Gabriel Haas

A Model for Stochastic Neural Computation and Learning with Memristive Devices

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur
Master's degree programme: Telematics

submitted to

Graz University of Technology

Institute for Theoretical Computer Science (IGI)
Inffeldgasse 16b/I, 8010 Graz

Thesis Advisor
Assoc.Prof. Dipl.-Ing. Dr.techn. Robert Legenstein

Graz, August 2015

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz,	August 20, 2015	
	Date	Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am	20. August 2015	
	Datum	Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

Recently, neuroscience in general and the field of neuromorphic engineering in particular receive an increasing attention. Although some of the concepts, such as learning within artificial spiking neural networks based on plastic synapses, are well understood from a theoretical perspective, implementing the corresponding models in hardware has proven challenging. This is the case, since, when implemented based on classic CMOS technology, the involved complex mathematical model descriptions at least partially conflict with tight constraints imposed on both chip area as well as energy consumption. As a result, a persistent interest for alternative substrates to base implementations of neuromorphic hardware circuits upon has been ever-present.

With the discovery of memristive behavior within nano-scale electronic devices, recently a promising candidate for these alternative substrates was identified. Based on this, the presented work seeks to capture and model plasticity within memristive synapses as well as related topics at different levels of abstraction, reaching from the detailed device level up to an abstract machine learning perspective.

In computer simulations, the most important properties are identified, which a given memristor model has to offer, in order to render it a potential candidate foundation to build a memristive synapse implementation upon. Moreover, as turns out in final simulations at network level, spiking winner-take-all (WTA) networks of voltage-based memristive synapses are indeed capable of solving demanding tasks such as the autonomous classification of handwritten digits.

Kurzfassung

Sowohl für das Fachgebiet der Neurowissenschaften im Allgemeinen, als auch das Teilgebiet des neuromorphic Engineerings im Speziellen, ist in der näheren Vergangenheit ein steigendes Maß an Interesse zu beobachten. Obwohl komplexe Themengebiete, wie etwa das maschinelle Lernen in künstlichen neuronalen Netzwerken mit plastischen Synapsen, in theoretischer Hinsicht bereits sehr gut verstanden werden, stellt die Umsetzung ebensolcher Netzwerke in neuromorphen Schaltungen nach wie vor eine große Herausforderung dar, da beispielsweise starke Einschränkungen in Hinblick sowohl auf die nutzbare Chipfläche als auch die Leistungsaufnahme oft im Widerspruch zur Komplexität entsprechender auf CMOS-Technologie basierender Schaltungen stehen. Um dem Rechnung zu tragen, besteht seit jeher Interesse an alternativen Grundmaterialien für die Umsetzung neuromorpher Schaltungen.

Die Entdeckung von memristivem Verhalten in kleinsten elektronischen Bauteilen förderte in den letzten Jahren schlussendlich eine vielversprechende Grundlagen für künstliche plastische Synapsen zu Tage. Basierend darauf wird in der vorliegenden Arbeit versucht, synaptische Platizität in künstlichen memristiven Synapsen auf unterschiedlichem Abstraktionsniveau zu erfassen und zu modellieren.

Dazu werden in Computersimulationen die wichtigsten Eigenschaften identifiziert, die von Memristormodellen erfüllt werden müssen. Wie anschließende Simulationen auf Netzwerkebene zeigen, können memristive spikende Winner-Take-All (WTA) Netzwerke anspruchsvolle Aufgaben, wie die autonome Klassifizierung von handgeschriebenen Ziffern, lösen.

Acknowledgements

First and foremost, I want to thank my advisor Robert Legenstein for enabling this thesis as part of the PNEUMA Project as well as for the great ideas and the incredible expertise he contributed to this work.

Secondly, I am deeply indebted to Johannes Bill for the countless hours of valuable discussion he spent with me. Without his unyielding patience and ever undiluted view for the big picture, this thesis would not have been possible.

Next, I want to thank Mario Prantl and Harald Wesenjak, two of my former teachers at HTL Innsbruck Anichstraße, who, through the excellent education they bestowed upon me, undoubtedly to a large extent laid the academic foundation for my studies in Graz.

Moreover, I want to thank all my fellow students, especially Tim Hell and Markus Mayerwieser, for the fruitful and pleasant time we spent together at university.

In addition to this, I also want to thank my family and friends, especially my sister Theresa and my girlfriend Marie-Theres, for their support, their patience and the solid foundation they provided me to rest my educational efforts upon.

Last but not least, I want to express my deepest gratitude to my parents Elisabeth and Robert for raising me in an education oriented environment, enabling my studies in Graz and especially for the loving care and support I received from them over all these years – danke Mama, danke Papa!

Contents

List of Figures

List of Tables

1 Introduction

Over the past few years, the field of neuroscience has been experiencing an increasing attention. A growing interest can in particular be observed for the field of neuromorphic engineering. This field tries to capture neural behavior as can be observed in biological computational units such as neurons and synapses and attempts to model and replicate this behavior, for instance, in dedicated hardware circuits.

A typical property all these hardware implementations have in common is the extensive demand for complex support circuitry [BillLegenstein, 2014]. Among other things, this support circuitry is mainly responsible for handling the communication between the constituent neural entities as well as the implementation of appropriate synaptic plasticity rules. Due to tight constraints on both energy consumption as well as plain chip space, this demand bears one of the biggest challenges most circuit implementations of state-of-the-art proposals of neuromorphic hardware models face. This is the case, since especially the implementation of a biologically plausible synaptic plasticity rule supporting a decent form of weight dependence involves complex mathematical operations. Mapping these operations to classic CMOS-based technology is difficult and requires a considerable amount of chip area per synapse [Fieres, 2008]. Practically relevant neuromorphic chips, however, are supposed to be small in size and still offer a couple of million of plastic synapses, since most spiking neural network motifs connect their respective input and output neurons in an all-to-all fashion.

With the recent discovery of memristive behavior within nano-scale devices, however, a promising substrate for the implementation of plastic synapses in hardware was found, which is believed to elegantly satisfy the above-mentioned constraints. Memristors and memristive devices in general are circuit elements whose electric resistance is changed persistently based on the physical history the device has experienced. This physical history includes, for instance, the time course of the input current that flew through the memristive device or the voltage that was applied across its terminals as driving force behind non-volatile resistance changes. Besides their extremely small size, this intrinsic property of persistent and history-dependent state changes as are required by the principles of synaptic plasticity, renders memristive devices promising candidates for future hardware implementations of plastic synapses for artificial neural networks. Evidence which supports this perception was provided by Linares-Barranco and Serrano-Gotarredona [Linares, 2009], who showed that the concept of memristance can basically explain spike time dependent plasticity (STDP), a powerful and biologically plausible plasticity rule, within biologic neural synapses.

Instead of investigating biological synapses and neurons, the focus of this work lies on artificial, biologically inspired replicas of neural structures. Specifically, we focus on spiking winner-take-all (WTA) networks, a common motif for artificial neural networks, based on plastic synapses that adhere a weight-dependent STDP rule. Unsupervised learning within these spiking WTA networks is well understood from a machine learning perspective as it can be explained through online Expectation Maximization [Nessler, 2013]. While a link between the two fields of unsupervised learning within spiking WTA networks and memristive synapses was already established by Bill and Legenstein [BillLegenstein, 2014], a joint investigation of the constituent building blocks including a detailed view on the utilized memristive synapses at circuit-level has been lacking so far. This detailed and joint investigation is contributed by this work.

More precisely, this work tries to capture plasticity within memristive synapses at different levels of abstraction, reaching from the low-level circuit description of the internal dynamics of a voltage-based memristor up to the high-level behavior of the resulting spiking neural circuit which can be described by means of probability and machine learning theory. By doing so, this work establishes a further intensification of the above mentioned link between memristors, neural networks and the backing learning theory and identifies some of the key components the overall network setup has to offer. In addition to this, in final computer experiments, the proposed memristive synapses are combined with stochastically spiking neurons and assembled to a spiking WTA network circuit. As turns out in these final computer simulations at network level, spiking WTA networks based on the proposed memristive synapses are indeed capable of solving complex tasks such as the autonomous classification of images of handwritten digits.

In the following chapters, we will first present a review of the theoretical background behind memristors in general as well as the employed learning theory (Chapter 2). This review is followed by a documentation of the software and simulation framework which was created for this work (Chapter 3). Chapter 4 presents a comprehensive summary and description of different problems and implications identified during the process of studying a candidate memristor model according to is suitability for the implementation of memristive synapses. This model is fine-tuned and used as foundation for network level simulations in Chapter 5. Finally, a discussion of the presented work and the different results is given in Chapter 6. Appendices A to C provide additional material, which accompanies the previous chapters and explains some of the specifics in greater detail.

2 Background

This chapter is devoted to the discussion of the theoretical background behind the two main components this work is based on. These components include memristors in general as well as the underlying learning theory. We start with a short historical outline, continue with the introduction of two different memristor models as well as a short review of the backing learning theory and conclude by restating the goals of this work in the light of this theory background.

2.1 Historical Outline

Speaking of passive circuit elements, one usually thinks of resistors, capacitors and inductors. At least this was the common conception for almost one century after the postulation of Maxwell's famous equations [Johnsen, 2012]. This conception, however, began to totter in the early

1970s, when Leon Chua argued that there should be a fourth and equally fundamental circuit element. Back then, Chua had studied the relations between the four axiomatic circuit variables (that is, the voltage v, the current i, the charge q and the flux φ) and had observed that not all of their possible combinations had led to established relations yet.

One of the possible relations, the one between the charge q and the flux φ, was still undefined. Chua justified the relevance of this missing relation from both a logical point of view demanding symmetry as well as for the sake of completeness. He described the circuit element implementing the missing relation as a device, whose resistance depends on the history it has experienced. More precisely, he postulated this fourth fundamental circuit element to act as a resistor with memory in that it was capable of establishing a link between the physical state of the device in terms of the amount of charge that had flown in a specific direction and its resistance experienced by external circuitry. This intrinsic property of acting as a **mem**ory **res**i**stor** is what Chua derived his postulated device's name *memristor* from [Chua, 1971].

Initially, however, due to the absence of real-world examples and observations, memristive devices received only little attention. In fact, it was not until almost four decades later, that the first appearance of memristive behavior in hardware was documented. More precisely, with the emerge of nano-scale electronics, devices showing evidence for the above-mentioned behavior of the resistance changing persistently as different inputs are being applied to the device, were observed.

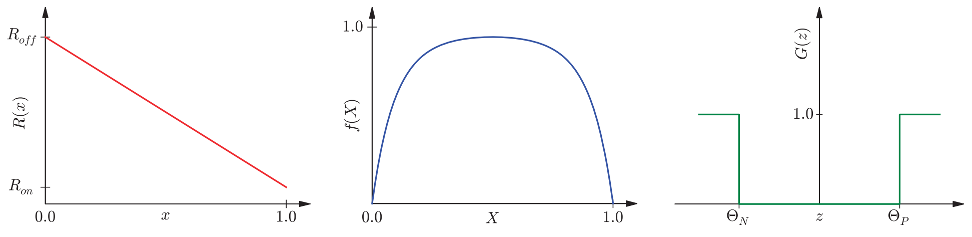
As turned out in the recent past, memristors are a useful means for modeling various processes in biology and neuroscience [Johnsen, 2012]. Spike time dependent plasticity in biological neural synapses, for instance, as was already indicated in the introduction, can be explained through memristive effects [Linares, 2009]. Consequently, it seems natural to try and utilize the now existing memristors to build synapses for neuromorphic hardware implementations of artificial neural networks.

Obviously, explaining plasticity and especially learning within artificial as well as biologic neural networks in a theoretical manner, requires an appropriate learning theory framework. One of the first and probably most famous learning rules ever proposed in this context (originally for biological neural networks) is the simple rule *"What fires together, wires together"* suggested by Hebb [Hebb, 1949]. Later, inspired by observations made with biologic synapses, this rule was refined to the concept of spike time dependent plasticity (STDP). In contrast to Hebbian learning, STDP explicitly demands a certain cause-effect relationship between the pre- and post-synaptic activity in order for the respective synaptic plasticity reactions, that is, increasing or decreasing the efficacy of the respective synapses, to be triggered. To do so, STDP focuses on the exact spike times of the pre- and post-synaptic neurons [DanPoo, 2004].

Based on this general and more historically inspired overview, the two main building blocks this work is based on shall now be discussed more thoroughly. In the following two sections we will first investigate different mathematical models describing the behavior of memristive hardware devices and then continue with a short review of the employed learning theory.

2.2 Memristor Models

Over the past few years, various formal descriptions for memristive devices were proposed. Generally speaking, depending on which of the circuit variables is regarded as being responsible for triggering updates to the internal state of the memristor (and thus ultimately the device's input/output behavior), one can basically distinguish voltage-based and current-based models. In this section, we are going to introduce a representative for both of them. We will also consider implications and limitations which are intrinsic to and arise directly from the respective architectures.



(a) $R(x)$ (b) $f(X)$ (c) $G(z)$

Figure 2.1: State mapping and parameter functions. In the current-based memristor model by Berdan et al. [Berdan, 2014], $R(\cdot)$ establishes a linear mapping between the internal state variable x and the resistance of the device as encountered by external circuitry. $f(\cdot)$ confines the values of x and y to the unity interval and $G(\cdot)$ introduces thresholds Θ_N and Θ_P which need to be exceeded in order for non-volatile changes to be enabled.

2.2.1 Current-Based

The first memristor model that was investigated for this work was proposed by Berdan et al. [Berdan, 2014]. This memristor model is built upon a set of three internal state variables. The first one of these, x, determines the instantaneous resistance R of the device as

$$R(x) = x\,(R_\mathrm{on} - R_\mathrm{off}) + R_\mathrm{off}\,, \tag{2.1}$$

where R_on and R_off denote the resistances of the device when being in the on- (most conductive) and off-state (least conductive), respectively.

Following Ohm's law, at any instant of time, the current I_mem that flows through the device in response to a voltage V being applied across its terminals is given as

$$I_\mathrm{mem} = \frac{V}{R(x)}\,. \tag{2.2}$$

This mapping is illustrated in Figure 2.1a.

The exact value of x at any instant of time, in turn, can be decomposed into a volatile and a non-volatile portion. More precisely, on the one hand, a certain volatile fraction determined by the amount of current that flew through the device in the recent past is present. On the other hand, in case no input current I_mem according to Equation 2.2 is present, x decays

Name	Value	Name	Value
R_{on}	$1\,\Omega$	C_z	$1\,F$
R_{off}	$100\,k\Omega$	R_z	$2\,\Omega$
μ_v	$1 \cdot 10^{-9}\,\frac{m^2}{s\,V}$	C_y	$1\,F$
D	$20\,nm$	Θ_{P}	$1.65\,nV$
C_x	$0.32\,F$	Θ_{N}	$-1.65\,nV$
R_x	$20\,\Omega$	P	10

Table 2.1: Parameters for the memristor model by Berdan et al. [Berdan, 2014]. Some of these quantities are related to physical parameter such as the device geometry or its material, while others are merely a result of fitting the presented dynamic model to data obtained from measurements conducted on real memristive hardware devices.

exponentially towards y, the non-volatile second internal state variable. Consequently, y is responsible for capturing persistent changes in the device's resting resistance. This is accomplished through the help of the third state variable, z, which integrates the recent current flow I_{mem}. It is compared against a positive and a negative threshold and in case one of them is exceeded, y receives either a positive or a negative update.

Formally, the evolution of the three internal state variables is described by a system of three coupled differential equations:

$$C_x \frac{dx}{dt} = -\frac{x-y}{R_x} + I_0(x) \tag{2.3}$$

$$C_y \frac{dy}{dt} = G(z)\, I_0(y) \tag{2.4}$$

$$C_z \frac{dz}{dt} = -\frac{z}{R_z} + I_{\text{mem}}\,, \tag{2.5}$$

with capacitances C_x, C_y and C_z, which basically define the strength of the coupling between the respective internal state variables x, y and z and their governing quantities.

In the system given in Equations 2.3 to 2.5, the functions $I_0(\cdot)$ and $G(\cdot)$ are defined as

$$I_0(\xi) = \frac{R_{\text{on}}\,\mu_v}{D^2}\, f(\xi)\, I_{\text{mem}} \tag{2.6}$$

and

$$G(z) = H\left(I_{\mathrm{mem}}\right) \, H\left(z - \Theta_{\mathrm{P}}\right) + H\left(-I_{\mathrm{mem}}\right) \, H\left(-\left(z - \Theta_{\mathrm{N}}\right)\right) \, , \tag{2.7}$$

respectively, where $H(\cdot)$ denotes the Heaviside step function. The quantities D and μ_v used in Equation 2.6 are constant and related to the device's material and geometry. Typical values for these as well as all the other parameters used by the model are listed in Table 2.1.

The window function $f(\cdot)$, finally, is given as

$$f(\xi) = 1 - \left[\left(\xi - \frac{1}{2}\right)^2 + \frac{3}{4}\right]^P \, , \tag{2.8}$$

where P is a shape parameter which determines the sharpness of the window function's edges. While $f(\cdot)$ confines the values of the affected state variables to the unity interval $(0, 1)$, $G(\cdot)$ implements the above-mentioned thresholds Θ_{P} and Θ_{N} which need to be exceeded in order to allow for non-volatile changes. An illustration of these two functions is given in Figures 2.1b and 2.1c, respectively.

Having a closer look at the system definition given in Equations 2.3 to 2.5 reveals that the input current I_{mem} affects all three internal state variables. Consequently, the memristor model by Berdan et al. [Berdan, 2014] can be considered current-based in that the input current I_{mem} is the driving force behind all volatile and non-volatile state updates the device experiences. As turns out, however, this current-based architecture is problematic for the learning framework this work aims for.

Namely, Equation 2.5 introduces a complex feedback into the system. More precisely, $\frac{dz}{dt}$ depends on I_{mem} which itself depends on $\frac{1}{R(x)}$ through Equation 2.2. As a consequence, on the one hand, a device in a high resistive state will hardly undergo any non-volatile changes according to Equation 2.4. This is the case since the applied voltage V is insufficient to open the gate defined by $G(z)$, that is, due to their amplitudes not being capable of driving enough current through the device, z remains below the thresholds Θ_{N} and Θ_{P} for most pulses. On the other hand, a device in a low resistive state will, due to the strong current flow I_{mem}, exhibit non-volatile changes even for small voltages V. Since already in initial learning experiments this implied type of varying thresholds turned out to be very hard to control, the memristor model by Berdan et al. [Berdan, 2014] had to be ruled out as a possible candidate prematurely. Rather, a voltage-based architecture was identified as being favorable for the given learning framework.

2.2.2 Voltage-Based

Another memristor model, which is somewhat related to the one presented in the previous section, was recently proposed by Li et al. [Li, 2015]. To base our research on, we were kindly permitted premature access to an early draft version of this model by Serb et al. [Serb, 2014], our PNEUMA project partners at the University of Southampton. This is why we are going to describe the draft version, rather than the published one here. Nevertheless, compared to this model, the publicly published one by Li et al. [Li, 2015], anyway features merely some slight modification to some of the internal details, while the overall system behavior remains almost unchanged.

The memristor model by Serb et al. [Serb, 2014] splits up the system description into five different modules, each of which describes a specific aspect of the overall system behavior. According to the authors, this modular system design was chosen in order to increase the flexibility by allowing for certain modules to be replaced easily to match new types of memristive hardware as they emerge [Serb, 2014]. One of the main differences between this model and the one discussed in the previous section is that this model relies on a voltage-based architecture. As indicated in the previous section, memristive devices built on top of voltage-based architectures are potentially better suited for the implementation of memristive synapses in the learning setup utilized for this work. In combination with the modular setup, this rendered the memristor model by Serb et al. [Serb, 2014] a promising candidate to base this work on.

Name	Value	Name	Value	Name	Value
M_{min}	$1\,\Omega$	R_z	$3\,m\Omega$	U_-	$-5 \cdot 10^{-10}$
M_{max}	$100\,k\Omega$	C_w	$1\,F$	U_+	$5 \cdot 10^{-10}$
C_x	$5\,mF$	R_w	$600\,m\Omega$	α	0.5
R_x	$1\,\Omega$	k	10^6	K_p	0.7
C_y	$0.15\,F$	B_-	$-3.5 \cdot 10^{-10}$	K_n	1
C_z	$1\,F$	B_+	$3.5 \cdot 10^{-10}$	p	10^8

Table 2.2: Parameters for the memristor model by Serb et al. [Serb, 2014]. Some of the quantities have a physical interpretation while others rather result from fitting the presented model to data obtained from measurements performed on memristive hardware devices.

In the following, we will take a closer look at the different modules in order to gain some insight into the memristor model's internals. The parameters used by these modules are jointly summarized in Table 2.2.

Module I: Memristor Interface M

The first module defines the memristor's interface with respect to external circuitry:

$$i_M(t) = \frac{V(t)}{M(t)}, \tag{2.9}$$

where

$$M(t) = M_{\text{max}} - V_x(t)\,\Delta M \tag{2.10}$$

and

$$\Delta M = (M_{\text{max}} - M_{\text{min}}) \tag{2.11}$$

As such, Equation 2.9 describing the time course of the current i_M that flows through the memristor resembles a time-dependent version of Ohm's law: Not only the voltage V applied to the device but also its resistance M is a function of the time. More precisely, as is indicated by Equation 2.10, the instantaneous memristance M at time t encountered by a voltage V being applied across the terminals of the memristor, depends on the current value of some internal state variable V_x.

Since, as we shall see in the next section, the value of V_x is confined to the interval $(0, 1)$ and as a result of Equation 2.11, M_{min} and M_{max} are the minimum and maximum values the instantaneous memristance $M(t)$ according to Equation 2.10 can attain.

Module II: Volatile Changes V_x

Module II accounts for the volatile memristor dynamics, that is changes in the instantaneous memristance $M(t)$ which decay over time. The system equation for this module reads

$$C_x \frac{dV_x}{dt} = i_x(t) - \frac{V_x(t) - V_y(t)}{R_x}, \tag{2.12}$$

where

$$i_x(t) = k\, i_M(t)\, f\left(V_x\left(t\right)\right) \tag{2.13}$$

and

$$f(V_x) = \begin{cases} 1 & \text{if } 0 < V_x < 1 \\ 0 & \text{else} \end{cases}. \tag{2.14}$$

As can be seen from Equation 2.12, the driving force behind these short term changes is the imaginary current $i_x(t)$. This imaginary current has no physical equivalent in the system but is closely related to the voltage applied across the memristor's terminals as becomes clear from Equation 2.13. The constant k used in this equation captures the influence of both the device's geometry as well as its fabrication on the overall system. As indicated above, V_x is confined to $0 < V_x < 1$ by the window function $f(\cdot)$ defined in Equation 2.14. This holds, because as V_x approaches the bounds of the interval $(0,1)$, its updates according to Equation 2.12 are restricted to changes away from these bounds, towards inside the interval.

Since V_x is the only link of the memristor model's internals to the corresponding memristor interface defined in Module I, Module II is actually also responsible for implicitly forwarding the non-volatile memristance changes to the outside. This is accomplished by the second term of the right hand side in Equation 2.12, which makes V_x decay exponentially towards V_y if $i_x(t)$ is equal to zero, that is, in case no input is present. The time constant τ_x of this exponential decay is determined by the constants C_x and R_x. As we shall see later in Section 4.1, besides its contribution to the decay time τ_x, C_x also defines the strength of the coupling between V_x and the input voltage $V(t)$.

Module III: Non-Volatile Changes V_y

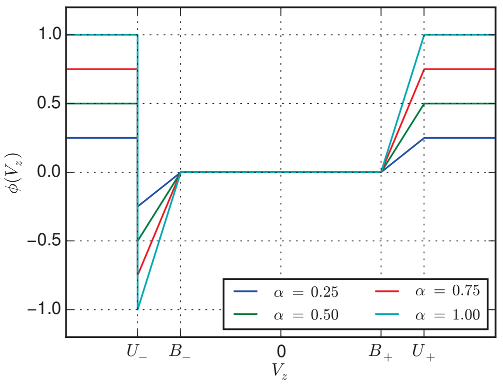
As indicated above, the purpose of Module III is capturing the non-volatile memristor dynamics, that is, long term changes in the memristor's resting memristance. For this task, Serb et al. [Serb, 2014] suggested the following system equation:

$$C_y \frac{dV_y}{dt} = i_y(t) = \phi(V_z, B_+, B_-, U_+, U_-)\, g(V_w)\, f(V_y)\,, \tag{2.15}$$

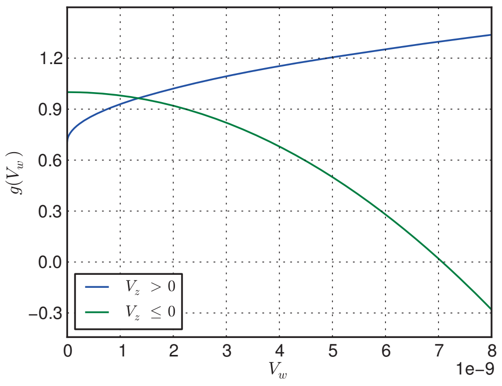
where

$$\phi(V_z, B_+, B_-, U_+, U_-) = \begin{cases} 0 & \text{if } V_z \in [B_-, B_+] \\ \alpha \frac{V_z - B_-}{U_+ - B_+} & \text{if } V_z \in [U_-, B_-) \\ \alpha \frac{V_z - B_+}{U_+ - B_+} & \text{if } V_z \in (B_+, U_+] \\ \alpha & \text{else} \end{cases}\,, \tag{2.16}$$

$$g(V_w) = \begin{cases} K_p \sqrt{p\, V_w} + 1 & \text{if } V_z > 0 \\ K_n \left[1 - 2\, (p\, V_w)^2\right] & \text{else} \end{cases} \tag{2.17}$$



(a) $\phi(V_z)$ for different values of α.



(b) $g(V_w)$ with its different branches.

Figure 2.2: Parameter functions $\phi(\cdot)$ and $g(\cdot)$. $\phi(\cdot)$ implements certain thresholds $B_\pm$ and $U_\pm$ which need to be exceeded in order to allow for non-volatile changes controlled by V_y. $g(\cdot)$ accounts for the activity dependence of the system. It is a purpose-built function that was fitted to data measured for biological synapses.

and $f(V_y)$ according to Equation 2.14. The other parameter functions $\phi(\cdot)$ and $g(\cdot)$ are illustrated in Figure 2.2. α, K_p and K_n used in their definitions in Equations 2.16 and 2.17 are scaling factors.

As can be seen from Equation 2.15, non-volatile changes in V_y (and thus the resting memristance) are only possible if all three functions $\phi(\cdot)$, $g(\cdot)$ and $f(\cdot)$ yield function values different from zero. Each of these parameter functions serves a different purpose. The first one, $\phi(\cdot)$, introduces two pairs of thresholds $B_\pm = \pm 3.5 \cdot 10^{-10}$ and $U_\pm = \pm 5 \cdot 10^{-10}$ which jointly specify the neutral, bipolar and unipolar regions of $\phi(\cdot)$. The driving effort V_z (see the next section) needs to exceed at least $B_\pm$ in order to enable non-volatile changes. Next, the parameter function $g(\cdot)$ accounts for the activity dependence of the non-volatile memristance changes. According to Serb et al. [Serb, 2014], rather than being directly related to memristive hardware, the exact shape of this function was fitted in order to match data observed for biological synapses. Finally, similar as in the previous section, the window function $f(\cdot)$ serves the purpose of confining the state variable V_y to the interval $(0, 1)$, forcing the instantaneous memristance M to be smaller than M_{max} and greater than M_{min}, respectively. The strength of the coupling between the product of the three parameter functions $\phi(\cdot)$, $g(\cdot)$ and $f(\cdot)$ and the non-volatile responses triggered in Module III is, as can be seen from Equation 2.15, determined by the constant C_y.

Module IV: Driving Effort V_z

The fourth module defined by Serb et al. [Serb, 2014] is responsible for integrating all the external inputs $V(t)$ applied across the memristor's terminals in order to obtain the driving effort V_z. This driving effort is ultimately responsible for triggering non-volatile memristance changes through the parameter function $\phi(\cdot)$ described in the previous section. The system equation for Module IV is as follows:

$$C_z \frac{dVz}{dt} = i_z(t) - \frac{V_z(t)}{R_z}, \tag{2.18}$$

where

$$i_z(t) = \frac{V(t)}{M_{\text{mid}}} \tag{2.19}$$

and

$$M_{\text{mid}} = \frac{M_{\text{max}} - M_{\text{min}}}{2}. \tag{2.20}$$

As can be seen from Equation 2.18, Module IV basically implements a low-pass filter as a leaky integrator. The input of this leaky integrator according to Equation 2.19 is also where the above-mentioned voltage-based nature of this memristor model becomes evident. Since both C_z as well as M_{mid} defined in Equation 2.20 are constants, the leaky integrator integrates a certain fraction of the input voltage $V(t)$. In case the applied input voltage $V(t)$ is zero, the leaky integrator implemented by C_z and R_z discharges with a time constant $\tau_z = C_z R_z$. Contrary to the memristor model by Berdan et al. [Berdan, 2014] described earlier, this makes the input voltage rather than the input current the driving force behind non-volatile changes in the resting memristance for this model.

Module V: Activity Dependence V_w

Module V, finally, integrates the absolute power dissipation through a leaky integration process defined by the following differential equation:

$$C_w \frac{dV_w}{dt} = |i_w(t)| - \frac{V_w(t)}{R_w} \tag{2.21}$$

where

$$i_w(t) = i(t)\, V(t) \tag{2.22}$$

Together with the purpose-built parameter function $g(\cdot)$ described above, the governing state variable V_w of this module is supposed to account for activity dependent effects introduced by Joule heating in Equation 2.15.

This concludes the discussion of the internals of different memristor models. As turned out, a voltage-based model is potentially better suited for the task of implementing memristive synapses for spiking WTA networks. A candidate memristor model was found in the proposal by Serb et al. [Serb, 2014]. In the following section, we will take a closer look at synaptic plasticity as well as the learning theory this memristor model is supposed mimic.

2.3 Plasticity and Learning Theory

As indicated in the introduction to this chapter, the second major building this work is based on is formed by synaptic plasticity as well as the SEM learning theory. Generally speaking, plasticity rules are a way of explaining how the individual synapses of a neural network are supposed to adapt the encoded synaptic weight in response to a certain type of neural activity within the network. Over the past few years, various different plasticity rules were proposed. As indicated in the introduction, one of the more famous ones is *spike timing dependent plasticity* (STDP for short). Besides being comparatively simple, STDP is experimentally well supported as well as considered plausible from a biological point of view. In the following, we will first introduce this plasticity rule and then establish a link to SEM, the learning theory assumed as foundation for this work.

2.3.1 Spike Time Dependent Plasticity

By itself, the basic STDP rule is simple. In a nutshell, for each of the synapses, it focuses on the respective pre- and the post-synaptic neurons only. The adaption of the weight encoded by the connecting synapse is then made according to the relative timing of the spikes these two neurons send through it. More precisely, for the STDP rule, one considers pairs of pre- and post-synaptic spikes and determines which of the two occurred first in terms of the exact absolute spike time [DanPoo, 2004]. According to this, one can basically distinguish the two cases

- *pre-synaptic neuron spiked before post-synaptic neuron* and
- *post-synaptic neuron spiked before pre-synaptic neuron*.

Instances of these cases are assumed to increase (*“potentiate”*) and decrease (*“depress”*) the weight of the affected synapse, respectively. For each of the observed spike pairs, the exact degree of potentiation and depression, that is the amount the encoded synaptic weight is incremented or decremented, depends on the timespan elapsed between the respective pre- and post-synaptic spikes. As a general rule, however, smaller timespans will lead to stronger reactions.

2.3.2 SEM Learning Theory

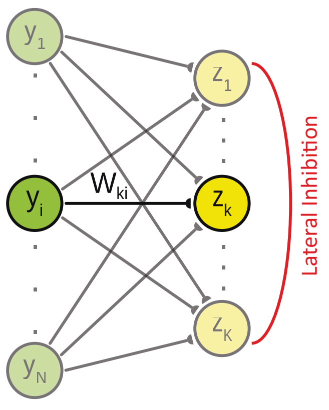
As indicated above, the role of STDP for the plastic adaption of the synaptic junctions within networks of neurons is well supported by findings from experimental biology. Still, the specific way how information is encoded and processed within these synapses as well as the exact influence of STDP on the plastic adjustment, remains largely unknown.

There exists, however, evidence that probabilistic inference is a key component of the underlying information processing machinery. More precisely, it is believed that neural networks maintain prior distributions as well as likelihood models for hidden causes in their network parameters. Among others, these network parameters include the strengths of the synaptic junctions between neurons, which are assumed to implicitly encode different likelihood models [Fiser, 2010]. During their operation, given some observed input, spiking neural networks then combine the priors with the likelihoods in order to obtain posterior distributions of hidden causes according to Bayes' theorem and, by doing so, infer the cluster an applied input vector belongs to. This, of course, raises the question how the prior and likelihood models can be acquired by networks of spiking neurons in a purely unsupervised manner.

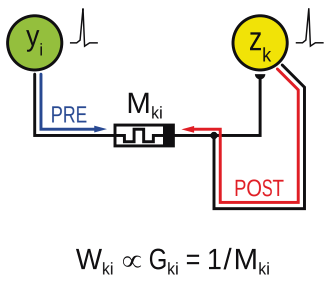
An explanation to this puzzle can be found in the *SEM* (Spike-based Expectation Maximization) learning theory proposed by Nessler et al. [Nessler, 2013]. Based on a weight dependent STDP rule, for which the adaption of a synapse's weight depends on its current weight, they showed that within spiking WTA networks the ongoing weight refinement process gives rise to an online approximation of Expectation Maximization. SEM assumes pre-synaptic spikes to trigger post-synaptic potentials (PSPs), pulses of a certain amplitude and a given duration τ. Post-synaptic neurons are assumed to react to these PSPs through point events, the post-synaptic spikes. Consequently, according to the SEM theory, one can basically distinguish four different cases:

- *PRE-pulse only,*
- *POST-spike only,*
- *POST-spike before PRE-pulse* and
- *PRE-pulse starts at most τ before POST-spike.*

In terms of the induced synaptic plasticity, the first one of the above cases is not supposed to trigger any reactions, while the second and the third ones are assumed to act depressing, that is, reduce the synaptic weight.



(a) WTA network model



(b) Memristive synapse

Figure 2.3: Illustration of the underlying network architecture. The spiking WTA network motif illustrated in (a) features N spiking input neurons y_i and K spiking network neurons z_k. The (pre-synaptic) input neurons are connected with the (post-synaptic) network neurons in an all-to-all fashion through plastic synapses with weights W_{ki}. Due to lateral inhibition, the network neurons are in competition to fire for a given input pattern. For this work, the connections between the neurons are implemented as memristive synapses as shown in (b). *PRE* and *POST* denote pulses of specific shape which are triggered by spikes of the respective neurons. Both illustrations are adapted from [BillLegenstein, 2014].

Finally, the fourth configuration shall result in a potentiation, that is, an increment of the affected synaptic efficacy.

2.4 Goals

Based on the theoretical foundation presented in the previous sections, the primary goals of this work can be specified more precisely. Given a spiking WTA network, this work aims to find a memristor model and, if necessary, additional building blocks required in order to approximate STDP- and SEM-like behavior in hardware. In addition to this, if an appropriate model is found, it shall be determined whether the resulting memristive WTA circuits are capable of learning and performing meaningful neural computation.

Figure 2.3a illustrates the spiking WTA network model used for this purpose. WTA networks, which are a common and well established spiking network motif, in general feature N spiking (pre-synaptic) input neurons y_i and K spiking (post-synaptic) network neurons z_k. Through lateral inhibition, only a single network neuron z_k can be active firing a spike at any instant of time. Consequently, the network neurons are in competition for firing in response to any given input. The input and the network neurons y_i and z_k, respectively, are connected in an all-to-all fashion through plastic synapses of a specific strength, the synaptic weights W_{ki}. Depending on the spiking activity of the neurons they connect, these plastic synapses are assumed to adapt their weight according to a weight-dependent STDP rule as well as the SEM learning theory.

Obviously, instead of using an idealized plastic synapse model, for this work we aim to use memristive synapses, which are, at every instant of time, assumed as establishing a relation between their instantaneous conductance G_{ki} and the synaptic weight W_{ki} they encode. This is illustrated in Figure 2.3b, where *PRE* and *POST* denote appropriate pulse shapes, capable of triggering different responses in terms of the induced memristance changes within the memristive synapses. These PRE- and POST-pulses are assumed as being triggered by the pre- and post-synaptic spikes. With regard to the post-synaptic events, this is, of course, a substantial deviation from the setup the SEM learning theory is built around, which assumes the events being emitted by the post-synaptic neurons as having infinitesimal small extent. Since this, however, is not practically feasible in a hardware implementation, for the purpose of this work we simply demand the POST-pulses as being short compared to the PRE-pulses. Finally, and in compliance with the SEM theory again, the spikes emitted by the constituent neurons are generated according to various stochastic process.

This concludes the discussion of the theoretical background and the motivation of this work. In the following chapter, we will have a closer look at the software framework which was created for this work.

3 Software

The memristor model described in Section 2.2.2 was originally implemented by Serb et al. as a SPICE circuit model being controlled by different MATLAB scripts. For this work, however, it was decided to re-implement the memristor model in Python[1]. This decision was motivated by Python being free and open-source software with a large variety of ready-to-use libraries for scientific computing as well as plotting being available. Moreover, using Python was found to increase the flexibility and extensibility of the simulation framework as well as to ensure easy integration into pre-existing code frameworks such as the implementation of a spiking WTA network.

[1] https://www.python.org/

3.1 Python Framework

As indicated above, flexibility and extensibility were two of the key design objectives considered for the memristor model's Python implementation written for this work. These objectives were met by choosing a modular framework design which allows for easy exchangeability of the different building blocks. More precisely, this modular design splits up and distributes the functionality of the memristor model among multiple entities including classes as well as modules and procedures. Since all entities of a given type (for instance pulse shapes or parameter sets) offer the same interface, one can switch from one implementation to another quite easily. In fact, most of the simulations can be configured using simple command line arguments which then select the appropriate entity implementations.

In the following we will take a brief look at some of the more important entity types. Class diagrams illustrating these entities as well as their relation to each other are presented in Section 3.1.6.

3.1.1 Memristors

The `Memristor` class shields the internals of the underlying memristor model away from other parts of the software framework. It offers various methods which allow other entities to interact with the implementation of the memristor model's dynamics. These methods are listed in Table 3.1 along with a short description of their functionality.

Currently, only one implementation, the one for the memristor model by Serb et al. [Serb, 2014] described in Section 2.2.2, exists of the `Memristor` class. In addition to providing the above-mentioned unified interface to other software components, in the chosen implementation the `Memristor` class is also responsible for keeping track of the different internal state variables defined in Serb et al.'s memristor model.

Name	**Function**
`Delta_g()`	Calculate and retrieve conductance change ΔG triggered by the preceding input stimulus.
`M()`	Retrieve current instantaneous memristance M.
`inject_state()`	Set internal state variables to specific values.
`recording()`	Retrieve recording of the evolution of the memristance as well as the memristor's internal state variables over time as triggered by the preceding input stimulus.
`state()`	Retrieve the current values of the memristor's internal state variables.
`update()`	Update memristor state according to an external input `V` which is assumed to be constant for a duration of `dt` seconds.

Table 3.1: Methods exposed by the `Memristor` class. The `Memristor` class is the main interface other entities can use in order to interact with the memristor model's dynamics. It is intended to shield away the internals of the underlying memristor model from other parts of the software framework.

3.1.2 Modules

As shown in Section 2.2.2, the memristor model defined by Serb et al. [Serb, 2014] is composed out of five modules. The purpose of most of these modules is solving differential equations describing different parts and functionalities of the memristor model. Since no closed form solution exists for most of these differential equations, in the software framework they have to be approximated by means of numerical procedures. During the course of this work, two different procedures for this purpose were evaluated (see Appendix A for details). In addition to this, the model itself was slightly modified. More precisely, the original window function used by one of differential equations was replaced for a different one (details follow in Section 4.4). Ultimately, in combination with the different numerical approximation procedures, this lead to various different versions of the modules. To account for this while ensuring flexibility for the simulations, different flavors of the various modules were implemented and grouped together in separate classes. The names of these classes all start with

Name	**Description**
`Modules_default`	Default implementation using plain Euler Integration.
`Modules_mult`	Implementation using multiplicative updates where applicable.
`Modules_mult_no_g`	Implementation as for `Modules_mult`, but with $g(V_w) \equiv 1$.
`Modules_mult_no_g_m0`	Identical to `Modules_mult_no_g`, but with additional pre-resistor M_0.
`Modules_mult_no_g_m0_fy`	Similar to `Modules_mult_no_g_m0`, but with alternative window function $\psi(\cdot)$ with $\varkappa_{\mathrm{P}} = \varkappa_{\mathrm{D}} = \varkappa$.
`Modules_mult_no_g_m0_2fy`	Identical to `Modules_mult_no_g_m0_fy`, but with full-blown window function $\psi(\cdot)$ with two different values for $\varkappa_{\mathrm{P}}$ and $\varkappa_{\mathrm{D}}$.

Table 3.2: List of module implementations (`Modules_xxx` classes). The memristor model introduced in Section 2.2.2 relies on five different modules. For the purpose of this work, different versions of these modules were implemented, which all inherit from the `Modules_default` class.

Name	**Function**
`f_x()`	Implementation of the window function $f(\cdot)$ according to Equation 2.14 as used by Module II.
`f_y()`	Implementation of the window function $f(\cdot)$ according to Equation 2.14 as used by Module III.
`Phi()`	Implementation of the parameter function $\phi(\cdot)$ according to Equation 2.16 as used by Module III.
`g()`	Implementation of the heating function $g(\cdot)$ according to Equation 2.17 as used by Module III.
`dV_x__dt()`	Implementation of the update rule for Module II according to Equation 2.12.
`dV_y__dt()`	Implementation of the update rule for Module III according to Equation 2.15.
`dV_z__dt()`	Implementation of the update rule for Module IV according to Equations 2.18 and 2.19.
`dV_w__dt()`	Implementation of the update rule for Module V according to Equation 2.21.
`update()`	Perform update of the state variables V_w, V_x, V_y and V_z for a given external input `V` which is assumed to be constant for a duration of `dt` seconds and return result.

