Ceca Kraišniković, Dipl.-Ing. Dipl. inž. elektr. i računar.

Spike-based models for cognitive computations and robust training of memristive neural networks

Doctoral Thesis

to achieve the university degree of

Doctor Technicae

submitted to

Graz University of Technology

Advisor:

Univ.-Prof. Dipl.-Ing. Dr. techn. Robert Legenstein

Institute of Theoretical Computer Science

Graz, May 2023

Affidavit

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 indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date

Signature

Abstract

Models of neural networks used in current artificial intelligence research can solve demanding cognitive tasks and reach excellent performances. However, the abstraction level of these models limits our ability to use them to deepen our understanding of computation in biological neuronal systems. Spiking Neural Networks (SNNs) provide a neural network model that is closer related to the biological counterpart. We enriched the standard SNN model with a Spike Frequency Adaptation (SFA) mechanism, and trained them on complex cognitive tasks. We found that the inclusion of SFA enabled SNNs to learn tasks involving the manipulation of sequences of symbols according to rules and even dynamically changing rules, arithmetic computations, and comparisons. More importantly, we studied neural codes of a trained network solving abstract tasks, and found that they resemble those from biological experiments, e.g., how neurons specialize for encoding abstract variables representing the task. Our model also offers insights into the organization of computation during arithmetic tasks and principles of working memory in the brain.

Inspired by the energy efficiency of the brain, the emerging field of neuromorphic computing attempts to build electronic devices resembling the neuron-synapse structure of the brain. In the second part of this thesis, we focused on nano-scale hardware devices — memristors — that mimic biological synapses. The promise of these devices to enable energy-efficient co-located computing and storage similarly as in the brain, is yet challenged by their non-ideal behavior. When used as synaptic elements in neural networks, faulty memristors cause significant performance drops. To overcome this problem, we developed a robust training scheme for the detection of faulty behaviors of memristors and their pruning on the fly. This training scheme showed success in mitigating the detrimental effects of faulty memristors on network training.

Zusammenfassung

Modelle neuronaler Netze, die in der aktuellen Forschung zur künstlichen Intelligenz verwendet werden, können anspruchsvolle kognitive Aufgaben lösen und hervorragende Leistungen erzielen. Die Abstraktionsebene dieser Modelle schränkt jedoch unsere Fähigkeit ein, sie zur Vertiefung unseres Verständnisses von Berechnungen in biologischen neuronalen Systemen zu nutzen. Spikende neuronale Netze (SNNs) stellen ein neuronales Netzwerkmodell dar, das dem biologischen Gegenstück näher steht. Wir haben das Standard-SNN-Modell um einen Mechanismus der Spikefrequenzanpassung (Spike Frequency Adaptation, SFA) erweitert und sie auf komplexe kognitive Aufgaben trainiert. Wir fanden heraus, dass SNNs durch die Einbeziehung von SFA in der Lage sind, Aufgaben zu lernen, die die Manipulation von Symbolfolgen nach Regeln und sogar sich dynamisch ändernden Regeln, arithmetische Berechnungen und Vergleiche beinhalten. Was noch wichtiger ist: Wir haben die neuronalen Codes eines trainierten Netzwerks untersucht, das abstrakte Aufgaben löst, und festgestellt, dass sie denen aus biologischen Experimenten ähneln, z. B. wie sich Neuronen auf die Codierung abstrakter Variablen, die die Aufgabe repräsentieren, spezialisieren. Unser Modell bietet auch Einblicke in die Organisation des Rechnens bei arithmetischen Aufgaben und in die Prinzipien des Arbeitsgedächtnisses im Gehirn.

Inspiriert von der Energieeffizienz des Gehirns versucht das aufstrebende Gebiet des neuromorphen Rechnens, elektronische Geräte zu bauen, die der Neuronen-Synapsen-Struktur des Gehirns ähneln. Im zweiten Teil dieser Arbeit konzentrierten wir uns auf Hardwaregeräte im Nanomaßstab — Memristoren — die biologische Synapsen nachahmen. Das Versprechen dieser Bauelemente, energieeffiziente, integrierte Rechen- und Speicherkapazitäten ähnlich wie im Gehirn zu ermöglichen, wird jedoch durch ihr nicht ideales Verhalten in Frage gestellt. Wenn sie als synaptische Elemente in

neuronalen Netzen eingesetzt werden, verursachen fehlerhafte Memristoren erhebliche Leistungseinbußen. Um dieses Problem zu überwinden, haben wir ein robustes Trainingsverfahren entwickelt, um fehlerhaftes Verhalten von Memristoren zu erkennen und sie im laufenden Betrieb auszusortieren. Dieses Trainingsverfahren konnte die nachteiligen Auswirkungen fehlerhafter Memristoren auf das Netztraining erfolgreich abmildern.

Acknowledgments

First, I would like to express my deep gratitude to my supervisor, Robert Legenstein, for his support, guidance, and patience through this journey of obtaining a doctoral degree. I am very thankful for having the opportunity to work on exciting research topics, the knowledge that I gained, and the encouragement to attend any educational training or event that I expressed my wish for. Second, I would like to thank Wolfgang Maass, for the collaboration with him and his team from whom I learned a lot. In addition, thanks to Stefano Vassanelli, for agreeing to be the second referee of this thesis.

