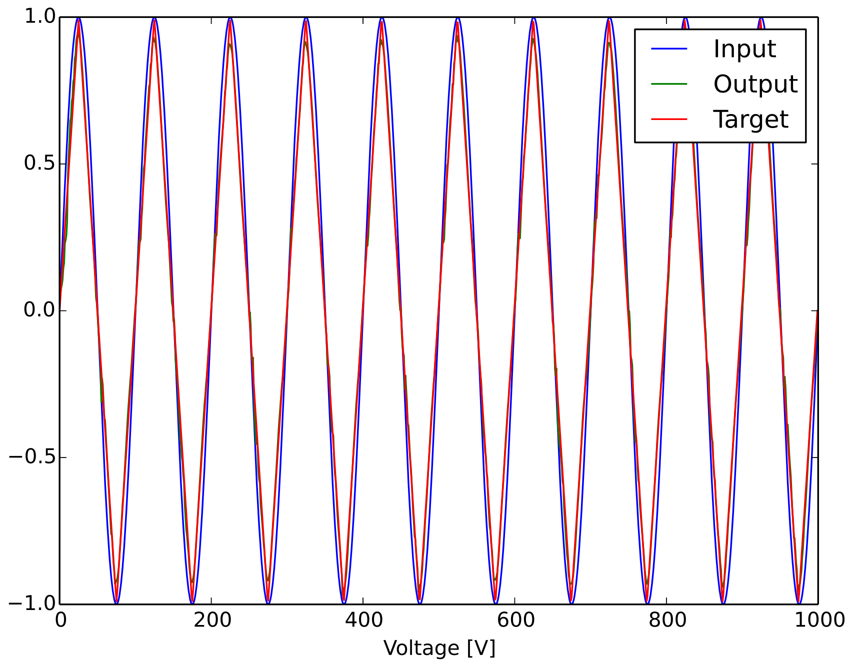


Figure 4.3: Higher harmonics generation of a triangle wave with perfect match between output and target signal.

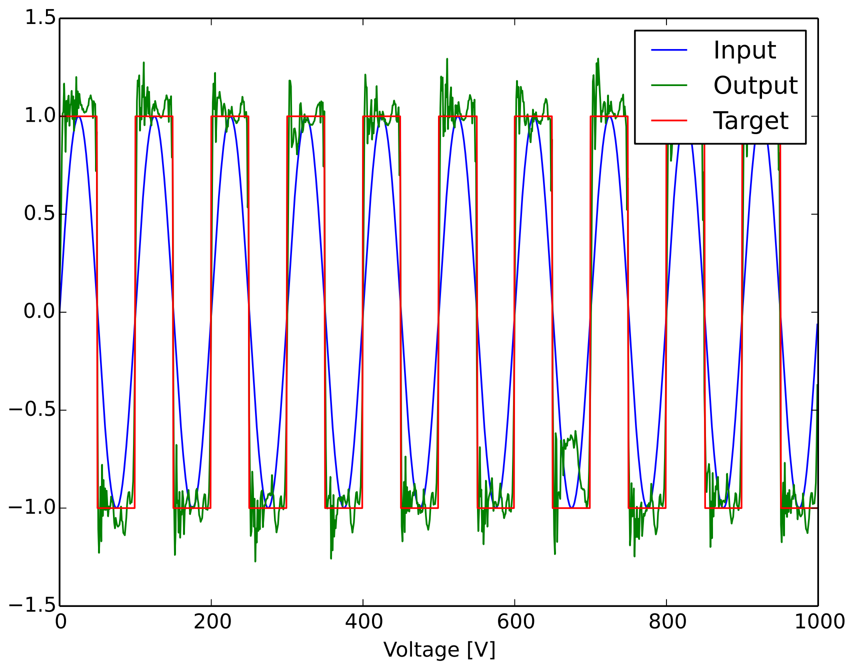
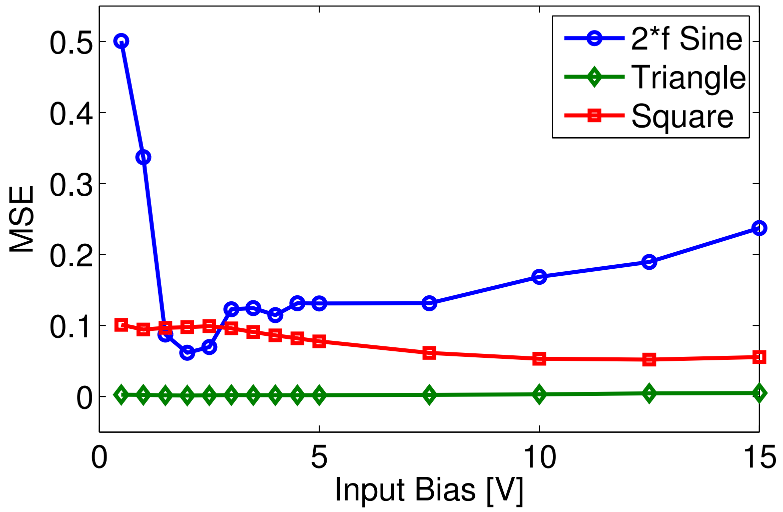


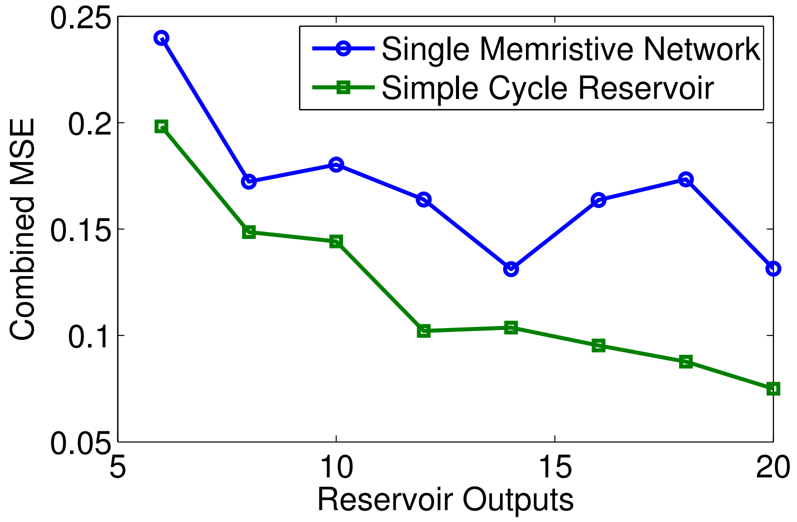
Figure 4.4: Higher harmonics generation of a square wave highlighting the higher frequency components as over- and undershoots.

the corresponding architecture. For the SCR this is the same as the number of nodes. We can observe that the signals read from the SCR allow a better generation of the target signals. Due to the physical separation of the SCR nodes, the signals are less correlated and provide more features compared with signals all read from the same memristive network.

The NARMA-10 task, due to the need of memory, poses a difficulty that single memristive networks, as presented here, are not capable of dealing with. Experiments to verify this were done for different single memristive network sizes (75 to 350 devices) with best resulting NRMSE values of around 10 indicating this difficulty. The reasons are found in the dynamics of passive electrical networks. With increasing network sizes each the relative device voltages will be reduced and state correlations increase. A possible solution to increase the performance of single networks is by applying input signals to multiple distant network nodes to increase individual device voltages. However, the additional



(a) MSE as function of input bias



(b) MSE as function of readout signals

Figure 4.5: (a) Wave generation MSE of single memristive network with 16 readout nodes. The curves for the individual signals are in shape similar to the ones published in [96], even though the scale is different. (b) Comparison of cumulative MSE of single memristive network with single cycle reservoir of memristive networks as a function of readout signals. The SCR, due to the physical separation of the memristive networks, can produce less dependent signals and achieve better performance.

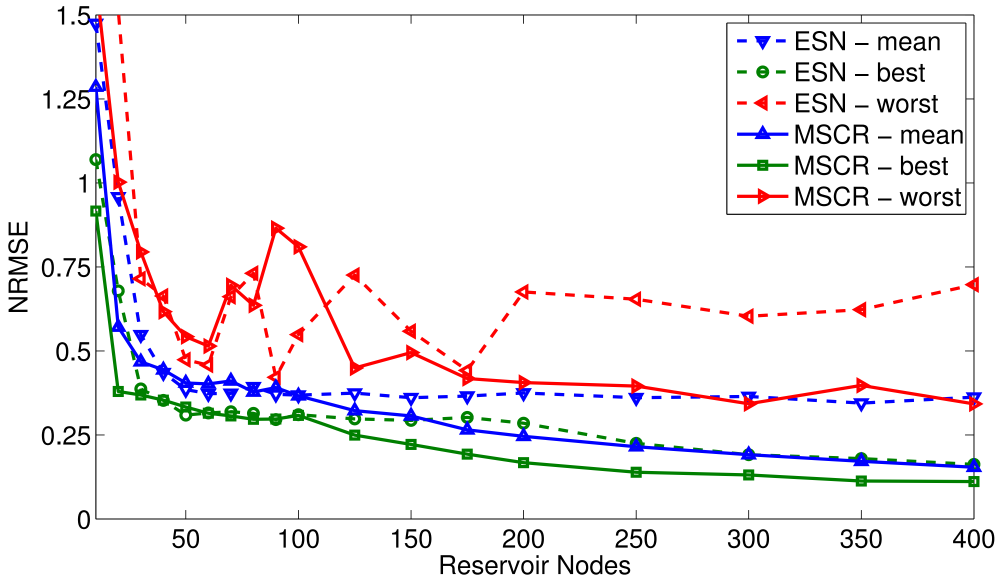


Figure 4.6: NARMA performance for increasing SCR sizes. The dashed lines are obtained with regular sigmoidal neurons (ESN) and $\nu = 0.01$ and $\lambda = 0.75$. The memristive SCR (MSCR) data was obtained based on $\nu = 0.5$. $\lambda = 1.7$, and 52 memristive devices in average per node.

resources on the CMOS level would likely not provide any area benefits over a modular approach.

Figure 4.6 compares the results for our memristive SCR and a regular sigmoidal neuron SCR. The low input signal range implies that the nodes of both implementations behave mostly linearly. However, due to a spectral radius greater than one and the resulting signal amplification, memristive networks experience some low frequency internal device state changes (at a lower rate than the input signal), which adds some nonlinear processing to the reservoir. As a result of this, with a growing number of nodes, the memristive SCR continues to improve while the sigmoidal SCR plateaus at around 100 nodes. We attribute this continuous performance improvement for increasing reservoir sizes to the heterogeneity of the input-output mappings (activation functions) of the memristive networks, the low frequency memristive state changes, and the resulting diverse signals used by the readout layer.

4.2.4 Discussion

The presented results give answer to research question 4.1 on a scalable random memristive architecture. By utilizing a set of small scale random memristive networks that are connected by an underlying CMOS area one can achieve more heterogeneous signals and harness the different topologies of the individual networks. The presented results have shown that the hierarchical approach allows for further scaling of the computational capabilities of random memristive networks. The results for the higher harmonics task correlate with the results published in [96] except for the triangular signal. This might be an outcome of the different memristor model used, some network dynamics that cannot be described by modeling individual devices only, and/or a difference in network topologies. The results for the NARMA-10 task indicate that random memristive networks provide both, computation and memory. Memory is implemented on the individual memristive network level as well as by the simple-cycle-reservoir structure in the underlying CMOS layer. This two-level implementation could explain the improved performance in comparison to the ESN SCR implementation.

4.3 MEMRISTIVE NETWORK MORPHOLOGIES

In this section I will present a modeling and simulation framework that enables a detailed analysis of resistive switch network morphologies as determined by nanowire lengths, distributions, and density. The computational capabilities of different network morphologies are analyzed with respect to the compressibility of the measured network signals. I put the computational capabilities into perspective by comparing to corresponding energy-consumption data. This comparison outlines a trade-off between computation and energy consumption. Based on the used modeling parameters, future fabrication of random memristive networks can be guided to achieve a desired trade-off between computational capacity and energy consumption.

4.3.1 Network Modeling

While the application of memristive networks to reservoir computing was simulation based [16, 17, 62], physical assemblies of atomic switch networks (atomic switches are a sub-category of memristive devices) were reported in [4, 27, 96, 98, 99]. A simple modeling approach with focus on localized conductance changes of such networks was shown in the supplementary documentation of [96].

Here we expand the modeling of such networks with the aim to investigate the relation between network morphologies, computational capabilities, and energy consumption. Network morphology is defined by the average length of nanowires. Their density can be controlled by the underlying copper seed posts (Fig. 4.1a. In [96] the effects of copper seed posts' shape, size, and pitch on the network's nanowire length and density were described. Smaller seed posts lead to long wires while larger seed posts lead to more fractal local structures. The pitch of the seed posts is a control parameter for the network density.

The connection of the random assembly of nanowires with the size of the seed posts implies that the distributions of the nanowire lengths can be described in terms of some probability density function. Large seed posts would cause more localized connections while small seed posts would result in long-range connections. Hence, we use a *probability density function* (PDF) that can capture different distributions. We use a beta-distribution for our purposes. A beta-distribution B produces values in the interval of $[0, 1]$ and is controlled by the parameters α and β that define the mean value and the skewness around the mean.

$$\begin{aligned} P(x,\alpha,\beta) &= \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} \quad \text{(PDF)} \\ \mu &= \frac{\alpha}{\alpha+\beta} \qquad \text{(Mean value)} \\ S &= \frac{2(\beta-\alpha)\sqrt{\alpha+\beta+1}}{(\alpha+\beta+2)\sqrt{\alpha\beta}} \quad \text{(Skewness)} \end{aligned} \tag{4.6}$$

$B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx$ is a normalization factor ensuring that $P(x,\alpha,\beta)$ is a probability measure.