Table 3.3: Methods and lambdas exposed by the different `Modules_xxx` classes. The different modules defined in the memristor model by Serb et al. [Serb, 2014] are built around a set of differential equations. In the software simulations, these are approximated by means of iterative numerical procedures. These procedures are implemented by the above methods and lambdas.

`Modules_` which is followed by abbreviations indicating the exact version the class implements. A full list of all these classes is given in Table 3.2 along with a short description.

Similar as the `Memristor` class described in the previous section, also the different `Modules_xxx` classes have a unified interface which other software components (that is, primarily the `Memristor` implementations) can use in order to interact with the modules in a standardized manner. These interactions include calls used to update the internal state variables in response to an external input as well as to retrieve the current memristance. Since some of the modules require additional parameter

Name	**Description**
`Params_default`	Default parameter set as listed in Table 2.2.
`Params_10_cx`	Parameter set for $C'_x = 10\,C_x$ and $R'_x = \frac{R_x}{10}$.
`Params_100_cx`	Parameter set for $C'_x = 100\,C_x$ and $R'_x = \frac{R_x}{100}$.
`Params_real`	Parameter set for realistic time constants $\tau_x = 100\,ms$ ($C_x = 500\,mF$ and $R_x = 200\,m\Omega$).

Table 3.4: List of parameter set implementations (`Params_xxx` classes). For the simulations conducted as part of this work, different parametrization for the modules defined by the memristor model were evaluated. In the software simulations, these parameterizations were implemented as separate `Params_xxx` classes all derived from `Params_default`.

functions, also the internal interface of the classes is unified in order to allow for single functions to be easily replaced by means of subclassing. The same applies to the functions used by the iterative update processes. All these methods and lambdas are listed in Table 3.3 together with a short description of their functionality.

3.1.3 Parameter Sets

In order to increase the flexibility, the values for the parameters used by the modules described in the previous section were not hard-coded in their implementations. Rather, different parameter sets were implemented as separate `Params_xxx` classes. Each of these classes represents a specific parameter configuration, such as the ones described in Section 4.1. When a new `Memristor` instance is created, one of these parameter set classes is selected and the underlying entity implementations retrieve the parameter values from it.

3.1.4 Initializers

Based on the parameter sets mentioned in the previous section as well as the modules described above, the memristor's internal state variables can be set to different initial states. In the software framework, this is

Name	Description
`init_max_g()`	Initialization to maximum conductance (same as `init_min_m()`).
`init_max_m()`	Initialization to maximum memristance.
`init_mid_g()`	Initialization to midrange conductance (without M_0).
`init_mid_g_m0()`	Initialization to midrange conductance (with M_0).
`init_mid_m()`	Initialization to midrange memristance.
`init_min_g()`	Initialization to minimum conductance (same as `init_max_m()`).
`init_min_m()`	Initialization to minimum memristance.
`init_frac_max_g_m0()`	Initialization to a certain fraction ρ of the maximum conductance (with M_0).

Table 3.5: List of initializer implementations (`init_xxx()` functions). Some of the simulations described throughout this work require the initialization of the memristor to a specific initial state defined by specific values for the internal state variables V_w, V_x, V_y and V_z. This task is accomplished by the above initializer functions.

accomplished by different initializer methods. These initializer methods are methods which determine the initial values for the memristor model's internal state variables V_w, V_x, V_y and V_z according to different aspects (for instance $M(t=0) \overset{!}{=} M_{\text{min}}$ or $G(t=0) \overset{!}{=} G_{\text{mid}}$) and return the corresponding values. See Table 3.5 for a summary of the different initializers and corresponding descriptions. A derivation of the different values the initializers assign to the state variables is presented in Appendix C.

3.1.5 Pulses

The framework entities mentioned in the previous sections were all part of the memristor model itself. During the course of this work, however, also different pulse shapes to be used in combination with the memristor model were developed. In order to allow for these pulse shapes to be easily plugged into various types of simulations, also the different pulse

Name	Description
d_pulse	Implementation of D-Pulses according to Section 4.3.1.
t_pulse	Implementation of T-Pulses according to Section 4.3.2.
s_pulse	Implementation of S-Pulses according to Section 4.3.3.

Table 3.6: List of pulse implementations (x_pulse classes). During the course of this work, different pulsing schemes were evaluated with respect to their suitability for the implementation of STDP within memristive synapses. In order to increase the flexibility, these pulsing schemes were implemented in separate classes.

Name	Function
post()	Retrieve the voltage the POST-pulse attains at a specific time t relative to the origin.
pre()	Retrieve the voltage the PRE-pulse attains at a specific time t relative to the origin.
sample_post()	Sample the POST-pulse at a given dt and return an array containing the resulting voltage train.
sample_pre()	Sample the PRE-pulse at a given dt and return an array containing the resulting voltage train.
T1()	Retrieve the overall duration of the PRE-pulse.
T2()	Retrieve the overall duration of the POST-pulse.

Table 3.7: Methods exposed by the x_pulse classes. Other parts of the software framework can rely on these methods as a defined interface in order to interact with pulse implementation according to different pulsing schemes.

types described in Section 4.3 were implemented as classes with a unified interface. In addition to the pulse implementations listed in Table 3.6, also a pulse class was implemented which all the other classes are derived from. This class offers the main interface consisting of the two methods get_name() and sample_pair() which other framework components such as the simulation scripts can call. The sample_pair() method, in turn, relies on some of the methods listed in Table 3.7 in order to sample PRE- and POST-pulses which are then combined to pulse pairs. The implementations of these methods are provided by the x_pulse classes. Although these methods are primarily intended to be called internally (for instance by sample_pair()), they are exposed to and can be used safely by the rest of the framework.

3.1.6 Class Diagrams

Figures 3.1 and 3.2 show class diagrams of the memristor framework as well as the pulse library described above.

3.2 Verification

Once a first version of the memristor model's Python implementation described above was finished, it had to be verified for correct functionality. In order to do so, the test protocols Protocol 1 and Protocol 2 defined by Serb et al. [Serb, 2014] and described in Sections 3.3.1 and 3.3.2 were run on the Python implementation. The results were then compared to the ones obtained from the SPICE/MATLAB implementation.

During this test phase, initially different discrepancies were identified and had to be resolved. Appendix A presents a comprehensive summary of the efforts which were taken in order to do so and align the results yielded by the different implementations. As we can see from Figure 3.3, after resolving all the discrepancies, finally both the Python as well as the SPICE/MATLAB version yielded almost identical results.

3.3 Test Protocols

Besides the memristor model itself, also various test protocols were implemented as part of the software framework. These test protocols were defined in order to allow for the investigation of the different problems which will be discussed in Chapter 4. In addition to the new test protocols, as mentioned above, also two others defined by Serb et al. [Serb, 2014], were adopted for this work.

One characteristic all of these protocols have in common is that they employ inputs which are composed of pulses of two different types. These pulses occur with a certain timespan between each other. In this context

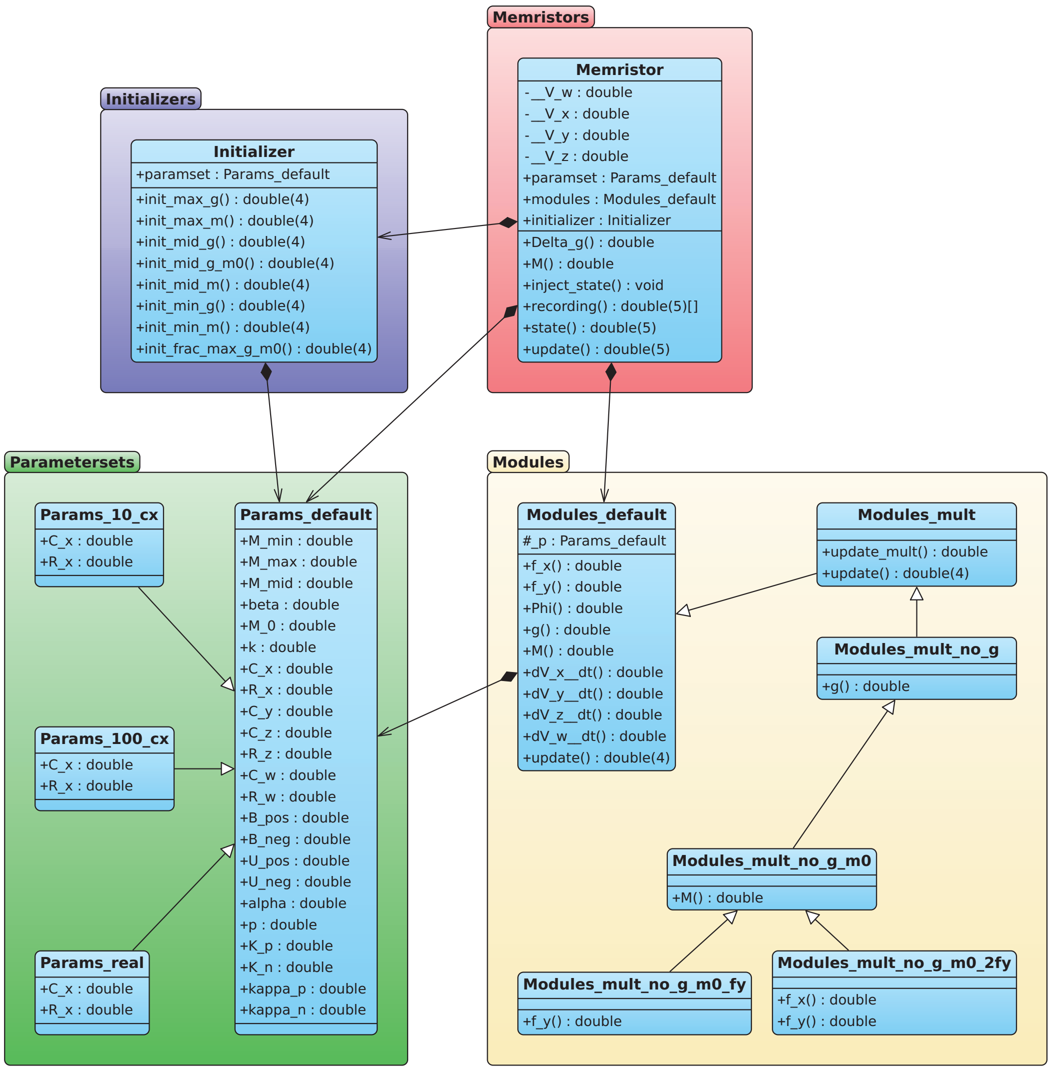


Figure 3.1: Class diagram illustrating the memristor framework. As can be seen, the memristor framework primarily consists of four different classes, each of which is responsible for covering a specific aspect of the given memristor model's internals. A comprehensive description of these different classes is given in Sections 3.1.1 to 3.1.4.

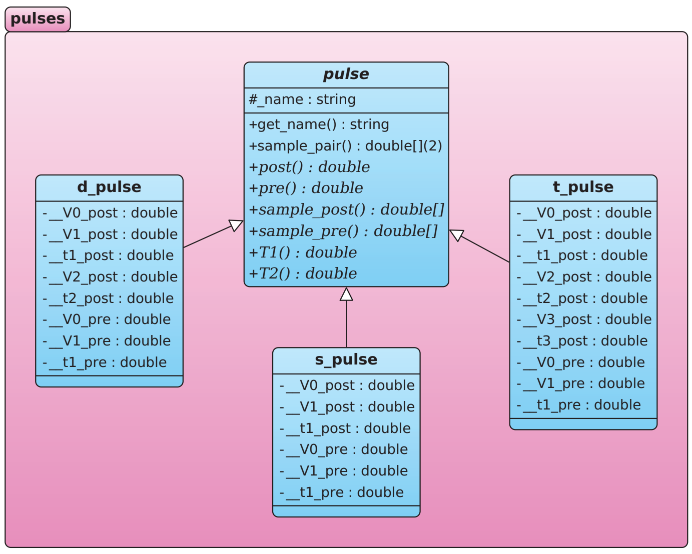


Figure 3.2: Class diagram illustrating the pulse library. The pulse library comprises three different implementations of the abstract `pulse` interface. These implementations represent different types of pulses as will be introduced in Sections 4.3.1 to 4.3.3.

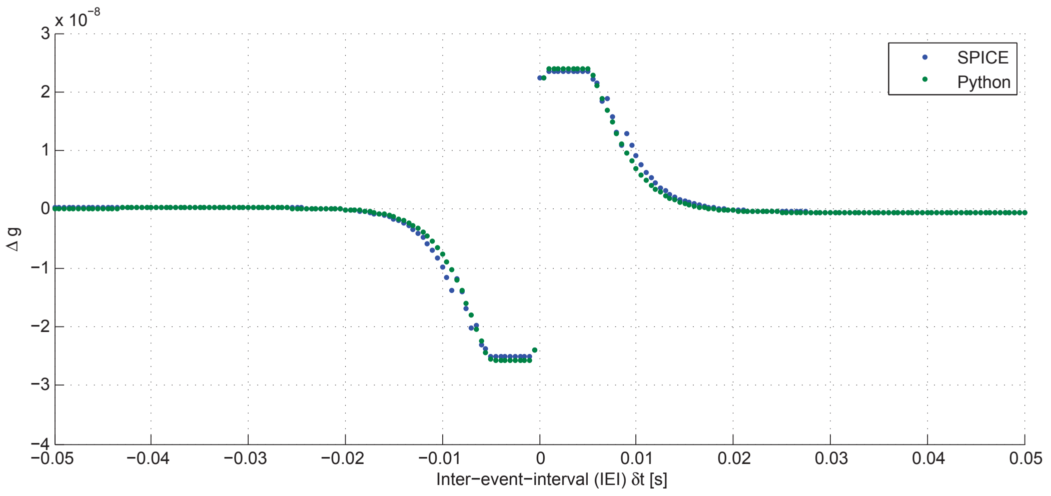


Figure 3.3: Comparison of the SPICE and the Python implementation. The plot compares the results for Protocol 1 as yielded by the original SPICE and the final Python implementation. The process of aligning the two implementations with each other is described at length in Appendix A.

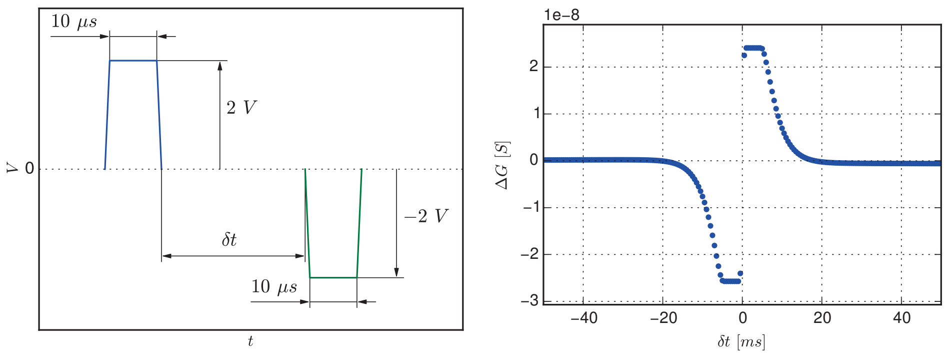
it should be pointed out, however, that there is a fundamental difference in the way the original protocols by Serb et al. and the new ones proposed in this work define this timespan. More precisely, on the one hand Serb et al. define the inter event interval as the time between the end of the first pulse and the beginning of the second pulse. Obviously, given an arbitrary inter event interval, this definition alone does not allow for the determination of the exact order in which the respective pulses occurred. This order, however, is substantial with respect to the learning theory in spiking neural networks. Most likely, this is why a second definition stating that negative inter event intervals correspond to the negative pulse occurring first was added by Serb et al.. For $\delta t = 0$, however, this definition is still ambiguous which is probably why Serb et al. excluded this case by definition in their software framework.

On the other hand and in contrast to [Serb, 2014], for the new protocols the inter event interval is defined as the timespan between the beginning of the respective pulses of a pulse pair. Since these times correspond to the spike times of the pre- and post-synaptic neurons, this approach is better aligned with spiking neural network theory. Moreover, this conception of the inter event interval is necessary in order to allow for overlapping pulses, which is a crucial point with respect to the theory this work is based on.

In order to allow for a better distinction of these two definitions, the inter event interval according to Serb et al. will be denoted as δt throughout this work, while Δt, in contrast, refers to the inter event interval according to the new definition given above. Given these definitions, we will now take a closer look at the implementation of some of the simulation protocols used for this work.

3.3.1 Protocol 1

One of the first steps taken for this work was porting the memristor model proposed by Serb et al. [Serb, 2014] to Python. As indicated earlier, in addition to the memristor model itself, Serb et al. also defined two test

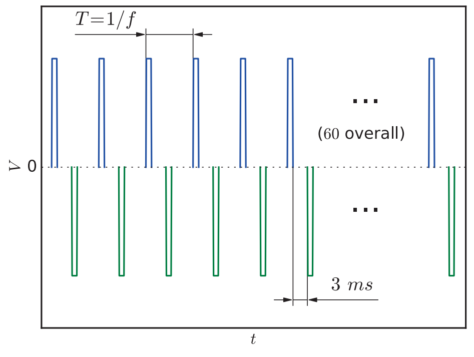


(a) Protocol (b) Example result

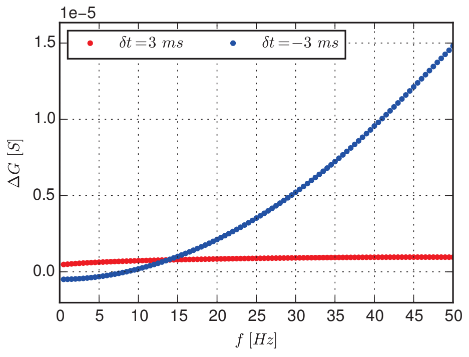
Figure 3.4: Protocol 1 as defined by Serb et al. [Serb, 2014]. Pairs of two pulses with fixed amplitudes of $-2\,V$ and $2\,V$ and fixed durations of $10\,\mu s$ but a varying inter event interval δt are applied to a memristor. The conductance changes ΔG resulting from these stimuli are then plotted against the inter event interval.

protocols. Both of them were adopted for this work in order to allow for a comparison of the Python and the reference implementation.

The first one of these two protocols, Protocol 1, is illustrated in Figure 3.4a. It defines a pair of a positive and a negative rectangular pulse both with durations of $t = 10\,\mu s$. In order to account for realistic hardware limitations and to meet the original SPICE implementation, also rise and fall times of $1\,\mu s$ are considered. The amplitudes of the pulses were chosen as $V_1 = 2\,V$ and $V_2 = -2\,V$, respectively. Based on this setup, the time between these pulses, the inter event interval (IEI) δt, is varied from $\delta t = -50\,ms$ to $\delta t = 50\,ms$ in $0.5\,ms$ steps. In this context, as we mentioned in the introduction to this section, negative and positive values for the IEI translate to the negative and the positive sub-pulse occurring first, respectively. The resulting voltage trains are then applied to a memristor initialized at midrange memristance. For each of these inputs, the difference ΔG between the final and the initial instantaneous conductance is computed and plotted against the corresponding δt values, yielding plots as the one shown in Figure 3.4b. This type of simulation allows insight into the influence of the exact timing between the positive and the negative pulses on the resulting change in conductance.



(a) Protocol for $\delta t = 3\,ms$



(b) Example results

Figure 3.5: Protocol 2 as defined by Serb et al. [Serb, 2014]. 60 pairs of pulses with the same properties as the ones defined in Protocol 1 are generated at different frequencies f and applied to a memristor. Two different inter event intervals $\delta t = -3\,ms$ and $\delta t = 3\,ms$ are used. The conductance changes ΔG resulting from these stimuli are recorded and plotted against the frequency for each of the inter event intervals.

3.3.2 Protocol 2

As indicated in the previous section, besides Protocol 1 also the second test protocol defined by Serb et al. [Serb, 2014], Protocol 2, was adopted for this work. Similar to Protocol 1, also Protocol 2 is based on pairs of square pulses as the ones shown in Figure 3.4a.

Figure 3.5a, however, illustrates that Protocol 2 applies multiple pulse pairs rather than single ones. In addition to this, the (absolute) time interval between the pulses forming a pulse pair is fixed. Based on these fixed quantities, Protocol 2 varies the time between the pulse pairs in order to match certain pairing frequencies. Put differently, Protocol 2 chooses the time $T = {}^{1}/_{f}$ between the pulse pairs such that the resulting 60 paired events occur at specific frequencies f. These frequencies are varied from $f = 0.5\,Hz$ to $f = 50\,Hz$ in $0.5\,Hz$ steps, yielding values for the timespan T between the pulse pairs of $T = 20\,ms$ to $T = 2\,s$. The input trains resulting from each of the frequencies are then applied to memristors, while keeping track of the instantaneous conductance $G(t)$. The difference ΔG between the conductance present at the end and at the beginning of this stimulation is recorded for each of the frequencies f.

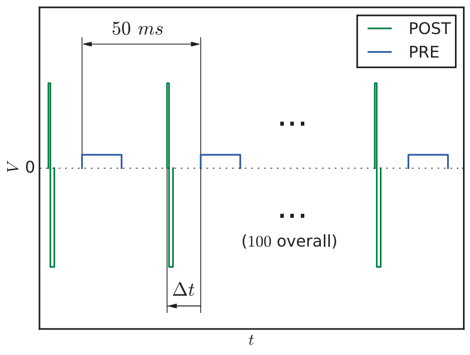
The resulting array of ΔG values is finally plotted against f yielding plots as the one shown in Figure 3.5b.

As we mentioned in the previous section, for the protocols defined by Serb et al. the sign of the inter event interval δt specifies which of the two pulses occurs first. In this section we stated that for Protocol 2 the absolute inter event interval is fixed. Given the remaining protocol setup described above, this allows for two different simulation runs for $\delta t = 3\,ms$ and $\delta t = -3\,ms$ indicated in red and blue in Figure 3.5b.

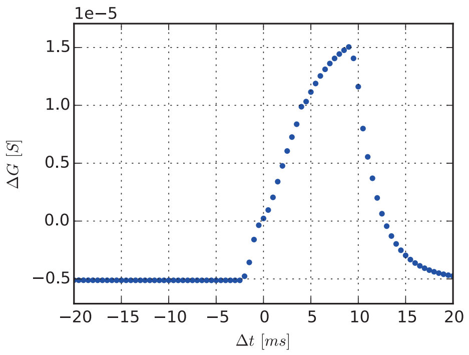
3.3.3 Protocol 3

The protocols described in the previous two sections formed the foundation for the first new protocol defined during the course of this work. Consequently, Protocol 3 can be seen as a combination of Protocol 1 and Protocol 2. More precisely, it performs a sweep of the inter event interval and, for each of the inter event interval values, generates multiple pulse pairs at a given frequency. There are, however, also a few significant differences between the protocols described in the previous sections and Protocol 3.

For instance, in contrast to the protocols defined by Serb et al. [Serb, 2014], Protocol 3 is not restricted to pulses of a specific shape. Rather, it allows for almost any arbitrary pulse shapes such as the ones defined in Section 4.3 to be plugged into the simulation. Hence, as such, Protocol 3 merely defines the simulation environment of having 100 pulse pairs which occur at a specific pairing frequency $f = {}^1\!/_T = 20\,Hz$. Besides this, also the sweep of the inter event interval between the two pulses of a pulse pair is part of the protocol definition. In contrast to the protocols proposed by Serb et al. [Serb, 2014], however, as becomes clear from Figure 3.6a Protocol 3 is the first test protocol which features the alternative definition of the inter event interval Δt we mentioned in the introduction to this section.



(a) Protocol for a negative Δt



(b) Example result

Figure 3.6: Protocol 3 for a negative Δt and D-pulses. 100 pairs of pre- and post-synaptic pulses are generated (according to one of the pulse shapes listed in Section 4.3) with a varying inter event interval Δt at a frequency of $f = 20\,Hz$. The resulting trains are then applied to a memristor while recording the conductance change ΔG for each of the inter event intervals.

During the simulation of Protocol 3, the values for Δt are varied from $\Delta t = -20\,ms$ to $\Delta t = 20\,ms$ in $0.5\,ms$ steps. For each of these Δt values, a corresponding pulse train is generated comprising 100 pulse pairs with a pairing frequency of $f = 20\,Hz$. This pulse train is then applied to a memristor initialized at a specific initial memristance, again keeping track of the instantaneous conductance $G(t)$. Similar as for Protocol 1, the difference ΔG between the final and the initial conductance is computed for each of the pulse trains. The resulting array of ΔG values is finally plotted against the corresponding Δt values, yielding plots as the one shown in Figure 3.6b. Hence, similar as Protocol 1, also Protocol 3 provides some insight into how the change in conductance depends on the relative timing of the pulses of a pulse pair. Since this is closely related to the theory of STDP, we will also refer to plots as the one shown in Figure 3.6b as *STDP plots*.

Finally it should be mentioned that, besides the pulse shapes actually used by the protocol, also almost any other (fixed) parameter can be adjusted for Protocol 3. For instance, for some of the simulations presented in later chapters, only single pulse pairs were used.

3.3.4 Protocol 5

Protocol 5 is closely related to Protocol 3 discussed in the previous section. In contrast to Protocol 3, however, no sweep of the inter event interval Δt is performed. Rather a fixed value for Δt is used. This fixed value is either specified at the beginning of the simulation or chosen from observations made with Protocol 3. More precisely, if no value for Δt is specified explicitly, the inter event interval is chosen from an array containing the Δt values for which the maximum responses in ΔG were observed for previous runs of Protocol 3.

Using this inter event interval Δt, similar as for Protocol 3, 100 spike pairs are generated at a pairing frequency of $f = 20\,Hz$. These spike pairs are then interpreted as triggers for pre- and post-synaptic pulses according to different pulse shapes again. Afterwards, the pulse trains resulting from this interpretation are fed to a memristor, keeping track of the instantaneous memristance $M(t)$. In contrast to Protocol 3, however, also the evolution of all internal state variables is recorded.

At the end of the simulation, the time courses of all the internal state variables V_w, V_x, V_y and V_z as well as the applied external voltage V resulting from the pulse trains and the instantaneous memristance M are plotted. Figure 3.7 shows an example for plots of this type. Obviously, since these plots depict all the internal state variables along with the resulting input/output behavior of the memristor, Protocol 5 can be used in order to gain insight into the memristor's internals, how they affect the overall system and how these internals give rise to certain input-output-behavior.

Similar as Protocol 3, also Protocol 5 allows for full customization of almost any of the simulation parameters. For instance, besides specifying the pulse shapes, the pre- and post-synaptic pulses can be selectively disabled in order to simulate different cases with respect to the SEM learning theory.

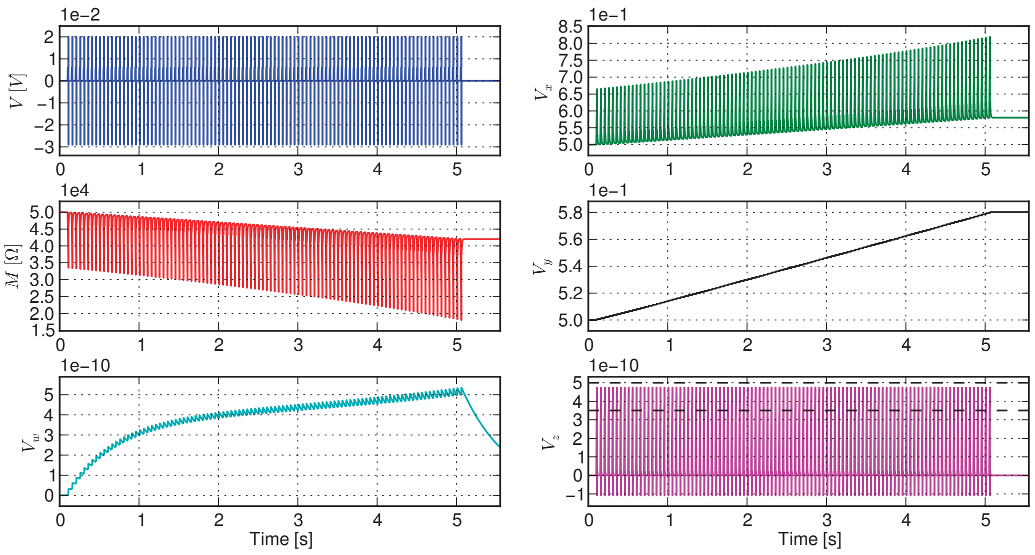
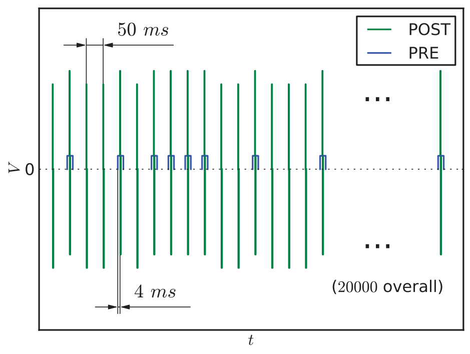


Figure 3.7: Protocol 5 for $\Delta t = 9\,ms$ and D-pulses. A number of pulse pairs (default: 100) with a fixed inter event interval is generated and applied to a memristor. At the end of the simulation, the time course of all the different internal state variables is plotted. Pre- and post-synaptic pulses may be disabled selectively.

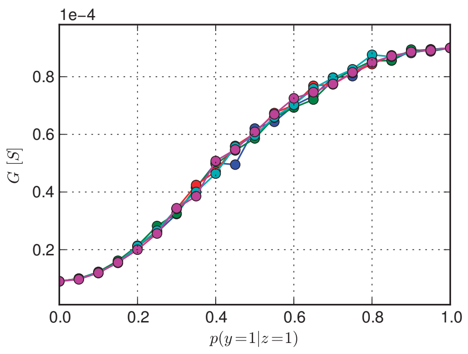
3.3.5 Protocol 7

Basically, Protocol 7 is closely related to one of the experiments described in [BillLegenstein, 2014]. More precisely, Protocol 7 defines a number of $n = 20000$ POST-pulses with a pairing frequency $f = 1/T = 20\,Hz$ which trigger corresponding POST-pulses. Again, the exact shape of these POST-pulses is not part of the protocol definition, but rather specified at the beginning of the simulation. From these n POST-pulses, a certain subset of size m is selected at random. This random subset of POST-pulses is paired with an additional PRE-pulse with a fixed inter event interval $\Delta t = 4\,ms$.

As we shall see in Section 5.2.2, in a spiking WTA network, two random variables Y and Z exist, which model the activity of the pre- and post-synaptic neurons, respectively. In this probabilistic framework, the case of overlapping PRE- and POST-pulses translates to $Y = 1$ and $Z = 1$. Hence, the fraction $\frac{m}{n}$ can be considered approximating the probability



(a) Protocol



(b) Example result

Figure 3.8: Protocol 7 for D-pulses and $p(Y = 1 \,|\, Z = 1) = 0.5$. Pulse trains encoding different probabilities $p(Y = 1 \,|\, Z = 1)$ are generated by creating $n = 20000$ POST-pulses of a given pulse shape (compare Section 4.3) and overlaying a PRE-pulse to a certain subset of them. The resulting stimuli are then then applied to memristors The resulting final convergence conductance values are then plotted against the probabilities.

$p(Y = 1 \,|\, Z = 1)$, that is, the probability of a PRE-pulse given a POST-pulse.

Given the fixed quantities n, f and Δt, Protocol 7 performs a sweep of $p(Y = 1 \,|\, Z = 1)$ from 0.0 to 1.0, corresponding to none and all of the events being of the PRE-and-POST type, respectively. For each of the probabilities, an input pulse train is generated, encoding the respective probability $p(Y = 1 \,|\, Z = 1)$ (an example for D-pulses and $p(Y = 1 \,|\, Z = 1) = 0.5$ is illustrated in excerpts in Figure 3.8a). Each of the resulting input pulse trains is then applied to a memristor. At the end of each of these simulations, the instantaneous conductance $G(t)$ is recorded. In order to allow for a more generalized view and to eliminate outliers, the simulation is repeated for different memristor initializations. Finally, the convergence conductances (that is, the final instantaneous conductance $G(t)$) obtained in the different simulation runs are plotted against the corresponding probabilities $p(Y = 1 \,|\, Z = 1)$, yielding plots as the one shown in Figure 3.8b. Obviously, plots like this allow for an evaluation of the mapping between the probabilities $p(Y = 1 \,|\, Z = 1)$ and the corresponding convergence conductances. As we shall see later in Sections 4.4 and 4.5, these plots play a key role in assessing convergence and initialization issues.

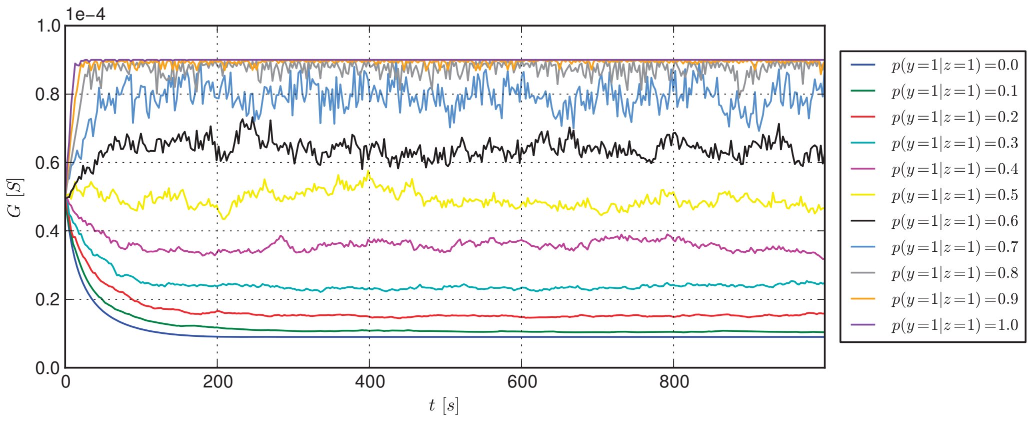


Figure 3.9: Alternative visualization of Protocol 7. Instead of plotting the final (convergence) conductances for each of the probabilities $p(Y = 1 \mid Z = 1)$, the evolution of the instantaneous conductance G over time is plotted as a whole.

In addition to the plots described above, also an alternative visualization for Protocol 7 exists. For this visualization, the evolution of instantaneous conductance G over time is recorded as a whole for each of the probabilities $p(Y = 1 \mid Z = 1)$. The resulting arrays of $G(t)$ values are then plotted against the time. As can be seen from Figure 3.9, this allows for some insight into the exact convergence behavior of the memristor as a function of the applied PRE/POST statistics as well as the memristor's initial state. Based on this, the mapping plots shown earlier can be considered as a snapshot of the evolution plots illustrated in Figure 3.9 taken at the end of the simulation.

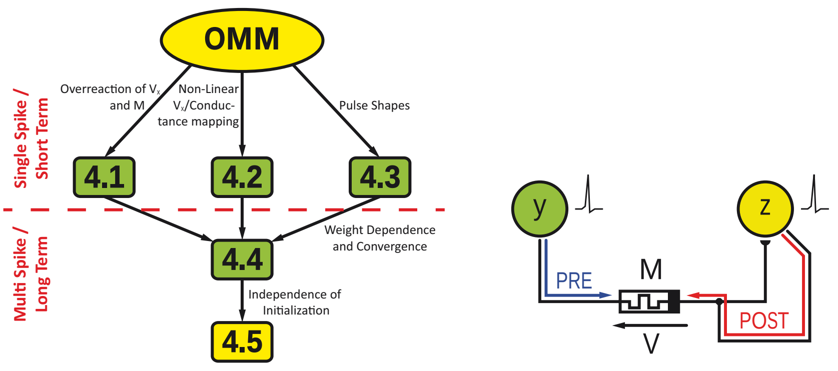
With $n = 20000$ POST-pulses involved, the focus of Protocol 7 lies on the investigation the system's long term behavior, making simulation runs for Protocol 7 consume significantly more computation time than the protocols described earlier. To account for this, Protocol 7 was implemented in a distributed, cluster-enabled manner.

This concludes the documentation of the software framework written for this work. In the next chapter, we will take a closer look at different problems investigated base on this software framework.

4 Problems and Solutions

During the process of studying the memristor model by Serb et al. [Serb, 2014] (see Section 2.2.2) in the context of its suitability for the implementation of synapses for spiking WTA network circuits, different problems were encountered. Some of these problems were found to be related with each other, while others turned out to be independent. In this section it was tried to decompose all of them into separate and almost independent subproblems as well as in any way possible. In the following, each of these decomposed subproblems will be discussed in its own section, starting with problems related to single spike and short term dynamics. These discussions are followed by considerations on issues encountered in the context of long term and convergence studies involving up to a couple of thousand spike pairs.

Figure 4.1a illustrates the corresponding subproblem-section mapping, where *OMM* denotes the original memristor model by Serb et al. [Serb, 2014]. These subproblems include the following:


OMM
Overreaction of V_x and M
Non-Linear V_x/Conductance mapping
Pulse Shapes
Single Spike / Short Term
4.1
4.2
4.3
Multi Spike / Long Term
Weight Dependence and Convergence
4.4
Independence of Initialization
4.5
y
z
PRE
M
POST
V

(a) Subproblem-section mapping (b) Simplified simulation setup

Figure 4.1: Different subproblems and simplified setup. In (a), *OMM* denotes the original memristor model as described in Section 2.2.2. Arrows are labeled with the name of the problems and point to the section in which these problems are discussed in this work. Rectangles with rounded corners, name these sections. The problems above the dashed red line are related to single spike (short term) setups, while those below it are specific to multi spike (long term) environments. (b) illustrates the simplified single-synapse setup which is used for the simulations described in this chapter to allow for an uncluttered notation (illustration adapted from [BillLegenstein, 2014]).

- Section 4.1: Overreaction of V_x and M to the input voltage.
- Section 4.2: Non-linearity of the mapping between V_x and the synaptic weight represented by the memristor's conductance.
- Section 4.3: Finding an appropriate pulsing scheme that is suitable for implementing LTP and LTD depending on the STDP timing Δt.
- Section 4.4: Weight dependence of the conductance change in order to achieve convergence.
- Section 4.5: Independence of the convergence points from the memristor's initialization.

As is indicated by Figure 4.1a, the investigations of the subproblems related to the system's short term dynamics are all based on the original memristor model as described in Section 2.2.2. The insights gained in the corresponding experiments were then jointly used to implement a slightly modified but fine-tuned memristor model, serving as foundation for the multi spike experiments. In order to allow for an uncluttered notation as well as to facilitate an easy analysis, the different subproblems were investigated based on a simplified network setup consisting of only a single synapse as illustrated in Figure 4.1b.

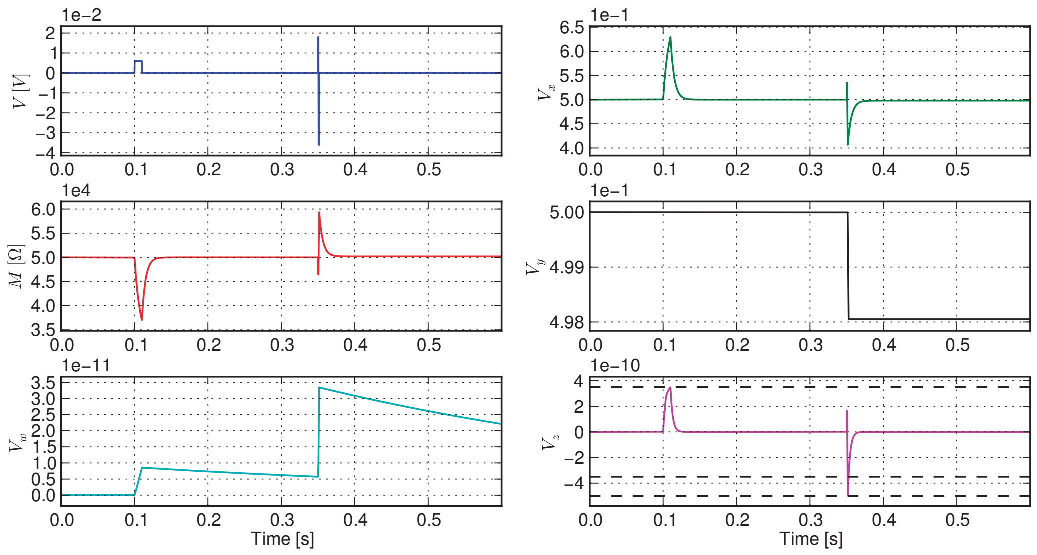


Figure 4.2: System response to events of two different types. Clearly, V_x modeling the volatile memristance changes shows heavy reactions to both the unipolar as well as the bipolar event. V_y governing the non-volatile portion of the instantaneous memristance reacts less violently.

4.1 Overreaction of V_x and M

As turned out during experiments involving pulses of different shapes, the original memristor model described in Section 2.2.2 shows heavy reactions to various types of input. More precisely, in the original memristor model the instantaneous memristance $M(t)$ follows spikes in the applied input voltage $V(t)$ immediately. This situation is illustrated in Figure 4.2. In the corresponding simulation, two events of different type were applied to a memristor initialized with standard parameters. These events are illustrate in the top left plot of Figure 4.2: While the first event is unipolar and comprises a single plain square pulse with an amplitude of $6\,mV$ and a duration of $10\,ms$, the second event is bipolar and features a combination of a positive and a negative square sub-pulse. These sub-pulses have amplitudes of $18\,mV$ and $-36\,mV$ and durations of $0.5\,ms$ and $1\,ms$, respectively.