I extend my thanks to all my colleagues that took part in this "PhD marathon run", for their cheerfulness and positivity even during difficult times. Here I would like to acknowledge: Anand Subramoney and Darjan Salaj, for their advice on how to approach and solve problems; Michael Müller, for being a great Journal Club (co-)organizer, and a very kind office neighbor, offering an active listener-ear and help whenever challenges were ahead of me; Thomas Limbacher, for running (many real, sometimes quite challenging runs) and going for wonderful hikes; Ozan Özdenizci, for discussions about teaching materials, and proofreading my thesis; Florian Unger and Arjun Rao, for numerous chat sessions; Daniela Windisch-Scharler and Oliver Friedl, for providing answers to any of my questions related to administration and life in Austria.

I am also grateful for the encouragement of my friends, particularly, Tanja Linkermann Vasilev, Emina Hadrović, and Elias Hajek. Finally, many thanks to my family without whom I would not be who I am today, for believing in me and for their endless support to fulfill my dreams.

I dedicate this thesis to my late grandfather Živan, known and remembered as a hard-working and patient person.

Contents

Abstract v

1. Introduction 1
- 1.1. Computational models for cognitive tasks . . . 3
- 1.2. Working memory . . . 6
- 1.3. Neural codes and Spike Frequency Adaptation . . . 7
- 1.4. Training algorithms for neural networks . . . 8
- 1.5. Neuromorphic computing systems and memristive technologies 9
- 1.6. Research questions studied and organization of the thesis . . 12

2. Spike frequency adaptation supports network computations on temporally dispersed information 15
- 2.1. Introduction . . . 16
- 2.2. Results . . . 19
  - 2.2.1. Network model . . . 19
  - 2.2.2. SFA provides working memory simultaneously for many pieces of information, and yields powerful generalization capability . . . 22
  - 2.2.3. SFA improves the performance of SNNs for common benchmark tasks that require nonlinear computational operations on temporally dispersed information . . . 28
  - 2.2.4. Comparing the contribution of SFA to temporal computing with that of other slow processes in neurons and synapses . . . 29
  - 2.2.5. SFA supports demanding cognitive computations on sequences with dynamic rules . . . 31
  - 2.2.6. SFA enables SNNs to carry out complex operations on sequences of symbols . . . 34
- 2.3. Discussion . . . 39

2.4. Materials and Methods 44
2.4.1. Network models 44
2.4.2. Training methods 48
2.4.3. Tasks 49

3. Spike-based symbolic computation on bit strings and numbers 59
3.1. Introduction 60
3.2. Neural networks of the brain and spiking neural networks . . 62
3.2.1. The leaky integrate and fire (LIF) neuron model 63
3.2.2. Enhancing temporal computing capabilities of RSNNs with Spike Frequency Adaptation (SFA) 66
3.2.3. Training RSNNs with e-prop 67
3.3. Symbolic computation and arithmetics in the brain 68
3.3.1. Representation of numbers in the brain 68
3.3.2. Circuits for arithmetic in the brain 69
3.3.3. Symbolic computation 71
3.4. Prior work on symbolic computations on bit strings and numbers with SNNs 73
3.5. New results on spike-based symbolic computation on bit sequences and numbers 74
3.5.1. Spike-based symbolic computation on bit strings . . . 77
3.5.2. Evaluation of nested arithmetic expressions 78
3.6. Conclusions 84
3.7. Technical details to new results 85
3.7.1. Network architecture and neuron model 85
3.7.2. Training method e-prop 86
3.7.3. Tasks 86

4. Fault Pruning: Robust Training of Neural Networks with Memristive Weights 91
4.1. Introduction 92
4.2. Results 94
4.2.1. Training of Memristive Neural Networks with Faulty Memristors 96
4.2.2. Fault Pruning for Memristive Neural Networks 99
4.3. Methods 105
4.3.1. Memristor Model 105

4.3.2. Training Schedule 105
4.3.3. Estimation of the Fault Factors $\hat{f}_i$ 107
4.3.4. Details to Computer Simulations 108
4.4. Conclusions 109

A. Appendix to Chapter 2: Spike frequency adaptation supports network computations on temporally dispersed information 113
A.1. Autocorrelation-based intrinsic time scale of neurons trained on STORE-RECALL task. 113
A.2. sMNIST task with sparsely connected SNN obeying Dale's law 114
A.3. Google Speech Commands 115
A.4. Delayed-memory XOR 118
A.5. The 12AX task in a noisy network 120
A.6. Duplication/Reversal task 122

Bibliography 127

List of Figures

2.1. Experimental data on neurons with SFA, and a simple model for SFA. . . . 22
2.2. 20-dimensional STORE-RECALL and sMNIST task. . . . 28
2.3. Solving the 12AX task by a network of spiking neurons with SFA. . . . 33
2.4. Spiking activity of an SNN with SFA trained to carry out operations on sequences. . . . 37
2.5. Analysis of an SNN with SFA trained to carry out operations on sequences. . . . 38

3.1. Neuron dynamics for LIF neuron without (left) and with SFA mechanism (right). . . . 64
3.2. Population coding of numbers. . . . 70
3.3. Comparison of two bit strings. . . . 80
3.4. Spike-based arithmetic on numbers and variables. . . . 83