We use this PDF to model the distribution of the relative nanowire lengths within a random network. To translate the PDF into nanowire lengths, we create a map with normalized Euclidean distances ($[0, 1]$) from every seed post within the network. Starting from an initial node, we add a nanowire to the network by drawing a value from $P(x,\alpha,\beta)$

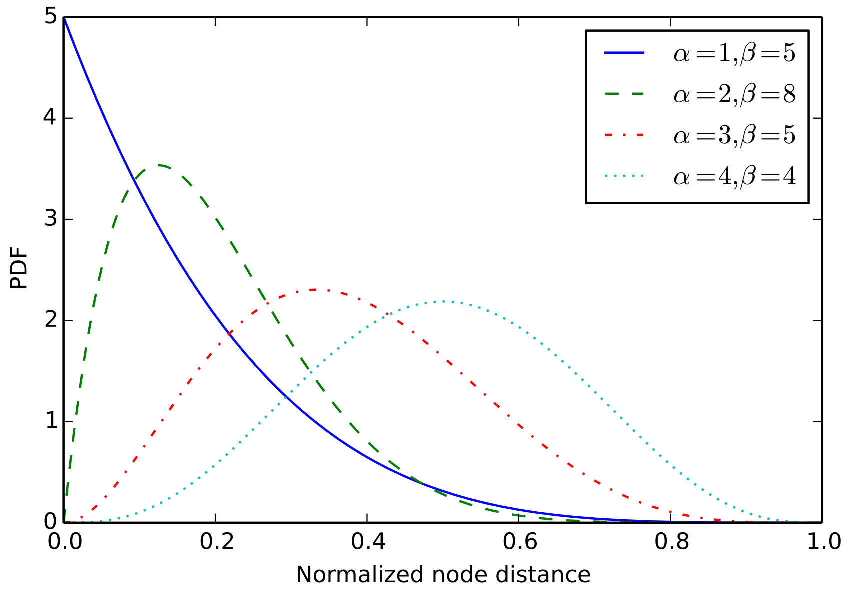


Figure 4.7: Different beta distributions and the resulting relative nanowire lengths. The x-axis represents the normalized nanowire length and the y-axis the probability of creating a nanowire of that length.

that determines the nanowire length. Beta-distributions with a focus on local connections ($\alpha < \beta$) will produce more fractal structures as nanowires branch out around the selected starting node for the wire. Distributions favoring long-range connections ($\alpha > \beta$) will connect distant parts of the underlying grid of seed posts. In Fig. 4.7 we show some examples of $P(x, \alpha, \beta)$ as a function of the control parameters α and β.

For the modeling of the underlying seed post grid we define two post types. Interface seed posts allow the application and reading of network voltages. These posts are established at a more coarse-grained pitch. The second grid is a supporting grid and more fine-grained than the interface grid. This supporting grid improves the formation of fractal structures for localized wire growth [27] and the establishment of more complex structures between the interface nodes; without a supporting grid establishment of multiple devices between two interface nodes will not be favourable, which limits the morpholog-

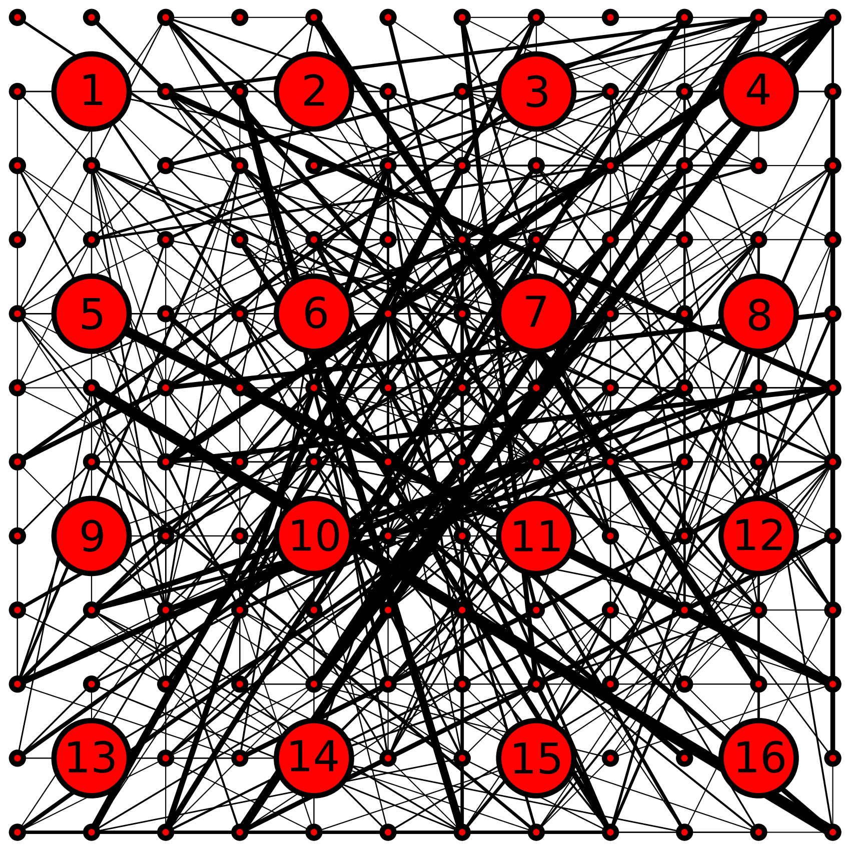


Figure 4.8: Example graph of modelling a random 16_2 memristive/atomic switch network with $\alpha = 2, \beta = 5$, and $\xi = 4$. The large circles represent the interface nodes to the underlying CMOS layer. The small circles create a narrower supporting grid that guides the density and morphology of the network structure. The graph edges represent memristive devices and the edge width is an indicator for the connectivity range, with increasing edge widths representing longer-range connections between nodes.

Table 4.2: Examples of network parameters

ID	α	β	ξ	**Connection length**	**Density**
N1	1	1	2	uniform	sparse
N2	1	10	8	short	dense
N3	10	1	4	long	semi-sparse
N4	10	10	6	medium (small σ)	semi-sparse
N5	5	5	2	medium (medium σ)	sparse

ical diversity (the supporting grid does not exist physically, but only in the simulation). The density of a network is defined by the indegree ξ of a node. The parameter ξ defines the average number of devices connected to each grid node. The number of resulting devices, and hence the density, is then given by the total number nodes in the supporting grid multiplied by ξ. The fabrication of networks with different densities was demonstrated in [96]. In Fig. 4.8 we show an example network model with 16 interface nodes, a supporting grid at a third of the pitch of the interface grid, and a mix of short and long-range connections. For all simulations we have used 16 interface nodes arranged in a 4×4 grid as shown in Fig. 4.8. If not mentioned otherwise we have used a supporting grid at half the pitch of the interface grid. Throughout this chapter I will refer to this network as of size 16_1, meaning that it has 16 interface nodes and one node between two interface nodes (half the pitch).

For a more intuitive understanding of how the defined network parameters inter-relate, we give five examples and describe the resulting network morphology (Table 4.2). In Fig. 4.9 and 4.10, we added the network IDs to the presented results to illustrate the relation between network morphology and information processing capacities.

We simulate the memristive networks by treating them as temporarily stationary resistive networks that can be solved efficiently using the *modified nodal analysis* (MNA) algorithm [68]. After calculating one time step using the MNA, we update the memristive devices based on the node voltages present in the network to account for the dynamic state changes of memristors.

4.3.2 Computational Capacities

As was outlined by Demis *et al.* [27], the structural similarities of *atomic switch networks* (ASN) and biological brains (i.e., fractal branching similar to dendritic trees) suggests that complex random assemblies provide hardware platforms for efficient brain-inspired computing. A characteristic feature of cognitive architectures is the nonlinear transformation of an input signal into a high-dimensional representation more suitable for information processing [14, 85]. In particular, for $N > M$, the input signal $\mathbf{u} \in \mathcal{R}^M$ is transformed to $\mathbf{x} \in \mathcal{R}^N$ by the dynamics of the cortical microcircuits driven by sensory inputs. In the case of random memristive networks, this means that the measured signals at the interface nodes are ideally nonlinearly related to the input signal and then provide a platform for brain-inspired computing.

A suitable transformation of the input requires rich dynamics that can preserve relevant distinctions between different input signals in the high-dimensional space [74]. This property is attributed to the dynamics of a system in the critical regime where the distinctions between states do not diverge or converge [9, 63, 97]. To provide such a rich dynamics, the activity of the nodes should show the least amount of redundancy. In other words, the dynamics of the nodes should be as uncorrelated as possible. The question is how one could measure that and how this measure would be related to the parameters of the system.

Here we introduce a simple measure that can describe the dynamics of random network signals in a way that is meaningful in the context of information processing. We study the compressibility of the network dynamics as a proxy for its richness. Specifically, we use *principal component analysis* (PCA) to transform the system dynamics into a principal component space [12].

The distribution of variations in the principal component space indicates the amount of redundancy between the different dimensions of the original system. The variation in each principal component is given by the corresponding normalized eigenvalue of the covariance matrix of the system. To calculate these, we record the network dynamics on all interface nodes as a result of an applied network input. All network signals are expressed in the network state matrix X. The covariance matrix of the network dynamics is then given by $C = X^T X$, and the eigenvalues are obtained by diagonalizing, $C = U \Lambda U^{-1}$. The diagonal elements of Λ, $\{\Lambda_1, \Lambda_2, \ldots, \Lambda_N\}$, are the eigenvalues of the corresponding dimension i and can be normalized as $\lambda_i = \frac{\Lambda_i}{\sum_{i=1}^{N} \Lambda_i}$. Since λ_i are normalized as a probability measure, we can describe their evenness using a single number $\mathcal{H} = -\sum_{i=1}^{N} \lambda_i \log_2(\lambda_i)$. This is an entropy measure and describes how evenly the λ_i are distributed. In one extreme case, where the system nodes are all maximally correlated, only one Eigenvalue will be one and the rest will be zero, and the resulting entropy will be $\mathcal{H} = 0$. In the other extreme case, where the nodes are maximally uncorrelated, the Eigenvalues will be identical and equal to $\frac{1}{N}$, and the resulting entropy will be $\mathcal{H} = \log_2 N$. In the latter case, every node in the network represents something unique about the properties of the input signal that cannot be described by a combination of the rest of the nodes.

In the following we present entropy measurements as a function of the network morphology. As outlined in section 4.3.1, the morphology is modeled based on a beta distribution with parameters α and β as well as the indegree ξ that defines the device density. We apply a 5Hz sine wave to the upper left interface node of a 16_1 network and connect the lower right node to ground (0V). Fig. 4.9 shows the network entropies as a function of α,β, and ξ. Across all densities, the highest entropies, and hence the least

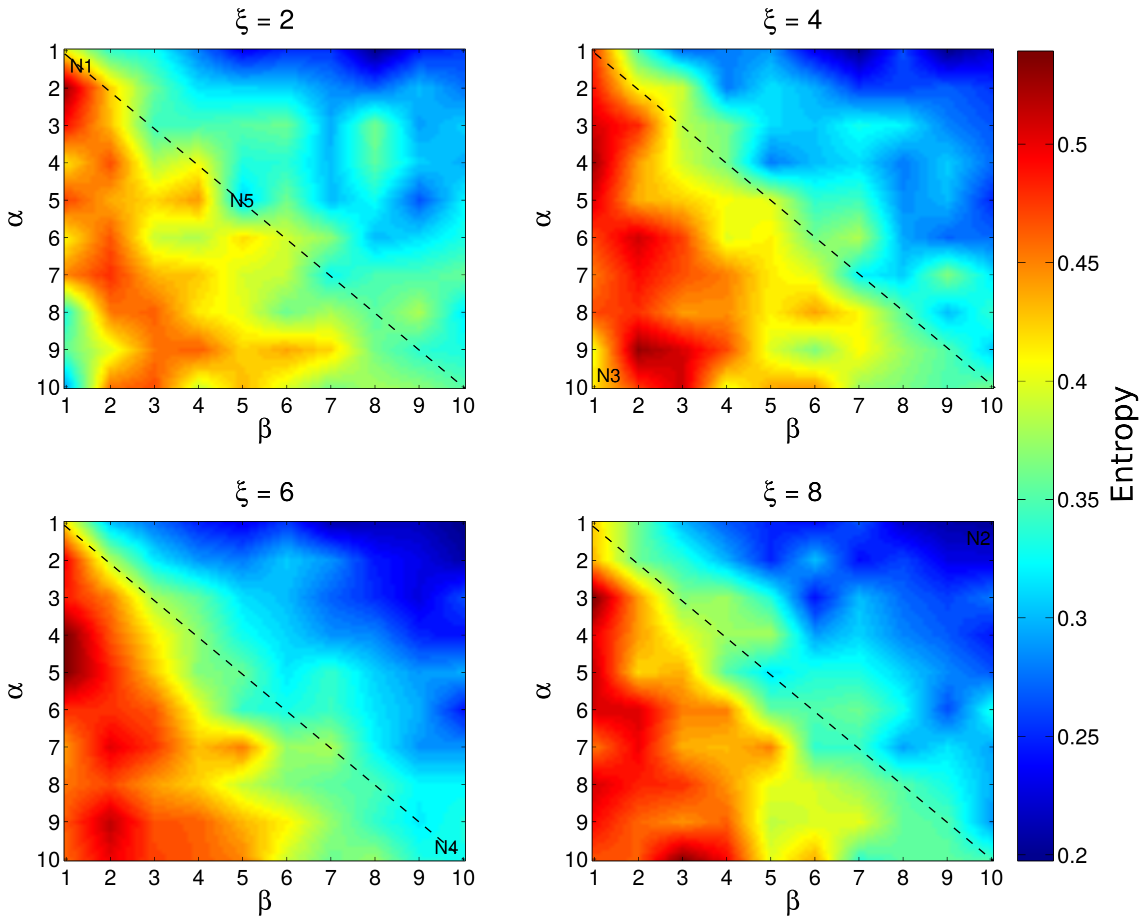


Figure 4.9: Entropy as a function of the network topology defined by the beta distribution values α, β and the indegree ξ ($v = 8V$). Independent of the density, best entropies are achieved for more globally connected network morphologies described by $\alpha \geq \beta$ (lower triangles of the plots). With respect to the network density an indegree of $\xi = 6$ has resulted in best entropies. Lower ξ values provide fewer signal paths and hence less network activity. Beyond $\xi = 6$, additional connections do not contribute any further to the effective network morphology. Average numbers of network devices were $150, 250, 350, 450$ for $\xi = 2, 4, 6, 8$, respectively. Markers Nx refer to table 4.2.

linearly-dependent network states are achieved with a majority of the nanowires being equal or longer than half the normalized maximum euclidean distance (lower left triangles in Fig. 4.9 where $\alpha \geq \beta$). Long-range connections spatially distribute the input signal across the network without much voltage drop. Hence, different areas of the network experience a sufficient bias voltage to exhibit switching dynamics useful to information processing. Increasing network densities, which in other words describes the number of devices per area, creates more signal paths and due to device parallelism also higher conductive connections. This also leads to better distribution of the input signal and to an expansion of the morphologies for which larger entropies can be achieved.