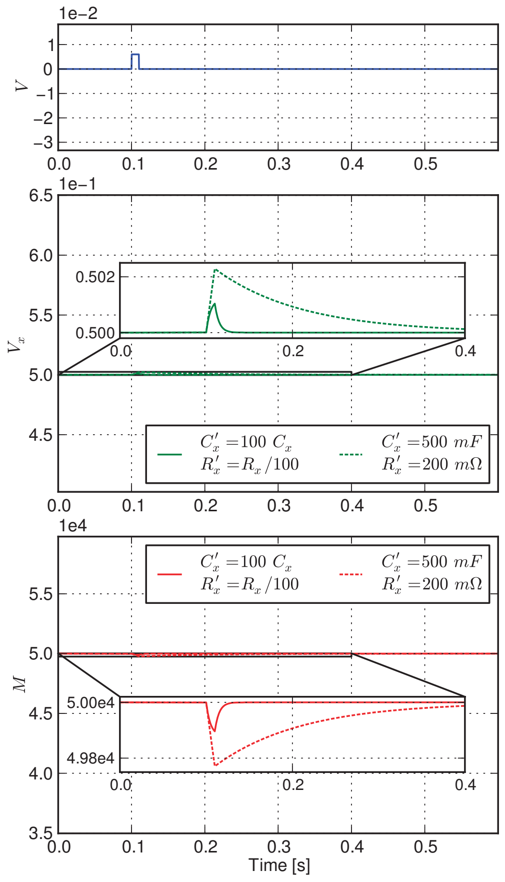
As becomes evident from the plots shown in Figure 4.2, although the memristor reacts heavily to both the bipolar as well the unipolar event, in terms of the internal state variables these overreactions are limited to V_x covering the volatile aspects of the memristor. The non-volatile part of the memristance V_y, in contrast, reacts to the pulses in the input voltage much less violently, merely dropping by less than one percent in case of the bipolar event. Both events' impact on V_x, however, is severe. More precisely, in a memristor initialized at midrange memristance, a single unipolar event drives V_x from the initial value $V_x = 0.5$ to a maximum of $V_x \approx 0.63$. Similarly, the first sub-pulse of a bipolar event initially increases V_x from $V_x = 0.5$ to $V_x \approx 0.54$, before the second sub-pulse drives the value down to $V_x \approx 0.4$. According to Equation 2.10, these ranges translate to variations of the instantaneous memristance from $M(t) \approx 37\,k\Omega$ to $M(t) \approx 50\,k\Omega$ and $M(t) \approx 46\,k\Omega$ to $M(t) \approx 60\,k\Omega$, respectively. For memristors with less neutral initial states (that is, states further away from midrange memristance), this could cause the instantaneous memristance $M(t)$ to be driven into its upper or lower bounds regularly by comparatively non-invasive events. Although finally these peaks in the memristance curve decay quickly for both pulse types, this behavior is highly undesirable in spiking WTA networks consisting of several hundreds of interconnected memristive synapses as it could compromise the overall system stability.

Based on the observation that only V_x is affected, the transient charging process defined in Module II was identified as a potential cause of the overreactions. More precisely, it was supposed that the coupling between the input voltage $V(t)$ and the volatile state $V_x(t)$ was too strong. According to Equation 2.12, the strength of this coupling is basically defined by the capacitor C_x, where larger capacitance values generally lead to weaker coupling. To account for this, it was decided increase C_x. In terms of the underlying system dynamics, this adjustment was supposed to slow down the before-mentioned transient charging process of V_x triggered by the input voltage $V(t)$. Trying to keep the influence of this parameter adjustment on the overall system behavior as small as possible, it was decided, however, to preserve the module's original time constant $\tau_x = C_x\,R_x$ by choosing an appropriate new value for R_x.

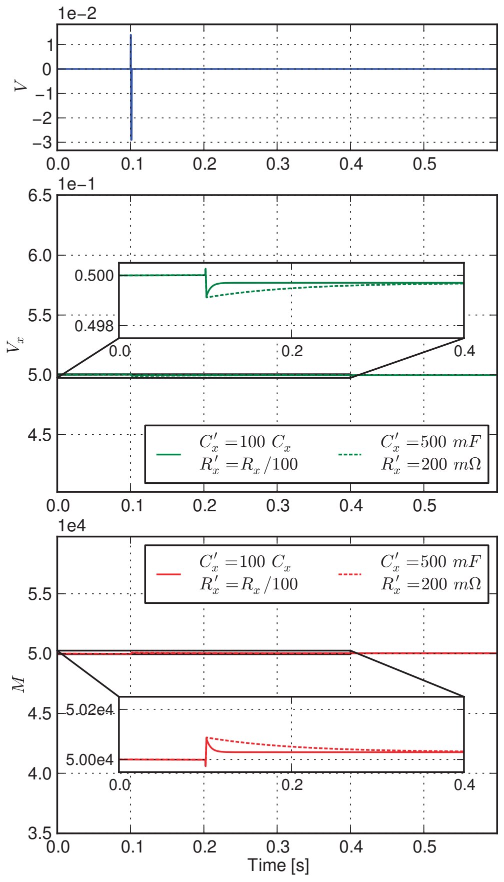
As a starting point, $C'_x = 100\,C_x$ and a corresponding $R'_x = \frac{R_x}{100}$ were chosen. The V_x- and M-curves associated with this first adjusted parameter configuration are indicated with solid lines in the middle and bottom plots of Figures 4.3a and 4.3b. As can be seen, for both input events the results are perfectly aligned with our expectations. Indeed, the memristor's volatile reactions represented by V_x are much smoother, ultimately resulting in a less spiky instantaneous memristance $M(t)$. Actually, the remaining peaks in the $V_x(t)$ curve are so small that they are not even visible when using the same scale as in Figure 4.2, which is why zoomed insets with an appropriate scale were added.

While undoubtedly reducing the strength of the coupling between the input voltage $V(t)$ and the instantaneous memristance $M(t)$ and thus solving the problem of having V_x overreact to spiky input, one might legitimately have serious concerns about this parameter adjustment. Even if preserving the original time constant τ_x, increasing C_x just like that seems a rather invasive strategy. Talking to Serb et al., however, it turned out that the adaption of C_x and the thereby induced reduction in the sensitivity of Module II controlling the volatile memristance, was well supported by their latest research findings[Bill, 2014a]. In addition to this, Serb et al. reported that their most recent experiments gave rise to the assumption that in real memristor hardware volatile processes related to the instantaneous memristance $M(t)$ occurred with time constants in the range of hundreds of milliseconds up to a few seconds[Bill, 2014a]. In order to account for these new insights, the time constant τ_x was adjusted once again, choosing $\tau_x = 100\,ms$ by setting $C'_x = 500\,mF$ and $R'_x = 200\,m\Omega$.

After this adjustment, the simulations described above were re-run. The V_x and M-curves resulting from this second alternative parameter configuration are illustrated in dashed lines in Figures 4.3a and 4.3b. As we can see, while having slightly stronger coupling than for the previous and arbitrarily chosen value for C'_x, the reaction of V_x to the input events is still clearly weaker than for the original parameter configuration. Comparing the dashed and the solid V_x-plots we notice, however, that the decay time of $V_x(t)$ has increased notably, which results from the new time constant τ_x being twenty times larger than the original one. Since



(a) Unipolar event



(b) Bipolar event

Figure 4.3: Reaction of V_x and M to events of two different types. Solid lines indicate the modified parameter set with $C'_x = 100\,C_x$ and $R'_x = \frac{R_x}{100}$, while dashed lines represent the "realistic" parameter set with $C'_x = 500\,mF$ and $R'_x = 200\,m\Omega$. Zoomed insets were added in order to allow for the same scale as used in Figure 4.2.

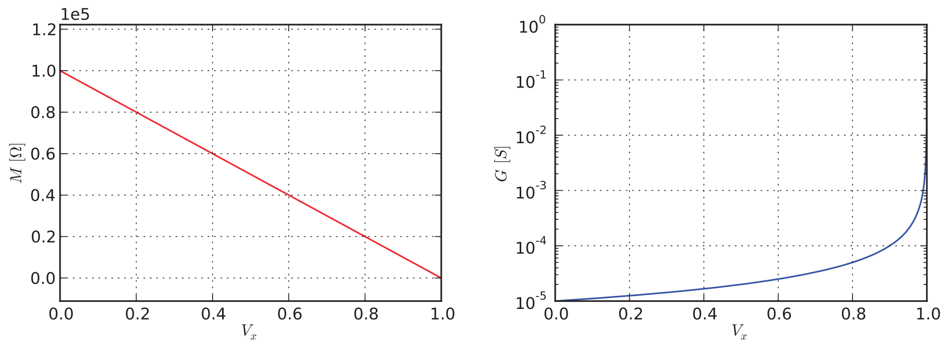
the instantaneous memristance $M(t)$ is shaped by $V_x(t)$, the same applies to the corresponding $M(t)$-plots in Figures 4.3a and 4.3b. Nevertheless, as turned out in experiments we are going to discuss in later sections, there is no harm in the increased decay time introduced by the "realistic" parameter set, making it an acceptable choice. Consequently, it was ultimately adopted into the final memristor model.

These considerations conclude the discussion of the efforts that were made in order to reduce the overreactions of the instantaneous memristance $M(t)$ to spiky input. As we have seen, adjusting the parameter C_x according to different aspects helped in making the reactions of the system less violent, increasing the overall stability. In the next section we will take a closer look at the mapping of internal state variables onto a corresponding conductance value.

4.2 Non-Linear V_x/Conductance Mapping and Dynamic Range

In Section 2.4 we stated that the ultimate goal of this work was constructing novel memristor-based synapses for spiking WTA networks. Most modern neuromorphic hardware implementations of these networks represent neural spikes as voltage pulses of different shapes. Based on this, the most natural approach is to establish a proportional relation between the conductance of the junctions of pairs of neurons as the respective synapses' weight. As a result, for our memristor-based synapses it can be considered preferable to have the affected synapses' response in terms of the conductance changes triggered by the individual pulses well under control. In the underlying memristor model, this requirement translates to the need of having the state variables governing the memristor's characteristics drive the device's conductance through the available dynamic range linearly and evenly.

Recalling the models presented in Section 2.2, we notice however that the available state-of-the-art memristors seem to be constructed around



(a) Memristance M (b) Conductance G

Figure 4.4: Memristance M and conductance G as functions of V_x. While the mapping of V_x onto a corresponding memristance $M(V_x)$ is perfectly linear, the relation between V_x and the conductance $G(V_x)$ is highly non-linear (notice the logarithmic scale of plot (b)).

a linear mapping between a subset of their internal state variables and the instantaneous memristance $M(t)$, meaning that they are resistance-based. None of them allows for our desired linear mapping of the memristor's internal state onto its instantaneous conductance $G(t)$. While this resistance-based mapping is not of harm for some applications, one runs into a fundamental problem when implementing memristive synapses for spiking WTA networks.

In their memristor model, Serb et al., for instance, defined the mapping as shown in Equation 2.10 through the internal state variable V_x yielding the perfectly linear relation between V_x and the instantaneous memristance M as illustrated in Figure 4.4a. The resulting mapping of V_x onto a corresponding conductance G, however, is highly non-linear. More precisely, as we can see from Figure 4.4b, more than 99 % of the conductance's dynamic range is covered by less than 10 % of V_x. For synaptic weights, this translates to a huge dynamic range being available to choose the weights from, but significant weight changes only happening within a very narrow range. This makes the reaction of the synaptic junctions highly sensitive to the applied input. After staying almost the same for very long time, the weight of a memristive synapse would then change from a relatively low to the highest possible value almost instantaneously. Al-

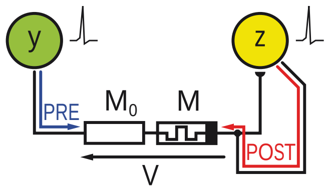
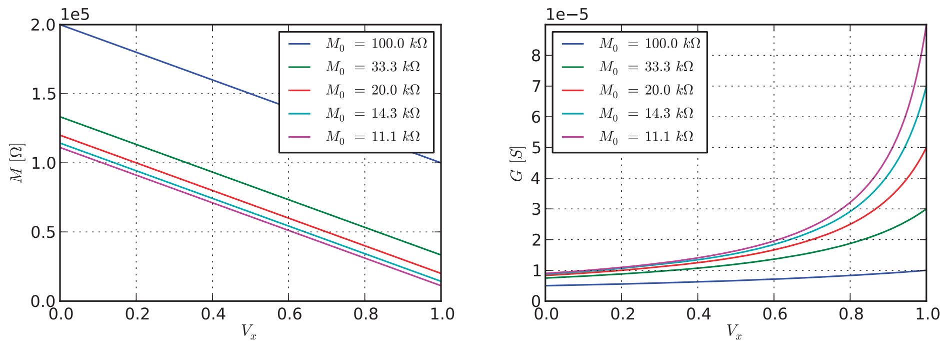


Figure 4.5: Additional ohmic pre-resistor M_0. Connecting M_0 aids in reducing the dynamic range of both the memristance as well as the conductance, thus improving the linearity of the mapping between V_x and the conductance (illustration adapted from [BillLegenstein, 2014]).

though compound memristive synapses based on stochastically switching, bistable memristive devices were proposed and shown to be viable by Bill and Legenstein [BillLegenstein, 2014], this behavior is highly undesirable for the approach we pursue for this work. As indicated in Chapter 1, for this work we aim for single-memristor synapses with a continuous weight spectrum.

Besides the underlying system design, in the memristor model by Serb et al. [Serb, 2014], the above-mentioned effect of the conductance changing from the least to the most conductive state almost instantaneously primarily results from the relatively large dynamic range of the memristor's memristance ($M_{\text{min}} = 1\,\Omega$ to $M_{\text{max}} = 100\,k\Omega$). Linearizing the system by narrowing this range down without any further ado, however, seemed to be an unnecessarily invasive strategy. Moreover, it was unknown whether changes like this are compatible with real memristive hardware. In an attempt to still reduce the dynamic range of the memristive synapse while acting less invasively, it was decided to introduce an additional, pure ohmic pre-resistor M_0. Connecting this resistor M_0 in series to the memristor representing the memristive synapse as shown in Figure 4.5 was supposed to bring the bounds of the memristance's dynamic range closer together. Consequently, also the bounds of the memristor's conductance would be closer together, thus reducing the non-linearity of the V_x/conductance mapping. Put differently, the installation of the additional pre-resistor M_0 was meant to reduce the non-linearity by widening the conductance range covered by a given V_x-range.

Looking at Figure 4.6b, however, we notice that, although reducing the



(a) Memristance M

(b) Conductance G

Figure 4.6: Memristance M and conductance G as functions of V_x for different values of M_0. Notice how the dynamic range controlled by V_x depends on the value chosen for M_0. Clearly, greater values for M_0 reduce the non-linearity of the mapping between V_x and a corresponding conductance $G(V_x)$.

non-linearity of the mapping, the chosen value of this resistor M_0 turns out to be a crucial quantity. As we can see, larger M_0 values basically lead to less non-linearity in the resulting $G(V_x)$ plots. This increased linearity, however, is obtained at the expense of a large dynamic range. Choosing $M_0 = \frac{M_{\text{max}}}{9} = 11.1\,k\Omega$, for instance, leads to a minimum and maximum memristance of approximately $11.1\,k\Omega$ and $111.1\,k\Omega$, respectively. This translates to a synapse's ability of adjusting its weight by a factor of 10 during learning. For the more linear curve obtained with $M_0 = M_{\text{max}} = 100\,k\Omega$, however, there is only a factor of approximately 2 between the least and most conductive memristor state. This corresponds to the synapse being able to only double the strength of the corresponding synaptic junction anymore. In contrast to this, for the original memristor model as proposed by Serb et al., a factor of roughly 10^5 could be achieved without the additional resistor M_0. For the purpose of the simulations carried out during this project, $M_0 = \frac{M_{\text{max}}}{9}$ was chosen. As turned out, for this value both acceptable stability as well as a sufficiently large dynamic range can be achieved.

In addition to the need for an appropriate choice of the value of M_0, introducing the pre-resistor bears another potential drawback. As indicated in the introduction, the implementation of a spiking WTA network as neuromorphic hardware chips involves tight area constraints. One of the greatest advantages of using memristors rather than other classic semiconductor-based devices for such networks lies in the way plasticity is implemented. While for classic semiconductor-based solutions one has to design special circuits or even dedicated co-processors that are capable of doing the complex math required by the principles of synaptic plasticity, memristors implement these rules intrinsically. Hence, in general memristor-based WTA network solutions can reduce the required die space dramatically [Serrano, 2013]. On the one hand, this allows for additional hardware on the chip. On the other hand, however, one has to be careful not to recklessly spoil the gained spatial advantage by blindly adding supplemental circuitry such as M_0. As turned out, however, the crossbar architecture suggested in [Linares, 2009], comes in handy at this point. More precisely, one does not have to add the pre-resistors M_0 for each of the memristive synapses. It suffices to add them at the the crossbar's input, since each of the PRE input channels is connected to multiple POST neurons through the respective memristive synapses. Since this ensures that the hardware scales well, adding the additional pure-ohmic resistor M_0 to the final memristor model seemed reasonable.

This concludes the discussion of the non-linearity problem intrinsic to all state-of-the-art memristor models. As we have seen, the implementation of memristive synapses potentially suffers from the way up-to-date memristor models map their internal state variables onto the external input-output-behavior. As a solution to this problem, the introduction of additional pure-ohmic pre-resistors M_0 was identified. These pre-resistors are connected in series to the input of the memristive synapses, reducing the conductance's dynamic range, while leaving the internals of the actual memristor model unchanged. Similarly, also in the next section dealing with different pulse shapes which are to be used in combination with the memristive synapses, the internals of the underlying memristor model are not changed in any way.

4.3 Pulse shapes

While the previous sections dealt with problems related to the internals of the memristors used to the construct memristive synapses, this section is devoted to a topic more in the context of these synapses' environment within spiking WTA networks. More precisely, we are going to develop different pulse shapes suitable for the implementation of synaptic plasticity inspired by the findings of Zamarreño-Ramos et al. [Zamarreño, 2011] and Querlioz, Bichler, and Gamrat [Querlioz, 2011]. In contrast to Section 5.1, the approach taken here is solely based on general reasoning implied by the memristor and learning theory discussed in Chapter 2. Put differently, in this section we are only interested in finding pulse shapes which implement synaptic plasticity in a qualitative manner and skip a thorough discussion of quantitative aspects for the moment.

Obviously, besides the memristors themselves, the different pulse shapes utilized by the individual neurons are one of the most fundamental building blocks for the implementation of spiking WTA networks as memristor-based neuromorphic hardware circuits. In combination with the input-output-characteristics of the memristors, the employed pulse shapes are responsible for the correct enforcement of an appropriate synaptic plasticity rule. As mentioned in Section 2.3, in this work we aim for a network implementation of the SEM learning theory as a standard gauge to match the behavior yielded by our memristive synapses against.

Recalling Section 2.3.2, we remember that, with respect to the weight adaption in response to a given type of synaptic activity, the SEM learning theory basically distinguishes four different cases, which call for three different synaptic reactions to be triggered. These reactions are

- *increasing the synaptic weight*,
- *reducing the synaptic weight* and
- *leaving the synaptic weight unchanged*.

As mentioned in Section 4.2, in the model used for this work the current weight of the memristive synapses is encoded in the instantaneous conductance $G(t)$ of the underlying memristors. Given that $G(t)$ is the

Case / Event	**Reaction**
PRE-only	None
POST-only	Increase non-volatile memristance
POST-before-PRE	Increase non-volatile memristance
PRE-at-most-τ-before-POST	Reduce non-volatile memristance

Table 4.1: Different events and their expected effects. According to the SEM learning theory presented in Section 2.3.2, four different cases with respect to the combination of pre- and post-synaptic activity can be distinguished. These cases call for three different synaptic reactions, which need to be emulated by the memristive synapses.

inverse of the instantaneous memristance $M(t)$, we can formulate the main requirements for our desired pulse shapes with respect to the induced conductance changes as summarized in Table 4.1. Obviously, in terms of the underlying memristor model by Serb et al. [Serb, 2014], these qualitative statements with respect to changes in the non-volatile memristance basically translate to an appropriate sign of the temporal derivative $\frac{dV_y}{dt}$ being required for each of the four cases.

Recalling Equation 2.15, we remember that the sign of this derivative is determined by the parameter function $\phi(\cdot)$, since both $f(V_y)$ as well as $g(V_w)$ are always positive within the range the memristors are operated in. The sign of $\phi(\cdot)$, in turn, is determined by the driving effort $V_z(t)$, since all the other quantities involved in its definition in Equation 2.17 are constants. Consequently, considering the shape of $\phi(\cdot)$ shown in Figure 2.2a, with respect to our pulse shapes everything boils down to depressing events being required to drive V_z below B_- for some timespan, while for potentiating ones V_z needs to exceed B_+. For neutral events in terms of the induced weight change, finally, V_z shall stay within the interval (B_-, B_+).

Based on the definition of $\frac{dV_z}{dt}$ given in Equation 2.18, these requirements to the driving effort V_z ultimately imply certain courses (that is, amplitudes and durations) for the input voltages $V(t)$ for each of the events listed in Table 4.1. Having a closer look at these events and the corresponding reactions once again, we notice that POST-pulses are involved in three out of four events, which require reactions of two different types to be

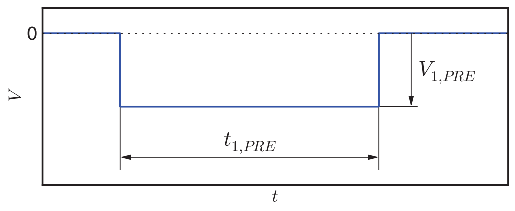


Figure 4.7: Qualitative illustration of the pre-synaptic pulse. Without any additional superimposed post-synaptic D- or T-pulse, the pre-synaptic pulse does not trigger any persistent conductance changes.

triggered. To account for this, it seems reasonable to aim for a bipolar design for the post-synaptic pulses, that is, pulses consisting of at least one positive and one negative sub-pulse. Moreover, given that pre-synaptic pulses are not supposed to trigger any non-volatile changes unless an overlap with a corresponding post-synaptic pulse exists, the simplest shape for the pre-synaptic event to satisfy both constraints is a single unipolar plain square pulse as shown in Figure 4.7.

Translating the qualitative constraints listed in Table 4.1 now into certain pulse shapes for the constituent sub-pulses of the different pulses requires an expression describing the evolution of V_z over time depending on the applied input voltage. This expression can be determined by solving the differential equation describing the temporal derivative of the driving effort V_z defined in Equation 2.18. Assuming a constant input $V(t) = A$ and an arbitrary initial preload $V_{z,0}$, we obtain

$$V_z(t) = V_{z,0}\, e^{\frac{-t}{C_z R_z}} + \frac{A\, R_z}{M_{\mathrm{mid}}} \left[1 - e^{\frac{-t}{C_z R_z}}\right] \tag{4.1}$$

(see Appendix B for a detailed derivation). A remarkable property of this expression is that, for the given assumptions, at any arbitrary instant of time the overall driving effort V_z can be interpreted as a superposition of two different forces. Namely, these forces are the decaying preload $V_{z,0}$ (the first expression of the sum) on the one hand and a transient charging process of the system driven by $V(t) = A$ (the second expression) on the other hand. As we shall see later, this interpretation comes in handy when dealing with multi-pulse input. In accordance with [Bill, 2014c], we identify the input voltage $V(t)$ yielded by the neurons as

$$V(t) = V_{\text{POST}}(t) - V_{\text{PRE}}(t)\,. \tag{4.2}$$

In hardware, this translates to the positive and negative terminals of the memristor being connected to the pre- and post-synaptic neurons, respectively.

Looking for appropriate values to be chosen for the PRE-pulses' free parameters $V_{1,\text{PRE}}$ and $t_{1,\text{PRE}}$, we start with the duration $t_{1,\text{PRE}}$. One of the main purposes of the unipolar pre-synaptic pulses is defining the width τ of the STDP learning window, during which the memristive synapse is receptive to potentiating updates. In order to match observations in this context made for real biological synapses, $t_{1,\text{PRE}} = 10\,ms$ was chosen. Based on this value for the pulses' width, one can determine a range of corresponding maximally allowable values for $V_{1,\text{PRE}}$. As explained above, for pre-only events V_z is required to stay within the range (B_-, B_+) for the duration of the whole pulse. Looking at Equation 4.1 we notice that, depending on the sign of the constant input voltage A, the evolution of V_z over time will be either monotonically increasing or decreasing. Hence, its local extremum will be reached at the end of the square pulse, that is at $t = t_{1,\text{PRE}}$. This implies that if V_z does not exceed the bounds of the just-mentioned range at the end of the square pulse and it has not at the beginning, we can safely assume that it has not exceeded the bounds anywhere in between either (see Appendix B.2 for a more detailed explanation).

Based on this observation and given the values for the positive bipolar threshold B_+ that must not be exceeded and an assumed initial preload $V_{z,0} = 0$, a lower bound for $V_{1,\text{PRE}}$ can be determined by rewriting Equation 4.1 as

$$V_{1,\text{PRE}} \overset{!}{>} -\frac{B_+\,M_{\text{mid}}}{R_z\left[1 - e^{\frac{-t_{1,\text{PRE}}}{C_z\,R_z}}\right]}\,. \tag{4.3}$$

Note that the negative sign of the right hand side of the above expression is a direct consequence of the way the input voltage is constructed out of the neurons potentials according to Equation 4.2. The upper bound for the amplitude of the pre-synaptic pulses is given as $V_{1,\mathrm{PRE}} < 0\,mV$, since Equation 4.2 implies a negative value for $V_{1,\mathrm{PRE}}$ in order to allow for the assigned responsibility of making V_y exceed B_+ in the case of potentiating events. Plugging the model parameters defined in Table 2.2 as well as the above mentioned value $t_{1,\mathrm{PRE}} = 10\,ms$ into Equation 4.3, we obtain a legitimate range for $V_{1,\mathrm{PRE}}$ of

$$-6.049\,mV < V_{1,\mathrm{PRE}} < 0\,mV\,. \tag{4.4}$$

As a slightly tighter constraint than the one satisfied by the above voltage range, we require that $V_z(t)$ stays within the range (B_-, B_+) even in the limit case, that is for $t_{1,\mathrm{PRE}} \rightarrow \infty$. This limit case approximates high spiking activity in the pre-synaptic neurons, resulting in a PSP being present almost all the time and yields

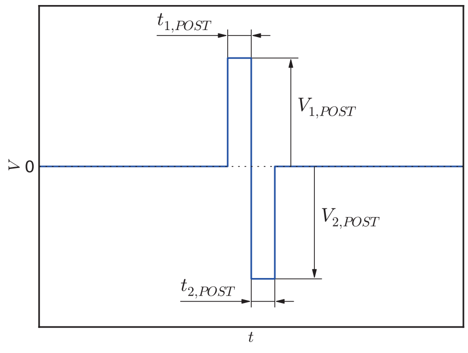
$$\boxed{-5.833\,mV < V_{1,\mathrm{PRE}} < 0\,mV}\,. \tag{4.5}$$

Based on these estimations, finally $V_{1,\mathrm{PRE}} = -5.83\,mV$ was chosen.

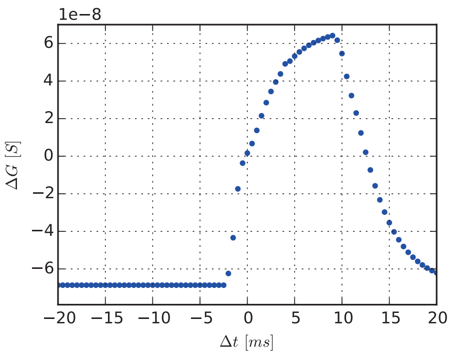
Together with $t_{1,\mathrm{PRE}} = 10\,ms$, this value for the amplitude fully describes the pre-synaptic pulse used in combination with post-synaptic D-pulses and T-pulses. These pulse shapes will be discussed in detail in the following (Sections 4.3.1 and 4.3.2, respectively) and compared with respect to their STDP behavior according to Protocol 3 described in Section 3.3.3 as well as few other aspects.

4.3.1 D-Pulses

As indicated in Section 4.1, one of the starting points in finding an appropriate POST-pulse shape suitable for the implementation of synaptic plasticity in memristive synapses was a pulse consisting of a positive and a negative sub-pulses as illustrated in Figure 4.8a. We will refer to this pulse type as *D-pulse*, where *D* is an abbreviation for *double* derived from the number of contributing sub-pulses. The idea behind the bipolar design is a direct consequence of the above-mentioned observation that post-synaptic pulses need to be capable of triggering two different synaptic



(a) POST-pulse



(b) STDP-plot

Figure 4.8: Qualitative illustration of the post-synaptic D-pulse and STDP plot. Indeed, as is indicated by the plot in (b), in combination with the PRE-pulse shown in Figure 4.7, D-pulses are qualitatively capable of triggering STDP: For negative inter event intervals Δt, the induced conductance change is negative, while it is positive for positive inter event intervals.

reactions depending on the relative timing to and the resulting overlap with respective pre-synaptic pulses.

According to the illustration presented in Figure 4.8a, with $V_{1,\text{POST}}$, $V_{2,\text{POST}}$, $t_{1,\text{POST}}$ and $t_{2,\text{POST}}$ there are four free parameters to be chosen for D-pulses. In order to derive specific ranges for each of them, we utilize the expression for the temporal evolution of the driving effort V_z given in Equation 4.1 together with the constraints mentioned earlier in this section as well as additional requirements. One of these additional requirements states that, in order to stay compatible with the SEM learning theory, the post-synaptic pulses have to be short compared to the pre-synaptic ones. To account for this, $t_{1,\text{POST}} = 0.5\,ms$ and $t_{2,\text{POST}} = 1\,ms$ was chosen.

Based on these durations, the qualitative constraints defined earlier in this section can be translated into voltage ranges the amplitudes $V_{1,\text{POST}}$ and $V_{2,\text{POST}}$ are not allowed to exceed. As mentioned earlier, POST-only events are supposed to trigger pure LTD without any fraction of LTP. Put differently, the drive effort V_z is not allowed to exceed B_+ for the duration of the whole POST-pulse. Since only positive input voltages can drive V_z towards B_+, this constraint allows for the derivation of an upper bound for $V_{1,\text{POST}}$. Similar as for the PRE-pulse, this can be achieved by plugging

B_+ and $t_{1,\mathrm{POST}}$ into Equation 4.1 and solving the resulting expression with respect to $V_{1,\mathrm{POST}}$:

$$V_{1,\mathrm{POST}} \overset{!}{<} \frac{B_+ \, M_{\mathrm{mid}}}{R_z \left[1 - e^{\frac{-t_{1,\mathrm{POST}}}{C_Z \, R_Z}}\right]} \tag{4.6}$$

The corresponding lower bound of $V_{1,\mathrm{POST}}$ can be derived from the PRE-and-POST case, which is supposed to trigger LTP. In order to match the theory, also small inter event intervals Δt should suffice to lead to a potentiation of an affected synapse's weight. Translating this into requirements to the pulse shapes, in contrast to the previous case we do want the superposition of the pre-synaptic pulse and the positive sub-pulse of the post-synaptic pulse to drive V_z beyond B_+. Solving Equation 4.1 with these parameters with respect to $V_{1,\mathrm{POST}}$ again, this results in

$$V_{1,\mathrm{POST}} \overset{!}{>} \frac{B_+ \, M_{\mathrm{mid}}}{R_z \left[1 - e^{\frac{-t_{1,\mathrm{POST}}}{C_Z \, R_Z}}\right]} + V_{1,\mathrm{PRE}} \, . \tag{4.7}$$

Plugging the values for the model parameter defined in Table 2.2 along with the other values we chose freely into Equations 4.6 and 4.7, we obtain the legitimate range for $V_{1,\mathrm{POST}}$:

$$\boxed{32.168 \, mV < V_{1,\mathrm{POST}} < 37.998 \, mV} \tag{4.8}$$

Based on this range, finally $V_{1,\mathrm{POST}} = 37 \, mV$ was chosen.

Given the value for the amplitude of the post-synaptic pulse's first sub-pulse, a range for the amplitude of the negative sub-pulse can be determined, again utilizing different constraints. The first of these constraints forces $V_{2,\mathrm{POST}}$ to be negative enough to drive V_z below B_- in the POST-only case, in order to trigger the required LTD. An appropriate range of amplitudes $V_{2,\mathrm{POST}}$, which satisfy this requirement, can be derived using the definition of the drive effort V_z given in Equation 4.1 again. In contrast to the previous ranges, however, a preload $V_{z,0}$ has to be considered in this case. This is due to effect of the post-synaptic pulse's first sub-pulse starting to decay at the moment the to be determined negative sub-pulse takes effect. One can account for this by interpreting the POST-pulse as defined by the D-pulses scheme as a succession of two piecewise linear and constant input voltages, both of which are applied for a limited amount of time. The still persisting effects of the first sub-pulse are then considered as corresponding pre-loads $V_{z,0}$ in Equation 4.1. In the case of the D-pulses, the preload corresponds to the value of the drive effort V_z at the end of the first sub-pulse, that is $V_{z,0} = V_z(t_{1,\text{POST}})$. In addition to considering this preload, we also perform an implicit time shift such that the second sub-pulse starts in the origin, that is at $t = 0$, of the new time axis. The term $V_z(t_{1,\text{POST}})$, however, shall still denote the drive effort V_z at time $t_{1,\text{POST}}$ relative to the old origin.

Rewriting the expression resulting from all these assumptions, we obtain the upper bound for the amplitude of the post-synaptic pulse's negative sub-pulse shown in Equation 4.9. This amplitude corresponds to the most positive voltage, only just capable of correctly triggering LTD in the POST-only case.

$$V_{2,\text{POST}} \overset{!}{<} \frac{\left[B_- - V_z(t_{1,\text{POST}})\, e^{\frac{-t_{2,\text{POST}}}{C_z R_z}}\right] M_{\text{mid}}}{R_z \left[1 - e^{\frac{-t_{2,\text{POST}}}{C_z R_z}}\right]} \tag{4.9}$$

Given this upper bound for negative sub-pulse's amplitude, the corresponding lower bound of $V_{2,\text{POST}}$ can be derived from yet another constraint. Namely, it was decided to demand that the drive effort V_z shall not exceed U_- in any case. This constraint was introduced due to the parameter function $\phi(V_z)$ suddenly changing its sign at $V_z = U_-$ (compare Figure 2.2a). First and foremost, because the sign of $\phi(V_z)$ turns from negative to positive when V_z exceeds U_-, the negative sub-pulse would trigger partial LTP even in the POST-only case. Having partial LTP in the POST-only case is highly undesirable, since it could threaten the overall system stability. Secondly, in addition to being undesirable from a theoretical point of view, the reason for the sudden sign change of $\phi(\cdot)$ could not be figured out. Hence, it was found to be a good idea to avoid coming into contact with it. Based on this, the corresponding lower bound

Name	Value	Name	Value
$V_{1,\mathrm{PRE}}$	$-5.83\,mV$	$t_{1,\mathrm{PRE}}$	$10\,ms$
$V_{1,\mathrm{POST}}$	$37\,mV$	$t_{1,\mathrm{POST}}$	$0.5\,ms$
$V_{2,\mathrm{POST}}$	$-43\,mV$	$t_{2,\mathrm{POST}}$	$1\,ms$

Table 4.2: Summary of the parameters describing D-pulses. Parameterizing the pulses illustrated in Figures 4.7 and 4.8a with the above parameter set yields a pulsing scheme capable of triggering qualitatively correct STDP in the presented memristor model.

for $V_{2,\mathrm{POST}}$ can be derived as

$$V_{2,\mathrm{POST}} \overset{!}{>} \frac{\left[U_{-} - V_z(t_{1,\mathrm{POST}})\, e^{\frac{-t_{2,\mathrm{POST}}}{C_z\,R_z}}\right] M_{\mathrm{mid}}}{R_z \left[1 - e^{\frac{-t_{2,\mathrm{POST}}}{C_z\,R_z}}\right]} . \tag{4.10}$$

Plugging the parameters defined in Table 2.2 along with the values for $V_z(t_{1,\mathrm{POST}})$ according to Equation 4.1 as well as $t_{2,\mathrm{POST}} = 1\,ms$ into Equations 4.9 and 4.10, we obtain a legitimate range for $V_{2,\mathrm{POST}}$ of

$$\boxed{-43.756\,mV < V_{2,\mathrm{POST}} < -34.937\,mV} . \tag{4.11}$$

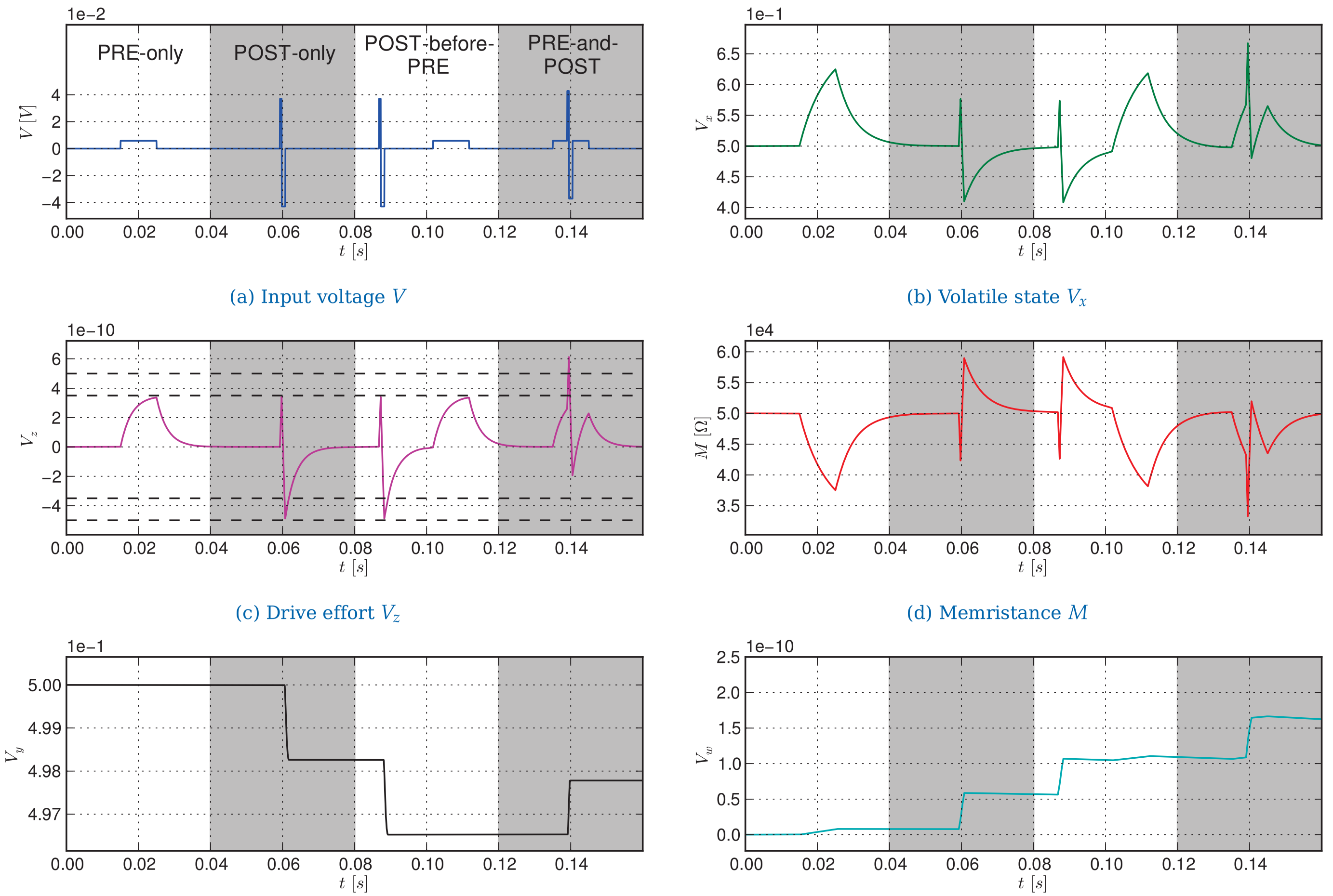
Ensuring a certain safety margin, finally an amplitude of $V_{2,\mathrm{POST}} = -43\,mV$ was chosen. Given this value, all four parameters required to fully describe the D-Pulses' post-synaptic pulses are defined. Table 4.2 summarizes all of them along with the parameters chosen for the pre-synaptic pulses.

In order to evaluate the system's reaction when being pulsed according to the D-scheme, different combinations of the pre- and post-synaptic pulses with regard to Table 4.1 were applied to a memristor initialized with standard parameters. This was achieved by configuring Protocol 5 described in Section 3.3.4 to generate either single pulses or single pulse pairs with an appropriate inter event interval Δt. Figure 4.9 shows the reactions of the memristor model's internal state variables as well as the corresponding input-output-behavior resulting from these events.

As can be seen from the $V_y(t)$- and $M(t)$-plots (Figures 4.9e and 4.9d), in combination with the PRE-pulses described above, the chosen D-POST-pulses are indeed capable of triggering the desired reactions in terms of the induced synaptic plasticity. In the POST-only case, for instance, the instantaneous memristance $M(t)$ is increased by a certain amount, corresponding to LTD being triggered. While this might not be that obvious from the red memristance plot, this circumstance can be inferred from the black V_y-plot. As we remember from Section 2.2.2, although V_x is the quantity defining the course of $M(t)$, the non-volatile portion of the memristance is governed by V_y which V_x decays to after some time. As we can see, in the POST-only case the post-synaptic pulse reduces V_y by a certain amount. Given the relationship between V_y and V_x according to Equation 2.12 as well as the definition of the memristor's interface in Equation 2.10, smaller V_y values translate to a greater instantaneous memristances $M(t)$, corresponding to weaker synaptic junctions. The same applies to the POST-before-PRE case. In contrast to this, in the PRE-and-POST case, the D-pulse scheme causes V_y to be boosted by a certain amount, translating to the instantaneous memristance being reduced and thus the corresponding conductance being intensified, which ultimately translates to LTP being triggered. Finally, without any additional overlapping post-synaptic pulse, just as expected, the PRE-pulse as defined in Table 4.2 does not trigger any non-volatile changes.