4.1. In-the-loop training setup. . . . 95
4.2. Feed-forward neural network trained on the MNIST data set. 97
4.3. Performances of feed-forward memristive neural networks trained on the MNIST data set. . . . 100
4.4. Performances of memristive CNNs on CIFAR-10. . . . 103
4.5. Connectivity in the network after pruning (in %). . . . 104

A.1. Histogram of the intrinsic time scale of neurons trained on STORE-RECALL task. . . . 114
A.2. sMNIST time series classification benchmark task. . . . 116
A.3. Delayed-memory XOR task. . . . 119
A.4. Effect of a noise current with zero mean and standard deviation 0.05 added to a single neuron in the network for the 12AX task. . . . 121

A.5. Effect of a noise current with zero mean and standard deviation 0.075 added to a single neuron in the network for the 12AX task. 122
A.6. A zoom-in of the spike raster for a trial solving Duplication task (left) and Reversal task (right). 123
A.7. Illustration of models for an inversely adapting ELIF neuron, and for short-term synaptic plasticity. 124
A.8. Distribution of adaptation index from Allen Institute cell measurements. 125

List of Tables

2.1. Recall accuracy (in %) of SNN models with different time constants of SFA (rows) for variants of the STORE-RECALL task with different required memory time spans (columns). . 26

3.1. Evaluation of nested arithmetic expressions with an SNN. . . 81

A.1. Google Speech Commands. 121

1. Introduction

Intelligence has many aspects, and accordingly, many definitions – the ability to learn, acquire and use knowledge, deal with novel situations, plan and reason, solve problems, and think abstractly, to name a few. Artificial Intelligence (AI), the intelligence of machines, is a research field that originally started as a subfield of computer science with the goal of writing intelligent programs (Turing, 1950). Nowadays many different scientific and engineering areas use AI and their progress is intertwined.

Here, in this thesis, we studied the questions of what we can learn from the neurobiology of the brain to improve models of neural networks performing cognitive computations, and what these models can tell us about the human brain.

The brain provides a resourceful pool of ideas with the potential to advance current AI models. For example, it informs us about the information coding principles and representations it utilizes, signal transmission, learning rules in the nervous systems, structures, functions, and cognitive processes of the brain (Hassabis et al., 2017; G. Li et al., 2023; Storrs & Kriegeskorte, 2020). Since the brain is very energy-efficient, it is also an inspiration for reducing the resources and energy consumption of AI models and implementation in unconventional hardware (Davies et al., 2018; Furber et al., 2014; Indiveri et al., 2013; Roy et al., 2019). AI contributes to the studies of the brain by explaining the phenomena of the brain and in attempts to understand the mechanisms and computations performed by the brain. In addition, using methods from AI — more specifically, machine learning — analysis of neuroimaging data and the analysis of the brain's connectome can give valuable insights into the nature of neural representations and connectivity patterns (Hassabis et al., 2017; G. Li et al., 2023; Markram et al., 2015; Storrs & Kriegeskorte, 2020).

Biological neural networks (BNNs) constitute living brains and inspire modeling efforts to simulate learning and computation in Artificial Neural Networks (ANNs). The communication in the brain is mediated by chemical signals known as neurotransmitters, and by electrical signals, known as action potentials transmitted between neurons. Neurons, the basic processing units of the nervous system, receive pieces of information, coming through dendrites, at locations called synapses. They process information according to "rules" in their cell body, i.e., soma, where also the cell's metabolic processes that maintain the neuron occur. Eventually, neurons pass information to other neurons or muscles through axons. Each stage of processing involves complex chemical processes, however, in modeling, often reductionist approaches are adopted that take into account different aspects of BNNs. Modeling typically considers very simplified neuron models, even though neurons differ with respect to the morphology, structure, and specialization within the nervous system (Gazzaniga et al., 2009; Gerstner & Kistler, 2002).

Commonly used ANNs in AI research are based on a rate-based neuron model organized in layers (LeCun et al., 2015; McClelland et al., 1987; Rosenblatt, 1958). Convolutional Neural Networks (CNNs) are a specialized model that employs the mathematical operation of convolution. CNNs are suitable for processing data in the form of multiple arrays, such as images, but are also applicable to time-series data. Recurrent Neural Networks (RNNs) are a model specialized in processing sequential data. RNNs process one element at a time while maintaining information about previous time steps, hence suitable for speech and language processing (Goodfellow et al., 2016; LeCun et al., 2015). Being a simplified abstraction of BNNs, these models can include different levels of detail that sometimes focus on achieving human-like performance on tasks these models were trained on (Krizhevsky et al., 2012; Sutskever et al., 2014), or, in other cases, to provide insights into mechanisms of the brain and give rise to new, testable predictions (Kriegeskorte, 2015; Mastrogiuseppe & Ostojic, 2018; Yamins & DiCarlo, 2016). A neuron model that can be better linked to neuroscience is provided by Spiking Neural Networks (SNNs) (Maass, 1997) consisting of Generalized Integrate-and-Fire (GIF) neurons (Allen Institute, 2017), or a specialization of it, Leaky Integrate-and-Fire (LIF) neurons (Gerstner & Kistler, 2002). In the first part of this thesis, we focus on recurrent SNNs and strive to link

them to neuroscientific findings. In the second part, we use feed-forward ANNs and CNNs to develop a training scheme suitable for implementations in hardware.