Besides the network morphology, the amplitude v of the input signal also greatly affects the network dynamics due to the exponential dependence of applied bias and either activation energy to form an atomic bridge (for atomic switches) or the velocity of ionic drift (for memristive devices). In Fig. 4.10 we show the entropy as a function of α, β, and v for networks with an indegree $\xi = 6$. It can be seen that the average entropy increases as we increase v. This is related to larger voltages enabling more devices to exhibit switching activity and hence affect the network dynamics. The similarity in the plots as compared to Fig. 4.9 implies that increased input voltages also allow better spatial distribution of the input and more areas of the network to exhibit switching activity. Note that these plots present qualitative results on the dependence of v and entropy, but the absolute values of v are device-dependent and can change for devices with different threshold behavior.

We also studied network sizes 16_0 and 16_2. The 16_0 networks have shown very similar results to the 16_1 networks. The morphological similarity between the 16_0 and the more globally connected 16_1 networks is that they do not form many local fractal structures. Contrary, the 16_2 networks are characterized by more complex morphologies between the interface nodes. While this might closely resemble complex structures in biological

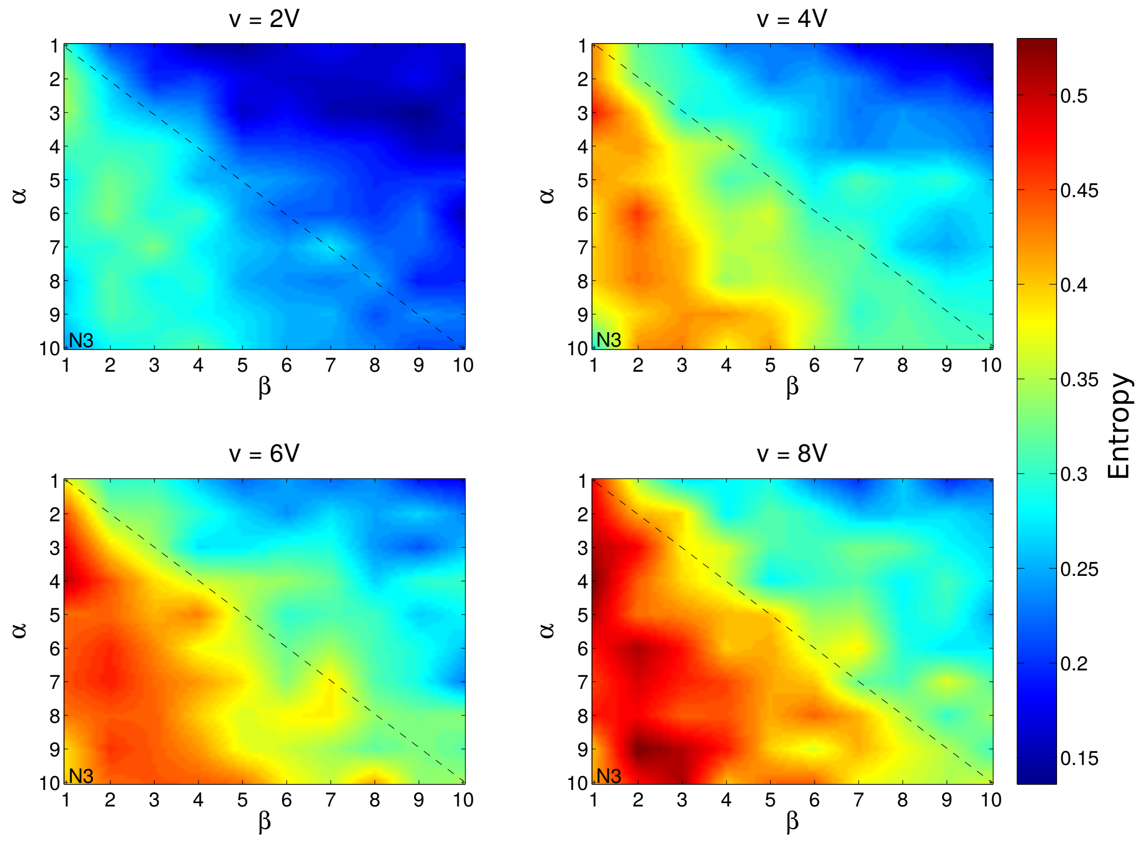
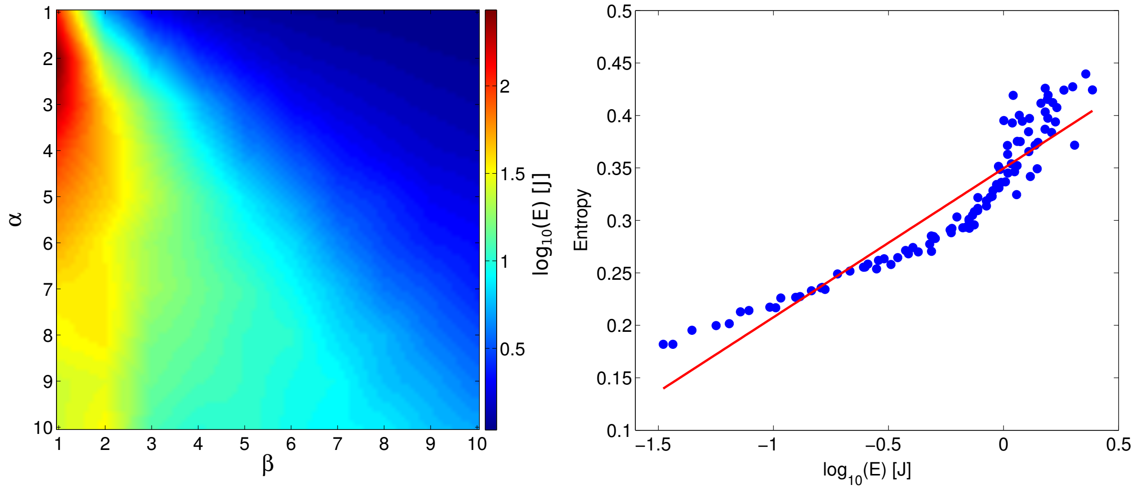


Figure 4.10: Entropy as a function of the network topology defined by the beta distribution values α, β and the input signal amplitude v ($\xi = 4$). Increasing signal amplitudes v lead to higher bias voltages for individual network devices and results in higher switching activity. Highest entropies are again achieved for more globally connected networks with $\alpha \geq \beta$ (lower triangles of the plots). Markers Nx refer to table 4.2.

brains, in the context of passive electrical networks, these fine-grained fractal structures with many devices in between interface nodes lead to alleviated individual switching dynamics. While this could be circumvented in simulations by increasing the input amplitude v, in practice this is not a viable approach for reasons of safe operation and energy consumption.

4.3.3 Energy Consumption

In the previous section we have shown that best computational capabilities are achieved for dense, globally connected networks ($\alpha \geq \beta$) and larger signal amplitudes v. The viability of these results has to be evaluated with respect to the energy that would be consumed by such networks. Based on the application of a 5 Hz sine wave with amplitude v, we calculate the total energy consumption of a network over time T as $E(t) = \int_{i=0}^{T} V_i(t) I_i(t) dt$, with $V(t)$ being the time-dependent signal amplitude and $I(t)$ the current drawn by the network. As the energy consumption is very application-specific, we consider our findings as qualitative measures that highlight the general relations between the network parameters and the consumed energy. Absolute energy numbers presented here are not of relevance, only the information on how drastically energy consumption changes with network parameters. Fig. 4.11a shows the resulting energy consumption for different α and β, averaged over different ξ and v. The distributions of the energy data across the α and β plane resemble the entropy distributions seen in Fig. 4.9 and 4.10, which confirms that high entropies come at the cost of high energy consumption (relative to the consumption at low entropies) caused by high conducting paths in denser networks. This finding is further supported by plotting energy vs. entropy (Fig. 4.11b). Here we plot the averaged energy against the averaged entropy for corresponding setups. We can see how entropy grows with energy. However, as the energy grows exponentially, increasing entropy can



(a) Average Energy Consumption (b) Energy vs. Entropy

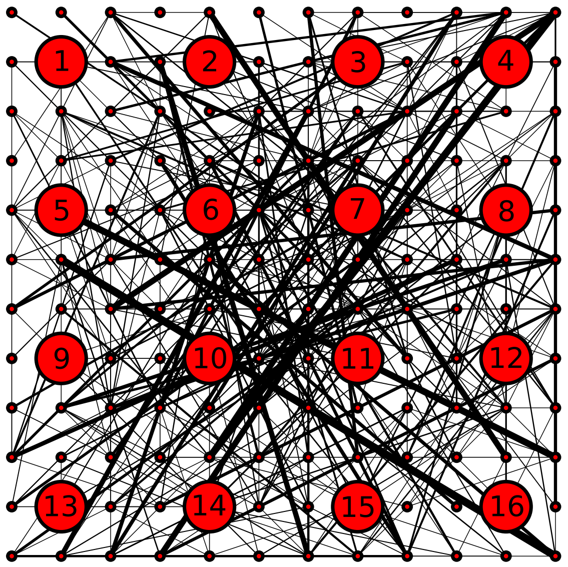
Figure 4.11: Energy consumption in random resistive switch networks. (a) Averaged energy consumption for different network morphologies expressed as $log_{10}(E)$. Energy grows exponentially when transitioning from locally connected ($\alpha = 1, \beta = 10$) to more globally connected networks ($\alpha \geq \beta$). The link between energy and entropy is shown in (b). Entropy grows linearly with exponential energy increase.

lead to over-proportional energy consumption.

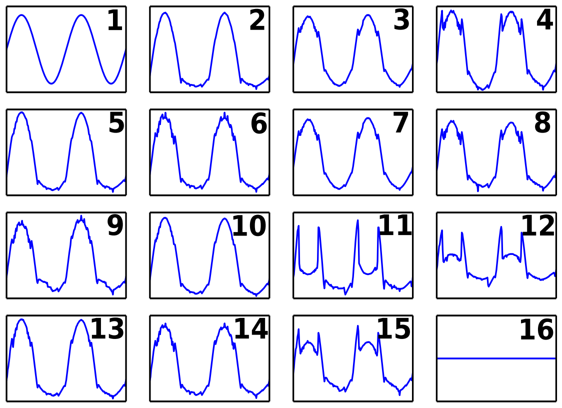
4.3.4 Applying Hierarchical Networks

As the computational performance of a single random network is limited by exponentially increasing energy consumption or strong linear dependence at low energies, we will outline an approach to increase entropy with linear growth of energy. In [99] a hierarchical approach of ASN was presented that combined multiple small-world networks on a single chip. Similarly, we have shown a hierarchical approach that embedded independent networks (similar sizes as presented here) in a reservoir computing architecture and showed that a memory and computationally demanding application could be solved [16].