Also, as can be seen from Figure 4.8b, the results in terms of the triggered STDP behavior according to Protocol 3 yielded by the D-pulses can be considered quite promising. Negative inter event intervals Δt, that is, instances of the POST pulse occurring before the PRE-pulse, induce a negative conductance change ΔG, translating to a reduction of the synaptic efficacy. Starting with $\Delta t = 0$, that is, in case the POST-pulse is triggered after the PRE-pulse, the conductance change turns positive. For $\Delta t > 9\,ms$, finally, the weight update triggered by a single pulse pair starts to become smaller, turning negative again for $\Delta t = 13\,ms$.

Having a closer look at $V_x(t)$ and the $M(t)$-plots, however, reveals a potential flaw in the proposed D-pulse shapes. In case of the POST-pulse occurring without a corresponding PRE-pulse, it takes $V_z(t)$ quite some time to return the neutral state $V_z(t) \approx 0$ again. Similar as for V_x, also for V_z transient processes like this are undesirable in hardware implementations of spiking WTA networks. The attempt to remedy this drawback leads



(a) Input voltage V

(b) Volatile state V_x

(c) Drive effort V_z

(d) Memristance M

(e) Non-volatile state V_y

(f) Absolute power dissipation V_w

Figure 4.9: System response to different events composed of D-pulses. As can be seen from (d), the effect of single events on the non-volatile portion of the resting memristance is small compared to the memristance's dynamic range. The persistent changes triggered by the respective events are, however, clearly visible from (e) illustrating the evolution of the non-volatile state V_y. Likewise, the exceedance of different thresholds $B_\pm$ and $U_\pm$ by V_z can be observed from (c), where the thresholds are indicated by the four dashed lines.

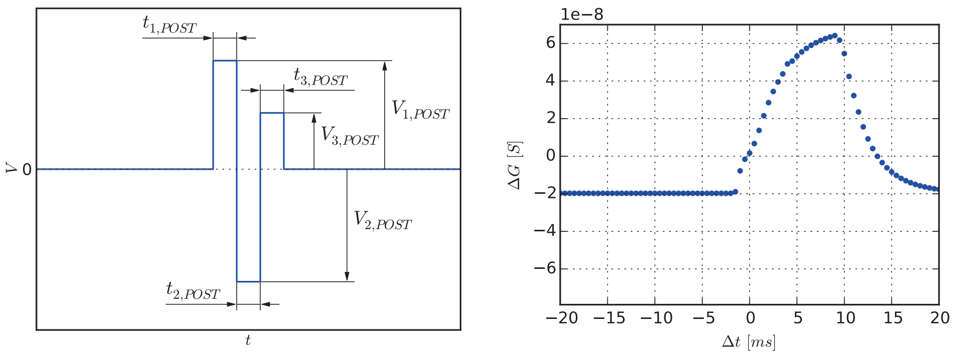
to the development of the *T-pulses*, an alternative type of POST-pulses, which will be discussed in the next section.

4.3.2 T-Pulses

The bipolar D-pulses basically satisfy, as we have seen in the previous section, all the requirements we defined for pulses being used in order to trigger synaptic plasticity in memristive synapses. In addition to this, as is emphasized by the STDP-plot presented in Figure 4.8b, also the results in terms of synaptic plasticity themselves obtained for D-pulses are promising. As already indicated, however, D-pulses introduce a potential drawback into the system. More precisely, after POST-only- and PRE-before-POST-events it takes the drive effort V_z some time to relax to the neutral state $V_z \approx 0$. As we mentioned earlier in Section 4.1, long durations for transient processes like this are undesirable in hardware implementations of spiking WTA networks, as they could lead to adverse superposition side-effects, ultimately reducing the overall system stability. In an attempt to account for this, it was decided to extend the existing D-pulses by a third sub-pulse, yielding *T-pulses*. By analogy with the previous section, the letter T is an abbreviation for *tripple* derived from the number of constituent sub-pulses featured by the POST-pulses according to this pulsing scheme.

This additional third sub-pulse is supposed to act as some kind of compensation pulse driving V_z back towards $V_z \approx 0$ in the POST-before-PRE and POST-only cases. Obviously, the assigned task of compensating a negative V_z calls for a positive amplitude for the to be determined third sub-pulse. This leads to the new pulse shape as illustrated in Figure 4.10a in a qualitative manner, with $V_{3,\text{POST}}$ and $t_{3,\text{POST}}$ as two additional parameters which need to be defined.

For both symmetry reasons as well as to keep the overall length of the POST-pulse short compared to the pre-synaptic one, $t_{3,\text{POST}} = 0.5\,ms$ was chosen. Based on this duration, an appropriate corresponding amplitude $V_{3,\text{POST}}$ can be determined. For this purpose, again a preload $V_{z,0}$ has to



(a) POST-pulse (b) STDP-plot

Figure 4.10: Qualitative illustration of the post-synaptic T-pulse and STDP plot. Comparing plot (b) with Figure 4.8b shows that for T-pulses the POST-pulse's third sub-pulse has clearly increased the strength of LTP.

be considered, which is set to the value of V_z at the end of the preceding sub-pulse. In addition to this, also the implicit time shift is performed again, such that the third sub-pulse starts in the origin. $V_z(t_{2,\text{POST}})$ shall, however, still denote the drive effort with respect to the old time axis. Trying to keep the relationships between the different amplitudes as simple as possible, it was decided to demand full compensation of the negative portion of V_z by the end of the POST-pulses' third sub-pulse in the POST-only case. Based on these requirements, $V_{3,\text{POST}}$ can be determined utilizing Equation 4.1 again as

$$V_{3,\text{POST}} \overset{!}{=} -\frac{V_z(t_{2,\text{POST}})\, e^{\frac{-t_{3,\text{POST}}}{C_z\, R_z}}\ M_{\text{mid}}}{R_z \left[1 - e^{\frac{-t_{3,\text{POST}}}{C_z\, R_z}}\right]} \,. \tag{4.12}$$

Plugging the model parameters defined in Table 2.2 along with the expression for the preload and the above mentioned value $t_{3,\text{POST}} = 0.5\,ms$ into Equation 4.12 yields

$$\boxed{V_{3,\text{POST}} = 44.768\,mV} \,. \tag{4.13}$$

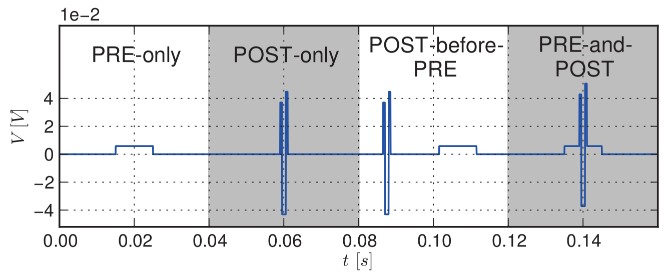
Based on this estimation, finally $V_{3,\text{POST}} = 44.7\,mV$ was chosen in order not to overcompensate the negative drive effort.

Name	Value	Name	Value
$V_{1,\text{PRE}}$	$-5.83\,mV$	$t_{1,\text{PRE}}$	$10\,ms$
$V_{1,\text{POST}}$	$37\,mV$	$t_{1,\text{POST}}$	$0.5\,ms$
$V_{2,\text{POST}}$	$-43\,mV$	$t_{2,\text{POST}}$	$1\,ms$
$V_{3,\text{POST}}$	$44.7\,mV$	$t_{3,\text{POST}}$	$0.5\,ms$

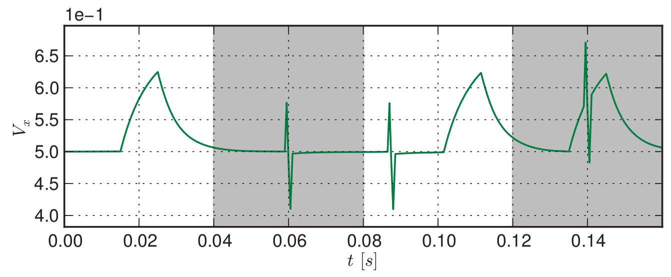
Table 4.3: Summary of the parameters describing T-pulses. Instances of the pulses illustrated in Figures 4.7 and 4.10a according to the above parameters are induce proper STDP behavior in memristive synapses. The additional compensation pulse ensures compensation of V_z in the POST-only and POST-before-PRE cases.

As an aside, a potential pitfall which has to be taken care of should be mentioned at this point. Inappropriately large values for $V_{3,\text{POST}}$ could lead to situations in which LTP is triggered even for POST-before-PRE events. More precisely, for very small negative inter event interval Δt, the remaining partial overlap of the PRE- and the POST-pulses could cause V_z to exceed B_+, if the amplitude of $V_{3,\text{POST}}$ is chosen too high. For the values listed in Table 4.3, however, this problem was not observed.

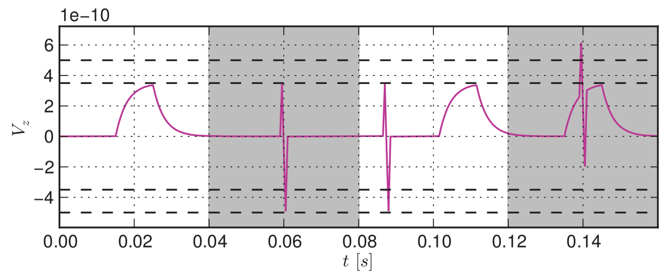
In order to allow for a direct comparison of this new pulsing scheme with D-pulses described in the previous section, the same simulations as described above were repeated. This time, however, the different events described in Table 4.1 were represented using T-pulses. Figure 4.11 shows the resulting evolution of the memristor model's different internal state variables as well as the input-output-behavior again. As becomes clear from Figures 4.11b and 4.11d, the additional third sub-pulse introduced with the T-pulse indeed aids in speeding up the relaxation of $V_z(t)$ after POST-only or POST-before-PRE events. Obviously, also the PRE-and-POST case is affected by the third sub-pulse. As we can see, compared to the plot shown in Figure 4.9c, the drive effort V_z illustrated in Figure 4.11c is pushed further towards the positive bipolar threshold B_+. Ultimately, compared to D-pulses, this will lead to LTP becoming stronger than LTD in a relative sense. This becomes also evident when comparing the STDP plots shown in Figures 4.8b and 4.10b. While for D-pulses the STDP curves' minimum and maximum are more or less symmetric with respect to the Δt-axis, this is not the case for T-pulses. More precisely, comparing the minimum and maximum values, for T-pulses LTP is approximately



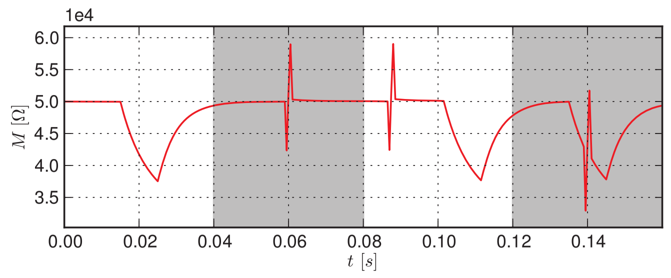
(a) Input voltage V



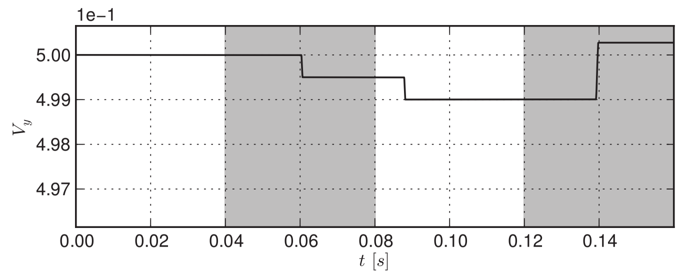
(b) Volatile state V_x



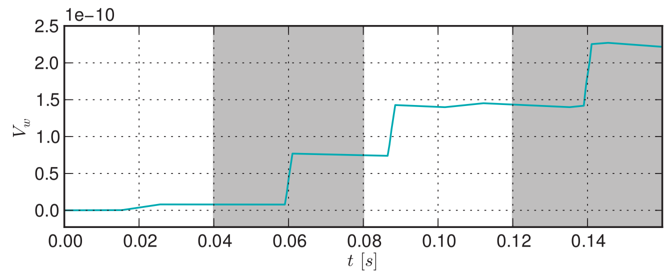
(c) Drive effort V_z



(d) Memristance M



(e) Non-volatile state V_y



(f) Absolute power dissipation V_w

Figure 4.11: System response to different events composed of T-pulses. Again, as can be seen from (d), the changes to the resting memristance are small compared to its dynamic range. Similar as in Figure 4.9, however, the persistent changes are again clearly visible from the illustration of V_y shown in (e) as well as the evolution of V_z shown in (c) (dashed black lines again illustrate the thresholds $B_\pm$ and $U_\pm$).

three times stronger than LTD. We will return to this observation later in Section 4.4 in the context of weight dependence and convergence. For the moment, however, we conclude the discussion of T-pulses, the second pulse shape investigated in this work.

4.3.3 S-Pulses

As a side note it should be mentioned that also experiments with a third pulse type, the so-called *S-pulses*, were made. In contrast to the D- and T-pulses described in the previous two sections, for this pulsing scheme both the proposed pre- and the post-synaptic pulses are unipolar. More precisely, on the one hand, similar to the D- and T-pulses, also for S-Pulses the amplitude of the pre-synaptic pulse was chosen such that the resulting driving effort V_z is bound to the range (B_-, B_+). Its polarity, however, was chosen to be positive. On the other hand and in contrast to D- and T-pulses, for S-pulses the post-synaptic pulse is limited to a *single* square pulse (which the abbreviating *S* is derived from). Its amplitude was chosen negative, such that the resulting V_z values are below B_-, close to but yet above U_-.

Given the parameter function $\phi(V_z)$ illustrated in Figure 2.2a, in theory this design allows for correct implementation of the four cases regarding synaptic plasticity as follows. On the one hand, similar to above, PRE-only would not trigger any change in the synaptic weight since $\phi(\cdot)$ is equal to zero in the affected V_z-range. POST-only as well as POST-before-PRE would lead to LTD by pulling the driving effort V_z into the negative bipolar region, resulting in negative values for $\phi(V_z)$. Finally, due to the positive amplitude of the pre-synaptic pulse and the way the memristor's input voltage is defined in Equation 4.2, V_z would be driven below U_- in the PRE-and-POST case. As can be seen from Figure 2.2a, the parameter function $\phi(\cdot)$ is positive for arguments smaller than U_-, resulting in positive values for $\frac{V_y}{dt}$ according to Equation 2.15. Hence, also in the PRE-and-POST case, S-pulses would be able to trigger the correct reaction in terms of synaptic plasticity by reducing the memristance, leading to LTP.

As mentioned earlier in Section 4.3.1, however, the reason for the sudden sign change of $\phi(\cdot)$ at U_-, which is utilized explicitly by S-pulses, is not known. This is why it was decided not to investigate this third pulse type any further although it seemed reasonable from a theoretical point of view.

These thoughts conclude both the discussion of the pulse shapes used in combination with memristive synapses in particular as well as the investigations of problems in the context of single spike setups in general. The following two sections are devoted to problems related to the long term behavior of the memristive synapses. That is, problems and requirements which arise in multi spike environments with several thousands of spikes or spike pairs. Since these spike pairs will be generated according to PRE/POST statistics, these setups introduce some degree of stochasticity. As indicated earlier, the models used for these experiments already incorporate all the insights gained in the previous three sections. That is, on the one hand the memristor model was configured to use the C_x value determined in Section 4.1 leading to a realistic parameter set. On the other hand, the memristive synapses were extended by the ohmic pre-resistor and driven by T-pulses as described in Sections 4.2 and 4.3.2, respectively. Based on this modified memristor model, the first problem to be investigated in the next section will be related to weight dependence as well as convergence issues.

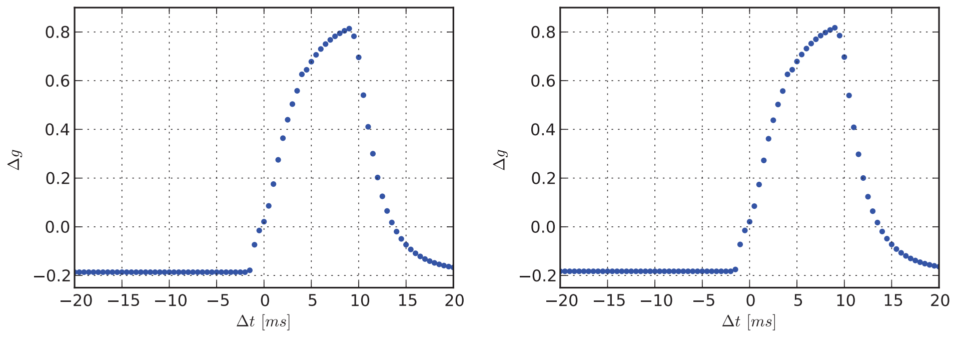
4.4 Weight Dependence and Convergence

In the previous section we identified two different pulse shapes that are capable of driving our memristive synapses in such a way that, depending on the relative timing of the respective neural spikes, different reactions in terms of synaptic plasticity are triggered. Although, as we have seen from the plots shown in Section 4.3, both D- and T-pulses basically trigger decent STDP, an additional building block is required for our setup in order to finally make memristors usable as synapses for spiking WTA networks backed by the SEM learning theory.

More precisely, besides having pulses of appropriate shape at hands, the memristor model itself has to offer support for weight dependence of some sort. In this context, the term weight dependence means that the amount a certain event changes the weight represented by a memristive synapse shall depend on the weight currently encoded by this synapse. This is a crucial point when it comes to encoding certain PRE/POST statistics in the synapse's weight. Put differently, this behavior is necessary in order to ensure convergence of the synapse's conductance to a certain value in response to any given PRE/POST statistics. Without weight dependence, over short or long the memristor's conductance would be simply driven into the bounds $\frac{1}{M_{\text{max}}}$ and $\frac{1}{M_{\text{min}}}$, respectively, depending on which type of event (that is, LTD or LTP) is predominant in the input being applied to the device.

As an aside, it should be emphasized at this point that our setup of having the synapses' weight being linked to the memristor's instantaneous conductance $G(t)$ actually already implements some sort of weight dependence. More precisely, as a direct consequence of the non-linear mapping between the internal state variables and the corresponding synaptic weight (compare Section 4.2), for larger conductance values the absolute weight increment or decrement triggered by a single potentiating or depressing event will be greater than for smaller ones. In this section, however, we are interested in a different type of weight dependence. What we are looking for here is a way of making one type of event (LTP or LTD) stronger than the other, depending on the weight currently encoded by the affected synapse.

Having a closer look at the memristor model by Serb et al. [Serb, 2014] we introduced in Section 2.2.2, we notice that out of the box the model as such does not provide support for this kind of weight dependence. This is also clearly visible from STDP plots shown in Figure 4.12. Note that, in contrast to the STDP plots shown in the previous section, these STDP plots are normalized, that is, they were scaled such that the STDP curves have unity height. As we can see, although for this simulation the synapses were initialized to entirely different initial states (midrange memristance for Figure 4.12a and midrange conductance for Figure 4.12b), the resulting



(a) Midrange memristance

(b) Midrange conductance

Figure 4.12: Normalized STDP plots for T-pulses and different initial states. In contrast to the STDP plots shown earlier, these plots are scaled to unity height. As can be seen, although the memristors were initialized to different states, their responses to Protocol 3 are almost identical, indicating that proper weight dependence is not supported by the original memristor model.

STDP curves are almost identical. Obviously, if the underlying memristor model had proper weight dependence in place, this would not be the case. Although, the shape of the curve itself would not change that much, one of them would be shifted by a certain vertical offset compared to the other one, illustrating that one of the two LTP and LTD is stronger for a given memristor state.

In addition to the STDP plots shown in Figure 4.12, the missing support for weight dependence becomes also evident from Figure 4.13 created using Protocol 7. As we can see, for the original memristor model as proposed by Serb et al. [Serb, 2014], the applied input consisting of 20000 POST-pulses (a certain fraction of which is overlayed a corresponding PRE-pulse) drives the conductance either into its upper or its lower bound. No dynamic equilibrium is reached, as would be the case if the memristor model offered support for weight dependence. Besides this, also another observation can be made from the plot in Figure 4.13. As we can see, for three out of eleven of the probabilities $p(Y = 1 \mid Z = 1)$ (that is, fractions of LTP events), the instantaneous conductance G runs into the maximum conductive state. This translates to LTP being considerably stronger than LTD for the chosen setup. This observation is perfectly aligned with what

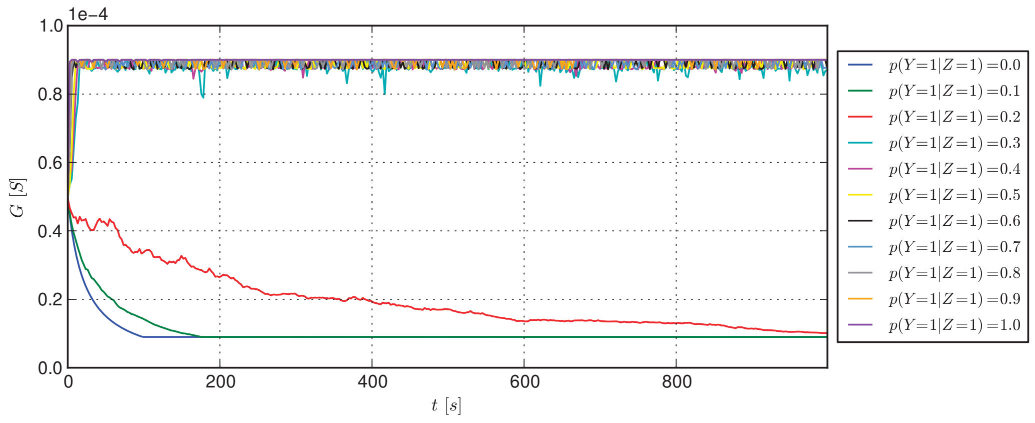


Figure 4.13: Temporal evolution of the conductance for different input statistics. Without support for proper weight dependence, the conductance $G(t)$ runs into the upper and lower bounds. No dynamic equilibrium is reached for any of the input statistics $p(Y = 1 \,|\, Z = 1)$. Since the conductance runs into its upper bound in the majority of the cases, LTP can be considered stronger in the given setup utilizing T-pulses.

we saw from the STDP plots shown in Figure 4.12. As we can see from Figures 4.12a and 4.12b, for the inter event interval $\Delta t = 4\,ms$ used by Protocol 7, the magnitude of the encountered Δg value is much greater for the LTP case than for any of the LTD cases (that is, any of the negative Δt values). This imbalance is, as already indicated in Section 4.3.2, primarily a result of the chosen T-pulses.

Looking now for an appropriate place for the weight dependence feature to be added to the existing model, one has to take a closer look at the system's dynamics. As we remember, the memristor model splits up the instantaneous memristance into a volatile and a non-volatile portion. Clearly the non-volatile part of the memristance defined in Module III is where long term weight dependence has to be implemented. As can be seen from the corresponding module definition given in Section 2.2.2 (Equation 2.15), the dynamics of Module III are governed by V_y. V_y, in turn, is controlled by the three parameter functions $\phi(\cdot)$, $g(\cdot)$ and $f(\cdot)$ being applied to the state variables V_z, V_w as well as V_y itself, respectively. The first one of these, $\phi(\cdot)$, describes certain thresholds that control whether the driving strength V_z suffices to trigger non-volatile memristance changes.

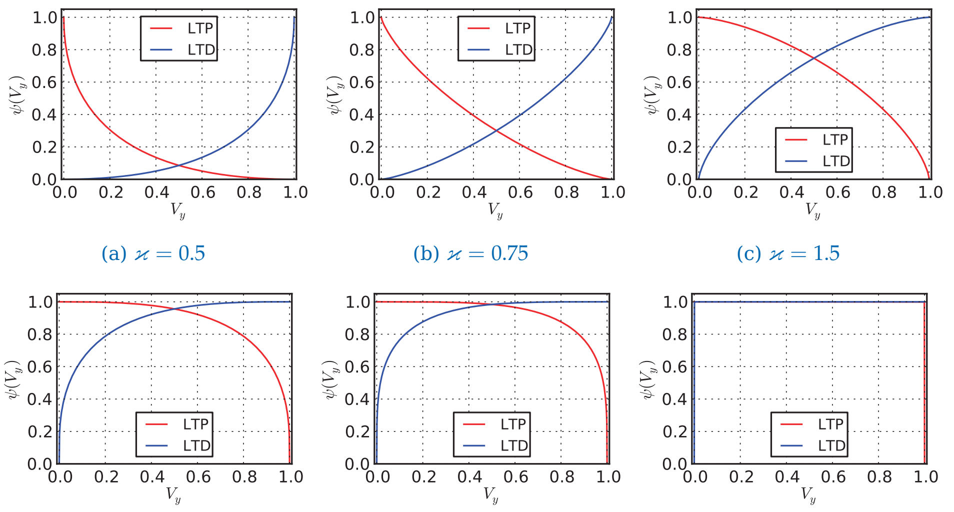
The role of this function was found to be too fundamental as to be changed in any way. In addition to this, with $g(\cdot)$ and $f(\cdot)$ better candidates for the implementation of weight dependence were available. This seemed reasonable, since on the one hand, as stated by Serb et al. [Serb, 2014], $g(\cdot)$ was not directly linked to observations made with real memristor hardware. Rather $g(\cdot)$ was fitted in order to match data measured for real biologic synapses. On the other hand, as mentioned in Section 2.2.2, the role of $f(\cdot)$ is merely confining the instantaneous memristance $M(t)$ to the bounds M_{min} and M_{max}. As we remember, this is achieved by implementing $f(\cdot)$ as a sharp-edged window function on the interval $(0, 1)$. However, as seems quite intuitive and as becomes evident from the plot shown in Figure 4.13, dynamic systems involving sharp edged window functions like $f(\cdot)$ tend to drive the affected system variables into their bounds. Besides being undesirable, behavior like this is not supported by data observed for hardware memristors. This gives rise to the assumption that also the window function $f(\cdot)$ could be an appropriate place for our desired weight dependence to be implemented without violating fundamental memristor principles.

In order to both keep the system simple as well as to get rid of the undesired sharp edged window function $f(\cdot)$, it was decided to set up a new parameter function as a combination of both $g(\cdot)$ and $f(\cdot)$, such that it undertakes both of the above-mentioned tasks. Put differently, it was decided to choose the new window function in such a way that it is capable of both introducing wight dependence to the memristor model as well as confining the instantaneous memristance to the above-mentioned bounds M_{min} and M_{max}.

A possible choice for this new and combined window function $\psi(\cdot)$ (note the new name which was chosen trying to avoid any ambiguities) that offers all the above-mentioned feature is shown in Equation 4.14:

$$\psi(V_y) = \begin{cases} \left(1 - V_y^{\varkappa_{\text{P}}}\right)^{\frac{1}{\varkappa_{\text{P}}}} & \text{if } V_z > 0 \\ \left(1 - \left(1 - V_y\right)^{\varkappa_{\text{D}}}\right)^{\frac{1}{\varkappa_{\text{D}}}} & \text{else} \end{cases} \tag{4.14}$$

As we can see, this function explicitly distinguishes between the LTP and LTD case. Similar as for the original parameter function $g(\cdot)$, in the model



(a) $\varkappa = 0.5$ (b) $\varkappa = 0.75$ (c) $\varkappa = 1.5$

(d) $\varkappa = 3.0$ (e) $\varkappa = 4.0$ (f) $\varkappa = 400.0$

Figure 4.14: Window function $\psi(\cdot)$ for different values of $\varkappa$. For the six plots, the symmetric version of $\psi(\cdot)$ was chosen, that is, $\varkappa_\mathrm{P} = \varkappa_\mathrm{D} = \varkappa$. As suggested by (f), for large values of $\varkappa$, the proposed window function $\psi(\cdot)$ degenerates to the original sharp edged window function $f(\cdot)$.

this distinction can be made by examining the current sign of the driving effort V_z. This holds, since V_z is ultimately responsible for triggering either LTP or LTD as, through $\phi(\cdot)$, it determines the sign of $\frac{dV_y}{dt}$. Plugging the new parameter function $\psi(\cdot)$ into Equation 2.15, we obtain the new system equation for Module III:

$$C_y \frac{dV_y}{dt} = i'_y(t) = \phi(V_z, B_+, B_-, U_+, U_-)\, \psi(V_y, V_z, \varkappa_\mathrm{P}, \varkappa_\mathrm{D}) \tag{4.15}$$

Besides having two branches, another remarkable property of the new parameter function $\psi(\cdot)$ is that the only parameters to be chosen in order to fully determine its exact shape are the values for $\varkappa_\mathrm{P}$ and $\varkappa_\mathrm{D}$. However, to keep things simple and uncluttered for the moment, we focus on the case $\varkappa_\mathrm{P} = \varkappa_\mathrm{D} = \varkappa$ in this section, that is, we use the same shape parameter value for both the LTP and the LTD branch of Equation 4.14. Figure 4.14 illustrates $\psi(V_y)$ for the case $\varkappa_\mathrm{P} = \varkappa_\mathrm{D} = \varkappa$ for different values of $\varkappa$.

As we can see, the function $\psi(\cdot)$ basically fulfills both of our main require-

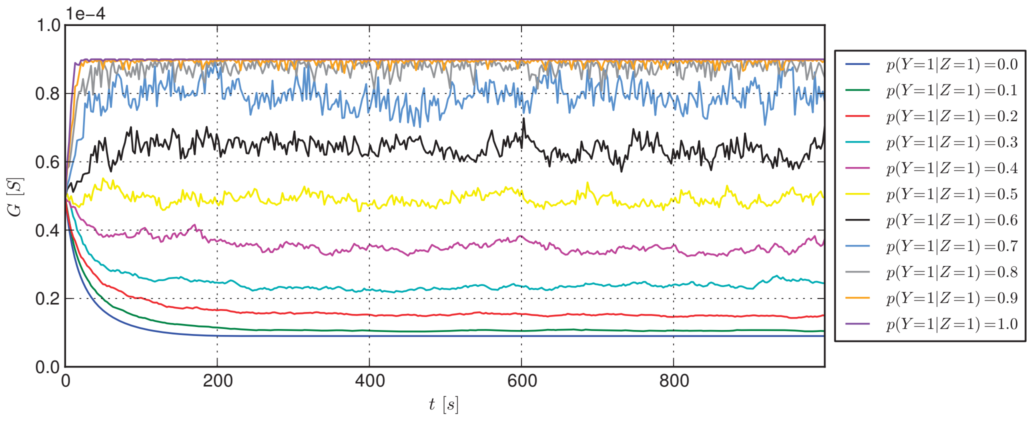


Figure 4.15: Temporal evolution of the conductance for different input statistics. Clearly, the adapted window function $\psi(\cdot)$ has introduced weight dependence into the system. A distinct convergence conductance is reached for almost any of the different input statistics $p(Y = 1 \mid Z = 1)$.

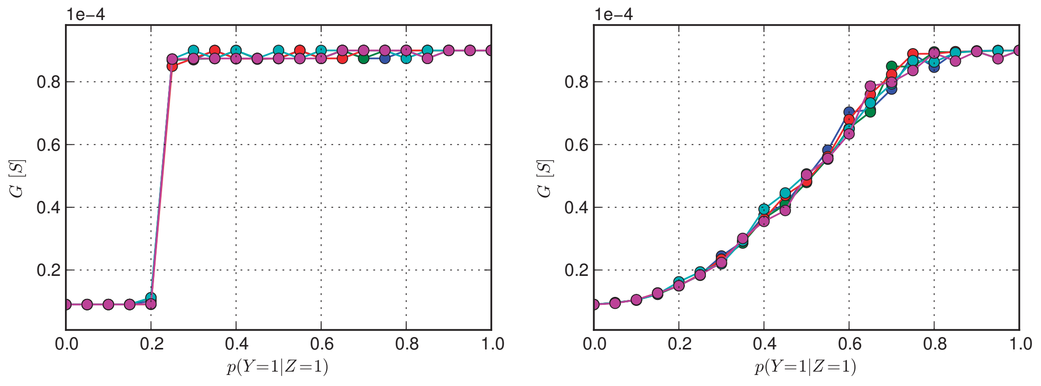
ments. On the one hand, it is bound to the interval $(0, 1)$. Hence, also V_y can be considered confined to the interval $(0, 1)$, thus forcing M to be between M_{min} and M_{max} again. On the other hand, also weight dependence seems to be implemented properly. For high conductive states (V_y values around 1.0), the LTP branch leads to much smaller function values of $\psi(\cdot)$ than the LTD branch does. Hence, in this situation (synaptic weight is already strong) the relative weight increment produced by a single potentiating event will be much smaller than the relative weight decrement induced by a single depressing event. This makes LTD dominant in this case. Likewise, if the memristor is in a low conductive state (V_y values around 0.0), the LTP branch leads to larger $\psi(V_y)$ values than the LTD branch does. Thus, in this situation (synaptic weight is already small), the relative weight increment produced by potentiating events will be larger than the weight decrement triggered by depressing events. Hence, LTP will be dominant in this situation. Overall, the interaction of both cases is supposed to lead to our desired type of weight dependence, that is, achieving a dynamic equilibrium. This, in turn, is supposed to make the model stable and converge to a certain conductance value for any given PRE/POST statistics.

In order to examine the actual suitability of the proposed window function

$\psi(\cdot)$ for adding this type of weight dependence to the existing memristor model, the simulation mentioned earlier in this section was repeated. Instead of using the original memristor model, however, the generated input was applied to a memristor whose parameter functions $g(\cdot)$ and $f(\cdot)$ were replaced for the new and combined window function $\psi(\cdot)$ as defined in Equation 4.14. The value for its free shape parameter $\varkappa$ was chosen as $\varkappa = 1.5$. All other simulation parameters (initial memristance state, number of pulses, inter event interval and so on) were left unchanged.

Figure 4.15 shows a plot illustrating the evolution of the conductance resulting from this setup. Comparing this plot to the one shown in Figure 4.13, we notice that the stability of the system has clearly improved. More precisely, almost any of the probabilities $p(Y = 1 \mid Z = 1)$ leads to a distinct convergence conductance. This indicates that, in contrast to the original memristor model, for the modified model featuring the newly proposed window function $\psi(\cdot)$, a dynamic equilibrium is reached for most of the input statistics. In a spiking WTA network, this translates to the quality of the $p(Y = 1 \mid Z = 1)$/weight mapping having improved. This mapping is illustrated in Figure 4.16 for both the model employing the original parameter functions $g(\cdot)$ and $f(\cdot)$ as well as the modified model featuring our proposed combined parameter function $\psi(\cdot)$. Obviously, the linearity of the mapping between the probabilities $p(Y = 1 \mid Z = 1)$ and the corresponding convergence conductances has improved significantly. While the mapping resembles the shape of a step function for the original memristor model it is almost linear for $p(Y = 1 \mid Z = 1) \leq 0.8$ in case of the modified model utilizing the new window function $\psi(\cdot)$. In addition to being almost linear, within this interval the mapping is more or less unique, allowing for bijectivity. That is, any arbitrary PRE/POST statistics imply a certain convergence conductance G and vice versa. In a spiking WTA network with memristive synapses, this translates to a unique and bijective mapping between the probability $p(Y = 1 \mid Z = 1)$ encoded in the applied input and the weight the affected synapse converges to.

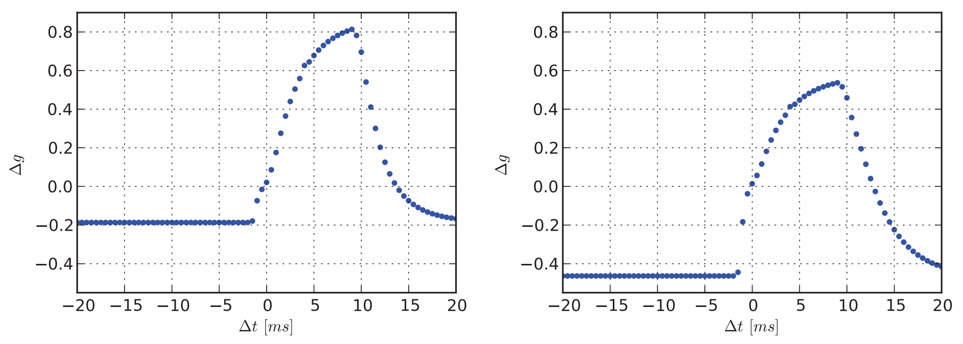
Although the simulations based on Protocol 7 allow for the examination of the system's long term behavior, the gained insights are limited to a single inter event interval Δt. In order to account for this and to gain further



(a) Model with $f(\cdot)$ and $g(\cdot)$ (b) Model with $\psi(\cdot)$

Figure 4.16: Mapping between input statistics and convergence conductances. The plots were generated by running Protocol 7 according to Section 3.3.5 on two memristors. One of them was set up to use the original combination of $f(\cdot)$ and $g(\cdot)$ as parameter functions, while the other one was assigned the modified window function $\psi(\cdot)$.

insights, the experiments based Protocol 3 described earlier in this section and related to investigation of the STDP behavior were repeated for the updated memristor model. The results obtained from these simulations can be seen in Figure 4.17. Again, two STDP plots were created by applying a single spike pair with varying inter event interval to two memristors which were previously initialized to two different initial states. As we can see, in contrast to Figure 4.12, for the memristor model featuring the new parameter function $\psi(\cdot)$, the normalized STDP plots yielded for different memristor states are not equal to each other anymore. Specifically, as indicated by Figure 4.17a for the already relatively low conductive state represented by the initialization to midrange memristance, LTP is clearly stronger than LTD. Since in our synapse model, low conductance corresponds to a low synaptic weight, the reaction of making LTP pulses dominant in this situation is perfectly aligned with what was required from the new parameter function $\psi(\cdot)$ earlier. Likewise, the STDP plot shown in Figure 4.17b suggests that for midrange conductance and an average inter event interval with respect to the width of the STDP learning window τ, both LTP and LTD are almost equally strong. Given that in our synapse model midrange conductance corresponds to a midrange weight,



(a) Midrange memristance (b) Midrange conductance

Figure 4.17: Normalized STDP plots for different initial states and the modified window function $\psi(\cdot)$. Clearly, the modified window function $\psi(\cdot)$ has introduced weight dependence into the system. While LTP is predominant for the relatively low weight represented by midrange memristance, LTP and LTD are almost equally strong in case of the initialization to midrange conductance and a medium inter event interval of $\Delta t \approx 5\,ms$.

also this reaction is aligned with our initial requirements.

These results obtained for the simplified version of the proposed new parameter function $\psi(\cdot)$ were found to be very promising with respect to the correct implementation of our desired weight dependence as required by SEM. As an aside it should be mentioned at this point that, in addition to apparently implementing proper weight dependence, the shape of the simplified version of $\psi(\cdot)$ in general was confirmed as being plausible in the context of real memristor hardware by an expert on memristor solid state chemistry [Bill, 2014b]. It remains to be seen, however, whether nano-scale memristive devices showing this type of weight dependence can be manufactured in big scale with the required reliability.

4.5 Independence from Initialization

In the experiments we described so far, we assumed that we had full control over the initial memristor state. More precisely, for different experiments, we specifically initialized the involved memristors to certain memristance or conductance values. In contrast to regular circuit elements, however, currently state-of-the-art hardware memristors can be in any of the possible states right after production. Since it would be rather time consuming, if not even impossible, to program each of the memristors of a highly integrated chip to a specific, known good state before they are assembled to a huge spiking WTA network, it is desired to ensure decoupling the memristors' convergence conductances from their initial states. Put differently, it is desirable to ensure that the convergence conductance of a given memristive synapse is independent of the initial memristor state. Rather, it should solely depend on the applied PRE/POST statistics. Together with the demand for convergence discussed in the previous section, this feature is supposed to aid in avoiding "un-trainable" hardware networks. That is, networks that are, due to adverse combinations of initial states, not capable of adjusting their synaptic weights properly as would be required to correctly resemble certain input statistics.

Looking for an appropriate test setup to check the final modified memristor model for this independence from the initial state, Protocol 7 was chosen again. In contrast to the simulations presented in the previous section, however, for these tests the same input trains were applied to multiple memristors initialized to different initial states. More precisely, for each of the PRE/POST statistics an appropriate input train encoding the respective probability $p(Y = 1 \mid Z = 1)$ was generated. This input train was then applied to five memristors, initialized to five different initial states. These states comprise

- *minimum conductance (maximum memristance)*,
- *midrange memristance without M_0*,
- *midrange conductance*,
- *midrange conductance without M_0* and
- *maximum conductance (minimum memristance)*.

The evolution of the instantaneous conductance G of the five differently initialized memristors was then recorded for each of the different input trains encoding one of the probabilities $p(Y = 1 \mid Z = 1)$ and finally plotted against the time, yielding the plot shown in Figure 4.18.

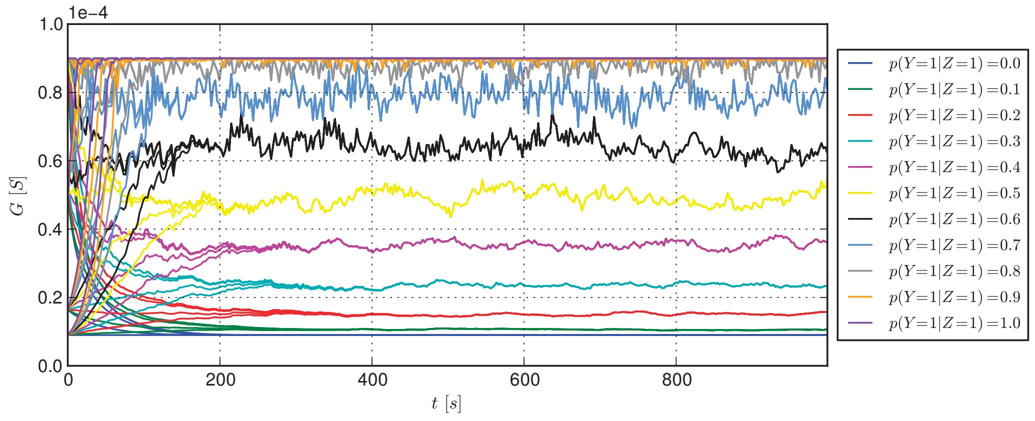


Figure 4.18: Temporal evolution of the conductance for different input statistics and initializations. Independent of the chosen initial state, the memristor converges to the same convergence conductance. Consequently, the the equilibrium state can be considered fully characterized by the input statistics $p(Y = 1 \mid Z = 1)$. The number of samples required to reach this equilibrium state, however, varies with both the applied input statistics as well as the chosen initial state.