The organization of this chapter is as follows: In Section 1.1 we will discuss cognitive control and processing in the brain. This section further introduces computational models and cognitive tasks used to study cognition. For solving cognitive tasks, working memory plays an important role. Its functions and neural coding principles are described in Section 1.2. To delve into the main functions of working memory, neural codes representing stimuli are commonly investigated. Neural adaptation, an observed phenomenon in neural codes, and a manifestation of it, Spike Frequency Adaptation (SFA), are introduced in Section 1.3. Learning is an important aspect of intelligent human behavior. We simulate learning in all our models, that is, we train models to solve cognitive tasks. An overview of training algorithms for models of neural networks, and versions of them that are more biologically plausible are briefly covered in Section 1.4. Another emphasis of this thesis concerns the implementation of learning in hardware. The research field that investigates neuro-inspired hardware, the field of neuromorphic engineering, and more specifically, devices that can be used to support learning *in silico* such as memristors, are discussed in Section 1.5. Finally, Section 1.6 gives an overview of the chapters of this thesis by stating the motivation and findings of the research questions we explored.

1.1. Computational models for cognitive tasks

A long-standing scientific question studied from many different points of view — from antique, through philosophy to cognitive science and neuroscience — is how the brain consisting of neurons gives rise to cognition. The emergence of cognitive capabilities in humans is related to the significant expansion of the frontal cortex throughout human evolution. In general, when comparing mammalian species, the larger the brain, the more flexible the behavior. In humans, half of the frontal lobe includes the prefrontal cortex – a brain region that is highly interconnected with other cortical and subcortical areas, and presumably plays a very important role in cognitive

control and coordination of cognitive processing (Gazzaniga et al., 2009; Miller & Cohen, 2001). The prefrontal cortex is involved when top-down processing is required, that is, for behaviors that are guided by internal states or intentions, and goal-directed (Botvinick & Cohen, 2014; Haque et al., 2021; Miller & Cohen, 2001; Rougier et al., 2005). Assisted by the prefrontal cortex, working memory is believed to be a mechanism where transient and task-relevant information is integrated with goals, perceptual information, and retrieved memories from past experiences (Gazzaniga et al., 2009; Haque et al., 2021). To serve such a purpose, working memory has to be dynamic and flexible: dynamic, to support the maintenance of task-relevant information, integrate information across different time scales, and ignore distractions, and flexible, to support switching between different options and contexts. Mechanisms of cognitive control that are involved in these functions include dynamic filtering, task switching, and the inhibition of response (Botvinick & Cohen, 2014; Gazzaniga et al., 2009). They become particularly relevant for context-dependent processing in which, for the same stimuli, the response might differ depending on the current goals, past experiences, environment, internal states, etc. The context acts as a currently active rule that determines how to process the stimuli (Rougier et al., 2005). Another definition of human cognition (i.e., at the psychological level) emphasizes symbols and symbol manipulation – the ability to represent, extract and generalize patterns, ultimately leading to abstract representations and rules for manipulating them (Marcus, 2003).

Since the inception of computer science and the seminal work of Alan Turing, researchers have tried to make computer programs learn and mimic the cognitive abilities of humans using models that manipulated symbols (Newell, Simon, et al., 1972; Shanker, 1995; Turing, 1950). A class of models known as production systems operate on symbols and uses rules and logic for symbol manipulation. However, the representations these models and other cognitive models use are abstract, implemented using *if-then* rules, and do not consider the neurobiology of the brain. On the contrary, models used in computational neuroscience focus rather on biologically plausible mechanisms to understand the emergent dynamics and phenomena of smaller circuits but neglect cognitive functions. The challenge of bridging both approaches can be tackled by employing biologically plausible artificial intelligence models that perform complex cognitive tasks (Krakauer et al.,

2017; Kriegeskorte & Douglas, 2018). Consequently, studying the content and format of brain representations, large-scale dynamics of brain activation and the computational mechanisms involved would become possible. Furthermore, the aforementioned mechanisms of cognitive control could be studied through computational models that solve cognitive tasks.

Apart from biologically plausible models at an appropriate level of detail, it is important to consider tasks that require cognitive abilities as humans have. As tasks that test the cognitive abilities of a model to flexibly manipulate symbols (possibly according to dynamically changing rules), tasks with sequences of symbols can be used (Lashley, 1951; Marcus, 2003). The simplest one is learning the identity function (e.g., repeat the sequence), in which humans excel even after being exposed to only a few examples, but computational models such as artificial neural networks struggle (Marcus, 2003). Another example task studied in humans that tests context processing and goal maintenance is the Continuous Performance Task (CPT-AX) (Barch et al., 2009). In this task, among sequentially presented letter stimuli, the subject has to detect occurrences of the target sequence A followed immediately by X. In a more complex version, the 12AX task, digits 1 and 2 determine two contexts for which the target sequence differs (Frank et al., 2001). Tasks that test the prefrontal cortex functions include the Stroop task and Wisconsin Card Sorting Task (Miller & Cohen, 2001). Both of them require the subject to selectively attend to one among more attributes of a stimulus before providing a response. Abstract reasoning, another hallmark of human intelligence, is effectively tested in humans using Raven's Progressive Matrices (Raven & Court, 1938), and Raven-style Progressive Matrices can be used to test neural networks in their ability to perform relational and analogical visual reasoning (Barrett et al., 2018; C. Zhang et al., 2019).