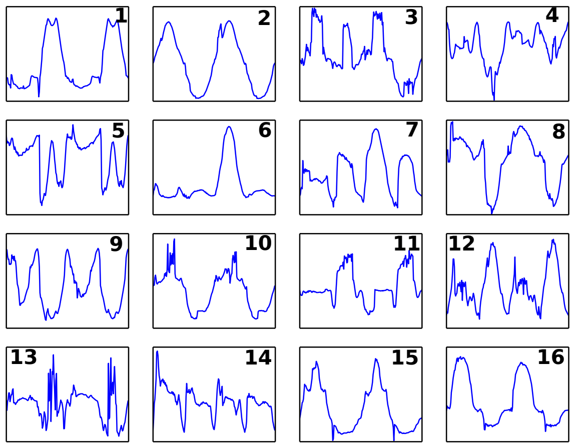
The concept relies on extracting only a subset of signals from each independent network. This allows harnessing the different processing caused by the different random



(a) Network Example



(b) Single Network



(c) Independent Networks

Figure 4.12: Examples of networks and node signals. (a) Example 16_2 network with $\alpha = 2, \beta = 5$, and $\xi = 4$ indicating the 16 interface nodes. (b) 16 Signals measured from a single 16_1 network with the input applied to node 1 and the ground node connected to node 16. The presented signals have an entropy of 0.37. The numbers correspond to the physical location of the nodes within the network. (c) Signals measured from 16 independent 16_1 networks. The entropy for this example is 1.79. Signals were measured as the difference between interface nodes 2 and 9. The numbers represent the network number. All networks were created with $\alpha = 1, \beta = 5, \xi = 4, v = 2V$

structures of the independent networks. Furthermore, in [16] we have extracted a differential signal from each network. By retrieving a network output as the difference of two network nodes, differences in the spatial and temporal dynamics within networks can be better captured than using signals measured with respect to a common ground. In Fig. 4.12 we show two examples of 16 network signals. Fig. 4.12b shows signals obtained from a single random network (qualitatively comparable to signals presented in [27]). Readouts from a single network show some nonlinearities; however, they are mostly characterized by strong linear dependence across the signals (low entropy of 0.37). In contrast, with 16 independent networks exposed to the same input, we can significantly increase the richness of the measured signals. In this example the entropy is 1.79. The total energy consumption of the hierarchical independent networks scales linearly with the number of networks. Considering the gain in entropy, this approach poses a much more viable approach for brain-inspired information processing than relying on single networks with high density and high signal amplitudes. A comparison of how energy and entropy relate in the two presented setups is shown in Fig. 4.13. The plotted data represents collected data from single networks as well as for the 16 networks approach with $\xi = 4$ and increasing v. The circled areas approximately mark the energy and entropy data obtained with a 2 V input signal. The average energy difference is 16 which corresponds to the number of networks used. However, compared with single networks that exhibit similar energy consumption (around $v = 4..6$) significantly higher entropies can be achieved.

4.4 MEMRISTIVE SWITCHING CHARACTERISTICS

The results so far have shown that depending on the task, adjusting system parameters that effect the memristive switching dynamics has a strong impact on the task-dependent

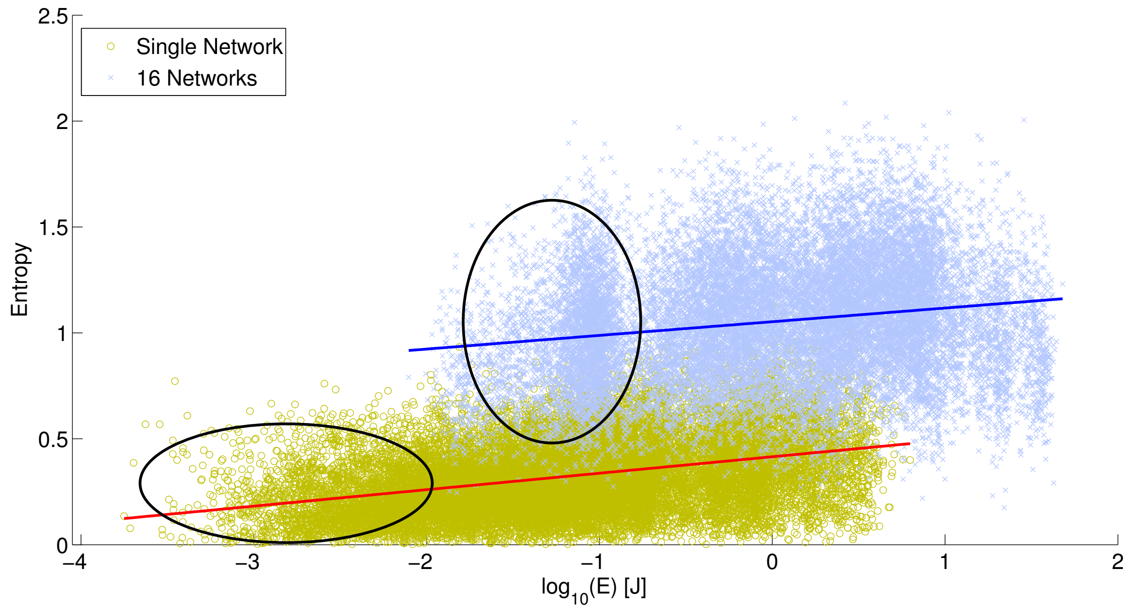


Figure 4.13: Energy vs. entropy for single networks and a hierarchical approach with 16 independent networks. The circled areas highlight an example of networks with the same network parameters ($v = 2, \xi = 2$). Scaling up the number of independent networks is more energy- as well as computationally-efficient as compared to increasing the entropy of a single network by means of scaling up either density or voltage. Data obtained with $v = 2$ and $\xi = 2 \ldots 8$.

performances. I could show that the higher harmonics generation tasks worked better with larger input voltages and hence more internal switching (Fig. 4.5). On the NARMA-10 task the reservoir performed better with lower voltages and less switching (Fig. 4.6). While the network modeling approach has shed light onto the network morphologies, it is not clear yet what the individual device dynamics within a network really look like. In this section I will discuss memristive device characteristics, their modeling, and how they affect the internal network dynamics.

4.4.1 Second-order Device Model

In this section I will describe the device model used in our simulations. Our focus is on capturing the fundamental switching characteristics of the discussed devices, not on precisely reproducing empirical data obtained from a specific device. As outlined in the preceding section, memristive devices are characterized by history-dependent nonlinear conductance changes and state decay.

We adopt a memristor model as presented for WO_x devices in [20, 22]. In the original model the conductance as well as the state change are defined as follows:

$$G = \left[(1-w)\,\epsilon\left[1-\exp\left(-\theta V\right)\right] + w\gamma\sinh\left(\delta V\right)\right]\frac{1}{V} \tag{4.7}$$

$$\frac{dw}{dt} = \lambda\sinh\left(\eta V\right) - \frac{w}{\tau} \tag{4.8}$$

Here the internal device state is modeled by w, V is the applied input bias, $\epsilon, \theta, \gamma, \delta, \lambda, \eta, \tau$ are the model parameters calculated from the experimental data (detailed experimental data is property of authors of [20, 22] and has not been released to the public). The non-

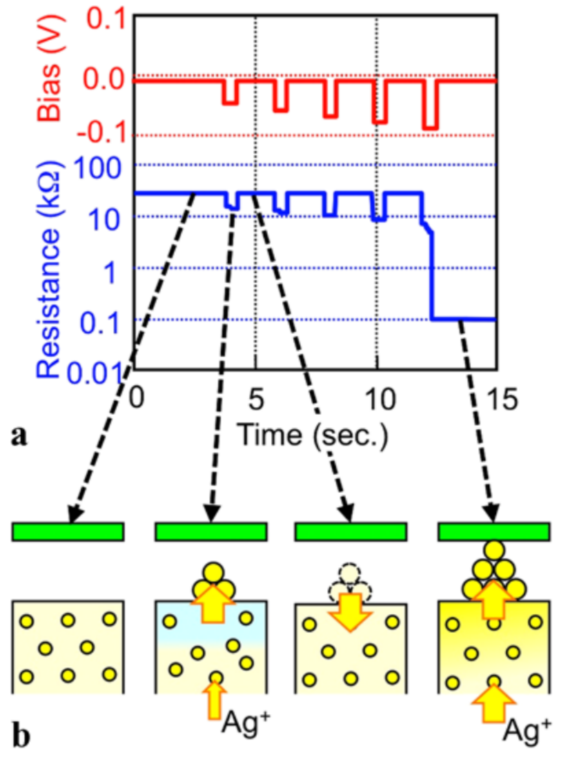


Figure 4.14: Atomic switch resistance changes based on formation and dissolution of the atomic bridge. Based on two state variables, only when an atomic bridge is established internally (first variable) will the external resistance (second variable) change measurably. (Image taken from [38].)

linear switching, as explained by the exponential ionic drift model [101], is modelled by the sinh term. This model captures well the nonlinear switching as a function of the applied input and the current state.

However, such a first-order model, using only one variable to describe the device conductance, does not capture effects such as Ag^+ cation concentration changes. Such a change affects a device's response to future bias signals, but does not reflect into the actual device conductance. In [38] this difference was described as the memristor using a single variable to model the size of an ion-doped area, while the atomic switch uses two variables, one to model the height of an Ag protrusion, and another to model the width of the atomic bridge that emerges after the Ag protrusion has reached a sufficient height (see Fig. 4.14). Recently a second-order memristor model was presented that follows a similar modeling approach to the atomic switch [60]. Here the second variable that functions as an enabler for a subsequent change in conductance is the internal device temperature. Application of an applied bias signal increases the internal device temperature due to Joule heating, which in turn affects drift and diffusion processes described earlier.

To account for effects such as Joule heating or Ag protrusions, we use equation 4.8 to model an internal state w' that does not directly reflect in the device conductance. Furthermore, we extend the model to implement the different state decays based on the device state. As shown in [21, 80], the rate of dissolution of the atomic bridge or the diffusion of ions to an equilibrium state is state-dependent and enables short- and long-term memory within a single device. We adopt equation 4.8 as well as describe state variable w, which models the device conductance as:

$$\frac{dw'}{dt} = \lambda \sinh(\eta V) - \frac{w'}{\tau}(1 - w') \tag{4.9}$$

$$w = f(w') \tag{4.10}$$

For our entropy and energy simulations we employed a binary switching function f that thresholds w' to create two distinct conductances. Binary behavior of atomic switches was shown in [80] and is also found in some other memristive devices [32]. Different levels of sub-surface Ag^+ concentrations are required to establish or dissolve an atomic bridge (length and width of Ag protrusion) [79]. This implies different threshold values

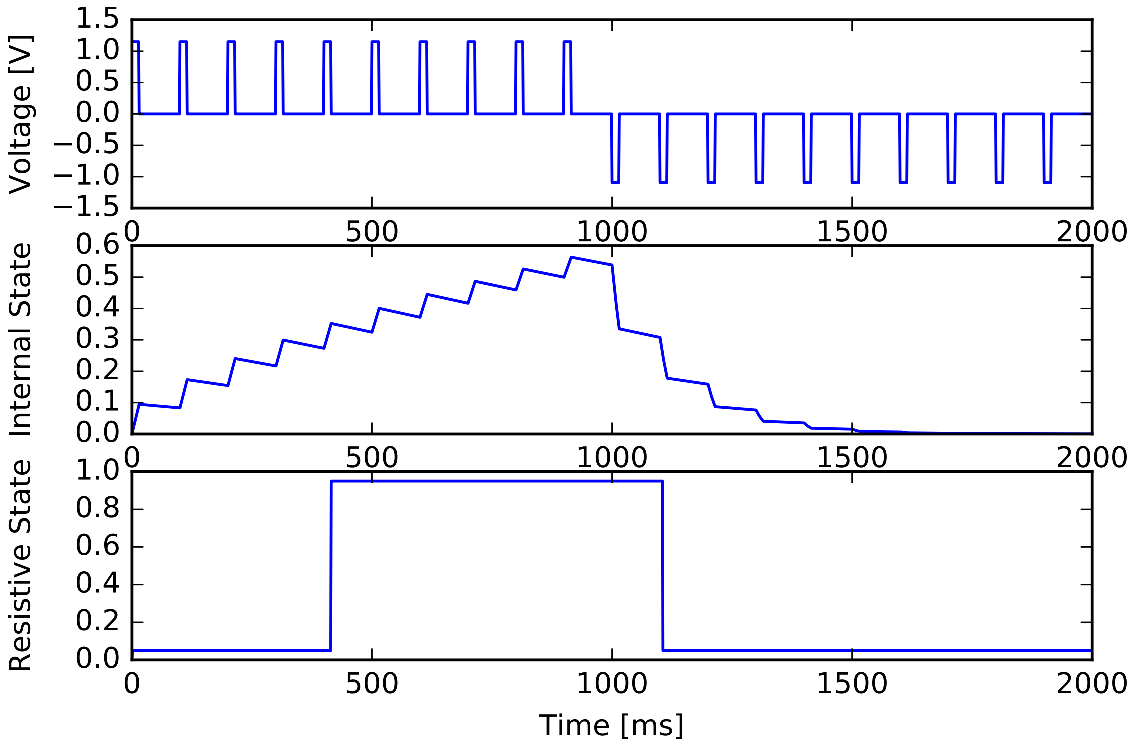


Figure 4.15: Second-order device model. Top: Applied sequence of positive and negative signals. Middle: Internal device state that is a direct function of the applied signals and their timing. Bottom: Resistive state of a binary device depending as a function of the internal device state. The resistive state here is a unit less number where 0 and 1 represent maximum and minimum resistance respectively.

for the internal state variable w' to perform device switching. We model this by applying a hysteresis function to w'. In the next sections we will compare different variations of the presented device model.

4.4.2 Switching Resolution

Based on the second-order device model, I will now discuss network behavior as a function of different switching characteristics. For this, three different models will be defined: (a) an analog switching model, (b) a binary model, and (c) a model with 10 resistive states. In the case of the analog model the resistive state of the device, that is the state relevant to the network dynamics, directly follows the internal device state. Every small change in the internal state will directly reflect in the measurable device resistance. Contrary, the binary device only exhibits two distinct resistive values, while the internal state is still modeled analog. Internal state changes don't necessarily reflect into the device's resistance right away, but require to pass a threshold level. The model with 10 resistive states functions similar to the binary model in the sense that it can experience internal state changes that don't directly reflect in the resistance. However, with more possible states and hence a higher resolution the resistive state of this model is more responsive to the internal state.