As we can see from this plot, after initial transients of different durations indeed the conductance runs into the same convergence value for any of the five different initializations. This gives rise to the assumption that our final memristor model supports the desired independence from the initial memristance state. Put differently, it is assumed that within certain limitations no special precautions with respect to the initial states have to be taken when assembling multiple memristors to a spiking WTA network. As can be seen from Figure 4.18, however, depending on the combination of the applied PRE/POST statistics and the chosen initial state, it takes up to approximately $400\,s$ of learning for the system to converge to the final conductance value. Given that in our test setup the POST-pulses were generated with $T = 50\,ms$ between each other, this translates to a need of up to 8000 events in order for the system to converge to the dynamic equilibrium from any arbitrary initial state. Finally, convergence tends to be the faster for PRE/POST statistics near the extremes $p(Y = 1 \mid Z = 1) = 0.0$ and $p(Y = 1 \mid Z = 1) = 1.0$ and is slowest for midrange values.

These observations conclude the investigation of the independence from the memristor's initialization as well as the chapter devoted to the discussion of different problems encountered during the process of examining the memristor model proposed by Serb et al. [Serb, 2014] with respect to its suitability for the implementation of synapses for spiking WTA networks. As we saw from the experiments presented in the previous five sections, basically the investigated memristor model can be indeed considered a promising foundation to build an implementation of memristive synapses upon. Nevertheless, some minor adjustments, a little bit of fine tuning as well as a few additional building blocks were found to be necessary in order to allow for a correct behavior of these synapses with respect to synaptic plasticity as required for the SEM learning theory. These include a slightly modified parameter set in order to reduce the overreactions of V_x to the applied input (see Section 4.1 for the details). Moreover, in Section 4.2 a linearization of the mapping of the internal memristor state onto a corresponding instantaneous conductance G of the memristor was found to be required in order to allow for the desired correspondence between the synaptic weight and the conductance as well as to ensure a sturdy coverage of the conductance's dynamic range by the internal state variable V_x. This linearization was achieved by adding a pure ohmic pre-resistor M_0 to the memristor. In addition to this, with the different pulse shapes discussed in Section 4.3 one of the most fundamental building blocks for the implementation of proper synaptic plasticity within memristors was identified. With D- and T-pulses, two possible candidates for pulse shapes capable of triggering appropriate synaptic plasticity were derived. Based on these adjustments and observations, weight dependence in terms of the ratio between the strength of LTP and LTD was added to the system in Section 4.4 by replacing the employed parameter functions $g(\cdot)$ and $f(\cdot)$ by a combined parameter function $\psi(\cdot)$. The chosen new parameter function $\psi(\cdot)$ turned out to allow for convergence and reasonable mappings between the applied input statistics and the convergence conductances. These observations finally motivated the experiments concerned with the convergence conductance's independence from the memristor's initial state described in this section. As we saw from the corresponding results, for the chosen setup the convergence conductance can be considered independent from the memristor's initial state.

5
Refinements and Experiments

Based on the memristor model described in Section 2.2.2, in the previous chapter, we step by step derived various improvements until we finally arrived at a synapse framework that was basically capable of mimicking spike time dependent plasticity in hardware. By doing so, we identified the utilized pulse shapes as well as the need for intrinsic weight dependence as two of the most important aspects for the implementation of memristor-based synapses for spiking WTA networks. The derivations of these two building blocks, however, were performed independently from each other. In addition to this, they were solely based on qualitative aspects and had more or less only proof-of-concept character.

This chapter, in contrast, presents refinements to the basic approaches taken in Chapter 4 in that a systematic and joint optimization of the parameters for the pulse shapes and the window function $\psi(\cdot)$ is performed. The resulting optimized parameter set is then used as foundation for final experiments at network level. These final experiments are intended to underline the relevance of the presented memristive synapse model. They include autonomous learning and classification of simple color gradients as well as handwritten digits. As turns out, both tasks can be solved successfully with comparably small networks of memristive synapses.

5.1 Systematic Parameter Search

As already indicated in the introduction to this chapter, the approach taken in Section 4.3 to derive suitable pulse shapes for the implementation of spike time dependent plasticity in memristive synapses according to the memristor model introduced in Section 2.2.2, was based on general reasoning implied by the learning theory as well as fundamental memristor principles. Consequently, as such, the approach was more of qualitative nature. In this section, in contrast, we ware going to take a much more systematic and formal strategy trying to optimize the parameters for the T-pulse pulsing scheme and the window function $\psi(\cdot)$ in a quantitative manner by means of an objective function $\mathcal{L}(\cdot)$. Since, however, a general, closed form analytical solution does not exist for all of the system equations defining the memristor model, this objective function $\mathcal{L}(\cdot)$ will be the foundation for an iterative refinement process rather than subject to direct and analytical optimization. More details on this topic follows in Section 5.1.3.

5.1.1 Requirements

Before the objective function $\mathcal{L}(\cdot)$ can be set up, we start with an exhaustive summary of the different requirements imposed on the pulse shapes as well as the memristor model used by the resulting framework. Some of these requirements were already mentioned briefly in Section 4.3, while others are new.

- PRE-only events should not change anything non-volatile to the memristance. This translates to V_z not being allowed to exceed B_+ and

B_- in the PRE-only case. Otherwise, PRE-only events would trigger either LTP or LTD. This, however, would be incompatible with the theory. Moreover, also very short exceedances of the thresholds $B_\pm$ are not allowed, since, in a network of multiple memristive synapses, they could accumulate and cause undesired behavior. This requirement shall also hold for the limit case, that is, for $T_1 \to \infty$.
$\Rightarrow B_- < V_z < B_+$ for the duration of the whole PRE-pulse T_1 (where $T_1 \to \infty$) in the PRE-only case.

- POST-only events should reduce the non-volatile conductance of the memristive synapse, that is, increase the non-volatile portion of the memristance. Hence, POST-pulses shall drive V_z below B_-. This is necessary, because from a theoretical point of view, POST-only events are supposed to trigger LTD.
$\Rightarrow V_z < B_-$ for part of the POST-pulse's duration in POST-only case.
- None of the POST-pulse's sub-pulses should cause V_z to exceed B_+ in case of a POST-only event. Otherwise, POST-only events would temporarily decrease the non-volatile part of the memristance (that is, increase the non-volatile conductance). Although this would not contradict the theory as long as the overall POST-only event triggers LTD, it is safer to require V_z to stay below B_+ for the duration of the whole POST-pulse in order to avoid similar effects as mentioned together with the first requirement.
$\Rightarrow V_z < B_+$ for the duration of the whole POST-pulse in the POST-only case.
- V_z should not exceed U_- for any of the events (PRE- or POST-only, PRE-and-POST). The meaning and the reason of the sudden sign change of the function $\phi(\cdot)$ at U_- are not known. Moreover, it could turn LTD into LTP, if the amplitude of POST-pulse's negative sub-pulse is chosen too large.
$\Rightarrow V_z > U_-$ for all pulse combinations.
- V_z should exceed B_+ (and/or U_+) in PRE-and-POST case. This is necessary to trigger a non-volatile decrement in memristance (that is, non-volatile conductance increment), which is necessary to achieve LTP for this constellation, as required by the theory. Already small positive Δt values should drive V_z beyond B_+ in order to trigger

LTP. Small negative Δt values should still trigger LTD. Similar to the POST-only case, where B_+ should not be reached, B_- shall not be exceeded in the PRE-and-POST case.
$\Rightarrow V_z > B_+$ or $V_z > U_+$ for part of the PRE-and-POST event (that is, $0 < \Delta t < T_1$). $\Rightarrow V_z > B_-$ for the duration of the whole PRE-and-POST event.

- PRE-pulses should have a duration of about $10\,ms$ in order to match observations concerning the STDP learning window and firing rates made with real biologic synapses.
  $\Rightarrow T_1 \approx 10\,ms$.
- POST-pulses should be "short" compared to PRE-pulses.
  $\Rightarrow T_2 \ll T_1$.
- The increase of the STDP plot around $\Delta t = 0\,ms$ should be steep. Its decay should be steady and start roughly at the end of the STDP learning window, that is, around $\Delta t \approx T_1$. After falling rapidly, the STDP curve should decay slowly towards its initial values (that is, those encountered for negative Δt values).
- V_z should return to $V_z \approx 0$ as fast as possible after the POST-pulse in the POST-only case.
- V_x should not show too big spikes, that is, V_x should decay to V_y as quickly as possible (PRE-only, POST-only, and PRE-and-POST case). Although this is more a matter of the parameter set, using smaller amplitudes can avoid huge ripples.
- Amplitudes should be in the range of several mV up to a few V ($\approx 5\,mV - 2\,V$). Amplitudes in a similar range are preferred.
- Proper weight dependence shall be enforced, that is, LTP shall be predominant for synapses in a low conductive state and LTD shall be predominant for synapses with high conductance.
- The conductance G of a synapse is supposed to converge to a specific value G^* for any given PRE/POST statistics $p(Y = 1\,|\,Z = 1)$. The mapping of the different probabilities $p(Y = 1\,|\,Z = 1)$ to a corresponding convergence conductance G^* shall be unique.
- The convergence conductance G^* shall be independent from the initialization of the memristor, that is, it shall only depend on the applied input statistics.

5.1.2 Objective Function $\mathcal{L}(\cdot)$

Based on the requirements summarized in the previous section, the objective function $\mathcal{L}(\cdot)$ tries to implement a quantitative assessment with respect to the memristor as well as the learning theory of different parameter configurations. From the perspective of the learning theory, the most important feature the memristive synapses have to offer is a unique convergence conductance G^*, which a given synapse attains in response to any arbitrary input statistics. In order to ensure independence of initialization, this convergence conductance G^* shall only depend on the applied PRE/POST statistics and be independent of any other meta-parameters such as the initial state. Besides the demand for unique convergence conductances implied by the learning theory, a second, more qualitative aspect related to the shape of the resulting STDP curves shall be considered by the to be defined objective function $\mathcal{L}(\cdot)$. As indicated in the previous section, we are aiming for a parameter configuration yielding STDP plots which are basically approximately rectangular. To account for these twofold requirements, the objective function $\mathcal{L}(\cdot)$ will be setup as the sum of two separate functions $\mathcal{L}_\mathrm{m}(\cdot)$ and $\mathcal{L}_\mathrm{s}(\cdot)$, assessing the quality of the **m***apping* and the **s***hape* of the resulting curves, respectively.

In a naive approach, the first one of these would simply measure the difference between the actual convergence conductances and some desired target values for multiple input statistics $p(Y = 1 \mid Z = 1)$. Besides this, $\mathcal{L}_\mathrm{m}(\cdot)$ would also assess the deviation of the convergence conductances obtained for different initializations. As indicated earlier, however, for the available memristor model the parameter optimization has to be based on an iterative process, since a closed form solution does not exist for all of the memristor model's system equations. As a result, in each of the iteration steps, the evaluation of $\mathcal{L}_\mathrm{m}(\cdot)$ would require multiple simulation runs of, for instance, the enormously time-consuming Protocol 7 to determine the convergence conductances. Due to this, in order to reduce the computational complexity of the optimization process, a more elaborate foundation for $\mathcal{L}_\mathrm{m}(\cdot)$ had to be developed. This alternative basis for $\mathcal{L}_\mathrm{m}(\cdot)$ was found in a probabilistic view of the convergence conductance.

More precisely, the key idea behind the chosen objective function $\mathcal{L}_{\mathrm{m}}(\cdot)$ is based on a closer inspection of the temporal evolution of a given synapse's conductance in a certain state. Generally, the average conductance change this given synapse encounters over a longer timespan in a certain state for any arbitrary input statistics is given as

$$\left\langle \frac{dG}{dt} \right\rangle \Bigg|_{G,p(\mathrm{LTP})} = p(\mathrm{LTP}) \cdot \Delta G_{\mathrm{LTP}}\,(G) + p(\mathrm{LTD}) \cdot \Delta G_{\mathrm{LTD}}\,(G)\ , \tag{5.1}$$

where $\langle \cdot \rangle$ denotes the expectation operator, $p(\mathrm{LTP})$ is shorthand for $p(Y = 1 \,|\, Z = 1)$ and $p(\mathrm{LTD})$ is given as $p(\mathrm{LTD}) = 1 - p(\mathrm{LTP})$. Finally, $\Delta G_{\mathrm{LTP}}(G)$ and $\Delta G_{\mathrm{LTD}}(G)$, denote the conductance increment and decrement triggered by a single potentiating or depressing event in a memristor with current conductance G, respectively.

Obviously, assuming that a given synapse has, in response to given input statistics characterized by $p(\mathrm{LTP})$ and $p(\mathrm{LTD})$, after some time reached its convergence conductance G^*, the expected value of the conductance change according to Equation 5.1 will be zero:

$$\begin{aligned} \left\langle \frac{dG}{dt} \right\rangle \Bigg|_{G=G^*,p(\mathrm{LTP})} &= 0 \\ \Leftrightarrow\ p(\mathrm{LTP}) \cdot \Delta G_{\mathrm{LTP}}(G^*) + p(\mathrm{LTD}) \cdot \Delta G_{\mathrm{LTD}}(G^*) &= 0 \end{aligned}$$

This expression can be rewritten as

$$p(\mathrm{LTP}) \cdot \Delta G_{\mathrm{LTP}}(G^*) = -p(\mathrm{LTD}) \cdot \Delta G_{\mathrm{LTD}}(G^*)\,. \tag{5.2}$$

The statement of the relation in Equation 5.2 seems rather intuitive, serving as a good illustration of the term *dynamic equilibrium*: Once a synapse's conductance has converged to a given target value G^*, that is, the point of dynamic equilibrium has been reached, the two forces LTP and LTD are, when averaging over a longer time interval, equally strong. The two forces are literally in a state of equilibrium. This state is characterized by a specific ratio between the conductance increment and decrement triggered by single events of each type (LTP or LTD), which can be determined by rearranging Equation 5.2 as

$$\frac{\Delta G_{\mathrm{LTP}}(G^*)}{\Delta G_{\mathrm{LTD}}(G^*)} = -\frac{p(\mathrm{LTD})}{p(\mathrm{LTP})}\,. \tag{5.3}$$

Assuming now that a convergence conductance G^* exists, the above argumentation also holds in the opposite direction: In order for the respective equilibrium state characterized by G^* to be reached in response to any given input statistics, there has to be a specific ratio between the conductance increment and decrement triggered by single instances of potentiating and depressing events. Since the quantities $p(\mathrm{LTP})$ and $p(\mathrm{LTD})$ are known from the given input statistics, this target ratio is uniquely defined according to Equation 5.3 as

$$\boxed{F := \frac{\Delta G_{\mathrm{LTP}}(G^*)}{\Delta G_{\mathrm{LTD}}(G^*)} \overset{!}{=} 1 - \frac{1}{p(\mathrm{LTP})}}\,. \tag{5.4}$$

Obviously, the above relation shall hold for any arbitrary convergence conductance G^* and corresponding input statistics $p(\mathrm{LTP})$. This insight can be utilized to elegantly translate the demand for a decent mapping of the probabilities $p(Y = 1 \,|\, Z = 1)$ onto corresponding convergence conductances G^* into a number of ratios F according to Equation 5.4. In the synapse model, these ratios F shall then be implemented by appropriate values for $\Delta G_{\mathrm{LTP}}(G^*)$ and $\Delta G_{\mathrm{LTD}}(G^*)$ yielded when the synapse is currently in the state of the convergence conductance G^*.

This observation gives rise to a computationally much less expensive objective function. After defining a set of desired convergence conductances

$\boldsymbol{G}^*$ for a set $\boldsymbol{p}$ of different probabilities $p(Y = 1 \mid Z = 1)$, one does not have to initiate long term simulations to determine the system's exact convergence conductances for different parameter configurations $\boldsymbol{\theta}$. Rather, given the probabilities $\boldsymbol{p}$, one determines a set of desired target ratios $\boldsymbol{F}^*$ according to the right-hand side of Equation 5.4. For each of the tested parameter configurations $\boldsymbol{\theta}$, these target ratios $\boldsymbol{F}^*$ are then compared to the ratios $\boldsymbol{F}(\boldsymbol{\theta})$ actually yielded by the model by means of a normalized sum squared error. Formally, this can be described as

$$\mathcal{L}_{\mathrm{m}}(\boldsymbol{\theta}) = \sum_{i=1}^{L} \left[\frac{F_i^* - F_i(\boldsymbol{\theta})}{F_i^*} \right]^2 , \tag{5.5}$$

where L denotes the number of design points, that is, the number of probability/target conductance pairs contained in $\boldsymbol{p}$ and $\boldsymbol{G}^*$, respectively. Since the actual ratios $\boldsymbol{F}(\boldsymbol{\theta})$ can be determined from simple STDP simulations as the ones performed by Protocol 3, the computational complexity of this objective function is considerably smaller compared to the naive approach outlined earlier.

The normalization of the absolute deviations as introduced with the division by F_i^* in Equation 5.5 ensures that all the design points receive the same importance in the partial objective function $\mathcal{L}_{\mathrm{m}}(\cdot)$. This is necessary because the target ratios $\boldsymbol{F}^*$ are spread across a considerably large range. $p(\mathrm{LTP}) = 0.1$, for instance, calls for $F^* = -9$, while $p(\mathrm{LTP}) = 0.9$ implies $F^* = -\frac{1}{9}$. As a result, without normalization, although having a much more severe impact when encountered for small F^* values, a given absolute error would be treated uniformly over the whole range of F. Using the division by F_i^*, the absolute misalignment is put in relation to the target ratio, allowing for a relative assessment of the encountered errors.

Besides the demand for a decent mapping between different input statistics and the resulting convergence conductances, the requirements listed in the previous section also include various restrictions with respect to the shape of the resulting STDP plots. In order to capture these additional, also more qualitative aspects in a quantitative manner, a second term $\mathcal{L}_{\mathrm{s}}(\cdot)$ is included in the final objective function. To keep things simple, the requirements for steep increases and decays of the STDP plots mentioned in the previous section were summarized as a rectangular shape for the STDP-curves being desirable. Based on this, the second term for the final objective function is given as the area between the two curves, that is the ideal rectangular STDP curve and the curve actually yielded by the simulation. Formally, this corresponds to

$$\mathcal{L}_{\mathrm{s}}(\boldsymbol{\theta}) = \sum_{i=1}^{L} \left[\frac{\int_{\Delta t_1}^{\Delta t_2} \mathcal{R}_{i,\boldsymbol{\theta}}(\xi) - \mathcal{S}_{i,\boldsymbol{\theta}}(\xi)\, d\xi}{\int_{\Delta t_1}^{\Delta t_2} \mathcal{R}_{i,\boldsymbol{\theta}}(\xi)\, d\xi} \right]^2 , \tag{5.6}$$

with $\mathcal{R}(\cdot)$ and $\mathcal{S}(\cdot)$ denoting a perfectly rectangular window of width $\Delta t_2 - \Delta t_1$ and the STDP curve yielded by the simulation, respectively. For each of the design points and parameter configurations, the hight of the rectangular window is chosen as $\max(\mathcal{S}_{i,\boldsymbol{\theta}})$. By doing so, one ensures that the integrand of the numerators of the constituent summands in Equation 5.6 is strictly non-negative. Finally, for each of the design points, Δt_1 and Δt_2 denote the inter event intervals for which the simulated STDP curve crosses the Δt-axis in rising and falling direction, respectively. Consequently, only the LTP portions of the STDP curves are considered by $\mathcal{L}_{\mathrm{s}}(\cdot)$.

By analogy with $\mathcal{L}_{\mathrm{m}}(\cdot)$, also the error measured by $\mathcal{L}_{\mathrm{s}}(\cdot)$ is normalized. Similar as for $\mathcal{L}_{\mathrm{m}}(\cdot)$, this is necessary in order to make deviations of the differently sized areas equally important in the overall objective function. In addition to this, by using this relative approach, one ensures that the compound objective function $\mathcal{L}(\cdot)$ can be constructed as a weighted sum of the partial ones.

Based on the expressions in Equations 5.5 and 5.6, we obtain the final objective function $\mathcal{L}(\cdot)$ as

$$\boxed{\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{L} \left\{ \mu \left[\frac{F_i^* - F_i(\boldsymbol{\theta})}{F_i^*} \right]^2 + \nu \left[\frac{\int_{\Delta t_1}^{\Delta t_2} \mathcal{R}_{i,\boldsymbol{\theta}}(\xi) - \mathcal{S}_{i,\boldsymbol{\theta}}(\xi)\, d\xi}{\int_{\Delta t_1}^{\Delta t_2} \mathcal{R}_{i,\boldsymbol{\theta}}(\xi)\, d\xi} \right]^2 \right\}} , \tag{5.7}$$

where μ and $\nu = 1 - \mu$ are used to fine-tune the relative importance of the mapping versus the shape aspects.

As we saw in Chapter 4, some of the requirements listed in Section 5.1.1 can easily be translated directly into certain value ranges various parameters such as pulse amplitudes or durations are not allowed to exceed. This fact was utilized to narrow down the search space which has to be covered by the iterative search process. Consequently, these requirements and implications are not explicitly contained in the definition of the objective function $\mathcal{L}(\cdot)$ given in Equation 5.7, but rather considered in the simulations backing the parameter search.

5.1.3 Search Strategies and Results

Given the objective function given in Equation 5.7, an iterative search process can be performed in order to find a set of model parameters $\boldsymbol{\theta}^*$ which are optimal with respect to $\mathcal{L}(\cdot)$. For the given setup (T-pulses and model featuring the full-blown window function $\psi(\cdot)$ with two separate shape parameters for LTP and LTD), this parameter vector $\boldsymbol{\theta}$ contains eight entries. These comprise the amplitudes $V_{1,\mathrm{POST}}$, $V_{2,\mathrm{POST}}$ and $V_{3,\mathrm{POST}}$, the durations $t_{1,\mathrm{POST}}$, $t_{2,\mathrm{POST}}$ and $t_{3,\mathrm{POST}}$ as well as shape parameters $\varkappa_{\mathrm{P}}$ and $\varkappa_{\mathrm{D}}$.

Although, with a dimensionality of 8, the parameter space by itself is not that big, assuming reasonable step sizes as well as realistic ranges for the various parameters, still results in more than 10^{14} parameter combinations. Consequently, despite the chosen objective function $\mathcal{L}(\cdot)$ being computationally way more efficient than the naive approach outlined in the introduction to the previous section, an exhaustive search of the parameter space is far from practically feasible. Due to this, an alternative search strategy had to be chosen. More precisely, instead of covering the whole parameter space, only a local search was performed. As indicated in the previous section, some of the requirements listed in Section 5.1.1 were directly incorporated into this search process by picking appropriate ranges for various parameters as well as excluding certain configurations beforehand without even evaluating the objective function $\mathcal{L}(\cdot)$.

i	$p(\text{LTP})$	F^*	$\rho^* = \frac{G^*}{G_{\max}}$
1	0.1	-9	0.15
2	0.5	-1	0.55
3	0.9	$-\frac{1}{9}$	0.95

Table 5.1: Design points used for the parameter search. For each of the design points, STDP simulations according to Protocol 3 are run on synapses initialized to $G^* = \rho^* G_{\max}$. The results yielded by these simulations are then used to evaluate the objective function $\mathcal{L}(\cdot)$ according to Equation 5.7, allowing for the quantitative comparison of different parameter configurations $\boldsymbol{\theta}$.

The design points used for this local parameter search are listed in Table 5.1. An important observation in this context is that not any target conductance G^*, or fraction ρ^* for that matter, may be chosen for the specification of these design points. More precisely, the minimum and maximum conductances of the underlying type of memristor model as well as the chosen value for the additional pre-resistor M_0 according to Section 4.2 impose certain limitations on this choice. For the memristor model this work is based on, for instance, ρ^* is limited to be within the range $0.100009 < \rho^* < 1$ (see Section C.4 for a detailed derivation of this range).

Another rather fundamental aspect of the parameter search process which was neglected so far, is the question how to exactly determine the conductance increments and decrements $\Delta G_{\text{LTP}}(G^*)$ and $\Delta G_{\text{LTD}}(G^*)$ from the results yielded by the STDP simulations. As we remember from Section 5.1.2, both quantities are, as they specify the ratio F, main building blocks for the evaluation of the objective function $\mathcal{L}_{\text{m}}(\cdot)$. While the LTD conductance decrement $\Delta G_{\text{LTD}}(G^*)$ can be determined relatively straightforward, this is not the case for ΔG_{LTP}. As can be seen from various STDP plots shown in Chapter 4, the ΔG values encountered for the LTD cases are, in general, not that different from each other. Instead, they are basically the same, for instance, for almost any value of $\Delta t < 0$. In contrast to this, the LTP cases $0 < \Delta t < \tau$ yield a broad variety of different ΔG values. Consequently, *"the"* conductance increment $\Delta G_{\text{LTP}}(G^*)$ cannot be determined that easily. For the purpose of this work it was decided to use the mean over all ΔG values within the STDP learning window τ, since

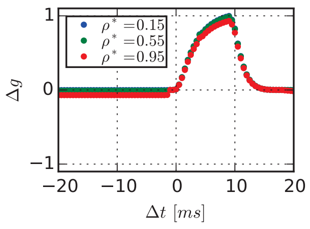
Name	Unit	Init	Step	Result θ^*
$V_{1,\text{POST}}$	mV	35	0.5	34.5
$V_{2,\text{POST}}$	mV	-35	0.5	-41
$V_{3,\text{POST}}$	mV	44.77	0.5	43.15
$t_{1,\text{POST}}$	ms	0.5	0.1	0.5
$t_{2,\text{POST}}$	ms	1	0.1	1.0
$t_{3,\text{POST}}$	ms	0.5	0.1	0.5
$\varkappa_\text{P}$	–	1.5	0.1	1.4
$\varkappa_\text{D}$	–	1.5	0.1	1.5

Table 5.2: Initialization and results of systematic parameter search. For each of the assessed parameter vectors $\boldsymbol{\theta}$, an appropriate value $V_{3,\text{POST}}$ was chosen (compare Section 4.3.2). Based on the above initializations, the parameter values listed in the rightmost column optimize the objective function $\mathcal{L}(\cdot)$ defined in Equation 5.7.

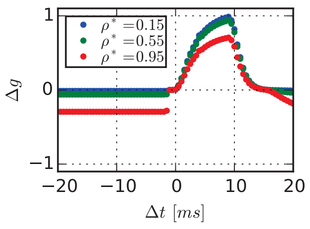
the inter event interval Δt was assumed to be distributed uniformly over the width of the PSPs triggered by pre-synaptic spikes, making each value equally likely in a network setup with stochastic spiking activity. Possible alternative choices include the mean, the median and the maximum ΔG values encountered for LTP as well as the ΔG values yielded for midrange inter event intervals Δt.

Besides the quantities directly related to the objective function, the parameter optimization process as such requires an additional building block. As indicated earlier, the optimization is based on an iterative local search strategy. Since the iterative local search works by varying the constituent entries of the parameter vector $\boldsymbol{\theta}$ by certain steps, obviously initialization values are required for each of its entries. These chosen values as well as the step sizes are listed in Tables 5.2. In order to avoid potential local optimums, these initial values were intendedly chosen different from the ones listed in Section 4.3.2..

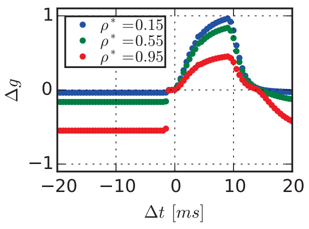
Figure 5.1 illustrates the iterative nature of the chosen optimization process. As can be seen, the local search strategy succeeds in driving the STDP responses for the different convergence values ρ^* apart, arriving at a local optimum after $n = 19$ iterations. At this point, the parameter set listed in the rightmost column of Table 5.2 was reached. The STDP



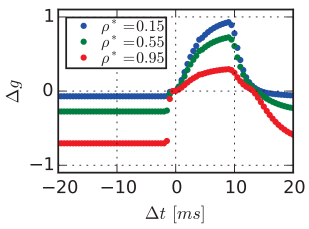
(a) $n = 1$



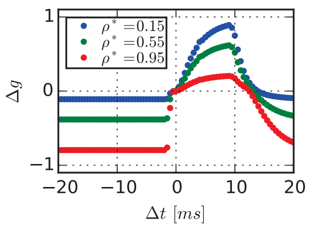
(b) $n = 5$



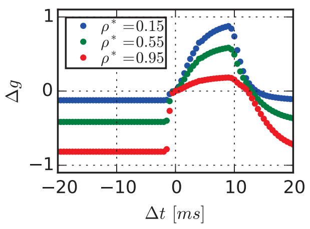
(c) $n = 8$



(d) $n = 11$



(e) $n = 14$



(f) $n = 19$

Figure 5.1: Gradual improvement of the STDP responses. The iterative local search process succeeds in optimizing the parameter set $\boldsymbol{\theta}$, finally yielding the results summarized in Table 5.2.

responses shown in Figure 5.1f illustrate, how this parameter configuration gives rise to a distinct local ratio F between the weight increments and decrements encountered for each of the desired convergence conductances. According to the above definition of the LTP weight increment, the ratios $\boldsymbol{F} = \{-4.75, -0.94, -0.15\}$ can be read off of these plots. These values are aligned with the target values $\boldsymbol{F}^* = \{-9, -1, -\frac{1}{9}\}$ listed in Table 5.1 acceptably well. In addition to this, also the more qualitative requirements for steep increases and decreases of the STDP plots are met quite decently.

In an attempt to assess the optimized parameter set in a qualitative manner, the convergence tests described in Section 4.4 were re-run on memristors using the optimized parameter vector summarized in Table 5.2. The results of this simulation are shown in Figures 5.2 and 5.3b. As becomes clear from Figure 5.2, still both proper weight dependence as well as convergence are guaranteed. In contrast to the unoptimized parameter set, for the optimized parameter vector especially the resolution of higher

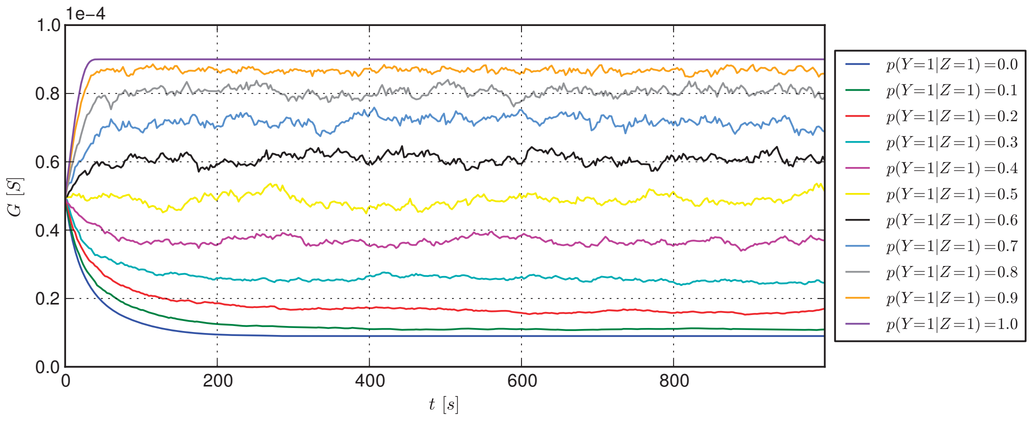
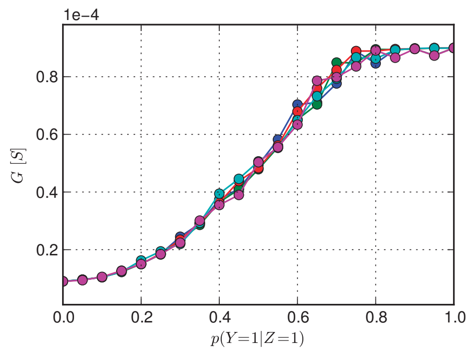


Figure 5.2: Temporal evolution of the conductance for different input statistics. Clearly, for the optimized parameter set summarized in the rightmost column of Table 5.2, the quality of the convergence conductances has improved. Compared to the evolution plot shown in Figure 4.15, especially the resolution of higher $p(Y = 1 \,|\, Z = 1)$ values has become better.

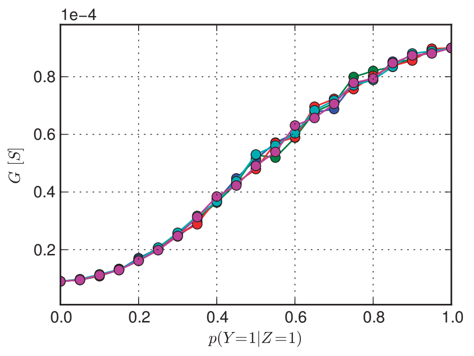
$p(Y = 1 \,|\, Z = 1)$ values has become better. Recalling that in the modified memristor model all non-volatile memristance changes are shaped by the proposed parameter function $\psi(\cdot)$, and given that according to Table 5.2 the optimization process yielded a smaller value for $\varkappa_P$ than was used in Section 4.4, this makes perfectly sense. The guessed value $\varkappa_P = 1.5$ as used in the unoptimized parameter set made LTP events unrightfully strong compared to LTD events for $p(Y = 1 \,|\, Z = 1) \geq 0.8$.

This becomes also clear when comparing the mapping plots shown in Figure 5.3. For the original parameter set shown in Figure 5.3a and probabilities $p(Y = 1 \,|\, Z = 1) \geq 0.8$, the conductance is driven into saturation. In contrast to this, for the optimized parameter set using $\varkappa_P = 1.4$, the linearity as well as the uniqueness of the mapping between probabilities $p(Y = 1 \,|\, Z = 1)$ and corresponding convergence conductances G has improved. As can be seen from Figure 5.3b, for the optimized parameter set the saturation effects have vanished.

These observations conclude the section devoted to the explicit optimization of the memristor model's and pulse shapes' joint parameter set. As turned out, applying a local search with respect to the joint parameter



(a) Parameter set as in Section 4.5



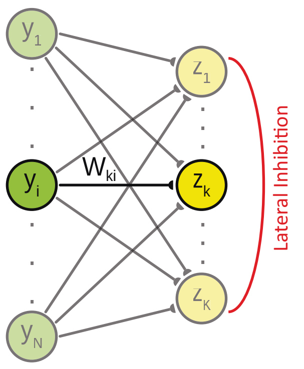
(b) Optimized parameter set

Figure 5.3: Mapping between input statistics and convergence conductances. Besides the evolution plot shown in Figure 5.2, also the mapping plot resembles the improvements yielded by the optimized parameter set. The saturation effects encountered for $p(Y = 1 \mid Z = 1) \geq 0.8$ for the original parameter set illustrated in (a), have vanished for the optimized parameter set underlying (b).

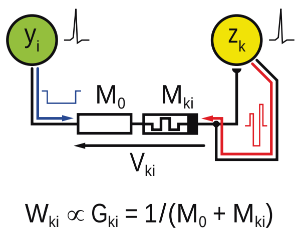
vector $\boldsymbol{\theta}$ and the objective function derived in Section 5.1.2 yields an improvement of the encountered convergence behavior. In fact, from a theoretical point of view the overall system behavior suggested by the mapping plot shown in Figure 5.3b, renders the modified memristor and pulse framework a promising combination for the implementation of memristor-based spiking WTA networks. Network level simulations seeking to confirm this impression follow in the next section.

5.2 Network Level Simulations

Returning to the ultimate goal of this work, that is, finding and evaluating a model for memristor-based plastic synapses to be used in spiking WTA networks, simulations at network level were performed. As indicated in the introduction to this chapter, these simulations include two different learning tasks for which multiple memristive synapses were assembled and combined with pre- and post-synaptic neurons to form spiking WTA network circuits. In the following, we will first take a quick look at the



(a) WTA network architecture



(b) Synapse model

Figure 5.4: Network architecture and synapse model. (a) shows an illustration of the WTA network architecture. As illustrated in (b), for the simulations described in this chapter, the connections between the neurons are implemented as memristive synapses according to the modified memristor model proposed in this work. Plasticity within these synapses is triggered by an appropriate pulsing scheme (indicated in blue and red). Both illustrations are adapted from [BillLegenstein, 2014].

general simulation setup before we turn to the implementation details as well as the results obtained for the two learning tasks.

5.2.1 Overview

As indicated, the idea behind the network level simulations is to evaluate the synapse framework based on the memristor model and pulsing scheme described in earlier chapters in the context of unsupervised learning within a spiking WTA network. For this purpose, a network as the one illustrated in Figure 5.4a is equipped with memristive synapses. These synapses are then pulsed according to the chosen T-pulse pulsing scheme in response to the spikes emitted by the respective neurons (compare Figure 5.4b). The parameters for this setup (that is, the parameters for the underlying memristor model as such as well as the pulses sent through the synapse) are chosen according to the optimized joint parameter vector $\boldsymbol{\theta}^*$ identified in the previous section.

An important concept the simulations at network level are based on is the introduction of two levels of detail. The first, more abstract level is concerned with synaptic transmissions and the generation of the input and network spikes according to the underlying stochastic processes. At this level, all quantities are idealized and unit-less. A pre-synaptic spike, for instance, is assumed to trigger a PSP of width τ and height 1. In the following simulations, this level is time-discretized at $dt_\mathrm{o} = 1\,ms$. Besides this abstract and idealized level, there exists a second, more detailed level, which aims to capture plasticity within the memristive synapses. This is done at a time scale sufficiently small to allow for a detailed simulation of the memristive hardware devices' underlying dynamic systems. The quantities involved on this level typically have physical equivalents in a possible circuit implementation of the network and thus also have physical units such as Volts or Siemens. Examples for such quantities are, for instance, the amplitudes of the involved pulses actually sent through the memristive synapses. Exceptions to this rule form, of course, some of the the memristor model's internal state variables which do not even have a physical representation in a hardware memristor device (compare Section 2.2.2). In order to capture the internal transient processes the memristor model is built upon at a decent fidelity, the time step size for this level of detail was chosen as $dt_\mathrm{i} = 50\,\mu s$.

The motivation behind this separation into two levels of detail is twofold. On the one hand, it facilitates fast and simple simulations since only those parts of the framework that require a comparably high fidelity are actually simulated at a detailed temporal resolution. On the other hand, the separation is necessary in order to allow for a distinction between effects that arise from plasticity within the neural memristive synapses and others which are results of neural interaction and integration in the neurons. The latter include, for instance, effects such as the influence of post-synaptic pulses on the membrane potential of the post-synaptic neuron that has just spiked. While the investigation of these considerations is subject to further research, the simulations carried out as part of this work focus on the plastic behavior of the memristive synapses and almost fully decouple plasticity from the spike generation.

5.2.2 Details and Results

Similar as in [BillLegenstein, 2014] and as indicated above, the simulations presented here were performed on an abstract stochastic neuron model. This neuron model is built around an abstract membrane potential $u_k(t)$, which is defined as

$$u_k(t) = b_k + \sum_{i=1}^{N} W_{ki}\, y_i(t)\,. \tag{5.8}$$

In the above equation, $y_i(t)$ denotes the inputs which are fed from the input neurons to network neurons z_k through the afferent weights W_{ki}. These inputs are generated by the input neurons which are assumed to send PSPs, rectangular voltage pulses of duration τ, through connected synapses when they spike. Consequently, the membrane potential $u_k(t)$ can be considered as linearly integrating the PSPs which arrive from all the input neurons through the synaptic weights W_{ki}.

The quantity b_k in Equation 5.8 denotes the intrinsic excitability, which can be interpreted as a network neuron's general disposition to fire [BillLegenstein, 2014]. Over the process of learning, these intrinsic excitabilities b_k are adapted according to a homeostatic plasticity rule [Habenschuss, 2012]. In a nutshell, the idea behind homeostatic plasticity is to ensure that all network neurons z_k show a certain target activity and thus all of them take part in the network response. In addition to a stabilizing effect, homeostatic plasticity and the accordingly adapted intrinsic excitabilities b_k ensure a certain degree of “fairness” among the K network neurons. Depending on the input that is presented to the network during learning, it might happen that some of the pattern classes induce a higher activity in the input neurons than others do. During WTA competition, without a regulation of the intrinsic excitability through homeostatic plasticity, neurons that became experts for strong patterns would have an advantage over others that have specialized on low activity patterns [BillLegenstein, 2014]. Imposing lower intrinsic excitabilities on the former can balance out this undesired effect to a certain extent.

The stochastic firing mechanism of the network neurons z_k constituting the network's response to a given input stimulus, is assumed as a Poisson process with an instantaneous firing rate $r_k(t)$. This firing rate is given as

$$r_k(t) = r_{\text{net}} \cdot e^{u_k(t) - u_{\text{inh}}(t)} , \tag{5.9}$$

where r_{net} is the network's overall firing rate. Basically, this firing rate translates to each network neuron z_k firing with probability $r_k(t) \cdot \xi$ within a small time interval ξ (for $\xi \to 0$). Finally, the quantity $u_{\text{inh}}(t)$ in the above equation models lateral inhibition and introduces WTA competition among the network neurons for firing in response to a given input pattern. It is defined as

$$u_{\text{inh}}(t) := \log \left[\sum_{j=1}^{K} e^{u_j(t)} \right] . \tag{5.10}$$

Following Bill and Legenstein [BillLegenstein, 2014], from a probabilistic perspective, the spiking activity of the input neurons y_i and network neurons z_k can be interpreted as samples of random variables Y_i and Z_k. The definition of the values of Y_i is straightforward. Setting $Y_i = y_i(t)$, in accordance with the concept of PSPs mentioned above, we identify $Y_i = 1$ if input neuron with index i spiked within the interval $[t - \tau, t]$ and $Y_i = 0$ otherwise. Consequently, the value of Y_i is defined for all times t. This is not the case for Z_k, which labels the winner of the WTA competition and thus is only defined at times, at which one of the network neurons spikes.