Employing biologically plausible models would make it possible to relate the insights they offer to neuroscientific findings, and to understand human cognition and its implementation in the brain.

1.2. Working memory

We experience the world around us through our sensory systems. Sensory information is rather short-lived and this form of memory takes place on timescales of milliseconds to seconds. Only attended information gets moved into short-term storage, and, if rehearsed, it can be further moved into long-term storage. Another term often used interchangeably to denote short-term memory is working memory, however, some authors distinguish between these two terms, defining short-term memory as a place where information is stored for periods up to a few seconds, and working memory as a place where information is also maintained and processed, or mentally manipulated (Gazzaniga et al., 2009; Goldman-Rakic, 1995). In this thesis, the term working memory is used as defined in the latter case.

It was long believed that the maintenance of information in working memory is achieved through sustained neuronal activity, that is, the activity of neurons that persistently fire throughout the stimulation and delay periods as long as the item is held in memory – a pattern of activity known as persistent activity. This view, however, is an incomplete picture of neuronal mechanisms that support information maintenance in working memory. Recent studies suggest that memory maintenance involves also activity-silent and dynamic coding principles. Activity-silent principle predicts that single neurons do not carry stimulus-specific information, but rather, the memories could be maintained in the short-term synaptic plasticity (Kamiński & Rutishauser, 2019; Masse et al., 2019; Mongillo et al., 2008; Stokes et al., 2013) and that hidden neural states (e.g., synaptic efficacy, membrane potentials, and extracellular transmitter concentrations) may play an important role (Wolff et al., 2017). The maintained item is possible to retrieve, however not from the same neurons that were persistently active, but through the activity of different neurons. This activity is rather dynamic across neurons and time (Kamiński & Rutishauser, 2019; Wolff et al., 2017).

Studies of underlying mechanisms and implementation of working memory are particularly relevant as they might provide insights into how the brain circuitry executes complex cognitive computations. For example, modeling work can help test whether the persistent activity is due to cell mechanisms and/or local recurrent connections, how it varies with cognitive demands

(Masse et al., 2019), what mechanisms achieve activity-silent and dynamic representations (Buschman & Miller, 2022; Lundqvist et al., 2023), or whether high-level and low-level representations in context-dependent processing have to be strictly separate representations (Rougier et al., 2005; Salaj et al., 2021).

For the AI community, the implementation of working memory has also been of great interest. The introduction of recurrent neural networks was the first step toward building architectures that explicitly maintain information over time (Elman, 1990; Hassabis et al., 2017). These architectures, displaying attractor dynamics and capable of processing sequential data, were then enriched by gating and control mechanisms that enabled a form of long short-term memory (Hochreiter & Schmidhuber, 1997), and became a basis for many successful applications in AI research (Bahdanau et al., 2015; Sutskever et al., 2014).

1.3. Neural codes and Spike Frequency Adaptation

Information encoding, maintenance, and manipulation are often studied through neural codes in which neural firing patterns represent a correlate of interest such as an external stimulus or a motor output. The neural codes are often not a one-to-one mapping from stimulus to the neural output but are history- and context-dependent, and are subject to processes involving neuronal, synaptic and network dynamics (Weber & Fairhall, 2019; Weber et al., 2019). Neural adaptation is a phenomenon in which a neuron's or population's response decays upon repeated or prolonged stimulation and thereby reflects history and context dependence (Benda, 2021; Weber & Fairhall, 2019). It is observed at all stages of processing and throughout the nervous system — from the sensory periphery, through the central nervous system, to the motor output. Spike frequency adaptation, a manifestation of this phenomenon, on a single neuron level, can be a result of various voltage- or calcium-gated ionic currents activated during action potentials. This can be reflected in an increase of the spiking threshold, but also in the short-term depression of the input synapses (Benda, 2021; Benda & Herz,

2003; Gutkin & Zeldenrust, 2014). On the network level, neural adaptation can be achieved as an interplay of excitatory and inhibitory connections with appropriately tuned timescales. Modeling work suggests that including synaptic- and membrane-related variables enables recurrent spiking neural networks to solve cognitive tasks (Y. Li et al., 2021; Salaj et al., 2021).

1.4. Training algorithms for neural networks

For humans, learning is defined as a process of acquiring new information (Gazzaniga et al., 2009), for computer programs, as acquiring experience in a task while improving a performance measure (Mitchell, 1997). The latter focuses on function optimization, and a learning algorithm commonly used for training ANNs and CNNs is the Backpropagation algorithm (Rumelhart et al., 1986). The Backpropagation algorithm calculates gradients of a cost function with respect to each parameter, i.e., errors, and uses them to iteratively update the parameters, in that way optimizing the cost function. Backpropagation Through Time (BPTT) is a version of this algorithm suitable for training Recurrent Neural Networks (RNNs) on time series and temporal computing tasks (Goodfellow et al., 2016; LeCun et al., 2015; Robinson & Fallside, 1987; Werbos, 1988).

There is little evidence that the brain performs backpropagation since it would require neurons to back-propagate precise information about computation errors of a long chain of neurons at contributing synapses. Nonetheless, the ongoing research aims to connect the fields of deep learning and neuroscience, and the search for biologically plausible learning rules and approximations of the backpropagation results in alternative online learning algorithms (Marblestone et al., 2016; Marschall et al., 2020; Richards et al., 2019). In online algorithms, the network parameters are updated as the input data arrive, in real-time.