In Figures 4.16, 4.17, and 4.18, I compare the network dynamics for the three different models when the network is driven with a constant input bias. The constant bias ensures that network devices have the required electrical signal to perform switching, but that the network state changes are not a result of a change in the input bias itself, but rather of the internal network dynamics. The results for the analog model (Fig. 4.16) show that the network states (voltages measured at the interface nodes), briefly after the application of the bias, settle at constant levels. These levels are a function of the network's topology

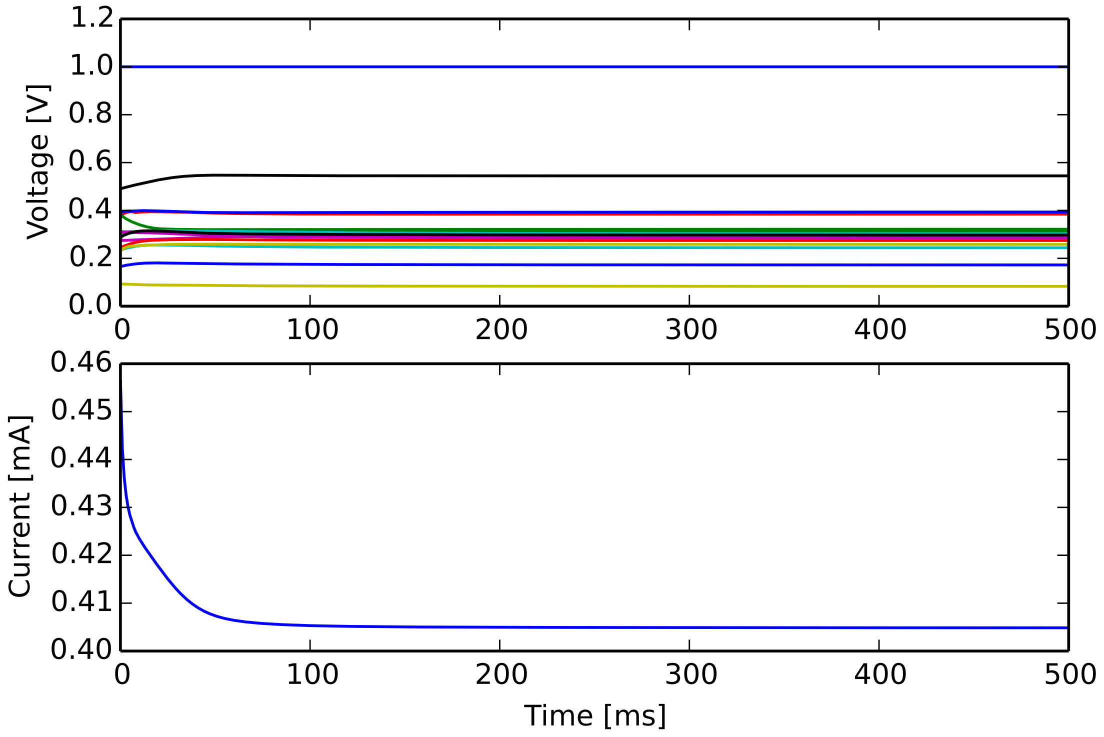


Figure 4.16: Voltages and current for a network with analog devices exposed to a constant input bias. The analog switching allows the internal devices to immediately update their state according to the applied signal. This direct, immediate response results in an equilibrium state.

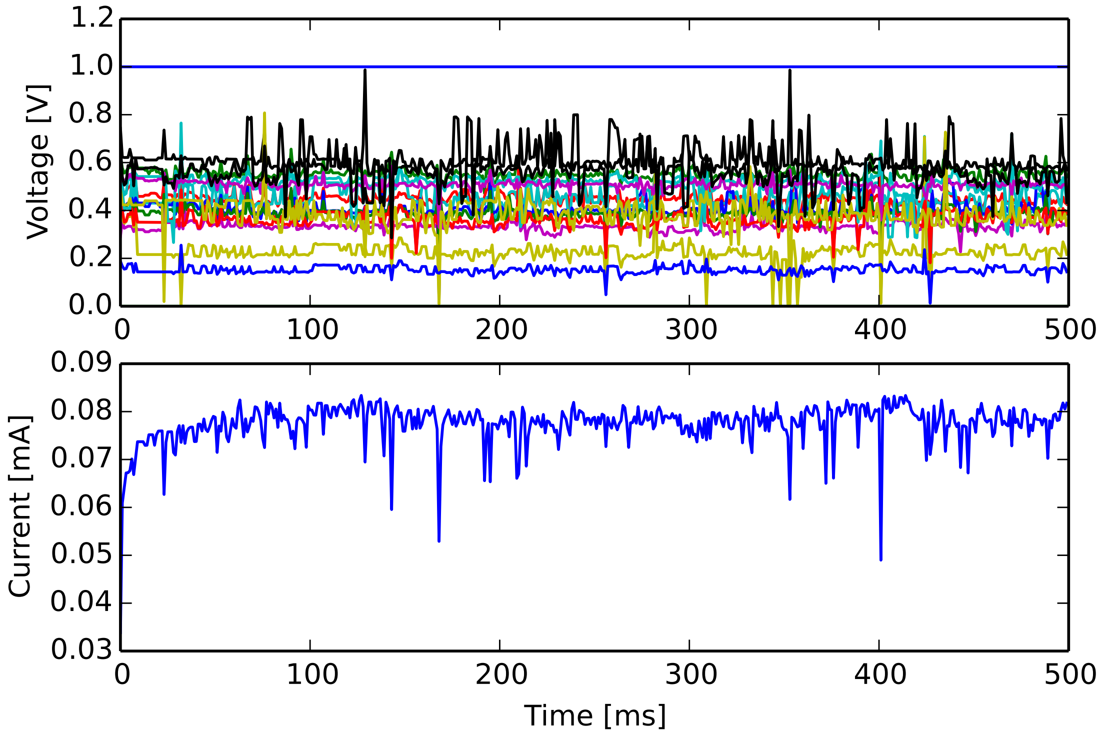


Figure 4.17: Voltages and current for a network composed of second-order binary devices. While internal device states integrate the applied bias, the binary switching function effectively implements a delayed/filtered response to the bias. With many devices competing for the applied bias, this results in a non-equilibrium state.

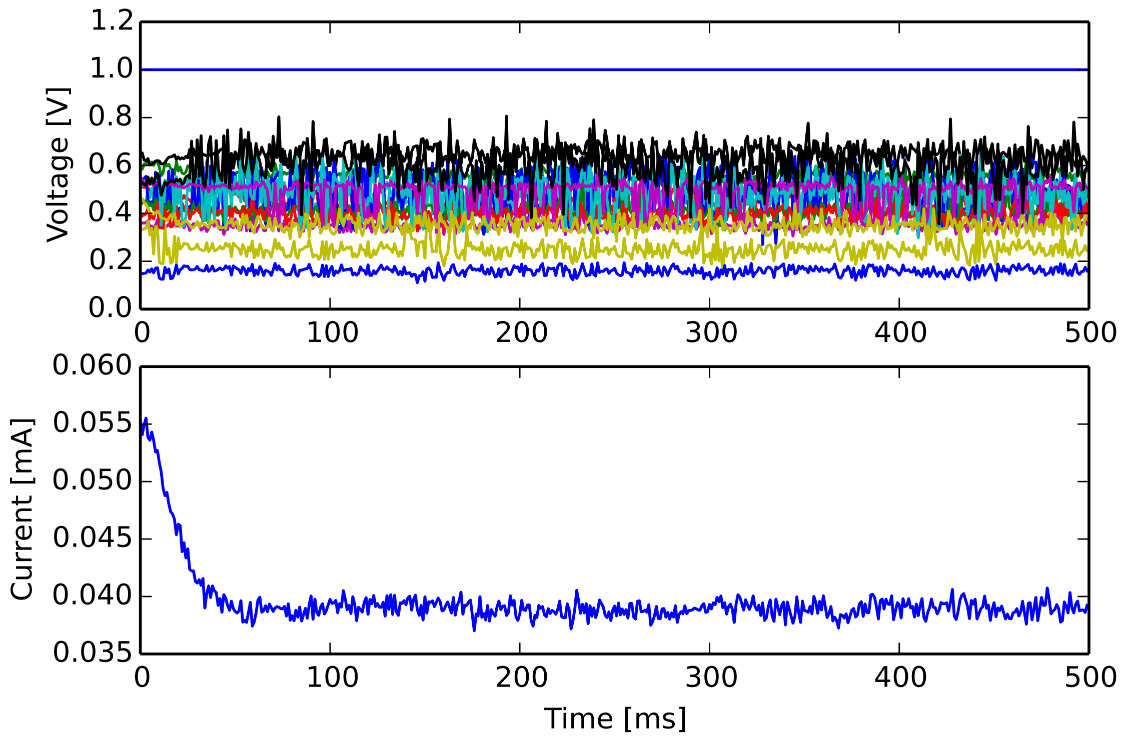


Figure 4.18: Voltages and current for a network with a 10-step second-order device model. Similar to the binary model with the difference that due to the increased amount of device states, more frequent resistive changes result in less pronounces voltage and current fluctuations.

and the respective distribution of resistances that govern the distribution of voltages. This behavior is the result of the network's ability to balance any competing memritive states due to the immediate response of these devices and reach an equlibrium state where all competition for voltage has been settled. The network composed of second-order binary device models exhibits very different behavior. Even under constant input bias the network does not reach an equilibrium state. This behavior is caused by delayed as well as abrupt changes in device resistances. While the internal device states are directly affected by the applied voltage, a thresholding function with two distinct resistive states causes sudden redistribution of voltages when individual devices switch. In turn, these re-distributions of voltages ensure the continuing oscillation of network voltages. Similarly, this is the case for the model with ten different resistive states (Fig. 4.18) where the network states do not reach an equilibrium state either. In contrast to the binary model, the overall magnitudes of voltage changes measured at the network nodes as well as the fluctuations in the current consumed by the network are lower. This can be explained by the smaller step sizes and a less delayed response of resistive changes.

After having shown how different device models affect network dynamics under a constant bias, I will discuss the physical plausibility and computational implications. While many memristors are described by first-order models, a second-order model captures more realistically a device's response to a wider variety of applied signals. Many models were designed with respect to a pre-defined signal with the aim to match experimental data with a mathematical description. However, when the shape, duration, magnitude, etc. of the input signal changes, not all models will remain accurate. The reason lies in the internal switching dynamics of memristive devices. In case of the atomic switch (gap-type device) [80] this can be illustrated as a metal atomic bridge having to be established first between the two terminals before the resistive state of the device changes.

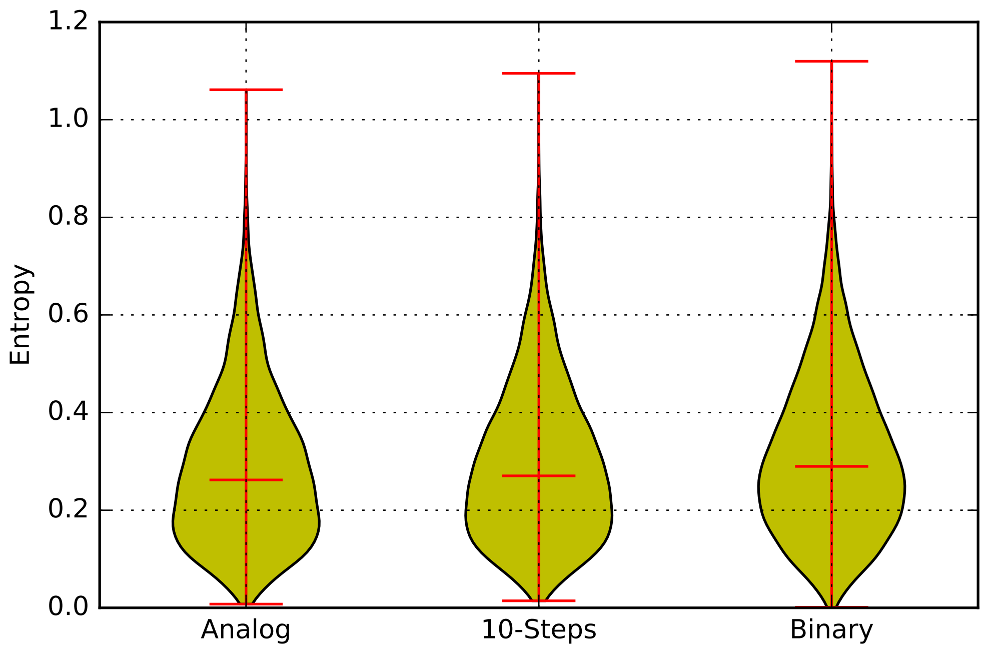


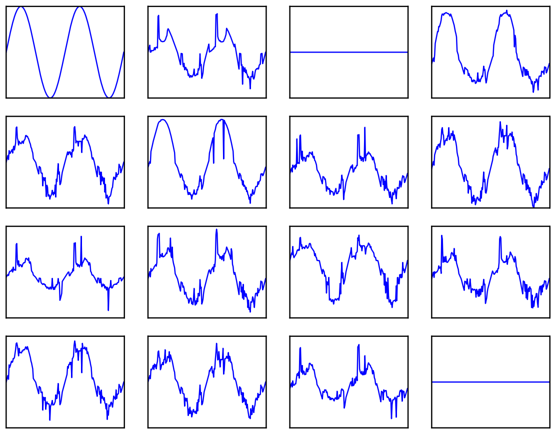
Figure 4.19: Entropy violin plot of simulations with different device models. Each violin is represents 25,000 data samples over a variety of network parameters. We can see that the medians (red line inside the violins) as well as the maximum values increase as the switching resolution decreases. The median values for the analog, the 10-step, and the binary model are 0.26, 0.27, and 0.29 respectively.