Formally, this can be described through the help of $s_j(t)$, the function defining the spike train of an arbitrary network neuron with index j. $s_j(t)$ is given as

$$s_j(t) = \sum_f \delta \left(t - t_j^f \right) ,$$

where $\delta(\cdot)$ denotes the Dirac delta function and t_j^f identifies the time at which the f^{th} spike of the network neuron with index j occurs. Provided now that $s_j(t) \neq 0$ for some network neuron with index j, then the values of Z_k label the winner of the WTA competition such that $Z_k = 1$ if $k = j$ and $Z_k = 0$ otherwise. This can be interpreted as the random vector $\boldsymbol{Z} := (Z_1, \ldots, Z_K)$, being an unrolled version of a multinomial random variable $\tilde{Z} \in \{1, \ldots, K\}$ with components $Z_k = 1$ for $\tilde{Z} = k$ and $Z_k = 0$ otherwise [BillLegenstein, 2014].

Together with the input random vector $\boldsymbol{Y} := (Y_1, \ldots, Y_N)$, this probabilistic interpretation of the spiking activity allows for the definition of a network response distribution $p_{\text{net}}(\boldsymbol{Z} \,|\, \boldsymbol{Y})$ of the outputs given on the inputs. Since, as mentioned above, the probability $p(Z_k = 1 \,|\, \boldsymbol{Y} = \boldsymbol{y}(t))$ for a single network neuron z_k to be active can be determined from the instantaneous firing rate $r_k(t)$, for any fixed input $\boldsymbol{Y} = \boldsymbol{y}(t) = (y_1(t), \ldots, y_N(t))$, the above response distribution $p_{\text{net}}(\boldsymbol{Z} \,|\, \boldsymbol{Y} = \boldsymbol{y}(t))$ can be determined by combining Equations 5.8 and 5.9:

$$\begin{aligned} p_{\text{net}}(Z_k = 1 \,|\, \boldsymbol{Y} = \boldsymbol{y}(t)) = \frac{r_k(t)}{r_{\text{net}}} = e^{u_k(t) - u_{\text{inh}}(t)} = \\ = \frac{e^{b_k + \sum_{i=1}^{N} W_{ki}\, y_i(t)}}{\sum_{j=1}^{K} e^{b_j + \sum_{i=1}^{N} W_{ji}\, y_i(t)}} \end{aligned} \tag{5.11}$$

Obviously, since the random vector $\boldsymbol{Z}$ is only defined at those instants of time at which one of the K network neurons z_k spikes, this is also the case for the above response distribution $p_{\text{net}}(\boldsymbol{Z} \,|\, \boldsymbol{Y})$.

As is outlined in [BillLegenstein, 2014], $p_{\text{net}}(\boldsymbol{Z} \,|\, \boldsymbol{Y})$ can be interpreted as the result of a Bayesian computation. Specifically, by hypothetically reversing the direction of the network computations, one can interpret the spiking network as a generative model. This view allows for the spiking activity of the network neurons z_k to be interpreted as hidden causes for the observed input patterns $\boldsymbol{y}(t)$. Moreover, this view implicitly gives rise to a prior distribution $p(\boldsymbol{Z})$ over the hidden causes Z_k as well as a set of likelihood distributions $p(\boldsymbol{Y} \,|\, Z_k = 1)$. As was shown by Nessler et al. [Nessler, 2013], these likelihood distributions are implicitly encoded in the weights W_{ki} each of the network neurons z_k maintains to all the input neurons y_i. In the network's real rather than reversed operation (that is, hidden causes Z_k which are to be inferred from input vectors $\boldsymbol{y}(t)$ rather than the other way round), the prior and the likelihood distributions are combined to obtain the posterior distribution as

$$p(Z_k = 1 \,|\, \boldsymbol{Y} = \boldsymbol{y}(t)) \propto p(Z_k = 1)\, p(\boldsymbol{Y} = \boldsymbol{y}(t) \,|\, Z_k = 1)\,. \tag{5.12}$$

In addition to showing that the likelihood distributions $p(\boldsymbol{Y} \mid Z_k = 1)$ are implicitly encoded in the synaptic weights W_{ki}, Nessler et al. [Nessler, 2013] also showed that the above posterior distribution is approximated by the network response distribution $p_{\text{net}}(\boldsymbol{Z} \mid \boldsymbol{Y})$ defined in Equation 5.11 and that the parameters W_{ki} of this likelihood model can be optimized in an unsupervised manner by a weight-dependent STDP rule [BillLegenstein, 2014].

Since the type of STDP weight dependence that is implemented by the memristive synapses used for this work differs significantly from the plain exponential weight dependence assumed by Nessler et al. [Nessler, 2013], as a naturally arising question one might ask, which exact likelihood model $p(\boldsymbol{Y} \mid Z_k = 1)$ is actually implemented by the proposed memristive synapses. As mentioned earlier, the proposed model sets the weights W_{ki} into direct proportion with the conductances G_{ki} of the respective synapses. As a result, an answer to the above question can be found through a closer inspection of the mapping between probabilities $p(y_i = 1 \mid z_k = 1)$ and corresponding convergence conductances G. Figure 5.5a illustrates this mapping for the proposed memristive synapse model.

In a first approximation, a linear model of the form

$$G(p(\text{LTP})) = \lambda\, p(\text{LTP})\,, \tag{5.13}$$

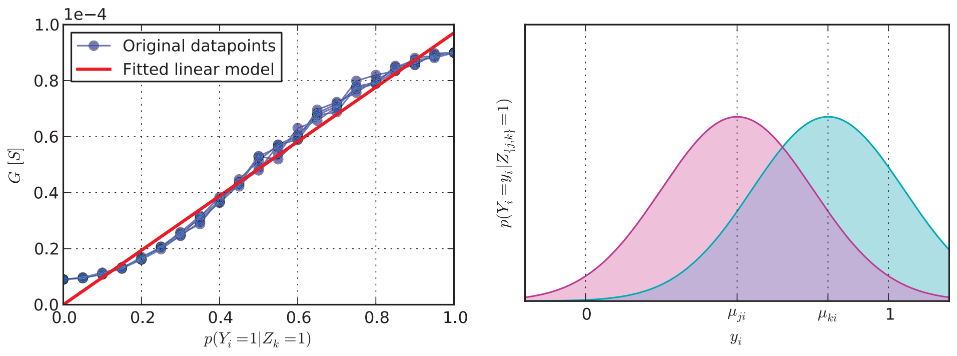
was fitted to these data points. In the above expression, $p(\text{LTP})$ is again shorthand for $p(Y_i = 1 \mid Z_k = 1)$ and λ is the fit parameter. For the original data points indicated in blue in Figure 5.5a, this parameter was identified as $\lambda \approx 9.7 \cdot 10^{-5}\, S$

Besides its simplicity, the choice of assuming a linear mapping model was further motivated by the fact that it gives rise to a Gaussian likelihood model for the distributions $p(\boldsymbol{Y} \mid Z_k = 1)$ of the form

$$p(\boldsymbol{Y} = \boldsymbol{y}(t) \mid Z_k = 1) = \prod_{i=1}^{N} p(Y_i = y_i(t) \mid Z_k = 1)\,, \tag{5.14}$$

with the likelihood distribution of each input y_i being given as

$$p(Y_i = y_i(t) \mid Z_k = 1) = \frac{1}{\sqrt{2\,\pi\,\sigma^2}}\, e^{-\frac{(y_i(t)-\mu_{ki})^2}{2\,\sigma^2}}\,. \tag{5.15}$$



(a) Maping and fitted model

(b) Gaussian likelihood model

Figure 5.5: Original mapping and fitted linear model. In a first approximation, a linear model of the form indicated in Equation 5.13 was fitted to the original data points obtained for the mapping between probabilities $p(y_i = 1 \,|\, z_k = 1)$ and corresponding convergence conductances G in simulations (see (a)). The plot in (b) shows an illustration of the Gaussian likelihood model according to Equation 5.15 for two hidden causes Z_j and Z_k (adapted from [BillLegenstein, 2014]).

An illustration of this distribution is shown in Figure 5.5b. The same model was already used by Bill and Legenstein [BillLegenstein, 2014], who identified the mean values μ_{ki} and the standard deviation σ used in Equation 5.15 as

$$\mu_{ki} = \frac{W_{ki}}{W_{\max}} \quad \text{and} \tag{5.16}$$

$$\sigma = \frac{1}{\sqrt{W_{\max}}} \, . \tag{5.17}$$

In the above expressions, the quantity $W_{\max}$ denotes the maximum achievable weight a given memristive synapse can attain.

As indicated in the previous section, for the purpose of this work, the simulation of plasticity within the memristive synapses is almost completely decoupled from the simulation of all the other, more abstract, parts of the network such as the neuron model. The only link between these two levels of detail are the values of the synaptic weights W_{ki} On the one hand, they are adapted through the simulation of the detailed hardware dynamics of the memristor. On the other hand, their values are used for the computation of the abstract membrane potentials $u_k(t)$ according to

Equation 5.8. While so far we merely stated that the weights W_{ki} are proportional to the memristive synapses' instantaneous conductance G_{ki}, we will now define them explicitly. In order to keep the model as simple as possible, we set up the mapping between conductances and weights as

$$W_{ki} = \zeta \, G_{ki} \, , \tag{5.18}$$

where ζ is a constant proportionality factor. The exact value for this factor represents a free parameter of the network model and may be determined based on the findings of Bill and Legenstein [BillLegenstein, 2014] mentioned above. Combining Equations 5.16 and 5.17 one can determine a relationship between the weights W_{ki}, the mean values μ_{ki} and the standard deviation σ:

$$W_{ki} = \frac{\mu_{ki}}{\sigma^2} \tag{5.19}$$

Assuming now that the full dynamic range of the memristive synapses' conductance shall be mapped to the complete dynamic range of the abstract synaptic weights W_{ki}, we can write

$$\zeta \; [G(\mu_1) - G(\mu_2)] \overset{!}{=} W(\mu_1) - W(\mu_2)$$

Plugging Equation 5.19 into the above expression, using Equation 5.13 and assuming $\mu_1 = 1$ and $\mu_2 = 0$ yields

$$\begin{aligned} \zeta \; [G(1) - G(0)] &\overset{!}{=} \frac{1-0}{\sigma^2} \\ \zeta \; [\lambda - 0] &\overset{!}{=} \frac{1}{\sigma^2} \, . \end{aligned}$$

Choosing $\sigma^2 = 1$, this translates to

$$\boxed{\zeta = \frac{1}{\lambda}} \tag{5.20}$$

Based on the value for λ determined through the fitted model, according to the above expression we identify ζ as $\zeta = \frac{1}{9.7 \cdot 10^{-5}\, S} \approx 1.03 \cdot 10^4 \, \frac{1}{S}$.

These considerations complete the discussion of the probabilistic interpretation of and pave the way for the actual network level simulations.

Learning of Simple Color Gradients

The first of these simulations aims to train a network with $K = 2$ network neurons such that it is able to distinguish and classify two different simple color gradients. Since these color gradients are set up according to known statistics, the ground truth of the presented input is known. Consequently, the results of this example, that is, the convergence weights, may be matched to the expected values according to the Gaussian likelihood distributions.

Figure 5.6a illustrates the color gradients used for this example. The brightness values of the 4×4 pixel cells $\boldsymbol{x}^s$ are transformed to spike trains. Similar as in [BillLegenstein, 2014], the spiking probabilities of the constituent input neurons y_i are determined according to

$$p_i^{\text{spike}} = 1 - (1 - x_i^s)^{\frac{dt_o}{\tau}} \ , \tag{5.21}$$

with i indexing the 4×4 grid in a row-wise manner and $s \in \{1, 2\}$ labeling the two patterns. According to the above probabilities, spikes of the input neurons are generated. These spikes are assumed as triggering the rectangular PSPs of duration τ. In accordance with [BillLegenstein, 2014], the resulting voltage trains are clipped to $[0, 1]$, in order to limit the effect of overlapping PSPs to an extension of their width. Similar measures are taken for the network's spike response. On the one hand, the number of post-synaptic spikes a given network neuron z_k can emit per simulation time step dt_o is limited to one. On the other hand, for memristive plasticity, once a post-synaptic pulse according to the chosen pulsing scheme is triggered, no additional pulse can be triggered for the duration of the first one. Consequently, for memristive plasticity, post-synaptic pulses are limited to one every $2\,ms$ for each of the network neurons.

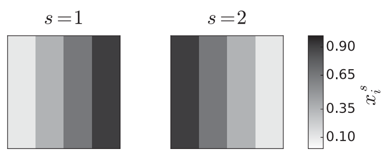
Similar as in [BillLegenstein, 2014], the network simulation time step and the width of the PSPs are chosen as $dt_o = 1\,ms$ and $\tau = 10\,ms$, respectively. The brightness values for the left pattern $\boldsymbol{x}^1$ in Figure 5.6a are defined as $\{0.1,\ 0.35,\ 0.65,\ 0.9\}$ (from left to right) and in reversed order for $\boldsymbol{x}^2$.

During learning, each pattern is presented to the input neurons for $200\,ms$, with a new random pattern being drawn after this time period. In contrast to [BillLegenstein, 2014], with $r_{\text{net}} = 20\,Hz$ a comparably low overall network firing rate was chosen in order to adapt the frequency at which the post-synaptic pulses are generated to the decay times of the memristor model's internal state variables' transient processes (see Section 4.1 for details). Choosing r_{net} to high bears the risk of glitches and adverse overlaps, which could, in turn, lead to the implementation of an incorrect STDP rule in the memristive synapses. As indicated earlier in this section, plasticity within the memristive synapse is simulated at a time step size $dt_{\text{i}} = 50\,\mu s$.

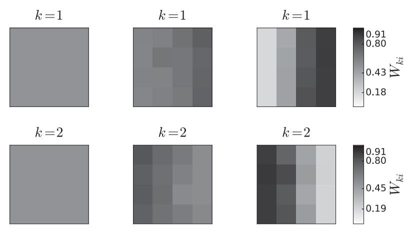
As can be seen from Figure 5.6b, in response to the input the network is presented with, indeed gradually prototypes of the input distribution emerge in the weights W_{ki} which were previously jointly initialized to a midrange value. More precisely, over the course of learning for $2000\,s$ (10000 samples being presented for $200\,ms$ each), the weights W_{1i} adapt to pattern $\boldsymbol{x}^1$, while the others, W_{2i}, learn to represent pattern $\boldsymbol{x}^2$. This specialization becomes already slightly apparent from the plots in the second column of Figure 5.6b, which show the weight matrices after $250\,s$ of learning and is clearly visible from the plots shown in the rightmost column, which illustrate the final weight matrices at the end of the $2000\,s$ learning procedure.

In addition to the illustration of the weight matrices, the gradual emergence of prototypes in the weights becomes also evident from the plots shown in Figures 5.6c and 5.6d. Instead of snapshots of the weights at specific times, these plots show the evolution of the weights W_{ki} over time for the duration of the whole learning process. After an initial phase of joint strengthening, the 2×16 afferent synapses of z_1 and z_2 split up and converge to four distinct weight levels (2×4 synapses per level). As becomes clear from the color codes of the plots, the mapping of the weight levels to synapses is symmetric. This is perfectly aligned with the input patterns $\boldsymbol{x}^s$ being also symmetric.

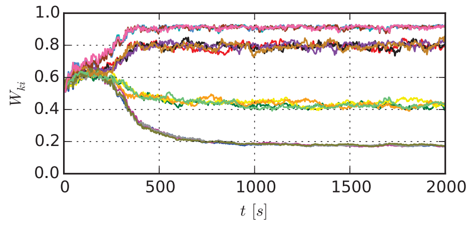
The convergence levels which can be read off of the plots illustrating



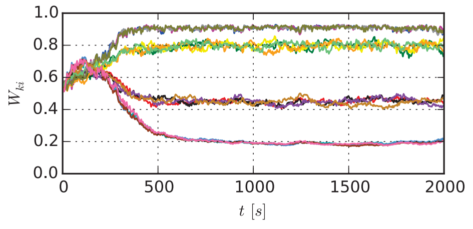
(a) Input patterns $\mathbf{x}^s$



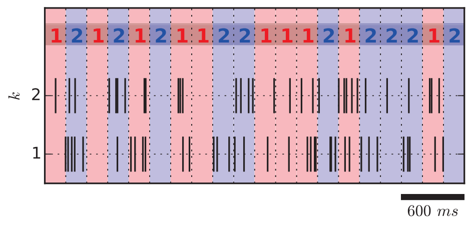
(b) Gradual emergence of prototypes in the weights W_{ki}



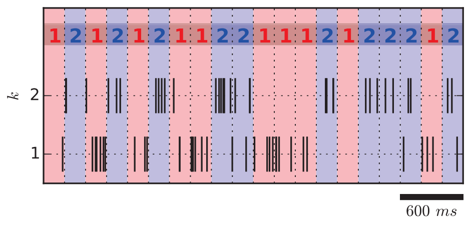
(c) Evolution of the weights W_{ki} over time ($k = 1$)



(d) Evolution of the weights W_{ki} over time ($k = 2$)



(e) Spike response before learning



(f) Spike response after learning

Figure 5.6: Learning of simple color gradients. The brightness values (0.1, 0.35, 0.65 and 0.9) of the 4×4 pixels x_i^s in (a) are interpreted as probabilities $p(y_i = 1)$ and translated into spike trains. These spike trains are then fed to the input neurons y_i of a WTA network. As can be seen from (b), during the process of unsupervised learning, prototypes of the presented input patterns gradually emerge in the weights W_{ki} (shown are the weight matrices $W_{ki}(t)$ for $t = 0\,s$, $t = 250\,s$ and $t = 2000\,s$). These prototypic weight configurations turn each of the network neurons z_k into a probabilistic expert [BillLegenstein, 2014] for one of the two input patterns. At the beginning of the learning process, for both network neurons the afferent weights W_{ki} are strengthened jointly, before the traces fan out and converge to four distinct levels (shown in (c) and (d)). This is perfectly aligned with the presented input patterns, which encode four discrete probabilities in the spikes patterns of y_i. As can be seen from (e), prior to learning the spike response of the network neurons z_k is more or less random. Contrary, as is illustrated in (f), the spike trains yielded after unsupervised learning for $2000\,s$ (10000 random instances of the patterns shown in (a) being presented for $200\,ms$ each) are sparse with only a single network neuron being active for a given input pattern most of the time.

the evolution of the weights also allow for a rough quantitative assessment of the probabilistic model actually encoded in the network. According to Equations 5.13 and 5.18, the convergence weights levels of W_{1i} shown in Figure 5.6c translate to probabilities $p(Y_i = 1 \mid Z_1 = 1) = \{0.18, 0.43, 0.80, 0.91\}$. These levels resemble quite well the probabilities the network input was generated according to. The reasons for the present deviations may on the one hand be found in the linear model that was fitted to the original data points (see Figure 5.5a). On the other hand, also the original data points may not have captured the actual mapping of probabilities $p(Y_i = 1 \mid Z_k = 1)$ to convergence conductances G with the necessary fidelity. As we remember, the underlying simulations according to Protocol 7 only consider a single inter event interval Δt. Consequently, also the resulting mapping is valid for a single inter event interval only. In the actual network level simulations, however, a multitude of different values for Δt is encountered, which may lead to a distortion of the mapping. In addition to this, also the saturation effects which arise from the underlying memristor not being capable of attaining arbitrarily low or high conductive states (compare Sections 2.2.2 and 4.2) are not captured appropriately by the linear model we assumed for the simulation.

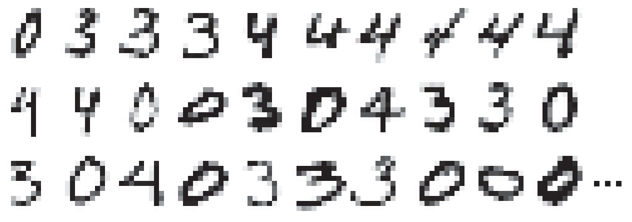
Nevertheless, as becomes clear from the spike responses illustrated in Figures 5.6e and 5.6f, the network is still able to adapt to the presented input. While prior to learning the spiking activity of the network neurons z_k appears to be quite random, after unsupervised learning for $2000\,s$ the network has learned to represent the input distribution quite well. As a result, the corresponding spike trains are sparse with only a single neuron being active for each of the input patterns most of the time. Plasticity within the memristive synapses in combination with an appropriately chosen pulsing scheme has turned each of the $K = 2$ network neurons into a probabilistic expert [BillLegenstein, 2014] for one of the patterns.

Learning of Handwritten Digits

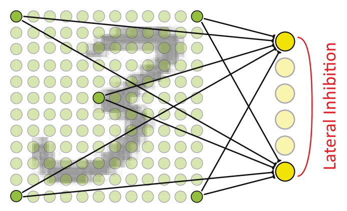
Since the results obtained for the small toy example of color gradients presented above appeared quite promising, a second, more realistic and practically relevant simulation was set up. This simulation attempts to autonomously train a classifier for images of handwritten digits taken from the MNIST dataset by Lecun et al. [Lecun, 1998] based on a WTA network with memristive synapses.

For this purpose, a limited subset consisting of 17896 images showing only 0s, 3s and 4s was extracted from the 60000 images contained in the MNIST training set. In the simulation, the resulting 28×28 pixel images are then cropped by a 2 pixel border and scaled down for 50 % to 12×12 pixels. Examples of the resulting images are shown in Figure 5.7a. Based on [BillLegenstein, 2014], the gray-scale values of these images are mapped to the interval $[0.05, 0.9]$, yielding a set of different input patterns $\boldsymbol{x}^s$. These input patterns are then transformed to spike trains according to Equation 5.21 by the input neurons y_i, where the pixels x_i^s are assigned to input neurons y_i in a row-wise manner (see Figure 5.7b for an illustration).

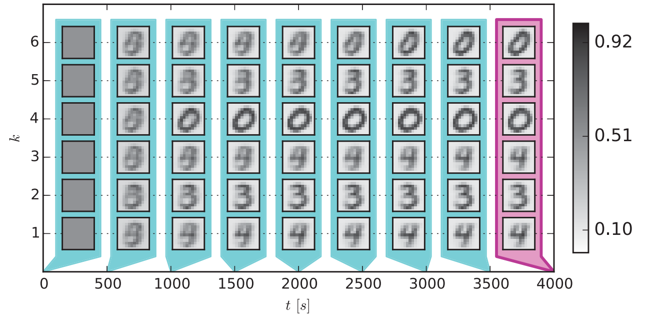
In this setup, over a period of $4000\,s$ random samples of the set of input patterns $\boldsymbol{x}^s$ are drawn and presented to the network for $200\,ms$ each (all other network and simulation parameters not explicitly mentioned here were chosen the same as for the color gradient simulation). Similar as for the color gradients described above and as is clearly visible form Figure 5.7c, starting from an initial midrange value, prototypes of the presented input patterns emerge gradually in the weights W_{ki}. While over the first couple of hundred seconds of unsupervised learning only the general and common area of interest, that is, the area in the center of the pixel cells, jointly emerges in the weight matrices for all network neurons (see for instance the weights after $500\,s$ shown in Figure 5.7c), after some time the memristive synapses have gathered enough spike pairs in order to turn each network neuron into a probabilistic expert for a given input pattern class. Since with $K = 6$ the number of network neurons is chosen greater than the number of different pattern classes actually contained in the presented input space, more than one network neuron z_k specializes on a given digit. More precisely, for instance, both z_4 as well as z_6 have learned to represent 0s. The exact versions of the 0s, however, are different. The same applies to neurons z_2 and z_5 as well as



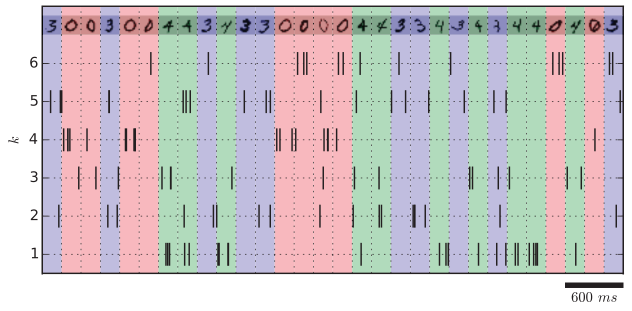
(a) Example 0s, 3s and 4s



(b) Network configuration



(c) Gradual emergence of prototypes in the weights W_{ki}



(d) Network spike response after learning

Figure 5.7: Learning of handwritten digits. 0s, 3s and 4s were extracted from the MNIST dataset of handwritten digits [Lecun, 1998], cropped to 24×24 pixels and scaled by 50 %, yielding 17896 training images (examples are shown in (a)). The gray-values of the resulting 12×12 pixel images are converted to spike trains and fed row-wise to the 144 input neurons y_i. As indicated in (b), these input neurons are connected in an all-to-all fashion to the network neurons z_k through memristive synapses (see Figure 5.4b). Each of the network neurons z_k specializes on a certain class of input patterns. Since with $K = 6$ there are more network neurons than different digits in input space, some of the neurons specialize on (slightly) different versions of the same digit (see (c)). After 4000 s of unsupervised learning (20000 training instances being presented for 200 ms each), the network has learned to convert the input generated from the images by the input neurons y_i into spike codes identifying the digit the image shows. As can be seen from (d), neuron z_4, for instance, exclusively spikes for 0s, while neuron z_1 has apparently specialized on 4s.

z_1 and z_3, which have pair-wisely specialized on two versions of 3s and 4s, respectively.

This specialization of the constituent network neurons on different input pattern classes becomes also evident from the spike response illustrated in Figure 5.7d. Most of the time, the network neurons z_k spike only for those test patterns (show in the upper area of the plot) they are, according to the weight matrices illustrated in Figure 5.7c, probabilistic experts for, and vice versa. Neuron z_1, for instance, fires almost exclusively for 4s, while neuron z_4 emits response spike only for 0s. In addition to this, also the specialization of multiple network neurons to a single digit class can be observed in the network spike response shown in Figure 5.7d. While both neurons specialized on 0s, neuron z_4 is more likely to fire for roundish versions of 0s, neuron z_6 predominantly spikes for narrower or slightly slanted instances. This is in perfect accordance with the weight matrices illustrated in Figure 5.7c. Instances of multiple neurons being active for a given input pattern as is the case, for instance, for the ninth "3" in the network response plot shown in Figure 5.7d, express uncertainty about the correct class a given input pattern belongs to.

Finally, it shall be emphasized that the mapping between digit class and the network neuron, which becomes an expert for it, happens on a purely random basis. It is a result of the underlying spike-based Expectation Maximization process [Nessler, 2013].

This concludes the discussion of the simulations at network level. As we saw from the results presented in the previous paragraphs, indeed the proposed synapses based on the slightly modified and fine-tuned memristor model we introduced are, in combination with an appropriate pulsing scheme, capable of implementing decent synaptic plasticity within spiking WTA networks. These WTA networks can be trained in order to solve demanding tasks such as the autonomous classification of handwritten digits.

6 Discussion

In this work we studied different memristor models according to their suitability for the implementation of plastic synapses for spiking WTA networks. During these studies, as one of the major findings, we identified voltage-based memristor architectures as being superior over current-based ones. This is the case due to a destructive and escalating feedback which is introduced by the latter ones in combination with the utilized learning environment.

In addition to this, based on a voltage-based memristor model proposal by Serb et al. [Serb, 2014], we identified five of the most important building blocks required for memristors in order to form a stable foundation for the implementation of the proposed memristive synapses. These major building blocks include low intensities as well as short decay times for transient processes within the memristors in response to plasticity pulses. Secondly, as the instantaneous conductance of the memristive devices is considered associated with the weight of the synapses they implement, a linear mapping of the internal memristor state onto the conductance G was identified as being preferable. Since most state-of-the-art memristor models combine a linear mapping between the internal state and the memristance M with a large dynamic range, an additional pre-resistor M_0 was identified as possible regulation for the resulting non-linearity. Next, an appropriate pulsing scheme which is capable of triggering the required reactions in terms of synaptic plasticity within the memristive synapses according to SEM is required. Inspired by Zamarreño-Ramos et al. [Zamarreño, 2011] and Querlioz, Bichler, and Gamrat [Querlioz, 2011], this pulsing scheme was found in the proposed *T pulses*, which consist of a single plain square pulse in the pre-synaptic and a combination of three square sub-pulses with different purpose in the post-synaptic case. Finally, the support for intrinsic stabilizing weight dependence was identified as probably the most important feature a memristor model has to offer in order to be a serious candidate for the implementation of memristive synapses. This weight dependence is closely related to the demand for a decoupling of the device's convergence conductance from its initial state.

Fine tuning (partly in a systematic parameter search) and slightly modifying the memristor model proposed by Serb et al. [Serb, 2014], finally indeed yielded a model which turned out to be capable of solving demanding tasks such as the recognition of handwritten digits in a spiking WTA network setup. It remains to be seen, however, whether memristive devices showing the exact properties we assumed for these final simulations can be fabricated with the necessary accuracy. Specifically, these properties include the stabilizing type of weight dependence as well as the continuous conductance spectrum, which has to be kept up over millions of programming cycles, we assumed.

Possible future work, as is also indicated in [BillLegenstein, 2014], heads for a full network implementation, which combines both synaptic plasticity as well as neural transmission. As mentioned in Chapter 5, these two mechanisms were decoupled for the purpose of this work, yielding, for instance, a distinction between the plasticity pulses and the input received by the abstract network neurons. In a combined setup, the pulses would likewise serve both purposes and trigger plasticity as well as be integrated to form the membrane potential of post-synaptic neurons. Simulations at this combined level would also include and consider the reaction of the (network) neurons to different pulse events.

Another possible link for future work is a more rigid investigation of the mapping between input statistics and convergence conductances. As we remember from Chapter 5, for the purpose of this work, we assumed a linear mapping in combination with a Gaussian likelihood model. Although these assumptions worked out well, they are basically no substitute for a rigid formal and mathematical analysis. As a starting point, this analysis could record the mapping curves in a more realistic setup by drawing the inter event intervals from a uniform distribution, rather than assuming a fixed value as was done here. The resulting data points could then serve as the foundation for identifying the probabilistic model actually implemented in the memristive synapses with higher accuracy.

Appendix

A SPICE vs. Python

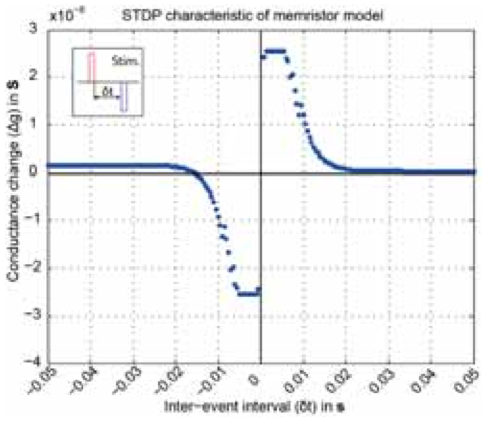
A.1 Introduction

This report aims to document the pitfalls that were identified during the comparison of two different implementations of a memristor model along with the corresponding insights that were gained. Both compared implementations are based on the same memristor model suggested in a paper draft by Serb et al. [Serb, 2014]. The investigated implementations were the original SPICE reference implementation the authors used to test their model on the one hand and a version written in Python on the other.

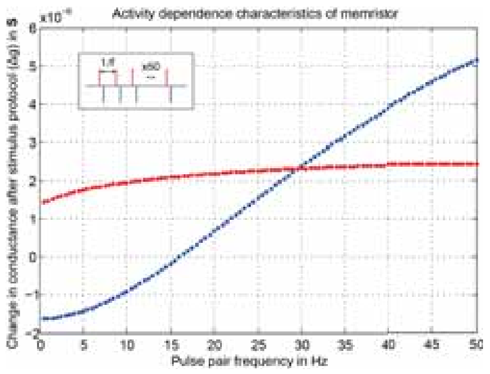
Originally, the Python implementation was thought to be identically equal to the SPICE reference implementation in that the produced results are the same for both versions. In an attempt to verify this, the experiments described by the two STDP test protocols defined in the Frontiers draft were run on the Python implementation. The comparison documented in this report was motivated by the observation that the results yielded by the initial Python implementation, however, deviated from the ones presented in the Frontiers paper draft in different ways. These discrepancies included the following [1]:

- *Shifting/Bias:* The STDP curve for Protocol 1 presented in {6} (compare Figure A.1a in this report) appeared to be shifted further into the positive half plane in the Python implementation. In other words, the STDP curve yielded by Python showed some positive bias, causing the flat, less responsive areas at the very left and right to be further away from the x-axis. Figure A.2a shows an example Python output.
- *Scaling:* In addition to the above-mentioned bias, the STDP curve for Protocol 1 also appeared to be scaled down by a factor of about 3 to 3.5 in the Python experiments. Also this phenomenon is clearly visible from the plots in Figure A.2a: If the STDP curve presented in this plot is imagined to be shifted down closer to the x-axis as it is supposed to be according to Figure A.1a, the amplitude of its positive peak is around 0.75. Looking at Figure A.1a again, we see that, according to the SPICE implementation, it should be somewhere around 2.5.
- *Curvature:* As can be see from the plots in Figures A.1b and A.2b, respectively, for the post-pre case of Protocol 2 (dotted blue) the curvature of the conductance change function was different in the Python implementation. In contrast to the SPICE reference results, the function seemed to be an exponential function of some kind, without any inflection points.
- *Zero-crossing:* Finally, the zero-crossing of the above mentioned conductance change function for the post-pre case of Protocol 2 was shifted closer to the origin in the Python experiments (compar. Figures A.1b and A.2b).

[1] To keep the descriptive texts short while preserving unambiguity, we decided to reference equations of the Frontiers draft by their numbers put in parentheses. *(11)*, for instance, should be read as *"Equation 11 in the Frontiers draft"*. Likewise, the numbers of figures contained in the Frontiers paper draft are put in curly brackets. *{6}* denotes *"Figure 6 contained in the Frontiers paper draft"*.



(a) Protocol 1



(b) Protocol 2

Figure A.1: SPICE results for Protocol 1 and Protocol 2 ({6} and {8} [Serb, 2014]).

In an attempt to track down the origin of these differences and to align the outputs of the two implementations, different tests were made. These tests will be described in the following sections along with the corresponding results and insights that were gained.

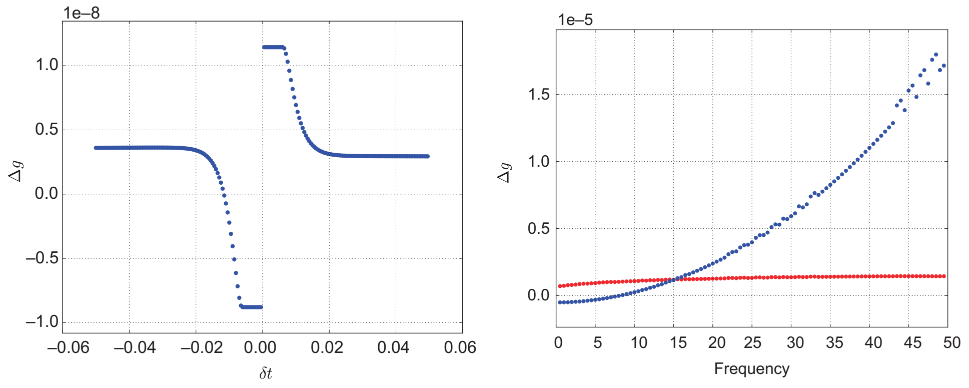
A.2 Experiments

A.2.1 Python Rewrite

In a first attempt to align the results of the two implementations, the Python version of the model was rewritten in accordance with the Frontiers paper draft. This, however, did not change the results – all of the above-mentioned discrepancies were still present.

A.2.2 Reimplementation according to SPICE Model

Since the rewrite of the Python implementation did not change the results at all, the authors of the Frontiers paper draft were contacted and

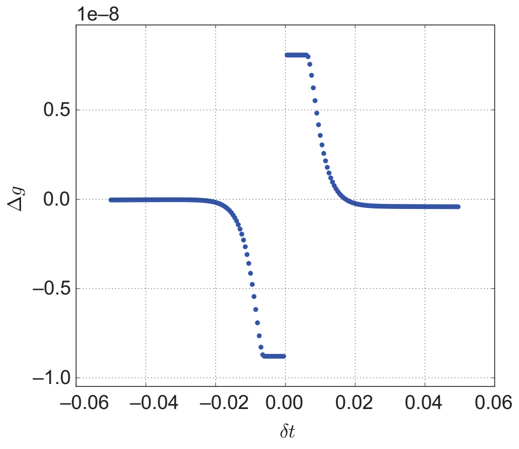


(a) Protocol 1 (b) Protocol 2

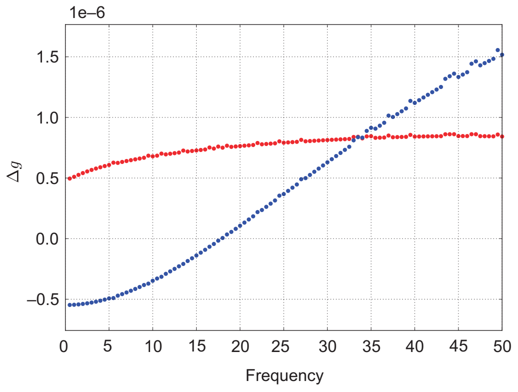
Figure A.2: Initial Python results for Protocol 1 and Protocol 2.

informed about the discrepancies. They rerun their SPICE simulations and more or less confirmed the results presented in the paper draft. In addition to the updated plots, the authors also provided the SPICE implementation their results were generated with. Thus it was possible to compare the SPICE and Python implementations against each other as well as the SPICE version with the model introduced in the paper draft. By doing so, additional scaling factors K_p and K_n for the two branches of $g(V_w)$ (compare (15)) were identified. These scaling factors were not mentioned in the paper draft, however the SPICE implementation used them. Moreover, the SPICE implementation used a different version of the differential equation defining the behavior of V_w (see (13)). More precisely, instead of using $|i(t) \cdot V(t)|$ as suggested in the paper draft, the SPICE implementation internally used $|i(t)|$. Serb et al. confirmed the scaling factors K_p and K_n as being correct, however identified the usage of $|i(t)|$ in the differential equation describing V_w as a mistake.

Updating the Python implementation according to this new information yielded the results shown in Figure A.3. As we can see, the bias problem for Protocol 1 appeared to be solved. The scaling issue, however, was still present (compare Figure A.3a). For Protocol 2, the results yielded by the updated Python implementation were also closer to the reference plots contained in the Frontiers draft in terms of the curvature. Also the zero



(a) Protocol 1



(b) Protocol 2

Figure A.3: Improved Python results for Protocol 1 and Protocol 2. After applying the changes described in Section A.2.2, the bias problem appeared to be solved. However, the scaling of the STDP curves was still not correct.

crossing was shifted away from the origin, closer to where it was expected to be according to the SPICE implementation (see Figure A.3b). Though the zero crossing was not exactly at the very same place.

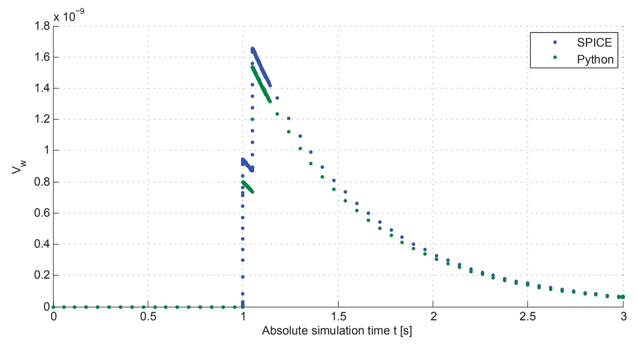
A.2.3 Direct comparison of the internal state variables

After informing Serb et al. about the new results which still deviated from the ones presented in the Frontiers draft, we were provided with their full SPICE/MATLAB-simulation framework (including the SPICE circuit simulator). So from this point on, we were able to run our own SPICE simulations. This situation was utilized in that a comparison script was written in MATLAB which was capable of comparing the actual evolution of all internal state variables V_w, V_x, V_y and V_z over time as estimated by the implementations in SPICE and Python. In order to keep things as simple as possible, only Protocol 1 (that is, a single pulse pair with a given inter event interval δt) was tested with this setup. The results of these comparisons are depicted in Figures A.4a to A.4d. As we can see from these results, the reactions of the various internal state variables are in general stronger in the SPICE implementation (that is, the peaks are higher). This is best visible from the plot showing the time course of V_y (see Figure A.4c). As we know from (10), on the one hand V_x decays towards V_y. On the other hand, we know from (9) that V_x ultimately influences the resting memristance. Put differently, we know that V_y has great influence on the resting memristance and the shape of the STDP curve of Protocol 1. This is why we found it worth taking a closer look at V_y in the experiments described in the next section.