An approximation of backpropagation for training feedforward networks, the Feedback Alignment algorithm (Lillicrap et al., 2016), for the update of weights uses a set of random feedback weights instead of the synaptic weights that contributed to the errors. This algorithm does not require knowledge of precise weights used in the forward pass, however, the errors

are still back-propagated through long computational chains. In online learning algorithms that approximate BPTT, unlike the transmission of errors that includes spatial and temporal gradient components (as in BPTT), the updates are ideally local in space and time, i.e., without a requirement for knowing the current values at all other synapses and in all processing steps, respectively. Example algorithms include Real-Time Recurrent Learning (RTRL) (Williams & Zipser, 1989), Unbiased Online Recurrent Optimization (UORO) (Tallec & Ollivier, 2018), Online Spatio-Temporal learning (Bohnstingl et al., 2022), to name a few. In spiking neural networks, due to the non-differentiability of the neuron's spiking output, BPTT cannot be readily applied for the training, but only after introducing a pseudo-derivative, as done in e.g., (Bellec, Salaj, et al., 2018; Salaj et al., 2021). Biologically plausible algorithms for training spiking networks include, for example, e-prop (Bellec et al., 2020), SuperSpike (Zenke & Ganguli, 2018), Random Feedback Local Online (RFLO) (J. M. Murray, 2019), SpikeProp (Bohte et al., 2000).

1.5. Neuromorphic computing systems and memristive technologies

Successful AI models are often trained on large data sets and require significant processing resources. Moreover, they are commonly trained on conventional computing hardware based on von Neumann architectures, which are known to suffer from the memory bottleneck (Davies et al., 2021; Legenstein, 2015; W. Zhang et al., 2020). Therefore, AI computing demands computing and power efficiency. High energy costs and dissipation arise from the fact that the memory unit is physically separated from the processing unit, and that data has to be moved from and to the memory unit, before and after being processed in the processing unit, respectively. Inspired by the principles of intelligent and energy-efficient information processing in the brain, significant research efforts aim at developing new computing paradigms such as neuromorphic systems (Christensen et al., 2022; Davies et al., 2018; Furber, 2016; Furber et al., 2014; W. Zhang et al., 2020). The research field of neuromorphic computing, also known as

neuromorphic engineering, started in the 1980s with the pioneering work of Carver Mead to create computing hardware that mimics information processing in biological nervous systems. Mead's early work was inspired by auditory and visual sensory systems, resulting in retina and cochlea silicon chips for vision and hearing, respectively. This work served not only as an illustration of how certain aspects of neural mechanisms can be successfully realized in analog electronic integrated-circuit technology, but also to link general principles of neural computation to electronics (Mead, 1990, 2020).

Important aspects of neuromorphic systems, similar to features of the brain, include a highly parallel architecture, i.e., neuron-synapse structure, in-memory computation, signal information encoding, and learning capabilities (Camuñas-Mesa et al., 2019; W. Zhang et al., 2020). In the brain, computing and memory are co-located and intermingled, with neurons representing the processing units, and synapses the storage elements underlying memory and learning. An implementational design that enables the transformation of inputs into outputs (analogous to matrix-vector multiplications between two layers of neurons in a neural network) can be achieved through memristive crossbar arrays consisting of a grid of horizontal and vertical metal wires, where each intersection exhibits memristive behavior, representing a synapse. When input voltages are applied to the horizontal wires, the driven currents flow through the memristive junction. Using Ohm's and Kirchhoff's laws, these currents can be summed across vertical wires and interpreted as the result of the computation. To enable learning, that is, synaptic weight changes, the crossbar intersections have to support the programming operation, i.e., that their resistance (conductance) is adjustable. Since memristors retain memories in their current states, they are also known as memory resistors, or shortly, memristors (Chua, 1971; Indiveri et al., 2013; Legenstein, 2015; Prezioso et al., 2015; Strukov et al., 2008). Memristors can be built using volatile or non-volatile technologies. Apart from their switching mechanisms and electrical properties, the main distinction between volatile and non-volatile memristive devices is the memory retention duration, with the former memristor technology being suitable for emulating certain synaptic and neural dynamics, and the latter technology e.g., for the implementation of matrix-vector multiplications (Zhou et al., 2022). Here, we will focus on non-volatile memory devices.

Non-volatile memristive devices can be realized using different technologies, such as Resistive Random-Access Memory (ReRAM), Phase-Change Memory (PCM), Magnetic Random-Access Memory (MRAM), and ferroelectric Random-Access Memory (FeRAM). Typically implemented as metal-insulator-metal nano cells, they differ with respect to the structure and material composition they are made of, and switching mechanisms between conductance states. To change resistance (conductance) states in ReRAM devices, set and reset voltage pulses are applied on top and bottom electrodes, which causes field-induced migration and diffusion of oxygen vacancies in metal oxides, or metallic ions from the electrodes. In PCM devices, the change between crystalline (low resistance) and amorphous (high resistance) states is induced thermally. In MRAM, the two ferromagnetic polarizations — parallel and antiparallel — define low and high resistance states, and the spin transfer torque mechanism can be used to flip between the states. In FeRAM, the resistance change is achieved by introducing a ferroelectric field-effect transistor (FeFET) structure that helps enable polarization switching in a ferroelectric material (Ielmini & Wong, 2018; Lanza et al., 2022).