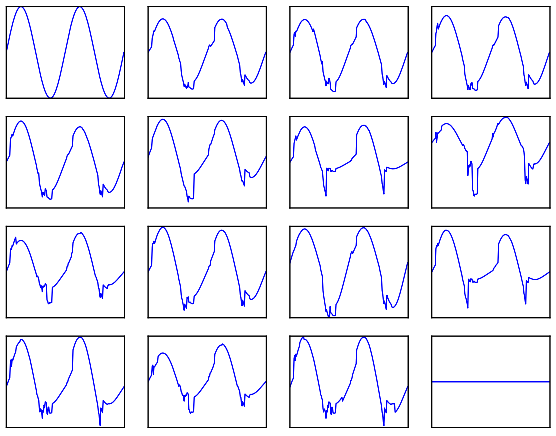
Hence the behavior shown by the analog model (first-order) is not physically plausible according to more recent publications on memristive switching and I argue that using second order models that don't behave perfectly analog should be preferred. From a computational point of view we are interested if the different switching characteristics affect the network performance. I illustrate such effects based on a violin plot in Fig. 4.19. This plot indicates minimum, maximum, median, and the general distribution of the sampled data. We can see that the median and maximum entropies are lowest for the analog model. Furthermore the widest part of the violin, indicating the most likely entropies, is further to the bottom of the plot. In contrast, the binary model shows a higher median entropy and a distribution that indicates the most likely entropy for a random network close to the median. Based on the physical plausibility for device models that do not exhibit arbitrary small resistive state changes as well as on the higher entropy values, I conclude that devices with a few stable resistive states are favourable for random memristive networks to exhibit nonlinear dynamics that can be harnessed by the reservoir computing approach.

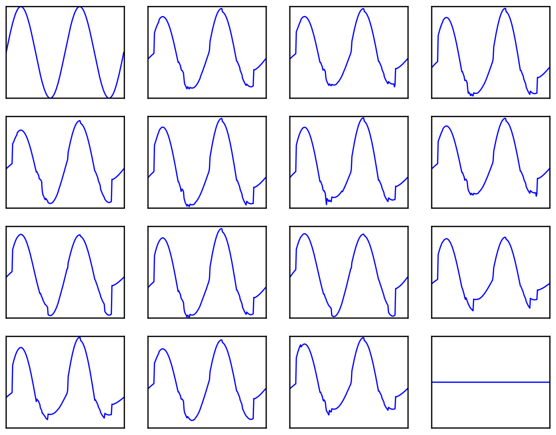
4.4.3 State Volatility

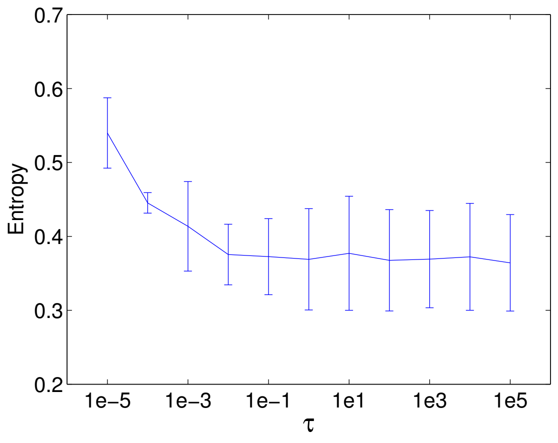
Another important aspect of memristive device characteristics is the state volatility. In the more researched field of applications where memristors are used as synaptic weight storage, state volatility is an undesired property as it would require periodic refreshing of the memristive states. In contrast to using memristors for storage and inference of the stored information, the memristive reservoir approach relies on extracting information based on state changes of memristors, not constant states. Hence, intuitively devices that exhibit volatility should lend themselves better to this approach. In this section I will present some data that underlines the need for volatile devices.

In Fig. 4.20, I show recorded network states for networks composed of memristive


(a) $\tau = 1e^{-}5$


(b) $\tau = 1e^{0}$


(c) $\tau = 1e^{5}$



(d) Entropy(τ)

Figure 4.20: Influence of the memristive volatility on the entropy. (a) Network states of a network composed of memristors with a fast state decay ($\tau = 1e^{-}5$). The fast decay reflects in fast competition over network voltages which is shown by the high frequency components modulated onto the sine wave input. (b) Network states when using devices with $\tau = 1e^{0}$. This network exhibits significantly less high frequency switching. (c) Network states when using devices with $\tau = 1e^{5}$. All non-linear dynamics within this network derive from actual state switching when a signal is applied and not from state decay. (d) Entropy as a function of τ. Data collected and averaged over 25,000 experiments with a variety of network and input signal settings.

devices that differ in their state volatility. Comparing very fast state decay rates ($\tau = 1e^{-5}$) to slower ones ($\tau = 1e^{0}$), and to practically non-existing state decay ($\tau = 1e^{5}$) we can see how faster state decay imposes more high frequency components onto the network states. Applying the entropy measure to these signals, high frequency components lead to less correlation and hence larger entropy values (Fig. 4.20d). I will explain the presented data on why faster decay leads to higher entropies on a short example. If a memristive device withing the network experiences enough of a voltage drop to switch it's state to a low-resistive state, the voltage drop over this device is suddenly much smaller. Hence, with no state decay this device would remain in that state and not further contribute directly to the network dynamics. With faster state decays, devices can perform more switching and therefore contribute more frequently to the competition and redistribution of voltages that leads to the dynamics of a network. However, this is just a general discussion. Optimal values for the state decay will depend on the applied input signal and the task at hand that determine the relevant time-constants.

4.5 DISCUSSION

In this chapter I have presented various aspects of utilizing random memristive networks for computation. This work has shed light onto relevant architectural design choices and computational capabilities of such networks when used in a reservoir computing paradigm.

The limited value of strong signal correlation within a single memristive network can be circumvented by applying a modular approach to memristive reservoir computing. Extracting signals from modular networks enhances the richness of the measured networks outputs, which benefits from the different random structures that determine a network's response to an input signal. Comparing 16 signals extracted from a single network to one signal extracted from 16 networks, 17.5% average increase in the richness could be observed at the same energy consumption. With the ability of modular fabrication [99] future work can move toward the integration of on-chip memristive networks as well as further study methods to increase the richness of the network signals.

With respect to the network morphology it was shown that by influencing the distribution of nanowire lengths one can control the computational capacities as well as the energy consumption of a network. The presented modeling approach has been the most detailed to this date and provides the basis for future work. With respect to the variation in the presented results for the same network morphologies, a much closer look at the resulting networks can be taken to determine the cause for variation. This will likely extend the understanding of the broader parameter exploration presented here and provide more insights into sub-parameter spaces and best/worst-case scenarios.

For switching resolution as well as the memristive state volatility we have seen that binary devices with fast state decays create the most diverse representations of an applied input signal. The increased entropies are a result of higher frequency components being part of the resulting signals. In future work it has to be evaluated if these higher frequency components can be efficiently harnessed for computation or if they turn out to represent noise rather than information.

5

CONCLUSION

The motivation for my research arose out of the growing possibilities of employing emerging nanoelectronics in neuromorphic hardware. With the long-term goal of building electronics that compare in computational complexity and energy consumption with that of biological brains, I saw potential in the use of memristive devices beyond synaptic weight storage. This evaluation led to my interest in defining architectures and algorithms that exploit memristive device properties, that are typically not considered desirable for synaptic weight storage, for information processing.

MEMRISTIVE CROSSBARS

In chapter 3, I set out to investigate the possibilities of computing within memristive crossbars. Device level fabrication primarily considering the crossbar structure will make this a mature architecture that can serve as the base for storage as well as computation. To this day, primarily used as synaptic weight storage, the regular and parallel structure of the crossbar is not a natural fit for dynamic information processing due to every device operating independent from all others. Based on state volatility of some memristive devices, which is a concept similar to synapses that provide temporal feature extraction [35], I presented a simple architecture that applies data to the crossbar rows and modulates the crossbar columns in order to achieve heterogeneous processing of the applied input data. Compared to another memristor-based architecture [83] my approach achieved the same classification rate of 93% on the MNIST data set while requiring 3 times less neurons and 15 times less weights to train.

Next, I focused on generalizing this approach as to better understand the details that contribute to the overall approach as well as to make it applicable to other data sets. Multiple aspects were shown to be important in utilizing the crossbar efficiently. The spatiotemporal code that the reference signal provides is essential to the feature extraction from input data. Deriving the reference signal from training data, PCA and class mean representations offer scalable methods to extract features and increase classification rates of up to 93% when utilizing a *tanh* activation function in connection with the readout operation. Using random reference signals can achieve identical classification rates if the sparsity constraint is chosen accordingly. Too sparse reference signals do not allow to extract enough features, while less sparse signals create features that are less descriptive of the applied input. While the design of the reference signals determines the integration of input data, methods to efficiently extract information from the crossbar for further processing have been presented. As a readout operation conflates multiple independent device states, a two-dimensional readout operation was introduced that allows to retrieve a more informative representation of the crossbar states. With one-dimensional readout operations along rows or columns leading to 14% and 60% correct classifications, respectively, a two-dimensional readout achieved 78% (smaller setup used than for the 93% classification rate). In addition to performing a two-dimensional readout it was shown that multiple readout operations with shorter readout vectors rather than a single readout with one long vector lead to better classification results. The reason is again found in the conflation of memristive states. The fewer devices involved in a readout operation, the more accurate the crossbar states can be retrieved.

While the computational considerations have introduced relevant methods and achieved competitive results, the viability of any architecture based on emerging nanoelectronics has to be evaluated considering the effects of physical device imperfections. Device variation describes the functional differences between different devices. Cycle variation describes a non-deterministic behavior of a single device over repeated application of the same signal. Relying on reservoir computing principles of exploiting dynamic state changes rather than predefined states, device variation is not a problem for the crossbar approach. I have shown that the classification rate remains stable up to 6 times the level of device variations reported so far. Beyond that, the drop of classification rate is attributed to a loss of nonlinear behavior that the memristive devices experience [17]. Cycle variation is more critical, yet within random fluctuations of up to 1% [2], as was shown for one memristive device, classification rates remain nearly constant. However, with the lack of available data on the phenomenon of cycle variation it remains to be evaluated for a broader set of devices.

As computing within crossbars is a new concept that has been introduced here, all work so far focused on the design and optimization of the used methods. The next steps expand the horizon by comparing to other methods and data sets. A benchmark experiment has been performed with an extreme learning machine implementation. Based on the same number of signals applied to a trainable readout layer, the crossbar approach shows advantages for smaller system sizes of less than 750 signals. However, the classification rates scale more favorably with the ELM approach and achieve better results beyond using 750 signals. It has to be noted though that the crossbar approach only requires 28 input neurons by utilizing MNIST as a spatiotemporal data set, while the ELM approach relied on 784 input neurons to the system. Another benchmark was performed to validate the crossbar approach on a different data set and compare it to an echo state network implementation. Based on 10 classes of jittered spike trains with added noise, the classification rates of the crossbar and the ESN were compared. Performing nearly identically for data samples containing sparse activity and no noise (0%) or almost only noise (50% of possible spikes), the crossbar approach achieves classification rates of up to 20% higher than the ESN for a wide range of signal sparsity and noise levels in-between the extreme cases of 0% and 50%. While all noise contained in the input data will enter and circulate within the ESN, the reference signals determining the spatiotemporal code for the crossbar to be receptive to input signals allows for most noise signals to not enter the crossbar and hence not corrupt the integrated signals.

A limitation of this approach has been shown in the scalability comparison with the ELM. In case of implementations with memory-less hidden neurons, such as the ELM that receives input data in a purely spatial manner, increasing the number of neurons allows for an increasing number of input feature combinations. In contrast, the fixed connectivity of the crossbar restricts each memristive device to input data along the temporal domain only. Another limitation concerns the input data format. All experiments considered binary input signals where information is encoded in the spike location and timing. This allowed to utilize state decay for temporal feature extraction. In analog encoded signals the information density is likely to be too high as for the crossbar to efficiently detect and extract relevant features. Therefore, this approach provides a viable solution to spatiotemporal spike detection as well as to complement other information processing architectures by providing a unique way to extract meaningful features in a compact architecture. Possible applications are inherently spatiotemporal problems such as speech recognition (encoded in spikes). In applications of image classification / object detection, where an applied image produces a sparse spiking activity over a range of neurons, the crossbar approach can also implement order-selective detection of these spikes to conclude on target features. In general, the scalability of the crossbar structure allows for a wide range of input data representations and hence for a wide range of data sets.

RANDOM MEMRISTIVE NETWORKS

A second architecture for computing with random assemblies of memristive devices was shown in chapter 4. Preliminary simulations and physical assemblies have demonstrated their ability for simple pattern classification and generation. Based on the introduction to memristive networks in section 2.3.2, it became clear that memristive networks are facing limitations in their ability to create a higher dimensional representation of the input data that could then be used to train a readout layer. I have introduced a hierarchical architecture that utilizes a set of small-scale memristive networks that are interconnected by an underlying CMOS layer. Information processing is done within the memristive networks. Signal amplification and routing is done in the CMOS layer. Based on the higher harmonics generation tasks that were shown for single memristive networks I demonstrated the hierarchical architecture's ability to benefit from independent networks with different topologies, and hence different transformations of the input data. For all three signals, sine, square, and triangle, the hierarchical approach achieved up to 6 times lower mean-square errors as compared to a single network. Based on the time-series prediction task NARMA-10, a task too complex for a single memristive network, I compared the hierarchical memristive network to a echo state network implementation. Based on the same number of signals extracted, the heterogeneity of the memristive networks allows to continuously improve the prediction error while the ESN approach stagnated. In its largest setup with 400 signals the memristive reservoir approach achieved 0.15 NRMSE while the ESN approach achieved 0.35. While these results made a strong case for computing with memristive reservoirs, it did not provide insights into how different network morphologies affect the computational results.