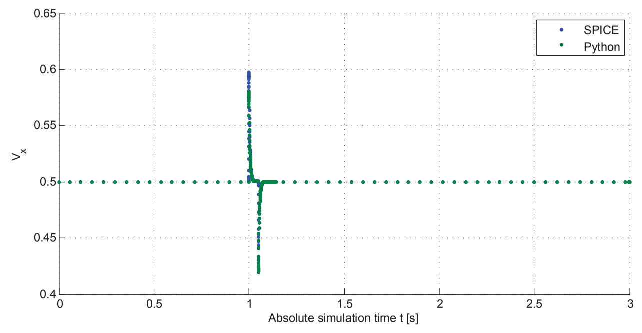
A.2.4 Slow/no decay

Having a look at (11), we notice that over $\phi(V_z)$ the dynamics of V_y, among other things, primarily depend on V_z. As can be seen from (14), $\phi(\cdot)$ involves the thresholds $B_\pm$ and $U_\pm$, respectively. According to Table I shown in the Frontiers draft, the values of these thresholds are $\pm 3.5 \cdot 10^{-10}$ and $\pm 5 \cdot 10^{-10}$, respectively. As we can see from the plot shown in Figure A.4d, the amplitudes of V_z are exactly within the range of these thresholds. In other words, slight differences (or imprecisions) in the computation of V_z performed by the two implementations of the memristor model could potentially give rise to dramatic differences in the evolution of $\phi(V_z)$ and ultimately (over (11)) V_y. This is the case, because the exact value of V_z could determine whether one of the above-mentioned thresholds is exceeded in the first place or not. As we know from the previous paragraph, V_y ultimately influences the resting memristance (and thus the STDP curves) over V_x. This is why in this experiment we concentrated on V_z in order to find out whether the still present discrepancies in the STDP curve for Protocol 1 resulted from differences in V_z.

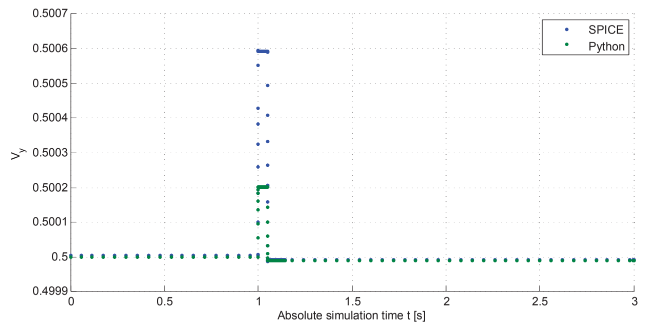
Given the dynamic description of V_z presented in (12), we know that V_z primarily depends on the input voltage $V(t)$. In order to be better able to see differences in V_z between the Python and the SPICE implementation, the model parameters were slightly modified. The rapid decay of V_z determined by the time constant $\tau_z = C_z R_z$ was slowed down while leaving



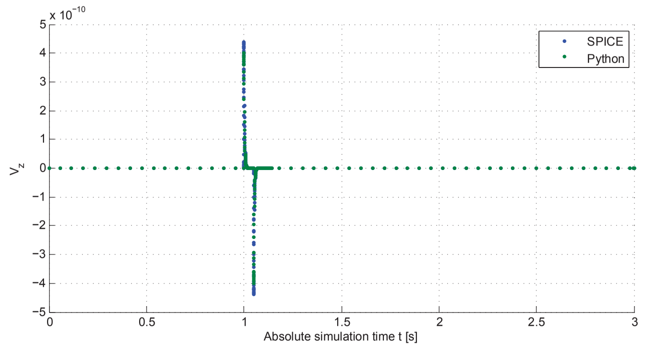
(a) Absolute power dissipation V_w



(b) Volatile state V_x



(c) Non-volatile state V_y



(d) Driving effortV_z

Figure A.4: Evolution of V_w, V_x, V_y and V_z for $\delta t = 50\,ms$ As can be seen from the plots, the SPICE implementation generally yields stronger reactions of the different state variables.

the initial response unchanged. This was achieved by choosing a much larger R_z and leaving C_z unchanged. In addition to this, the behavior of the input voltage was simplified. Instead of a spike pair with given IEI as suggested by Protocol 1, only a single positive spike with an amplitude of $V_{\text{pulse}} = 2\,V$ was used. Given the definition of the memristor model's dynamics, these simplifications allowed for an easy determination of the analytical solution of V_z.

We start with (12), that is, the dynamics of V_z, which reads

$$C_z \frac{dV_z}{dt} = i_z(t) - \frac{V_z(t)}{R_z} = \frac{V(t)}{M_{\text{mid}}} - \frac{V_z(t)}{R_z}\,, \tag{A.1}$$

where

$$M_{\text{Mid}} = \frac{M_{\text{max}} + M_{\text{min}}}{2}\,. \tag{A.2}$$

According to the Frontiers draft, $M_{\text{max}} = 10^5\,\Omega$ and $M_{\text{min}} = 1\,\Omega$.

Slowing down the decay of V_z, that is, choosing R_z sufficiently large (for instance $R_z = 3\,T\Omega$), makes the right-most term of Equation A.1 neglectable and turns the whole sub-circuit into a pure integrator. Thus, we can write

$$C_z \frac{dV_z}{dt} \approx \frac{V(t)}{M_{\text{mid}}} - 0 = \frac{V(t)}{M_{\text{mid}}}\,, \tag{A.3}$$

where, according to the Frontiers draft, $C_z = 1\,F$.

Solving this differential equation w.r.t. V_z we obtain

$$V_z(t) \approx \frac{1}{C_z\, M_{\text{mid}}} \int_0^t V(\xi)\, d\xi\,. \tag{A.4}$$

Applying an input waveform for $V(\xi)$ that comprises a single pulse at $t_{\text{spk}} = 1\,s$ with an amplitude of $V_{\text{pulse}} = 2\,V$ and a width of $t_{\text{active}} = 10\,\mu s$ and evaluating the resulting V_z according to Equation A.4 at $t = 3\,s$ yields:

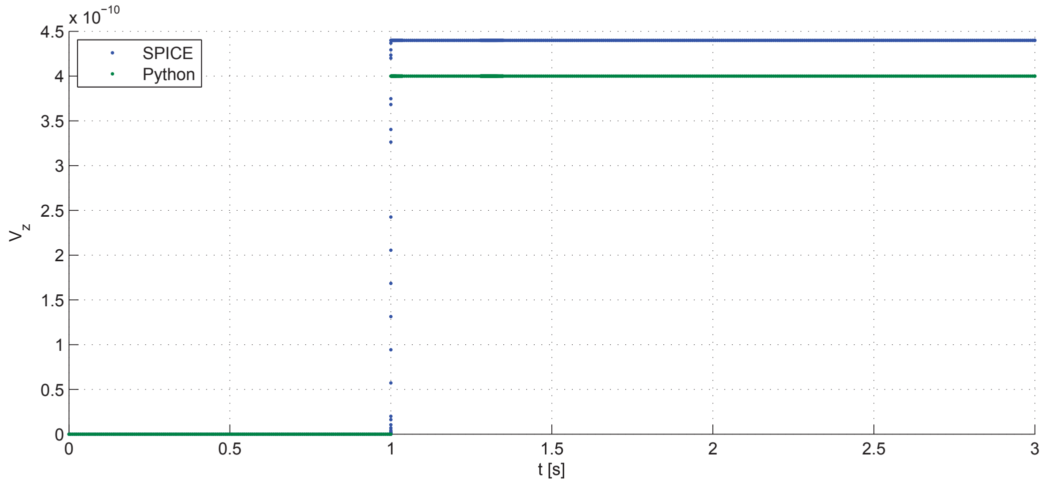
$$\begin{aligned} \underline{V_z(t)} &\approx \frac{1}{C_z\, M_{\text{mid}}} \left[0\big|_0^{t_{\text{spk}}} + V_{\text{pulse}} \cdot \xi\big|_{t_{\text{spk}}}^{t_{\text{spk}}+t_{\text{active}}} + 0\big|_{t_{\text{spk}}+t_{\text{active}}}^{t} \right] = \\ &= \ldots = \frac{4}{100001} \cdot 0.000010 = \underline{3.99996 \cdot 10^{-10}} \end{aligned} \tag{A.5}$$

Figure A.5 shows a comparison of the different results achieved for this test setup (including the adapted parameters) by SPICE and Python, respectively. In SPICE, the single input pulse was implemented using two different built-in voltage sources. On the one hand, the piecewise linear source (`PWL`) was used with the same parameters as in the original SPICE framework (rise and fall time of $1\,\mu s$). On the other hand, for the second simulation run the input waveform was generated by the built-in pulse source (`PULSE`). For the pulse source, the rise time was set to zero (important additional notes on this are discussed in Section A.2.5). The Python implementation used Euler integration and a simulation time step size $dt = 10\,\mu s$ to solve the differential equations numerically.

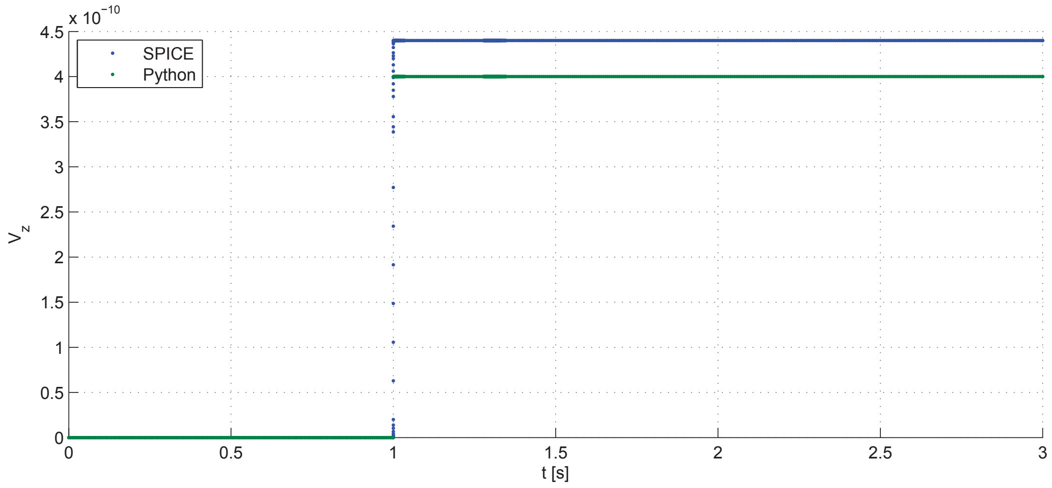
As we can see from the plots in Figure A.5, the results yielded by Python resembled the expected value for V_z we calculated in Equation A.5 closer. The results by SPICE, however, seemed to show deviations of the analytic solution determined in Equation A.5. Figure A.6 shows the evolution of V_z as determined by the Python implementation. In contrast to Figure A.5, for these plots smaller simulation time step sizes were chosen ($dt = 5\,\mu s$ and $dt = 1\,\mu s$). The difference between the evolution of V_z in these plots and the ones presented in Figure A.5, however, is not visible with the naked eye. This apparent numerical stability of the Python simulations as well as fact that the Python results were closer to the allegedly correct analytical solution, strengthened our confidence in the accuracy of the Python-based solution.

A.2.5 Finite slew rates

In addition to the alleged confirmation of the Python model mentioned in the previous section, the observation that both the piecewise linear and the pulse source had led to almost the same results in the SPICE simulations (see Figures A.5a and A.5b), gave rise to the assumption that the limited slew rate of the pulse source was without influence on the evolution (and especially the final value) of V_z.

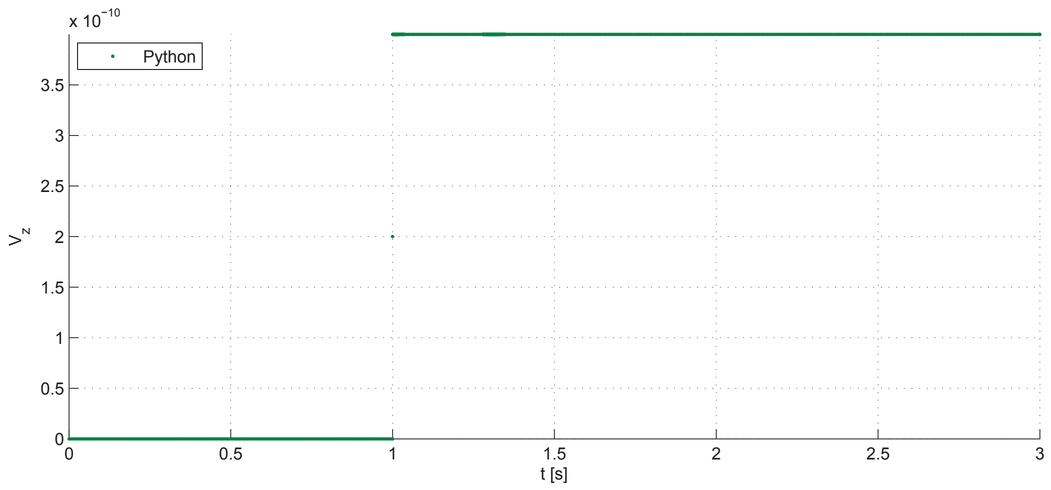


(a) PWL source

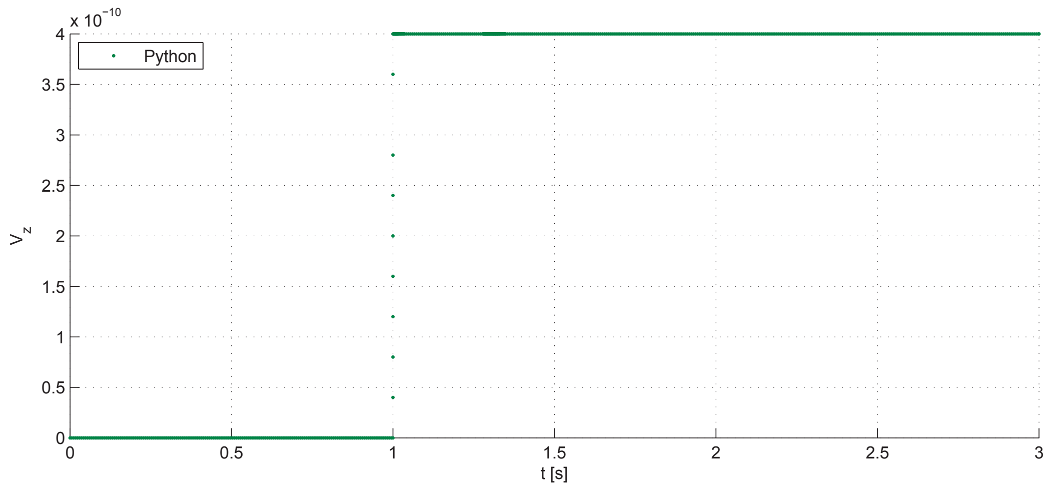


(b) PULSE source

Figure A.5: Comparison of the temporal evolution of V_z. The applied input comprises a single square pulse with an amplitude of $V_{\text{pulse}} = 2\,V$ being generated by voltage sources of two different types in case of the SPICE results. For the underlying simulation, a large decay time constant τ_z was chosen by setting $R_z = 3\,T\Omega$, turning the corresponding module qualitatively into a pure integrator.



(a) $dt = 5\,\mu s$



(b) $dt = 1\,\mu s$

Figure A.6: Evolution of V_z as yielded by Python. The plots show the evolution of the drive effort V_z for a single input pulse with an amplitude of $V_{\text{pulse}} = 2\,V$ for two different simulation time step sizes. Again, very large decay times τ_z was chosen.

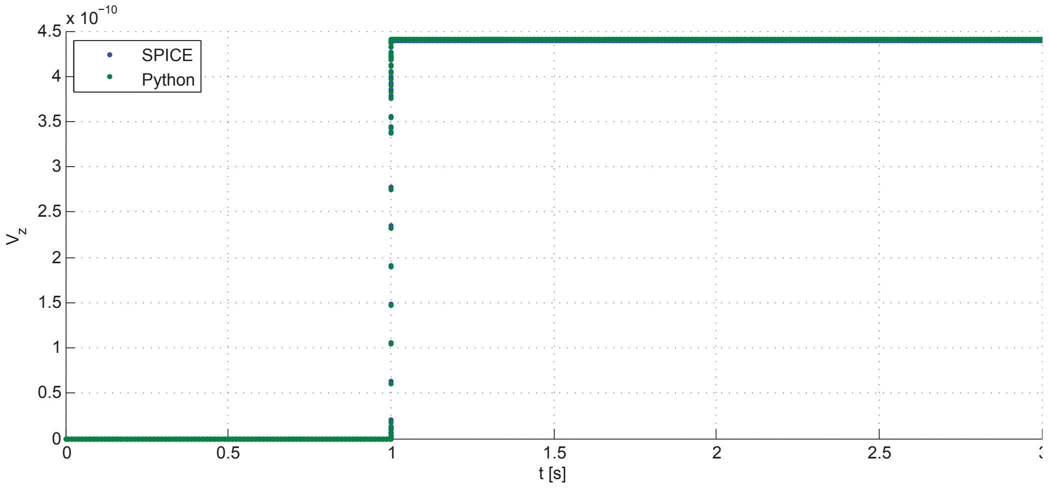
However, in another correspondence Serb et al. pointed out that they had used rise and fall times greater than zero in their original SPICE simulations. This made us rethink our conclusions and assumptions. First of all, we redid the math presented in the previous section. This time taking into account the limited slew rates (i.e. the rising and falling slopes). Adding these rise and fall times of $t_{\text{rise}} = t_{\text{fall}} = 1\,\mu s$ to the voltage pulse and having a closer look at the resulting shape of the input voltage revealed that, referring to the integral in Equation A.4, exactly one-tenth of the area of the perfect rectangle we had assumed so far was contained in the leading and trailing slopes. Obviously, adding these areas to the perfect rectangle notably increased the area of the pulse as well as the overall value of the integral in Equation A.4. This, in turn, of course meant that the analytical solution determined in the previous section was highly likely to be incorrect and that probably the Python implementation still did not produce the correct results. The analytical determination of V_z with the updated input voltage $V(t)$ including rising and falling edges with finite slew rate (and some simplifications for brevity) reads

$$\begin{aligned}\underline{V_z(t)} &\approx \frac{1}{C_z\, M_{\text{mid}}}\left[2\cdot\frac{V_{\text{pulse}}}{t_{\text{rise}}}\cdot\frac{\xi^2}{2}\bigg|_0^{t_{\text{rise}}} + V_{\text{pulse}}\cdot\xi\big|_0^{t_{\text{active}}}\right] = \\ &= \ldots = \underline{4.399959\cdot 10^{-10}}\,.\end{aligned} \tag{A.6}$$

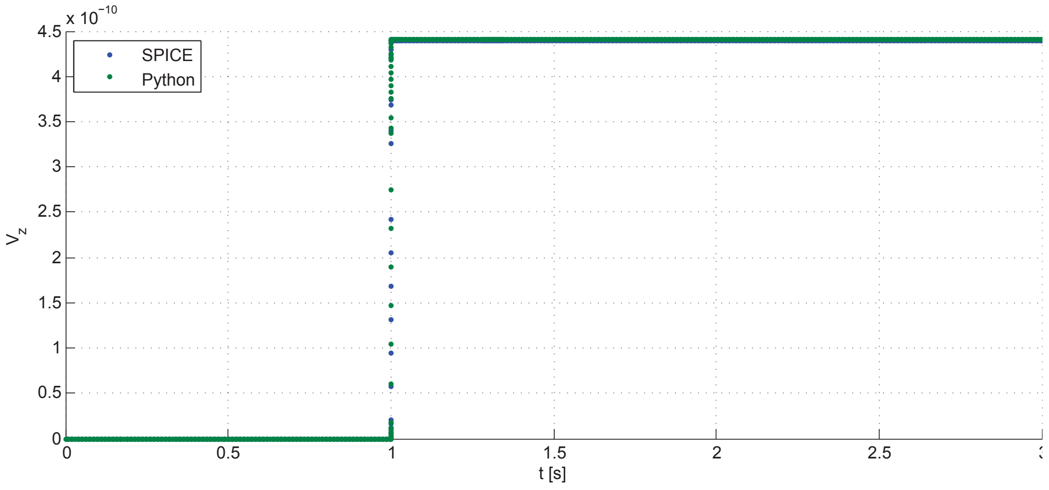
Comparing this new analytic result with the plots shown in Figures A.5a and A.5b again, we noticed that the SPICE results for both the pulse and the piecewise line source were better aligned with the expected value.

Rerunning the Python simulations with the adapted pulse shape (i.e. leading and trailing slopes with rise and fall times $t_{\text{rise}} = t_{\text{fall}} = 1\,\mu s$) yielded the results presented in Figure A.7. As we can see, the evolution of V_z over time was now almost identical for both the two SPICE and the Python implementation. In addition to this, all results resembled the analytical solution quite well.

These new observations, of course, raised the question why the results for both the piecewise linear and the pulse source were identical. Or, put differently, why the rise and fall times were without influence in the



(a) PWL source



(b) PULSE source

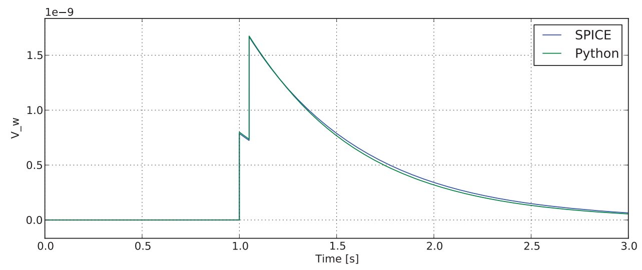
Figure A.7: Comparison of the temporal evolution of V_z. The plots show the response of the modified system in terms of the drive effort V_z to single input pulses with an amplitude of $V_{\text{pulse}} = 2\,V$ as yielded by SPICE and Python for a large decay time τ_z. In contrast to Figure A.5, finite slew rates were considered in the Python implementation.

SPICE simulations (recall that we had set both parameters to zero for the `PULSE` source), while they apparently definitely had an impact on the Python implementation. Rerunning the SPICE implementation with a pulse source and having a closer look at the log files revealed two rather interesting lines: `Limiting rise time of source v1 to 1e-006` and `Limiting fall time of source v1 to 1e-006`. So instead of using zero, SPICE had limited both T_r and T_f to $1\,\mu s$. In other words, SPICE had implicitly aligned the `PWL` and `PULSE` sources to behave identically. No wonder, the results were identical too. This also contradicted (or at least did not support) the assumptions concerning the missing influence of the rise and fall times we made in Section A.2.4.

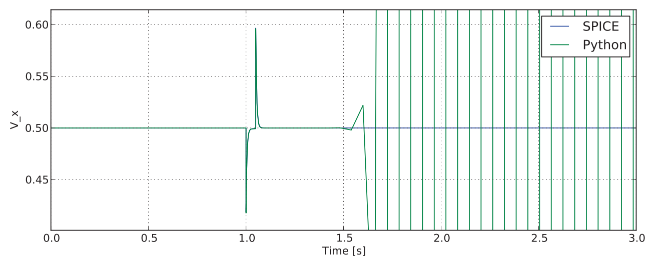
A.2.6 Multiplicative Updates

As described in the previous section, the results for the modified parameter set (slow/no decay) and the corresponding test setup (only a single positive spike) were now almost identical for both the SPICE and the Python implementation. When testing the results for Protocol 1, however, a new problem emerged. The adaption of the shape of the input voltage (finite slew rates) required the simulation time step size dt for the Euler integration employed by the Python implementation to be adapted too. Since we were uncertain about the correct choice of dt and various other attempts turned out to be impractical, we decided to choose it according to the SPICE simulation results (important additional notes on this topic are discussed in Section A.2.8). This, however, made the simulation numerically unstable. More precisely, V_z started to oscillate and made V_x oscillate too. Since the oscillation diverged, V_y ran into its boundaries. This behavior is depicted in the plots presented in Figures A.8b to A.8d. In addition to this, the relatively slow decay of V_w was also rather imprecise in the Python simulation now (see Figure A.8a).

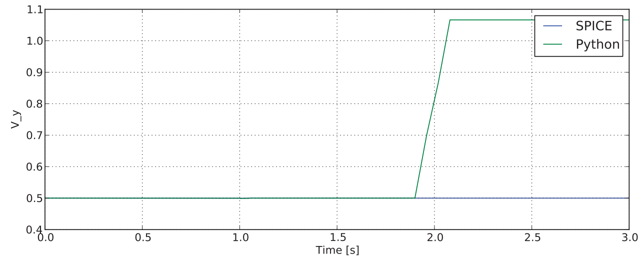
In an attempt to achieve better robustness against these numerical artifacts, in the next step some of the variable updates used in the Python implementation to solve the differential equations were changed to use a



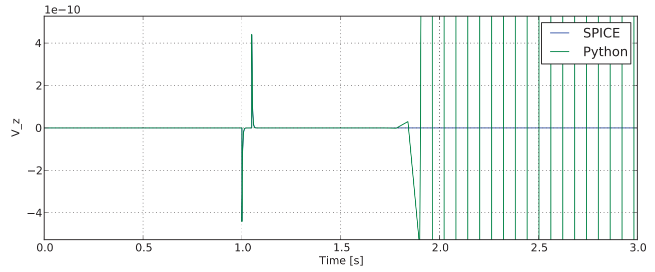
(a) Absolute power dissipation V_w



(b) Volatile state V_x



(c) Non-volatile state V_y



(d) Driving effor V_z

Figure A.8: Unstable evolution of V_w, V_x, V_y and V_z for $\delta t = -50\,ms$. For the Python implementation, plain Euler integration with simulation time step size dt according to the SPICE simulation was used. Clearly, this adjustment made the Python simulation unstable (V_x and V_z) and imprecise (V_w).

multiplicative update instead of pure Euler integration. The idea behind this multiplicative update rule can be described as follows:

We start with a differential equation of the form

$$\frac{dx}{dt} = -\gamma \cdot (x - a)\,, \tag{A.7}$$

where $x(t)$ is the variable, whose time course we are interested in. γ and a are real valued constants. The solution of this differential equation is given as

$$x(t) = C \cdot e^{-\gamma \cdot t} + a\,, \tag{A.8}$$

where $C = x(t)|_{t=0} - a$. Given the expression in Equation A.8, we can derive a relation for $x(t + dt)$, the value of $x(t)$ an arbitrary amount of time dt after time t:

$$\begin{aligned} \underline{x(t+dt)} &= C \cdot e^{-\gamma \cdot (t+dt)} + a \\ &= C \cdot e^{-\gamma \cdot t - \gamma \cdot dt} + a \\ &= \underbrace{C \cdot e^{-\gamma \cdot t}}_{x(t)-a} \cdot e^{-\gamma \cdot dt} + a \\ &= \underline{[x(t) - a] \cdot e^{-\gamma \cdot dt} + a} \end{aligned} \tag{A.9}$$

Put into words, Equation A.9 suggests that $x(t)$ can be updated to $x(t + dt)$ by a multiplication with factor $e^{-\gamma \cdot dt}$ and some addition operation $\pm a$. Notably, this type of numerical integration by itself is exact for any choice of dt. One limit of accuracy that remains is, of course, the precision of the floating point unit.

So far, however, we only considered the case without external input. Adding an arbitrary external input $I(t)$ to Equation A.7 yields

$$\frac{dx}{dt} = -\gamma \cdot (x - a) + I(t)\,. \tag{A.10}$$

In order to allow for a multiplicative update in a numerical solution of this differential equation as indicated in Equation A.9, we need to adjust this expression and make some assumptions. More precisely, we assume the external input $I(t)$ to be constant for short amounts of time dt. In addition to this, we rearrange Equation A.10 as follows:

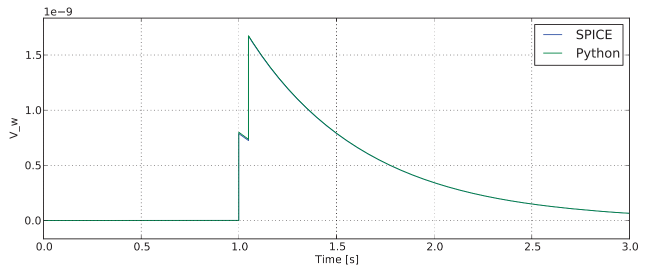
$$\begin{aligned}\frac{dx}{dt} &= -\gamma \cdot \left(x - \left[a + \frac{I}{\gamma}\right]\right) \\ &= -\gamma \cdot (x - A) \ ,\end{aligned} \tag{A.11}$$

where A is a constant. Again, the solution of this differential equation can be approximated with an approach similar to the one described in Equation A.9. In this case, however, the update (or more precisely the solution the iterative update process converges to) is not exact anymore. An additional disadvantage of this method is that it cannot be used for differential equations that describe pure integrators (for instance (11) describing the time course of V_y).

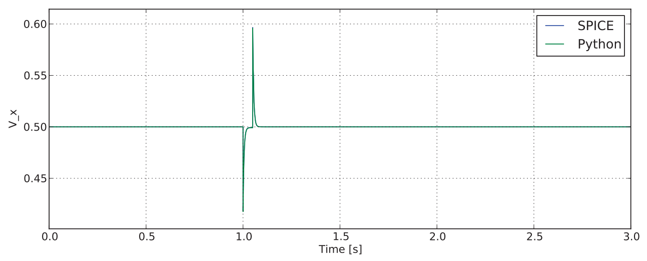
Nevertheless, as we can see from the plots shown in Figures A.9a to A.9d, the multiplicative update strategy helped in making the Python simulation stable. Neither did V_x and V_z oscillate anymore (see Figures A.9b and A.9d) nor ran V_y into the boundaries. Rather all quantities behaved very similar as in the SPICE simulation (compare Figure A.9c). Notably, compared to Figure A.4c, the Python results for V_y improved since the difference in the maximum peak amplitude of the time courses of V_y yielded by the SPICE and the Python implementation decreased. In addition to this, we also observed that the evolution of V_w (especially its relatively slow decay) matched the SPICE results quite close now (compare Figure A.9a). Finally, as can be seen from Figure A.10, also the results for Protocol 1 had improved. Although in the regions with low IEI δt now the Python implementation showed heavier reaction than the SPICE version, the relative difference between the two implementations had clearly become smaller.

A.2.7 Limited Precision

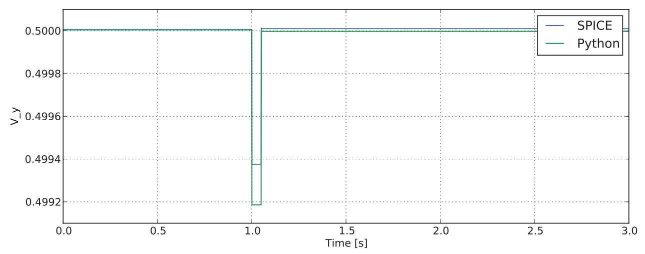
As mentioned in Section A.2.6, using multiplicative updates the Python simulation was now stable and resembled the SPICE results better than



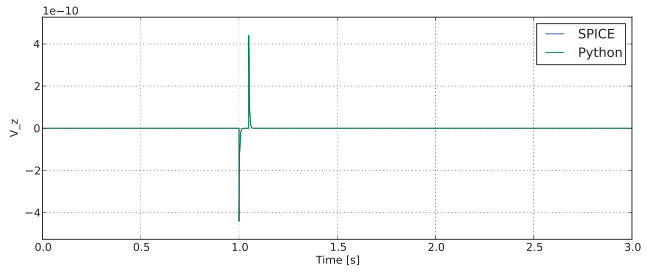
(a) Absolute power dissipation V_w



(b) Volatile state V_x



(c) Non-volatile state V_y



(d) Driving effort V_z

Figure A.9: Evolution of V_w, V_x, V_y and V_z for $\delta t = -50\,ms$. In contrast to Figure A.8 multiplicative updates were used where applicable for the Python implementation. The simulation time step size dt was chosen according to the SPICE simulation. Clearly, using the multiplicative updates had improved the results, even though V_y showed havier reactions than SPICE now.

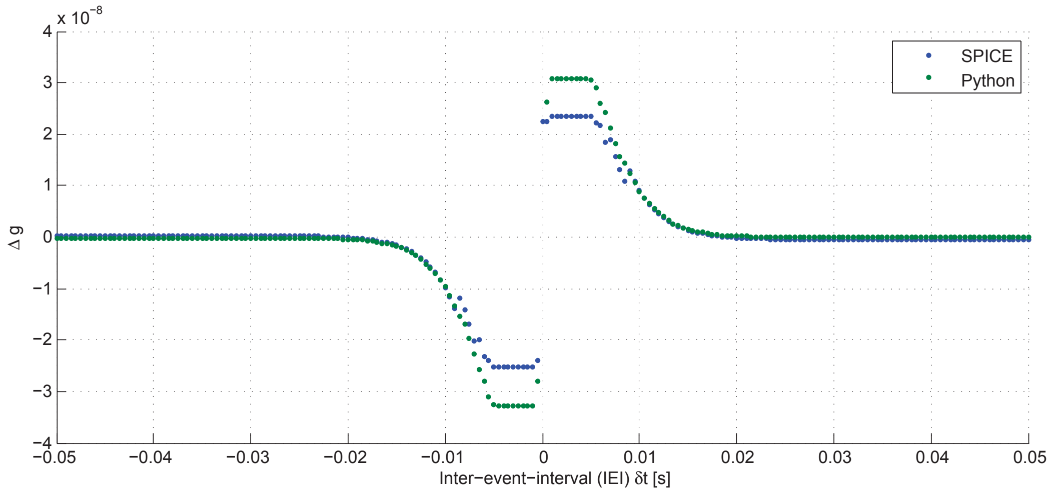


Figure A.10: Comparison of the results for Protocol 1. For Python, multiplicative updates were used where applicable. In addition to this, slew rates were considered to be finite.

ever before. There were, however, still some differences in some of the internal variables (especially in V_y, compare Figure A.9c). In an attempt to find the origin of these remaining discrepancies, we had a closer look at V_w and V_z, since both affect V_y over (11). First, we had a look at $g(V_w)$ and its evolution over time. Looking at Figure A.11, we noticed that the course of $g(V_w)$ after the positive spike was dramatically different for SPICE and Python. Comparing this observation with the definition of $g(\cdot)$, we presumed the root of this difference in the case differentiation in (15). Apparently, SPICE used a different branch of the piecewise definition of $g(\cdot)$ as Python did. As we can see from (15), one of the branches of $g(\cdot)$ is for $V_z > 0$, the other for $V_z \leq 0$. In order to check, which of the two branches was used at which instant of time in each of the implementations, the truth value of $(V \leq 0)$ was plotted against the time for both SPICE and Python. The results of this experiment are shown in Figure A.12.

From this plot we observed that in the SPICE implementation the truth value of $(V_z \leq 0)$ (and thus the branch of $g(\cdot)$) switched multiple times. This seemed rather odd, because in the area in question the course of V_z was characterized by a positive exponential decay. In theory, the value of this decay never exactly reaches zero. So the truth value of $(V_z \leq 0)$

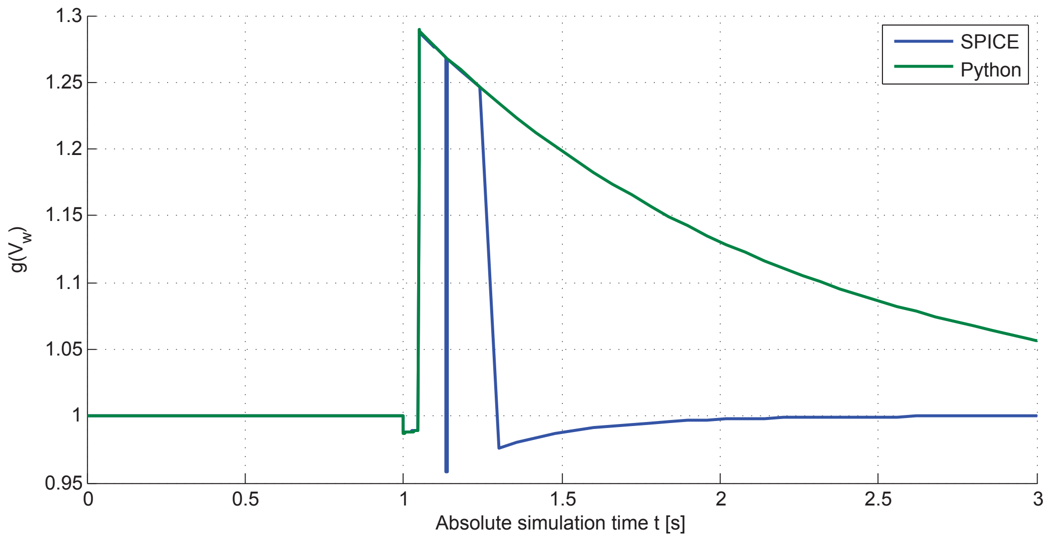


Figure A.11: Evolution of $g(V_w)$ for $\delta t = -50\,ms$. Multiplicative update for Python where applicable, dt according to the SPICE simulation.

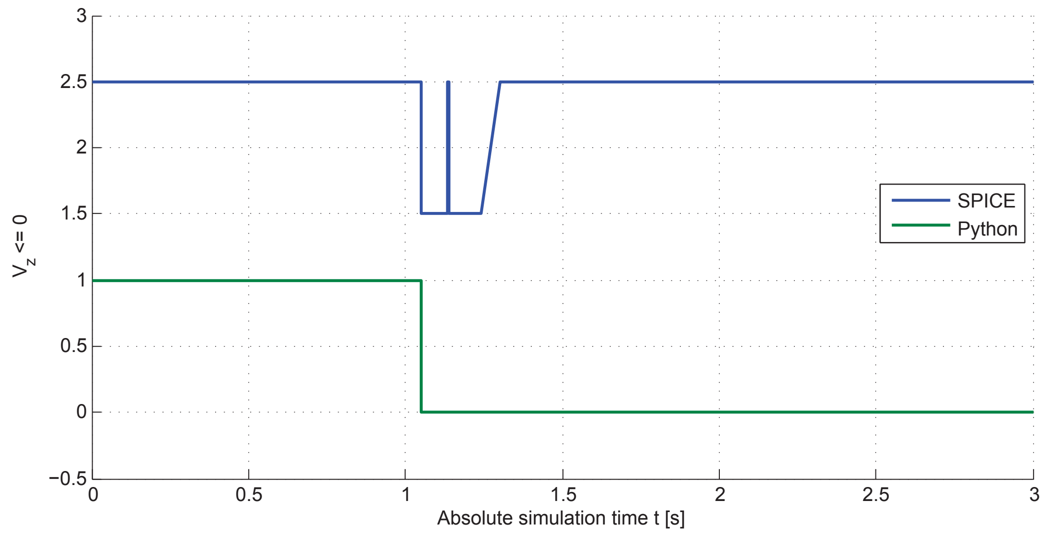


Figure A.12: Evolution of $(V_z \leq 0)$ for $\delta t = -50\,ms$. For SPICE, 1.5 and 2.5 and for Python 0.0 and 1.0 correspond `false` and `true`, respectively.

should always be `false` once the positive spike takes effect. Looking at the plot in Figure A.12 again, we noticed that this was only true for the Python implementation. In other words, in the SPICE implementation, the decay had at least reached zero (or even become negative). Having a look at the exact values of V_z revealed that in the SPICE implementation values of V_z with a magnitude below approximately 10^{-37} were truncated to 0.0, which caused the branch in (15) to be switched. Having a look at the definition of the IEEE 754 floating point format, we noticed that this value of about 10^{-37} was almost equal to the smallest non-zero value representable by an IEEE 754 32-bit single precision float. This led us to the conclusion that SPICE internally uses single precision floats, while our Python simulation used doubles. As we have seen, this leads to the problematic behavior of $g(V_w)$. This problem, however, is without influence at least for the discussed differential equation (11) describing the time course of V_y. This holds, because $\phi(V_z)$ is zero in the affected regions, making the wrong values of $g(V_w)$ unharmful.

A.2.8 Simulation Time Step Size

In a final attempt to track down the roots of the still present differences in V_y, we reduced the simulation time step size used for the Python implementation. As mentioned in Section A.2.6, so far we had chosen dt according to the simulation results yielded by SPICE. Now, however, we reduced the step size and chose a constant $dt = 5\,ns$. In order to speed up the simulations, we limited the time interval to the “interesting part” around the two spikes. A sample result of these simulations is shown in Figure A.13. As we can see, the difference in the maximum negative amplitude of V_y between the SPICE and the Python results became smaller. Hence, we found it conceivable that reducing the simulation time step size for the Python implementation even further could aid in increasing the accuracy (and better align the SPICE and the Python results). This, however, raised the question, why the time steps we had adopted from the SPICE simulation did not lead to accurate results in Python (or at least very similar ones as in SPICE). At this point, we remembered that a SPICE

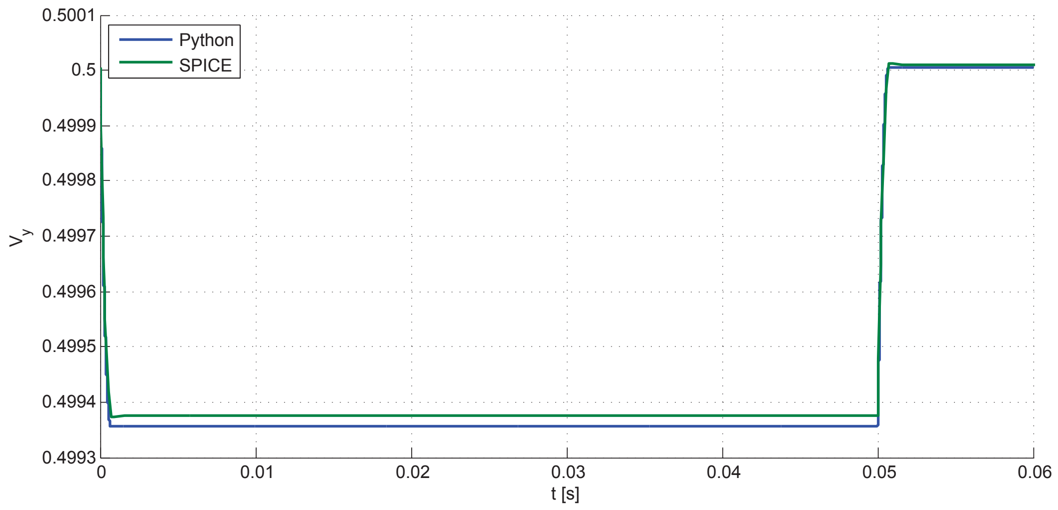


Figure A.13: Evolution of V_y. The plot shows the results as yielded by SPICE and Python for a simulation time step size $dt = 5\,ns$.

manual had stated that SPICE used two different time steps. One for the numerical circuit simulation and one for reporting the current circuit state. These two step sizes need not be the same and only the latter is accessible through the simulation output. So we had used SPICE's reporting time steps for the numerical solution of the differential equations in Python. Hence, the root of the still present differences in V_y as well as the fact that decreasing dt improved the results in Python could originate from this circumstance.