In this thesis, we concentrate on ReRAM devices. ReRAM devices are a promising technology that offers great conductance tuning performance, but is prone to yield and reliability issues, which is especially problematic for large-scale manufacturing and device downscaling.

Other aspects often taken into consideration include integration density, energy efficiency, and on-chip learning capabilities. Integration density is a metric that is concerned with the implementation of as many synapses/devices on the chip area as possible. Energy efficiency is measured in terms of time and energy consumption needed in the programming phase (more significant) and for read operations of synaptic memory. Learning capabilities, i.e., whether on-chip learning can be successfully supported by the chip, will become even more important in the future as a replacement for off-chip and hybrid approaches in which conventional computers are still used in the learning phase (W. Zhang et al., 2020).

1.6. Research questions studied and organization of the thesis

In this thesis, we used spike-based models and trained them to solve complex cognitive tasks. The complexity of these tasks was not only due to the requirement to maintain the memory content on a time scale of seconds, but also to manipulate the content according to dynamically changing or implicitly given rules, even for the content on different levels of abstraction. Maintenance of information on the time scale of seconds is already a very challenging task for SNNs, and we showed that it was possible to achieve even human-level performance on complex tasks by employing models of spiking neurons with spike frequency adaptation, implemented in our case as a transient increase of a neuron's firing threshold every time the neuron produced a spike. Moreover, SNNs were trained to solve these tasks, either using BPTT (with the pseudo-derivative of the spiking output function), or a biologically plausible training algorithm, e-prop. The resulting neural codes were in a biologically plausible regime, as suggested by the population's firing rates. This property is necessary when the aim is to ask questions related to the brain.

In Chapter 2, we addressed the questions of coding principles implemented by working memory and the computational benefits of the inclusion of neurons with SFA. Our results suggest a novel principle used by neurons with SFA, the negative imprinting principle, in which the neuron that was strongly active during the stimulus (e.g., STORE command) becomes less active when the response is needed (e.g., RECALL command). This negative imprinting is in accordance with the activity-silent coding principle of working memory. We also investigated the contribution of different mechanisms to the computing capabilities – of SFA, facilitating and depressing short-term plasticity, and enhanced excitability as opposed to reduced excitability (as in SFA) of a neuron. We found that depressing rather than facilitating mechanisms support better robust memory maintenance, and that models with SFA achieve higher performance. This suggests that the memory content is better protected from perturbations and noise when the neuron's firing is reduced. In the 12AX task, for which a hierarchical working memory is necessary, some studies suggested that separate neural representations for

encoding different levels of information are required, or special memory modules to solve the task successfully. We showed that it is not necessary to include modules resembling specific architecture in the brain, but a generic neural circuitry suffices to implement a form of hierarchical memory. In the tasks where strings of symbols were manipulated, we studied the neural codes representing variables of interest – serial position in the sequence, symbol and task identity. Similarly as in studies with primates, the majority of neurons showed mixed selectivity, but some neurons also specialized for encoding position in the sequence and symbols, and much less of them specialized for encoding the task identity. These codes could potentially be studied further, for example, as factorial codes for encoding abstract knowledge and their role in generalization capabilities.

In Chapter 3, we focused on SNNs with SFA trained by e-prop on tasks involving number comparison and simple arithmetics. These tasks also required temporal processing, as inputs were given to the networks in a sequential manner, hence SNNs with SFA were a suitable choice of a model. We considered SNNs not only because they are a biologically plausible model that can be related to neuroscience, but also as a suitable model for use in neuro-inspired hardware. Therefore, biologically plausible training algorithms that could be supported by hardware are also of interest to us, and here our goal was to test whether also complex tasks such as mathematical computations can be successfully learned by e-prop. Humans can perform computations in an approximate or exact manner. Performing computations in an exact manner requires the ability to abstract and use rules. Often, sophisticated sequence processing is required, for example, when comparing two numbers (first, comparing the length of sequences representing numbers, then comparing the most significant digits, and even the less significant ones until the answer becomes unambiguous). We trained SNNs to do similar comparisons with implicitly specified rules, and on tasks that required evaluation of nested arithmetic expressions. Arithmetic tasks that involve variables and concrete values for them require the network also to bind the lower-level and higher-level information in order to evaluate the arithmetic expression. The results showed that the trained spiking networks performed well even when novel examples were presented to the network, and there was no need to include special-purpose circuits for arithmetic.

In Chapter 4, we developed a robust training scheme for neural networks

useful when hardware is in the loop, and a digital computer is used for calculating the necessary updates. Since the focus here was on devices resembling synapses, i.e., memristors, and the training scheme that preserves performance when devices are susceptible to noise and nonidealities, we employed here standard artificial models (ANNs and CNNs) instead of SNNs, since they are easier to train. The approach is rather model-agnostic, and SNNs could be also easily considered. The use of memristors promises energy efficiency and a possible solution to overcome the memory bottleneck of von Neumann architectures. Also, as integration density increases, it is likely that robust training schemes will be crucial to overcoming harmful effects on computing accuracy.