Based on physical factors, such as the size, shape, and the distance of electrodes that guide the fabrication of random networks, I presented a comprehensive modeling approach that can control different network morphologies with a few parameters. A beta-distribution was used as probability density function to define the ratio of short and long range connections with the network. A grid approach allowed to control the growth of nanowires with a more fine-grained grid leading to more fractal structures. Following this modeling approach I investigated computational capacity and energy consumption of these networks. Computational capacity was described as the richness of the extracted network signals. A network has a rich set of signals if these are linearly uncorrelated so that each signal contains information that cannot be obtained by any other signal. It was shown that the highest computational capacities are obtained for networks that are more globally connected. Global connections allow the applied input signal to more evenly spread throughout the network and cause more devices to exhibit switching activity. The downside of this increased switching activity is the resulting energy consumption. As the voltage-current relationship of memristive devices follows nonlinear characteristics, the linear growth in computational capacities causes exponential growth in energy consumption. A solution to this problem is given by the hierarchical architecture. Relying on fewer signals per memristive network, but implementing more networks allows to harness the different morphologies and to increase computational capacities in average of 17.5%, as compared to single networks, at linear growth in energy consumption.

To create a complete picture of contributing factors to computational capacities, I took a look at the memristive switching characteristics. I implemented different switching behavior in order to gain an understanding how it impacts computational capacities. An analog model, where every change in the internal state causes a change in the measurable device resistance was compared to second-order devices. Such devices use two variables to model possible switching. One variable models the internal state, while the second variable maps the internal state to the external device resistance following a specific function. Here I used a binary as well as a denary (ten states) function. Under the application of a constant bias signal, I observed the network states. The analog model results in a fast equilibrium of the network states as it allows to immediately mediate between all competing voltage levels within the network. The binary and denary models exhibit ongoing network activity as a result of sudden memristive state changes and resulting redistributions of voltages within the network. Such a behavior is similar to synaptic activity in biological brains under constant external stimuli. However, while a difference in computational capacities could be observed, the differences were minor (entropies of 0.26, 0.27, 0.29 for the analog, denary, and binary model, respectively). Hence, more important to an efficient design of random memristive networks is the network morphology and the hierarchical utilization of independent networks.

The computation with hierarchical memristive networks involves design parameters determining the connectivity between memristive networks, the connectivity within memristive networks, device characteristics, input signal encoding, and network signal extraction. Due to this design complexity further research is required before the full potential of memristive reservoir computing can be understood. Based on the results presented here, that have shown memristive reservoirs to outperform an ESN implementation on the NARMA-10 task, we assume that this approach holds further potential. While it was shown that more globally connected networks lead to richer network activity, a more detailed look has to be taken at the randomness in the network creation that is likely to cause the deviations in the network behaviors. Once the network creation is fully understood, this approach can be applied to other applications. I.e., an ESN implementation for speech recognition was shown to lead to competitive results. Based on the hierarchical architecture, the scalability of this approach certainly permits tasks of such complexity.

5.1 LIST OF CONTRIBUTIONS

General

- My work is the first that was going significantly beyond the simple recognition of memristive devices as storage elements by showing combinations of architectural and algorithmic considerations demonstrating working examples of computing with memristive devices.

- This research extends possibilities of neuromorphic hardware architectures to operate more efficiently by exploiting inherent device properties.

- The demonstration of harnessing diverse device properties allows memristive device research to see value in device properties that in conventional architectures would be considered undesirable.

Memristive Crossbar

- With a mathematical interpretation of memristive crossbars that outlined relevant components to create heterogeneous state patterns within a highly parallel structure, a novel computing architecture based on emerging nanoelectronics has been presented.

- By considering intrinsic device properties as a potential for efficient computation, state volatility was shown to not be an undesired property, but one that can efficiently extract temporal information from input sequences.

- This approach was shown to work for inherently spatiotemporal problems as well as provide the opportunity to reduce the system size for applications with high-dimensional spatial input data.

- Utilizing nonlinear integration and state decay for computation, the crossbar performs useful pre-processing that allows to significantly reduce the training complexity.
- The reference matrix implements noise robustness in a unique way. As the spatiotemporal reference signal determines when the crossbar integrates an input signal, most noise won't even alter the memristive states.
- By utilizing the dynamic range of the device states rather than fixed predefined states, the crossbar approach provides an implementation that is indifferent toward realistic levels of device variation and even robust under realistic levels of non-deterministic cycle variation.

Random Memristive Networks

- I have introduced a scalable hierarchical approach that harnesses the dynamics of memristive networks for nonlinear transformations of input signals, and that avoids alleviated dynamics of large networks.
- Scalability and heterogeneous network morphologies have allowed to significantly increase task complexities to; (a) new levels for memristive reservoir computing and (b) to competitive levels among the general reservoir computing field.
- The detailed modeling framework of network morphologies has provided a clear understanding of the relations between network dynamics and energy consumption.

- Different device models and volatility rates have on the one hand demonstrated the effect on network dynamics and on the other hand provided an understanding of switching characteristics beneficial to memristive reservoir computing.

5.2 LIST OF PUBLICATIONS

1. J. Bürger and C. Teuscher. Variation-tolerant Computing with Memristive Reservoirs. In *Nanoscale Architectures (NANOARCH), 2013 IEEE/ACM International Symposium on*, pages 1–6, July 2013 [17]

2. J. Bürger and C. Teuscher. Volatile Memristive Devices as Short-Term Memory in a Neuromorphic Learning Architecture. In *Nanoscale Architectures (NANOARCH), 2014 IEEE/ACM International Symposium on*, pages 104–109, July 2014 [18]

3. W. Woods, J. Bürger, and C. Teuscher. On the Influence of Synaptic Weight States in a Locally Competitive Algorithm for Memristive Hardware. In *Nanoscale Architectures (NANOARCH), 2014 IEEE/ACM International Symposium on*, pages 19–24. IEEE, 2014 [113]

4. J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher. Hierarchical Composition of Memristive Networks for Real-Time Computing. In *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pages 33–38, July 2015 [16]

5. W. Woods, J. Bürger, and C. Teuscher. Synaptic Weight States in a Locally Competitive Algorithm for Neuromorphic Memristive Hardware. *IEEE Transactions on Nanotechnology*, 14(6):945–953, Nov 2015 [114]

6. J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher. Computational capacity and energy consumption of complex resistive switch networks. *AIMS Materials Science*, 2(4):530–545, 2015 [15]

7. W. Woods, M. M. A. Taha, S. J. D. Tran, J. Bürger, and C. Teuscher. Memristor panic - A survey of different device models in crossbar architectures. In *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pages 106–111, July 2015 [115]

Bibliography

[1] L. F. Abbott and S. B. Nelson. Synaptic plasticity: taming the beast. *Nature neuroscience*, 3:1178–1183, 2000.

[2] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23(7):075201, 2012.

[3] F. Alibart, E. Zamanidoost, and D. B. Strukov. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nature Communications*, 4(2072):1–7, 2013.

[4] A. V. Avizienis, H. O. Sillin, C. Martin-Olmos, H. H. Shieh, M. Aono, A. Z. Stieg, and J. K. Gimzewski. Neuromorphic atomic switch networks. *PloS ONE*, 7(8):e42772, 2012.

[5] M. F. Balcan, A. Blum, and S. Vempala. Kernels as Features: On kernels, margins, and low-dimensional mappings. *Mach. Learn.*, 65(1):79–94, 2006.

[6] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith. Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(48):1–13, 2014.

[7] B. Benjamin, P. Gao, E. Mcquinn, S. Choudhary, A. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. Arthur, P. Merolla, and K. Boahen. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[8] R. Berdan, T. Prodromakis, a. Khiat, I. Salaoru, C. Toumazou, F. Perez-Diaz, and E. Vasilaki. Temporal processing with volatile memristors. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 425–428. IEEE, 2013.

[9] N. Bertschinger and T. Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004.

[10] G. Bi and M. Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual Review of Neuroscience*, 24:139–166, 2001.

[11] E. Bingham and H. Mannila. Random Projection in Dimensionality Reduction: Applications to Image and Text Data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 245–250, New York, NY, USA, 2001. ACM.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[13] A. Blum. Random Projection, Margins, Kernels, and Feature-selection. In *Proceedings of the 2005 International Conference on Subspace, Latent Structure and Feature Selection*, SLSFS'05, pages 52–68, Berlin, Heidelberg, 2006. Springer-Verlag.

[14] D. V. Buonomano and W. Maass. State-dependent computations: spatiotemporal processing in cortical networks. *Nature Reviews. Neuroscience*, 10(2):113–125, 2009.

[15] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher. Computational capacity and energy consumption of complex resistive switch networks. *AIMS Materials Science*, 2(4):530–545, 2015.

[16] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher. Hierarchical Composition of Memristive Networks for Real-Time Computing. In *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pages 33–38, July 2015.

[17] J. Bürger and C. Teuscher. Variation-tolerant Computing with Memristive Reservoirs. In *Nanoscale Architectures (NANOARCH), 2013 IEEE/ACM International Symposium on*, pages 1–6, July 2013.

[18] J. Bürger and C. Teuscher. Volatile Memristive Devices as Short-Term Memory in a Neuromorphic Learning Architecture. In *Nanoscale Architectures (NANOARCH), 2014 IEEE/ACM International Symposium on*, pages 104–109, July 2014.

[19] J. P. Carbajal, J. Dambre, M. Hermans, and B. Schrauwen. Memristor Models for Machine Learning. *Neural Computation*, 27(3):725–747, 2015.

[20] T. Chang, S.-H. Jo, K.-H. Kim, P. Sheridan, S. Gaba, and W. Lu. Synaptic behaviors and modeling of a metal oxide memristive device. *Applied Physics A*, 102(4):857–863, 2011.

[21] T. Chang, S.-H. Jo, and W. Lu. Short-term memory to long-term memory transition in a nanoscale memristor. *ACS Nano*, 5(9):7669–76, 2011.

[22] T. Chang, Y. Yang, and W. Lu. Building Neuromorphic Circuits with Memristive Devices. *Circuits and Systems Magazine, IEEE*, 13(2):56–73, 2013.

[23] Y. Chen, G.-Y. Jung, D. a. a. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. a. Nielsen, J. F. Stoddart, and R. S. Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14(4):462–468, apr 2003.

[24] L. Chua. Memristor - The missing circuit element. *Circuit Theory, IEEE Transactions on*, 18(5):507–519, 1971.

[25] P. Coulibaly. Reservoir Computing approach to Great Lakes water level forecasting. *Journal of Hydrology*, 381(1-2):76–88, 2010.

[26] J. P. Crutchfield, W. L. Ditto, and S. Sinha. Introduction to focus issue: intrinsic and designed computation: information processing in dynamical systems–beyond the digital hegemony. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(3):037101, 2010.

[27] E. C. Demis, R. Aguilera, H. O. Sillin, K. Scharnhorst, E. J. Sandouk, M. Aono, A. Z. Stieg, and J. K. Gimzewski. Atomic switch networks – nanoarchitectonic design of a complex system for natural computing. *Nanotechnology*, 26(20):204003, 2015.

[28] L. Deng, G. Li, N. Deng, D. Wang, Z. Zhang, W. He, H. Li, J. Pei, and L. Shi. Complex Learning in Bio-plausible Memristive Networks. *Scientific Reports*, 5:1717–1724, 2015.

[29] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, April 2006.

[30] T. Driscoll, Y. V. Pershin, D. N. Basov, and M. Di Ventra. Chaotic memristor. *Applied Physics A*, 102(4):885–889, 2011.

[31] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. *SIGARCH Comput. Archit. News*, 39(3):365–376, 2011.

[32] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu. Stochastic memristive devices for computing and neuromorphic applications. *Nanoscale*, 5(13):5872–5878, 2013.

[33] W. Gerstner, R. Kempter, J. Van Hemmen, and H. Wagner. A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78, 1996.

[34] A. Ghofrani, M. A. Lastras-Montaño, and K. T. Cheng. A Model for Variation- and Fault-Tolerant Digital Logic using Self-Assembled Nanowire Architectures. In *Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 116–121. IEEE Press, 2014.

[35] V. Goudar and D. V. Buonomano. A model of order-selectivity based on dynamic changes in the balance of excitation and inhibition produced by short-term synaptic plasticity. *Journal of Neurophysiology*, 113(2):509–523, 2015.

[36] B. Govoreanu, G. Kar, Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo,

P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. Wouters, J. Kittl, and M. Jurczak. 10x10nm 2 Hf / HfO x Crossbar Resistive RAM with Excellent Performance, Reliability and Low-Energy Operation. In *Electron Devices Meeting (IEDM), 2011 IEEE International*, pages 31.6.1–31.6.4, 2011.

[37] S. Haeusler and W. Maass. A Statistical Analysis of Information-Processing Properties of Lamina-Specific Cortical Microcircuit Models. *Cerebral Cortex*, 17(1):149–162, 2007.