This conclusion motivated a final experiment. We returned to the choice of dt, the simulation time step size, and tried once again to come up with a rule describing its dependence on the shape of the input voltage $V(t)$. It was clear that in the affected regions dt had to be small enough to capture the rising and falling edges of the voltage as well as the rectangular pulses while being wide enough anywhere else in order to ensure reasonable simulation times. Hence, we decided to introduce three different step sizes

- $dt_{\text{fine}} = 100\,ns$ for the spikes' rising and falling edges (ensures 10

samples on the slopes),
- $dt_{\text{medium}} = 1\,\mu s$ for the square shape of the spikes themselves (10 samples) and the time between the spikes (the IEI δt) and
- $dt_{\text{sparse}} = 2\,ms$ for anywhere else.

Depending on the current “state” of the input voltage (that is, before spike, rising or falling edge, pulse peak, ...), we then switched back and forth between these different simulation time step sizes. An important task was to handle these switches properly and to prevent switching to a smaller dt too late or to a larger one too early. Employing this new rule for the choice of dt in combination with the multiplicative updates described in Section A.2.6, yielded the final results presented Figure A.14. As we can see, the STDP curve for Protocol 1 yielded by this final Python implementation was now almost identical to the one yielded by the SPICE reference implementation.

A.3 Summary and Conclusions

The observations we made in the experiments presented in the previous sections can be summarized as follows:

- When applying a single spike and removing/slowing down the decay of V_z, SPICE and Python yield almost identical results.
- SPICE limits the rise and fall times T_r and T_f for `PULSE` sources. Not taking into account the slopes of the input voltage V in the Python model made the simulations less responsive.
- Using the time step sizes as suggested by the SPICE simulation, the Python simulation is only stable when using multiplicative updates instead of plain Euler integration (where applicable) to solve the differential equations. This update method also improves the results for Protocol 1.
- SPICE internally apparently uses single precision floats, Python (at least) doubles. This causes unexpected branch switches functions, whose definitions involve case differentiations.

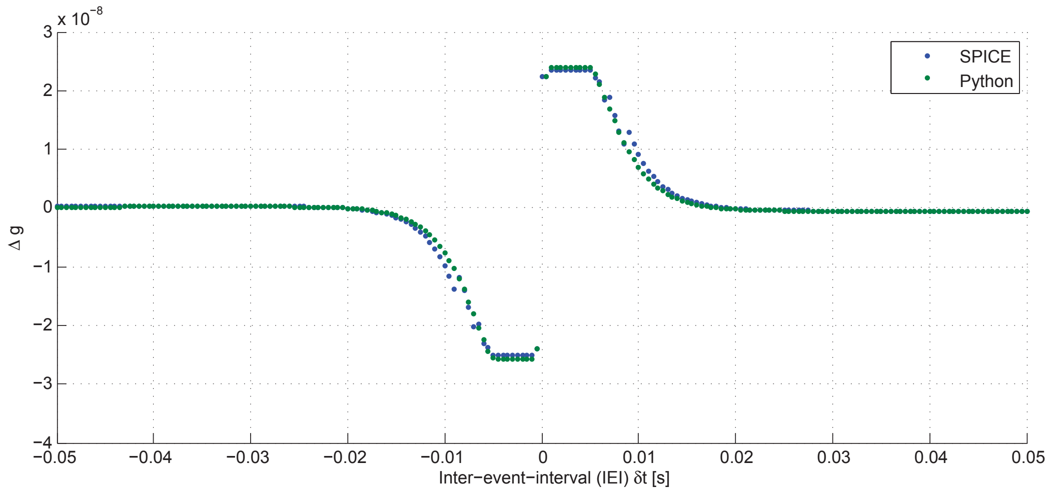


Figure A.14: Comparison of the final results for Protocol 1. The plot shows the results as yielded by the SPICE reference implementation as well as the final Python implementation.

- SPICE uses two different time step sizes: One to simulate the implemented circuit and solve corresponding differential equations and a second one for reporting the current circuit state. These seem to be different. Reducing the simulation time step size for Python improved the results.
- Choosing the Python simulation time step size according to the shape of the input voltage (that is, small dt for the rising and falling edges, medium dt for the spikes and the regions in between, large dt anywhere else) yields good results while preserving a reasonable simulation time.

Using all the techniques described in Sections A.2.1 to A.2.8, the Python and the SPICE implementation could be aligned almost completely. Hence, we decided to stop at this point and accept the Python model as it was. For Protocol 1, the final Python implementation yielded the results presented in Figure A.14.

Concluding with the “ultimate lessons learned” (or the most important conclusions that were made), three things can be said. Firstly, for the numerical solution of the differential equations of the presented memristor model, multiplicative updates are in general better suited. Secondly, it is important to impose some maximum simulation time step size dt if the external input $V(t) \neq 0$ in order to preserve both stability as well as accuracy. And finally thirdly, V_y is rather sensitive to changes in $B_\pm$ and V_z. This raises the question whether there is a way to adjust the model by selecting more stable parameters.

B

Derivation of $V_z(t)$

In this chapter we will take a closer look at one of the equations used heavily throughout this work. More precisely, we will derive an expression for the time course of the driving effort V_z and take a closer look at how it behaves in response to pulse shaped input voltages.

B.1 Generic Expression

We start by recalling Equations 2.18 and 2.19, describing the dynamic system governing the driving effort $V_z(t)$:

$$C_z \frac{dV_z}{dt} = \frac{V(t)}{M_{\text{mid}}} - \frac{V_z(t)}{R_z} \tag{B.1}$$

As we mentioned earlier, this equation is a first order differential equation of the generic type

$$\frac{dy}{dt} + a(t)\, y(t) = f(t)\,.$$

The solution for differential equations of this type is well known and given by

$$y(t) = \left[\int f(t)\, e^{\int a(t)\, dt}\, dt + c \right] e^{-\int a(t)\, dt}\,, \tag{B.2}$$

where c is an arbitrary constant. Rewriting Equation B.1 helps to identify the terms $a(t)$ and $f(t)$:

$$\frac{dV_z}{dt} + \underbrace{\frac{1}{C_z\, R_z}}_{a(t)}\, V_z(t) = \underbrace{\frac{V(t)}{C_z\, M_{\text{mid}}}}_{f(t)}$$

Plugging these expressions into the rule given in Equation B.2, we obtain:

$$V_z(t) = \left[\int \frac{V(t)}{C_z\, M_{\text{mid}}}\, e^{\int \frac{1}{C_z\, R_z} dt}\, dt + c \right] e^{-\int \frac{1}{C_z\, R_z} dt}$$

For a constant input with an amplitude of $V(t) = A$, this translates to

$$V_z(t) = \left[\int \frac{A}{C_z\, M_{\text{mid}}}\, e^{\int \frac{1}{C_z\, R_z} dt}\, dt + c \right] e^{-\int \frac{1}{C_z\, R_z} dt} \tag{B.3}$$

As we can see, in a nutshell the above expression consists two exponential functions with integrals in their exponent as well as some multiplications and another integral. Basically, both exponentials contain the same easy to solve integral:

$$\underline{\int \frac{1}{C_z\, R_z}\, dt} = \frac{1}{C_z\, R_z} \int dt = \underline{\frac{t}{C_z\, R_z}} \tag{B.4}$$

Plugging this expression into the exponential contained in the integrand of the outer integral yields

$$\begin{aligned}\underline{\int \frac{A}{C_z\, M_{\text{mid}}}\, e^{\frac{t}{C_z\, R_z}}\, dt} &= \frac{A}{C_z\, M_{\text{mid}}} \int e^{\frac{t}{C_z\, R_z}}\, dt = \\ &= \frac{A}{\cancel{C_z}\, M_{\text{mid}}}\, \cancel{C_z}\, R_z\, e^{\frac{t}{C_z\, R_z}} = \\ &= \underline{\frac{A\, R_z}{M_{\text{mid}}}\, e^{\frac{t}{C_z\, R_z}}} \end{aligned} \tag{B.5}$$

Rewriting Equation B.3 considering the expressions we obtained in Equations B.4 and B.5 yields

$$\begin{aligned}\underline{V_z(t)} &= \left[\frac{A\, R_z}{M_{\text{mid}}}\, e^{\frac{t}{C_z\, R_z}} + c\right] e^{\frac{-t}{C_z\, R_z}} = \\ &= \frac{A\, R_z}{M_{\text{mid}}}\, \cancel{e^{\frac{t}{C_z\, R_z}}}\, \cancel{e^{\frac{-t}{C_z\, R_z}}} + c \cdot e^{\frac{-t}{C_z\, R_z}} = \\ &= \underline{\frac{A\, R_z}{M_{\text{mid}}} + c \cdot e^{\frac{-t}{C_z\, R_z}}} \end{aligned} \tag{B.6}$$

Given Equation B.6, we have now specified the evolution of $V_z(t)$ for a given constant input A up to an arbitrary constant c. The value of this constant, however, can be determined by solving initial value problems. Generally, since Equation B.6 is supposed to hold for all possible times t, it has to be true also for $t = 0$. Assuming that at time $t = 0$, V_z had a certain preload $V_z(t = 0) = V_{z,0}$, we can derive a generic expression for c:

$$\begin{aligned} V_z(t=0) = \frac{A\, R_z}{M_{\text{mid}}} + c \cdot \underbrace{e^{\frac{0}{C_z\, R_z}}}_{=1} = \frac{A\, R_z}{M_{\text{mid}}} + c &\overset{!}{=} V_{z,0} \\ \Leftrightarrow c &\overset{!}{=} V_{z,0} - \frac{A\, R_z}{M_{\text{mid}}} \end{aligned} \tag{B.7}$$

Plugging the expression given in Equation B.7 back into Equation B.6

yields the final expression describing the evolution of V_z over time:

$$\begin{aligned} \underline{V_z(t)} &= \frac{A\,R_z}{M_{\text{mid}}} + \left[V_{z,0} - \frac{A\,R_z}{M_{\text{mid}}}\right] e^{\frac{-t}{C_z\,R_z}} = \\ &= \underline{V_{z,0}\,e^{\frac{-t}{C_z\,R_z}} + \frac{A\,R_z}{M_{\text{mid}}}\left[1 - e^{\frac{-t}{C_z\,R_z}}\right]} \end{aligned} \tag{B.8}$$

For the trivial case $V_{z,0} = 0$, this expression can be slightly simplified to

$$V_z(t) = \frac{A\,R_z}{M_{\text{mid}}}\left[1 - e^{\frac{-t}{C_z\,R_z}}\right] . \tag{B.9}$$

B.2 Smooth Transitions

For some of the derivations shown in this work, we quietly assumed the driving effort $V_z(t)$ to follow jumps in the input voltage $V(t)$ smoothly. The derivation of appropriate amplitude levels for the different pulse shapes presented in Section 4.3, for instance, was partly based on the assumption that V_z cannot exceed certain thresholds during a given time interval, if it is guaranteed to be below that threshold both at the beginning and at the end of the time interval. In this section we will take a closer look at why this assumption holds for the expression describing the time course of V_z shown in Equation B.8.

In Section 4.3 we stated that the input voltage resulting from a multi-amplitude pulse event can be interpreted as a quick succession of multiple inputs, each being applied for a limited amount of time. As we remember, the driving effort V_z resulting from this type of input can be determined recursively. More precisely, the amplitudes of the different sub-pulses are plugged into Equation B.8 as values for A along with the driving effort V_z present at the end of the previous sub-pulse as the new initial preload. Put differently, the driving effort V_z present at the end of each of the sub-pulses is used as the initial condition for the new sub-pulse. Consequently, at any instant of time, the current driving effort can be determined by considering the driving effort present at the nearest preceding sub-pulse boundary as well as the current input voltage. For a pair of two sub-pulses with amplitudes and durations of V_1 and V_2 and t_1 and t_2, respectively, at an arbitrary instant of time during the second sub-pulse this translates to

$$V_z(t_1+t) = V_z(t_1)\, e^{\frac{-t}{C_z R_z}} + \frac{V_2 R_z}{M_{\mathrm{mid}}} \left[1 - e^{\frac{-t}{C_z R_z}}\right], \tag{B.10}$$

where $0 \leq t \leq t_2$.

Assuming that the driving effort was zero before the first sub-pulse, this expression can be expanded according to Equation B.8 to

$$V_z(t_1+t) = \underbrace{\frac{V_1 R_z}{M_{\mathrm{mid}}} \left[1 - e^{\frac{-t_1}{C_z R_z}}\right]}_{V_z(t_1)} e^{\frac{-t}{C_z R_z}} + \frac{V_2 R_z}{M_{\mathrm{mid}}} \left[1 - e^{\frac{-t}{C_z R_z}}\right].$$

This expression can be rearranged as follows

$$\begin{aligned}
\underline{V_z(t_1+t)} &= \left[\frac{V_1 R_z}{M_{\mathrm{mid}}} - \frac{V_1 R_z}{M_{\mathrm{mid}}} e^{\frac{-t_1}{C_z R_z}}\right] e^{\frac{-t}{C_z R_z}} + \frac{V_2 R_z}{M_{\mathrm{mid}}} - \frac{V_2 R_z}{M_{\mathrm{mid}}} e^{\frac{-t}{C_z R_z}} = \\
&= \underbrace{\left[\frac{V_1 R_z}{M_{\mathrm{mid}}} - \frac{V_1 R_z}{M_{\mathrm{mid}}} e^{\frac{-t_1}{C_z R_z}} - \frac{V_2 R_z}{M_{\mathrm{mid}}}\right]}_{\Gamma = \mathrm{const}} e^{\frac{-t}{C_z R_z}} + \frac{V_2 R_z}{M_{\mathrm{mid}}} = \\
&= \underline{\frac{V_2 R_z}{M_{\mathrm{mid}}} + \Gamma \cdot e^{\frac{-t}{C_z R_z}}}
\end{aligned} \tag{B.11}$$

As we can see, the expression in Equation B.11 consists of the sum of a fraction which is constant and an exponential decay of a certain constant Γ. Obviously, since the exponential decay by itself is smooth, also the sum of a constant and an exponential decay as shown in Equation B.11 will be smooth. Consequently, depending on the values of V_1 and V_2, the above expression describing $V_z(t_1+t)$ will have its (local) maxima and minima for $t = 0$ and $t = t_2$, respectively, rather than anywhere in between.

C
Derivation of Initializers

In this chapter we will take a closer look at the derivation of the different initializers which were used heavily during the course of this work. As mentioned in Section 3.1, these initializers were required in order to prepare the memristors used in the various experiments with respect to their initial states.

As we saw in Section 2.2.2, in the memristor model this work was finally built upon, the memristor's instantaneous memristance $M(t)$ is split up into a volatile and a non-volatile portion. The volatile part modeled by V_x represents transient processes which finally decay towards the non-volatile portion of the memristance governed by V_y. Obviously, in order to ensure proper and reproducible results, one wants all transients to be over before the simulation starts. Nevertheless, since according to Equation 2.10, the instantaneous memristance $M(t)$ is primarily defined through V_x, this translates to finding an appropriate value for V_x and then setting $V_y = V_x$. This is the case, because setting $V_y = V_x$ translates to V_x having fully decayed to V_y and thus all transients being over.

Given this brief introduction, what we are concerned with in the following sections is finding different values for $V_x = V_y$ for each of the respective initialization strategies in order to ensure a certain initial memristor state. This memristor state is defined in terms of either a certain memristance or a specific conductance.

C.1 Maximum Memristance/Minimum Conductance

One of the simplest initialization strategies is choosing the initial memristor state such that the resulting instantaneous memristance $M(t)$ is equal to maximum achievable memristance. In this context it is irrelevant whether an additional pre-resistor M_0 as suggested in Section 4.2 is used for the actual memristor model or not. This is the case because the pure-ohmic pre-resistor M_0 goes into the equation as an additive constant. Consequently, if the memristance is maximal without M_0, it will also be maximal in combination with the pre-resistor M_0. Hence, in order to keep the derivations simple, we will neglect M_0 for the moment.

As we remember from Equation 2.10, the instantaneous memristance can be determined from the relation

$$M(t) = M_{\text{max}} - V_x \left(M_{\text{max}} - M_{\text{min}}\right) . \tag{C.1}$$

Since M_{max} is greater than M_{min}, the expression in brackets on the right is positive. Hence, choosing greater values for V_x will make $M(t)$ smaller. Consequently, the maximum instantaneous memristance $M(t)$ is achieved for the smallest possible value of V_x. As we remember from Section 2.2.2, the value of V_x is confined to the interval $(0, 1)$, making $V_x = 0 + \varepsilon$ the smallest possible value, where $\varepsilon > 0$. Hence, the maximum possible memristance is approximately M_{max} which is achieved for

$$\boxed{V_x = V_y \approx 0} . \tag{C.2}$$

Obviously, since the instantaneous conductance $G(t)$ is the inverse of the memristance $M(t)$, choosing V_x and V_y as shown in Equation C.2 will likewise result in the minimum possible conductance.

C.2 Minimum Memristance/Maximum Conductance

A similarly simple initialization strategy as the one described in the previous section is initializing the memristor to minimum memristance. Again considering that the conductance is the inverse of the memristance, this initialization corresponds to the initialization to maximum conductance.

Similar as in the previous section, also for this initialization the additional pre-resistor M_0 as introduced in Section 4.2 for linearization reasons can be neglected. This holds again, because M_0 is merely an additive term. Hence, if the memristance is minimal for a given V_x value in the model without M_0, it will also be minimal for the case with the additional pre-resistor. Consequently, we can concentrate on the original memristor model without the additional memristor M_0 again.

In order to find appropriate values for V_x and V_y, we start with Equation C.1 describing the instantaneous memristance $M(t)$ again. By analogy with what we stated in the previous section, the instantaneous memristance $M(t)$ will become smaller the greater V_x is. This holds, again, because the term $(M_{\text{max}} - M_{\text{min}})$ is positive. Given that V_x is confined to values between 0 and 1 $V_x = 1 - \varepsilon$ is the greatest possible value, where $\varepsilon > 0$. Hence, the smallest possible memristance will be achieved for

$$\boxed{V_x = V_y \approx 1}\,. \tag{C.3}$$

For this value, the corresponding memristance is given as

$$M(t) \approx M_{\text{max}} - (M_{\text{max}} - M_{\text{min}}) = \cancel{M_{\text{max}}} - \cancel{M_{\text{max}}} + M_{\text{min}} = M_{\text{min}}\,.$$

As indicated at the beginning of this section, a memristance of $M = M_{\text{min}}$ corresponds to a conductance of $G = G_{\text{max}}$.

C.3 Midrange Memristance

For the previous two initializers, the initial values for V_x and V_y were determined more or less directly from the equation describing the memristor's instantaneous memristance $M(t)$. In contrast to this, the initialization to midrange memristance requires some more derivations. To do so, first of all the term *midrange memristance* has to be defined. For the purpose of this work, midrange memristance shall refer to the arithmetic mean of the memristor's minimum and maximum memristances M_{min} and M_{max}, that is

$$M_{\text{mid}} = \frac{M_{\text{min}} + M_{\text{max}}}{2}\,. \tag{C.4}$$

Given this definition, we can start our derivation of V_x and V_y by equating the above expression with Equation C.1 defining the instantaneous memristance $M(t)$:

$$\underbrace{M_{\text{max}} - V_x\ (M_{\text{max}} - M_{\text{min}})}_{M(t)} \overset{!}{=} \underbrace{\frac{M_{\text{min}} + M_{\text{max}}}{2}}_{M_{\text{mid}}}$$

This equation can now be solved with respect to V_x:

$$M_{\text{max}} - \frac{M_{\text{max}} + M_{\text{min}}}{2} \overset{!}{=} V_x\ (M_{\text{max}} - M_{\text{min}})$$

$$\frac{\cancel{2}\,M_{\text{max}} - \cancel{M_{\text{max}}} - M_{\text{min}}}{2} \overset{!}{=} V_x\ (M_{\text{max}} - M_{\text{min}})$$

$$\frac{M_{\text{max}} - M_{\text{min}}}{2} \overset{!}{=} V_x\ (M_{\text{max}} - M_{\text{min}})$$

$$V_x \overset{!}{=} \frac{\cancel{M_{\text{max}} - M_{\text{min}}}}{2\,\cancel{(M_{\text{max}} - M_{\text{min}})}} = \frac{1}{2}$$

Hence, initialization of the system to midrange memristance M_{mid} can be achieved by setting

$$\boxed{V_x = V_y = 0.5}\,. \tag{C.5}$$

In order to keep the derivation simple again, the above reasoning was based on the original memristor model. Obviously, however, the resulting values for V_x and V_y shown in Equation C.5 are also valid for the extended memristor model featuring the additional pre-resistor M_0.

C.4 Fraction ρ of Maximum Conductance

This initializer sets the internal state variables V_x and V_y to values such that the memristor's initial conductance is a fraction ρ of the maximum achievable conductance G_{max}. In contrast to the previous sections, we concentrate on the memristor model featuring the additional pre-resistor M_0 as described in Section 4.2 here. In contrast to the Section 4.2, however, for the derivation presented here, a more general expression $M_0 = \frac{M_{\text{max}}}{\beta}$ was chosen for this pre-resistor.

Before we can start with the actual derivation of the relations defining the values for V_x and V_y required to achieve our desired initial conductance, we have to determine an expression for the maximum conductance G_{max}. Obviously, the conductance has its maximum when the memristance is at its minimum. As we remember from Section 2.2.2, in the memristor model by Serb et al. [Serb, 2014], the instantaneous memristance can be determined through the relation

$$M(t) = M_{\text{max}} - V_x \left(M_{\text{max}} - M_{\text{min}}\right) .$$

In the memristor model employing the additional pure-ohmic pre-resistor M_0, this translates to

$$\begin{aligned} M(t) &= M_0 + M_{\text{max}} - V_x \left(M_{\text{max}} - M_{\text{min}}\right) = \\ &= \frac{M_{\text{max}}}{\beta} + M_{\text{max}} - V_x \left(M_{\text{max}} - M_{\text{min}}\right) . \end{aligned} \tag{C.6}$$

Since V_x is confined to the interval $(0, 1)$, clearly the minimum overall memristance is achieved for $V_x \approx 1$. In order to keep the following derivations simple, however, we will assume $V_x \in [0, 1]$ from now on. Consequently, the minimum overall memristance is achieved for $V_x = 1$.

Plugging this value for V_x into Equation C.6, the minimum memristance achievable in our setup translates to the sum of the pre-resistor $M_0 = \frac{M_{\text{max}}}{\beta}$ and the memristor's lower memristance bound M_{min}. Hence, the corresponding conductance for this case, that is the maximum achievable conductance G_{max}, is given as

$$\begin{aligned} G_{\text{max}} &= \frac{1}{\frac{M_{\text{max}}}{\beta} + M_{\text{min}}} = \frac{1}{\frac{\beta\, M_{\text{min}} + M_{\text{max}}}{\beta}} = \\ &= \frac{\beta}{\beta\, M_{\text{min}} + M_{\text{max}}} \,. \end{aligned} \tag{C.7}$$

As mentioned earlier, what we are interested in now is a value for V_x which makes the instantaneous conductance $G(t)$ equal to a certain fraction ρ of G_{max}. While we have just determined an expression for G_{max}, a relation for $G(t)$ is still to be found. This can be achieved by taking the reciprocal of the expression given in Equation C.6:

$$\begin{aligned} G(t) &= \frac{1}{\frac{M_{\text{max}}}{\beta} + M_{\text{max}} - V_x\,(M_{\text{max}} - M_{\text{min}})} = \\ &= \frac{1}{\frac{(\beta+1)\, M_{\text{max}}}{\beta} - V_x\,(M_{\text{max}} - M_{\text{min}})} \end{aligned} \tag{C.8}$$

Given this expression, we can now begin with the derivation of V_x by equating the above expression with the one given in Equation C.7 multiplied with ρ:

$$\underbrace{\frac{1}{\frac{(\beta+1)\, M_{\text{max}}}{\beta} - V_x\,(M_{\text{max}} - M_{\text{min}})}}_{G(t)} \overset{!}{=} \rho\, \underbrace{\frac{\beta}{\beta\, M_{\text{min}} + M_{\text{max}}}}_{G_{\text{max}}} \tag{C.9}$$

Taking the inverse of this equation yields

$$\frac{(\beta+1)\, M_{\text{max}}}{\beta} - V_x\,(M_{\text{max}} - M_{\text{min}}) \overset{!}{=} \frac{\beta\, M_{\text{min}} + M_{\text{max}}}{\beta\, \rho} \,. \tag{C.10}$$

From this expression, V_x and V_y can now be determined in two steps:

$$-V_x\ (M_{\text{max}} - M_{\text{min}}) \overset{!}{=} \frac{\beta\, M_{\text{min}} + M_{\text{max}} - \rho\ (\beta + 1)\ M_{\text{max}}}{\beta\, \rho}$$

$$\boxed{V_y = V_x \overset{!}{=} \frac{\beta\, M_{\text{min}} - M_{\text{max}}\ [\rho\, (\beta + 1) - 1]}{\beta\, \rho\ (M_{\text{min}} - M_{\text{max}})}} \tag{C.11}$$

As we know from Section 2.2.2, the value of V_x is bound to the interval $(0, 1)$. In addition to this, the derivation given above is based on the extended memristor model featuring the pre-resistor $M_0 = \frac{M_{\text{max}}}{\beta}$. Consequently, obviously not any arbitrary value can be achieved for ρ. Based on this observation, in the following we will derive a certain range of achievable values for ρ. Especially for the reasoning the simulations presented in Section 5.1 are based on, this interval is of great interest.

Put differently, we want to determine a range the expression $\rho = \frac{G(t)}{G_{\text{max}}}$ can attain in our given setup. In order to do so, we take a closer look at Equation C.9 again. Since this expression essentially reads $G(t) \overset{!}{=} \rho\, G_{\text{max}}$, ρ can be determined by rearranging this expression:

$$\rho = \frac{\beta\, M_{\text{min}} + M_{\text{max}}}{(\beta + 1)\ M_{\text{max}} - \beta\, V_x\ (M_{\text{max}} - M_{\text{min}})} \tag{C.12}$$

In the above expression, V_x is the only non-constant quantity. Hence, the corresponding minimum and maximum values for ρ can be determined by plugging extremes $V_x = 0$ and $V_x = 1$ into Equation C.12. This yields

$$\rho_{\text{min}} = \frac{\beta\, M_{\text{min}} + M_{\text{max}}}{(\beta + 1)\ M_{\text{max}}} \tag{C.13}$$

and

$$\begin{aligned}\rho_{\text{max}} &= \frac{\beta\, M_{\text{min}} + M_{\text{max}}}{(\beta + 1)\ M_{\text{max}} - \beta\ (M_{\text{max}} - M_{\text{min}})} = \\ &= \frac{\beta\, M_{\text{min}} + M_{\text{max}}}{\cancel{\beta\, M_{\text{max}}} + M_{\text{max}} - \cancel{\beta\, M_{\text{max}}} + \beta\, M_{\text{min}}} = 1\end{aligned} \tag{C.14}$$

This makes sense, of course, since $G(t)$ cannot become greater than the maximum conductance $G_{\max}$.

Finally, plugging the value $\beta = 9$ chosen in Section 4.2 along with the parameters $M_{\min}$ and $M_{\max}$ according to Table 2.2 into the above Equation C.13 and taking into account that V_x is actually bound to the open interval $(0,1)$, yields

$$\boxed{0.100009 < \rho < 1} \tag{C.15}$$

Recalling Section 4.2, this range is perfectly aligned with the statement we made about memristive synapses in the chosen setup being able to adjust their conductance by a factor of approximately 10.

C.5 Midrange Conductance

In the previous section we derived expressions for V_x and V_y which made it possible to set the initial conductance to a certain fraction ρ of its maximum value $G_{\max}$, assuming the extended and linearized memristor model with the additional pre-resistor M_0. A prominent example for this type of initialization is the initialization to midrange conductance G_{mid}. In order to determine corresponding values for V_x and V_y according to Equation C.11, however, we need to derive an appropriate value for ρ first, because, against intuition, picking $\rho = 0.5$ is not the correct choice. This is due to the lowest possible value for ρ being different from zero, as we can see from the range defined in Equation C.15.

The correct value for the target fraction ρ leading to midrange conductance can be determined from Equations C.14 and C.13. More precisely, ρ_{mid} can be determined as the arithmetic mean of $\rho_{\min}$ and $\rho_{\max}$:

$$\begin{aligned} \underline{\rho_{\text{mid}}} &= \frac{\rho_{\min} + \rho_{\max}}{2} = \frac{\frac{\beta\, M_{\min} + M_{\max}}{(\beta+1)\, M_{\max}} + 1}{2} \\ &= \underline{\frac{\beta\, M_{\min} + (\beta+2)\; M_{\max}}{2\; (\beta+1)\; M_{\max}}} \end{aligned} \tag{C.16}$$

Plugging this expression into Equation C.11 we obtain:

$$V_x \overset{!}{=} \frac{\beta\, M_{\text{min}} - M_{\text{max}} \left[(\beta+1) \, \frac{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}{2\,(\beta+1)\, M_{\text{max}}} - 1 \right]}{\beta\, \frac{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}{2\,(\beta+1)\, M_{\text{max}}} \, (M_{\text{min}} - M_{\text{max}})} \tag{C.17}$$

This equation can now be simplified in order to obtain an easy to handle expression describing the target values for V_x and V_y leading to midrange conductance for the model featuring an additional pre-resistor $M_0 = \frac{M_{\text{max}}}{\beta}$.

$$V_x \overset{!}{=} \frac{\beta\, M_{\text{min}} - M_{\text{max}} \left[\cancel{(\beta+1)} \, \frac{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}{2\,\cancel{(\beta+1)}\, M_{\text{max}}} - 1 \right]}{\beta\, \frac{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}{2\,(\beta+1)\, M_{\text{max}}} \, (M_{\text{min}} - M_{\text{max}})}$$

$$V_x \overset{!}{=} \frac{\beta\, M_{\text{min}} - \cancel{M_{\text{max}}} \left[\frac{\beta\, M_{\text{min}} + \left(\beta+\cancel{2}\right) M_{\text{max}} \cancel{-2\, M_{\text{max}}}}{2\,\cancel{M_{\text{max}}}} \right]}{\beta\, \frac{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}{2\,(\beta+1)\, M_{\text{max}}} \, (M_{\text{min}} - M_{\text{max}})}$$

$$V_x \overset{!}{=} \frac{\beta\, M_{\text{min}} - \frac{\beta\, M_{\text{min}} + \beta\, M_{\text{max}}}{2}}{\beta\, \frac{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}{2\,(\beta+1)\, M_{\text{max}}} \, (M_{\text{min}} - M_{\text{max}})}$$

$$V_x \overset{!}{=} \frac{\cancel{2}\,\beta\, M_{\text{min}} - \cancel{\beta\, M_{\text{min}}} - \beta\, M_{\text{max}}}{2\,\beta\, \frac{\beta\, M_{\text{min}}\, (\beta+2)\, M_{\text{max}}}{2\,(\beta+1)\, M_{\text{max}}} \, (M_{\text{min}} - M_{\text{max}})}$$

$$V_x \overset{!}{=} \frac{\cancel{\beta}\, [M_{\text{min}} - M_{\text{max}}]}{\cancel{2}\,\cancel{\beta}\, \frac{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}{\cancel{2}\,(\beta+1)\, M_{\text{max}}} \, (M_{\text{min}} - M_{\text{max}})}$$

$$V_x \overset{!}{=} \frac{\cancel{[M_{\text{min}} - M_{\text{max}}]}\, (\beta+1)\, M_{\text{max}}}{[\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}]\, \cancel{(M_{\text{min}} - M_{\text{max}})}}$$

Hence, in order to achieve midrange conductance G_{mid} according to the extended memristor model, the initial values for V_x and V_y have to be chosen as

$$\boxed{V_y = V_x \overset{!}{=} \frac{(\beta+1)\, M_{\text{max}}}{\beta\, M_{\text{min}} + (\beta+2)\, M_{\text{max}}}}. \tag{C.18}$$

Choosing $\beta = 9$ as suggested in Section 4.2, the expression in Equation C.18 can be simplified to

$$V_y = V_x \overset{!}{=} \frac{10\,M_{\text{max}}}{9\,M_{\text{min}} + 11\,M_{\text{max}}}\,. \tag{C.19}$$

Given Equation C.19, the exact values for V_x and V_y can be determined by plugging in the values for M_{min} and M_{max} according to Table 2.2:

$$\boxed{V_x = V_y = 0.909} \tag{C.20}$$

C.6 Midrange Conductance without M_0

Finally, midrange conductance can also be defined for the original memristor model without the additional pre-resistor M_0. Similar as in the previous section, $\widetilde{G}_{\text{mid}}$ is defined as the arithmetic mean of conductance values in the least and the most conductive states. For the original memristor model and by analogy with Equation C.4 this, however, translates to the following expression:

$$\widetilde{G}_{\text{mid}} = \frac{G_{\text{min}} + G_{\text{max}}}{2} \tag{C.21}$$

As we remember from Sections C.1 and C.2, for the original memristor model the states of minimum and maximum conductance correspond to maximum and minimum memristance, respectively. Plugging this knowledge into Equation C.21, we can derive a more elaborate expression for $\widetilde{G}_{\text{mid}}$:

$$\begin{aligned} \underline{\widetilde{G}_{\text{mid}}} &= \frac{G_{\text{min}} + G_{\text{max}}}{2} = \frac{\frac{1}{M_{\text{max}}} + \frac{1}{M_{\text{min}}}}{2} = \frac{\frac{M_{\text{min}}+M_{\text{max}}}{M_{\text{max}}\,M_{\text{min}}}}{2} = \\ &= \underline{\frac{M_{\text{max}} + M_{\text{min}}}{2\,M_{\text{max}}\,M_{\text{min}}}} \end{aligned} \tag{C.22}$$

Similar as in the previous sections, we are now interested in values for V_x and V_y which lead to the conductance shown in Equation C.22. This can again be achieved by equating the above expression with another one describing the instantaneous conductance as a function of V_x. As indicated in the introduction, in this section we assume the original memristor model without any pre-resistor. Consequently, for this case the instantaneous conductance $G(t)$ can be determined by taking the inverse of Equation C.1. Likewise, in order to determine V_x one can also determine $\widetilde{M}_{\text{mid}}$ by taking the inverse of Equation C.22 and equate the resulting expression with Equation C.1:

$$\underbrace{M_{\text{max}} - V_x\,\left(M_{\text{max}} - M_{\text{min}}\right)}_{M(t)} \stackrel{!}{=} \underbrace{\frac{2\,M_{\text{max}}\,M_{\text{min}}}{M_{\text{max}} + M_{\text{min}}}}_{\widetilde{M}_{\text{mid}}}$$

This equation can now be solved with respect to V_x:

$$V_x\,\left(M_{\text{max}} - M_{\text{min}}\right) \stackrel{!}{=} M_{\text{max}} - \frac{2\,M_{\text{max}}\,M_{\text{min}}}{M_{\text{max}} + M_{\text{min}}}$$

$$V_x\,\left(M_{\text{max}} - M_{\text{min}}\right) \stackrel{!}{=} \frac{M_{\text{max}}\,\left(M_{\text{max}} + M_{\text{min}}\right) - 2\,M_{\text{max}}\,M_{\text{min}}}{M_{\text{max}} + M_{\text{min}}}$$

$$V_x \stackrel{!}{=} \frac{M_{\text{max}}^2 + \cancel{M_{\text{max}}\,M_{\text{min}}} - \cancel{2}\,M_{\text{max}}\,M_{\text{min}}}{\left(M_{\text{max}} + M_{\text{min}}\right)\,\left(M_{\text{max}} - M_{\text{min}}\right)}$$

$$V_x \stackrel{!}{=} \frac{M_{\text{max}}\,\cancel{\left(M_{\text{max}} - M_{\text{min}}\right)}}{\left(M_{\text{max}} + M_{\text{min}}\right)\,\cancel{\left(M_{\text{max}} - M_{\text{min}}\right)}}$$

$$V_x \stackrel{!}{=} \frac{M_{\text{max}}}{M_{\text{max}} + M_{\text{min}}} \tag{C.23}$$

Finally, plugging the values for M_{max} and M_{min} defined in Table 2.2 into Equation C.23, yields the values for V_x and V_y required in order to achieve midrange conductance $\widetilde{G}_{\text{mid}}$:

$$\boxed{V_x = V_y = 0.99999} \tag{C.24}$$

As an aside, the target value shown in Equation C.24 emphasizes one of the problems described in Chapter 4. As we remember from Section 4.2, the original memristor model suffers from an extremely non-linear mapping between V_x and the instantaneous conductance $G(t)$. This non-linearity becomes also clearly evident when looking at the above equation. More precisely, the value presented in Equation C.24 translates to $99.999\,\%$ of the V_x values controlling the first half of the conductance values. Consequently, only a vanishing $0.001\,\%$ of the dynamic range of V_x remain to adjust the second half of the conductance's dynamic range.

Bibliography

[Berdan, 2014]	Radu Berdan et al. “Qualitative SPICE modeling accounting for volatile dynamics of TiO2 memristors.” In: *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. June 2014, pp. 2033–2036 (cit. on pp. 8–11, 16).
[Bill, 2014a]	Johannes Bill. E-Mail communication with Alexander Serb. Oct. 10, 2014 (cit. on p. 47).
[Bill, 2014b]	Johannes Bill. Personal communication. Sept. 17, 2014 (cit. on p. 79).
[Bill, 2014c]	Johannes Bill. *TiO_2 memristors as plastic synapses for statistical model optimization*. Tech. rep. Institute for Theoretical Computer Science, TU Graz, Feb. 2014 (cit. on p. 56).
[BillLegenstein, 2014]	Johannes Bill and Robert Legenstein. “A compound memristive synapse model for statistical learning through STDP in spiking neural networks.” In: *Frontiers in Neuroscience* 8.412 (2014). ISSN: 1662-453X. DOI: 10.3389/fnins.2014.00412 (cit. on pp. 1, 3, 20, 40, 44, 51, 98, 100–110, 114).
[Chua, 1971]	L.O. Chua. “Memristor-The missing circuit element.” In: *Circuit Theory, IEEE Transactions on* 18.5 (Sept. 1971), pp. 507–519. ISSN: 0018-9324. DOI: 10.1109/TCT.1971.1083337 (cit. on p. 6).

[DanPoo, 2004] Yang Dan and Mu-ming Poo. “Spike timing-dependent plasticity of neural circuits.” In: *Neuron* 44.1 (2004), pp. 23–30 (cit. on pp. 7, 18).

[Fieres, 2008] Johannes Fieres, Johannes Schemmel, and Karlheinz Meier. “Realizing biological spiking network models in a configurable wafer-scale hardware system.” In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE. 2008, pp. 969–976 (cit. on p. 2).

[Fiser, 2010] József Fiser et al. “Statistically optimal perception and learning: from behavior to neural representations.” In: *Trends in cognitive sciences* 14.3 (2010), pp. 119–130 (cit. on p. 19).

[Habenschuss, 2012] Stefan Habenschuss, Johannes Bill, and Bernhard Nessler. “Homeostatic plasticity in Bayesian spiking networks as Expectation Maximization with posterior constraints.” In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 773–781 (cit. on p. 100).

[Hebb, 1949] Donald O. Hebb. *The Organization of Behavior*. New York: Wiley, 1949 (cit. on p. 7).

[Johnsen, 2012] Gorm Krogh Johnsen. “An introduction to the memristor-a valuable circuit element in bioelectricity and bioimpedance.” In: *Journal of Electrical Bioimpedance* 3.1 (2012), pp. 20–28 (cit. on pp. 5, 6).

[Lecun, 1998] Y. Lecun et al. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791 (cit. on pp. 110, 111).

[Li, 2015] Qingjiang Li et al. “A Memristor SPICE Model Accounting for Synaptic Activity Dependence.” In: *PLoS ONE* 10.3 (Mar. 2015), e0120506. DOI: 10.1371/journal.pone.0120506. URL: http://dx.doi.org/10.1371/journal.pone.0120506 (cit. on p. 11).

[Linares, 2009] Bernabé Linares-Barranco and Teresa Serrano-Gotarredona. “Memristance can explain spike-time-dependent-plasticity in neural synapses.” In: *Nature precedings* 1 (2009) (cit. on pp. 2, 6, 53).

[Nessler, 2013] Bernhard Nessler et al. “Bayesian Computation Emerges in Generic Cortical Microcircuits through Spike-Timing-Dependent Plasticity.” In: *PLoS Computational Biology* 9 (Apr. 2013) (cit. on pp. 2, 19, 102, 103, 112).

[Querlioz, 2011] Damien Querlioz, Olivier Bichler, and Christian Gamrat. “Simulation of a memristor-based spiking neural network immune to device variations.” In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE. 2011, pp. 1775–1781 (cit. on pp. 54, 114).

[Serb, 2014] A. Serb et al. “Memristors as synapse emulators in the context of event-based computation.” June 2014. URL: http://eprints.soton.ac.uk/362468/ (cit. on pp. 11, 12, 14–17, 24, 25, 27, 31, 34–37, 43, 51, 55, 71, 72, 74, 82, 113, 114, 119, 121, 155).

[Serrano, 2013] Teresa Serrano-Gotarredona et al. “STDP and STDP variations with memristors for spiking neuromorphic learning systems.” In: *Frontiers in neuroscience* 7 (2013) (cit. on p. 53).

[Zamarreño, 2011] Carlos Zamarreño-Ramos et al. “On Spike-Timing-Dependent-Plasticity, Memristive Devices, and building a Self-Learning Visual Cortex.” In: *Frontiers in Neuroscience* 5.26 (2011). DOI: 10.3389/fnins.2011.00026. URL: http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2011.00026/abstract (cit. on pp. 54, 114).