[38] T. Hasegawa, A. Nayak, T. Ohno, K. Terabe, T. Tsuruoka, J. K. Gimzewski, and M. Aono. Memristive operations demonstrated by gap-type atomic switches. *Applied Physics A*, 102:811–815, 2011.

[39] J. Hasler and B. Marr. Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in neuroscience*, 7:118, 2013.

[40] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology. *Science*, 280(5370):1716–1721, 1998.

[41] M. H. Hennig. Theoretical models of synaptic short term plasticity. *Frontiers in Computational Neuroscience*, 7(45), 2013.

[42] M. Hermans and B. Schrauwen. Memory in linear recurrent neural networks in continuous time. *Neural Networks*, 23(3):341–355, 2010.

[43] J. Hermiz, T. Chang, C. Du, and W. Lu. Interference and memory capacity effects in memristive systems. *Appl. Phys. Lett.*, 102(8):083106, 2013.

[44] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501, 2006.

[45] G. Indiveri and S. Fusi. Spike-based learning in VLSI networks of integrate-and-fire neurons. In *2007 IEEE International Symposium on Circuits and Systems*, pages 3371–3374, May 2007.

[46] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5(73):1–23, 2011.

[47] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology*, 24(38):384010, 2013.

[48] H. Jaeger. The ‘‘echo state’’ approach to analysing and training recurrent neural networks. GMD Report 148, German National Research Center for Information Technology, 2001.

[49] S. H. Jo, K.-H. Kim, and W. Lu. High-Density Crossbar Arrays Based on a Si Memristive System. *Nano Letters*, 9(2):870–874, 2009.

[50] S. Johnson. *Iterative Error Correction: Turbo, Low-Density Parity-Check and Repeat-Accumulate Codes*. Iterative error correction: turbo, low-density parity-check and repeat-accumulate codes. Cambridge University Press, Pittsburgh, PA, USA, 2009.

[51] S. Johnson, J. Marro, and J. J. Torres. Robust Short-Term Memory without Synaptic Learning. *PLoS ONE*, 8(1):e50276, 2013.

[52] W. B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[53] M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber. Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2849–2856, June 2008.

[54] A. Khiat, I. Salaoru, and T. Prodromakis. Resistive switching characteristics of indium-tin-oxide thin film devices. *physica status solidi (a)*, 211(5):1194–1199, 2014.

[55] J. Kim. *Two-dimensional Memory System Protection*. PhD thesis, Pittsburgh, PA, USA, 2008. AAI3326636.

[56] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 197–209, Washington, DC, USA, 2007. IEEE Computer Society.

[57] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu. A Functional Hybrid Memristor Crossbar-Array/CMOS System for Data Storage and Neuromorphic Applications. *Nano letters*, 12(1):389–395, 2012.

[58] K.-H. Kim, S. Hyun Jo, S. Gaba, and W. Lu. Nanoscale resistive memory with intrinsic diode characteristics and long endurance. *Applied Physics Letters*, 96(5):053106, 2010.

[59] S. Kim, S. Choi, J. Lee, and W. D. Lu. Tuning Resistive Switching Characteristics of Tantalum Oxide Memristors through Si Doping. *ACS Nano*, 8(10):10262–10269, 2014.

[60] S. Kim, C. Du, P. Sheridan, W. Ma, S. Choi, and W. D. Lu. Experimental Demonstration of a Second-Order Memristor and Its Ability to Biorealistically Implement Synaptic Plasticity. *Nano Letters*, 15(3):2203–2211, 2015.

[61] D. Kudithipudi, Q. Saleh, C. Merkel, J. Thesing, and B. Wysocki. Design and Analysis of a Neuromemristive Reservoir Computing Architecture for Biosignal Processing. *Frontiers in Neuroscience*, 9(502), 2016.

[62] M. S. Kulkarni and C. Teuscher. Memristor-based Reservoir Computing. In *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on*, pages 226–232. IEEE, 2012.

[63] C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1-3):12–37, 1990.

[64] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[65] E. Lehtonen, J. Poikonen, and M. Laiho. Two memristors suffice to compute all Boolean functions. *Electronics Letters*, 46(3):239–240, 2010.

[66] K. K. Likharev and D. B. Strukov. CMOL: Devices, Circuits, and Architectures. In *Introducing Molecular Electronics*, volume 680, pages 447–477. Springer-Verlag Berlin Heidelberg, 2005.

[67] E. Linn, R. Rosezin, C. Kügeler, and R. Waser. Complementary resistive switches for passive nanocrossbar memories. *Nature Materials*, 9(5):403–6, may 2010.

[68] V. Litovski and M. Zwolinski. *VLSI Circuit Simulation and Optimization*. Chapman & Hall, London, UK, 1997.

[69] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[70] M. Lukoševičius, H. Jaeger, and B. Schrauwen. Reservoir computing trends. *KI - Künstliche Intelligenz*, 26(4):365–371, 2012.

[71] W. Ma, L. Chen, C. Du, and W. D. Lu. Temporal Information Encoding in Dynamic Memristive Devices. *Applied Physics Letters*, 107(19), 2015.

[72] W. Maass. Motivation, theory, and applications of liquid state machines. In B. Cooper and A. Sorbi, editors, *Computability in Context: Computation and Logic in the Real World*, pages 275–296. Imperial College Press, Imperial College Press, 2011.

[73] W. Maass, R. Legenstein, N. Bertschinger, and T. U. Graz. Methods for Estimating the Computational Power and Generalization Capability of Neural Microcircuits. In *Advances in Neural Information Processing Systems*, pages 865–872. MIT Press, 2005.

[74] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–2560, Nov. 2002.

[75] M. D. McDonnell, M. D. Tissera, T. Vladusich, A. van Schaik, and J. Tapson. Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the extreme learning machine algorithm. *PLoS ONE*, 10(8):1–20, 2015.

[76] C. Merkel, Q. Saleh, C. Donahue, and D. Kudithipudi. Memristive Reservoir Computing Architecture for Epileptic Seizure Detection. *Procedia Computer Science*, 41:249–254, 2014.

[77] F. Merrikh Bayat, B. Hoskins, and D. Strukov. Phenomenological modeling of memristive devices. *Applied Physics A*, 118(3):779–786, 2015.

[78] T. Natschläger, W. Maass, and H. Markram. The "liquid computer": A novel strategy for real-time computing on time series. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8(1):39–43, 2002.

[79] T. Ohno, T. Hasegawa, A. Nayak, T. Tsuruoka, J. K. Gimzewski, and M. Aono. Sensory and short-term memory formations observed in a Ag2S gap-type atomic switch. *Applied Physics Letters*, 99(20):1–3, 2011.

[80] T. Ohno, T. Hasegawa, T. Tsuruoka, K. Terabe, and J. K. Gimzewski. Short-term plasticity and long-term potentiation mimicked in single inorganic synapses. *Nature Materials*, 10(8):591–595, 2011.

[81] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[82] L. Qu, Y. Akbergenova, Y. Hu, and T. Schikorski. Synapse-to-Synapse Variation in Mean Synaptic Vesicle Size and Its Relationship with Synaptic Morphology and Function. *The Journal of Comparative Neurology*, 514(4):343–352, 2009.

[83] D. Querlioz, O. Bichler, and C. Gamrat. Simulation of a memristor-based spiking neural network immune to device variations. *2011 Int. Jt. Conf. Neural Networks*, pages 1775–1781, 2011.

[84] D. Querlioz, W. S. Zhao, P. Dollfus, J. Klein, and Q. G. O. Rqgdphqwdoh. Bioinspired Networks with Nanoscale Memristive Devices that Combine the Unsupervised and Supervised Learning Approaches. In *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on*, pages 203–210. IEEE, 2012.

[85] M. Rabinovich, R. Huerta, and G. Laurent. Transient dynamics for neural processing. *Science*, 321(5885):48–50, 2008.

[86] A. Radwan, M. Zidan, and K. Salama. On the Mathematical Modeling of Memristors. In *Microelectronics (ICM), 2010 International Conference on*, pages 284–287, 2010.

[87] M. Rigotti, O. Barak, M. R. Warden, X.-J. Wang, N. D. Daw, E. K. Miller, and S. Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–90, 2013.

[88] M. Rigotti, D. B. D. Rubin, X.-J. Wang, and S. Fusi. Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses. *Frontiers in Computational Neuroscience*, 4(24):1–29, 2010.

[89] A. Rodan and P. Tino. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–44, 2011.

[90] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, 65(6):386–408, 1958.

[91] A. Roxin and S. Fusi. Efficient partitioning of memory systems and its importance for memory consolidation. *PLoS Computational Biology*, 9(7):e1003146, 2013.

[92] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[93] S. Saïghi, C. G. Mayr, T. Serrano-Gotarredona, H. Schmidt, G. Lecerf, J. Tomas, J. Grollier, S. Boyn, A. F. Vincent, D. Querlioz, S. La Barbera, F. Alibart, D. Vuillaume, O. Bichler, C. Gamrat, and B. Linares-Barranco. Plasticity in memristive devices for spiking neural networks. *Frontiers in Neuroscience*, 9(51):1–16, 2015.

[94] M. Salmen and P. G. Ploger. Echo State Networks used for Motor Control. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1953–1958, April 2005.

[95] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco. STDP and STDP variations with memristors for spiking neuromorphic learning systems. *Frontiers in Neuroscience*, 7(2):1–15, 2013.

[96] H. O. Sillin, R. Aguilera, H.-H. Shieh, A. V. Avizienis, M. Aono, A. Z. Stieg, and J. K. Gimzewski. A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing. *Nanotechnology*, 24(38):384004, 2013.

[97] D. Snyder, A. Goudarzi, and C. Teuscher. Computational capabilities of random automata networks for reservoir computing. *Phys. Rev. E*, 87:042808, Apr 2013.

[98] A. Z. Stieg, A. V. Avizienis, H. O. Sillin, C. Martin-Olmos, M. Aono, and J. K. Gimzewski. Emergent Criticality in Complex Turing B-Type Atomic Switch Networks. *Advanced Materials*, 24(2):286–293, 2012.

[99] A. Z. Stieg, A. V. Avizienis, H. O. Sillin, C. Martin-olmos, M.-l. Lam, M. Aono, and J. K. Gimzewski. Self-organized atomic switch networks. *Japanese Journal of Applied Physics*, 53(01AA02):0–6, 2014.

[100] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[101] D. B. Strukov and R. S. Williams. Exponential ionic drift: fast switching and low volatility of thin-film memristors. *Applied Physics A*, 94(3):515–519, 2009.

[102] D. Sussillo, T. Toyoizumi, and W. Maass. Self-Tuning of Neural Circuits Through Short-Term Synaptic Plasticity. *Journal of Neurophysiology*, 97(6):4079–4095, 2007.

[103] M. Talagrand. A New Look at Independence. *The Annals of Probability*, 24(1):1–34, 1996.

[104] J. C. Tapson, G. K. Cohen, S. Afshar, K. M. Stiefel, Y. Buskila, R. M. Wang, T. J. Hamilton, and A. van Schaik. Synthesis of neural networks for spatio-

temporal spike pattern recognition and processing. *Frontiers in Neuroscience*, 7(August):153, 2013.

[105] F. Triefenbach, K. Demuynck, and J. P. Martens. Large Vocabulary Continuous Speech Recognition With Reservoir-Based Acoustic Models. *IEEE Signal Processing Letters*, 21(3):311–315, March 2014.

[106] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J.-P. Martens. Phoneme Recognition with Large Hierarchical Reservoirs. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2307–2315. Curran Associates, Inc., 2010.

[107] A. van Schaik. Building blocks for electronic spiking neural networks. *Neural Networks*, 14(6-7):617–28, 2001.

[108] D. Verstraeten, B. Schrauwen, S. Dieleman, P. Brakel, P. Buteneers, and D. Pecevski. Oger: Modular Learning Architectures For Large-Scale Sequential Processing. *Journal of Machine Learning Research*, 13:2995–2998, 2012.

[109] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. V. Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528, 2005.

[110] J. D. Victor and K. P. Purpura. Nature and precision of temporal coding in visual cortex: a metric-space analysis. *Journal of neurophysiology*, 76(2):1310–1326, 1996.

[111] R. Wang, G. Cohen, T. Hamilton, J. Tapson, and A. van Schaik. An improved aVLSI axon with programmable delay using spike timing dependent delay plas-

ticity. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1592–1595, 2013.

[112] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Texts in Applied Mathematics. Springer New York, 2003.

[113] W. Woods, J. Bürger, and C. Teuscher. On the Influence of Synaptic Weight States in a Locally Competitive Algorithm for Memristive Hardware. In *Nanoscale Architectures (NANOARCH), 2014 IEEE/ACM International Symposium on*, pages 19–24. IEEE, 2014.

[114] W. Woods, J. Bürger, and C. Teuscher. Synaptic Weight States in a Locally Competitive Algorithm for Neuromorphic Memristive Hardware. *IEEE Transactions on Nanotechnology*, 14(6):945–953, Nov 2015.

[115] W. Woods, M. M. A. Taha, S. J. D. Tran, J. Bürger, and C. Teuscher. Memristor panic - A survey of different device models in crossbar architectures. In *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pages 106–111, July 2015.

[116] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams. Memristive switching mechanism for metal/oxide/metal nanodevices. *Nature nanotechnology*, 3(7):429–433, jul 2008.

[117] C. Zamarreño Ramos, L. a. Camuñas Mesa, J. a. Pérez-Carrasco, T. Masquelier, T. Serrano-Gotarredona, and B. Linares-Barranco. On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in Neuroscience*, 5(26):1–22, 2011.