Contributions of this thesis can be summarized as a work on models of cognitive computations and robust training. Once neuromorphic hardware is ready to support SNNs and training algorithms that drive synaptic weight changes directly on memristors, these results will provide valuable guidance in the implementation of (cognitive) brain-like neural networks *in silico*.

2. Spike frequency adaptation supports network computations on temporally dispersed information

Abstract For solving tasks such as recognizing a song, answering a question, or inverting a sequence of symbols, cortical microcircuits need to integrate and manipulate information that was dispersed over time during the preceding seconds. Creating biologically realistic models for the underlying computations, especially with spiking neurons and for behaviorally relevant integration time spans, is notoriously difficult. We examine the role of spike frequency adaptation in such computations and find that it has a surprisingly large impact. The inclusion of this well-known property of a substantial fraction of neurons in the neocortex — especially in higher areas of the human neocortex — moves the performance of spiking neural network models for computations on network inputs that are temporally dispersed from a fairly low level up to the performance level of the human brain.

This chapter was published as:

Salaj*, D., Subramoney*, A., Kraisnikovic*, C., Bellec, G., Legenstein, R., Maass, W. (2021). Spike frequency adaptation supports network computations on temporally dispersed information. Elife, 10, e65459.

* These authors contributed equally to this work.

Contributions My contributions in this publication include work on cognitive computations on sequences with dynamic rules, complex operations on sequences of symbols, analysis of emergent neural codes (Data curation, Software, Formal analysis, Validation, Investigation, Visualization), and writing (Writing - original draft, Writing - review and editing).

Acknowledgements This research was partially supported by the Human Brain Project (Grant Agreement number 785907 and 945539), the SYNCH project (Grant Agreement number 824162) of the European Union, and under partial support by the Austrian Science Fund (FWF) within the ERA-NET CHIST-ERA programme (project SMALL, project number I 4670-N). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Computations were carried out on the Human Brain Project PCP Pilot Systems at the Juelich Supercomputing Centre, which received co-funding from the European Union (Grant Agreement number 604102) and on the Vienna Scientific Cluster (VSC).

2.1. Introduction

Since brains have to operate in dynamic environments and during ego-motion, neural networks of the brain need to be able to solve "temporal computing tasks", i.e., tasks that require integration and manipulation of temporally dispersed information from continuous input streams on the behavioral time scale of seconds. Models for neural networks of the brain have inherent difficulties in carrying out such temporal computations on the time scale of seconds, since spikes and postsynaptic potentials take place on the much shorter time scales of milliseconds and tens of milliseconds. It is well known that biological neurons and synapses are also subject to a host of slower dynamic processes, but it has remained unclear whether any of these can be recruited for robust temporal computation on the time scale of seconds. We focus here on a particularly prominent one of these slower processes: spike frequency adaptation (SFA) of neurons. SFA denotes the effect that preceding firing activity of a neuron transiently increases

its firing threshold, see Figure 2.1A for an illustration. Experimental data from the Allen Institute (Allen Institute, 2018) show that a substantial fraction of excitatory neurons of the neocortex, ranging from 20% in mouse visual cortex to 40% in the human frontal lobe, exhibit SFA, see Appendix figure A.8. Although a rigorous survey of time constants of SFA is still missing, the available experimental data show that SFA does produce history dependence of neural firing on the time scale of seconds, in fact, up to 20 seconds according to (Pozzorini et al., 2015; Pozzorini et al., 2013). The biophysical mechanisms behind SFA include inactivation of depolarizing currents and the activity-dependent activation of slow hyperpolarizing or shunting currents (Benda & Herz, 2003; Gutkin & Zeldenrust, 2014).

SFA is an attractive feature from the perspective of the metabolic cost of neural coding and computation since it reduces firing activity (Gutierrez & Denève, 2019). But this increased metabolic efficiency comes at the cost of making spike codes for sensory stimuli history-dependent. Hence, it becomes harder to decode information from spikes if neurons with SFA are involved (Weber & Fairhall, 2019; Weber et al., 2019). This problem appears already for very simple input streams, where the same stimulus is presented repeatedly, since each presentation is likely to create a somewhat different neural code in the network. However, it has recently been shown that a careful network construction can ensure that stable neural codes emerge on the network level (Gutierrez & Denève, 2019).

A number of potential computational advantages of neurons with SFA have already been identified, such as cellular short-term memory (Marder et al., 1996; Turrigiano et al., 1996), enhancement of sensitivity to synchronous input and desirable modifications of the frequency response curve (Benda et al., 2010; Ermentrout, 1998; Gutkin & Zeldenrust, 2014; X.-J. Wang, 1998). On the network level, SFA may enhance rhythms and support Bayesian inference (Deneve, 2008; Kilpatrick & Ermentrout, 2011). The contribution of SFA to temporal computing capabilities of recurrent spiking neural networks (SNNs) had first been examined in (Bellec, Salaj, et al., 2018) and its role for language processing in feedforward networks was subsequently examined in (Fitz et al., 2020).

Here we are taking a closer look at enhanced temporal computing capabilities of SNNs that are enabled through SFA, and also compare the

computational benefit of SFA with that of previously considered slower dynamic processes in SNNs: short-term synaptic plasticity. Most experimental analyses of temporal computing capabilities of biological neural networks have focused on arguably the simplest type of temporal computing, where the response to a stimulus has to be given after a delay. In other words, information about the preceding stimulus has to be kept in a working memory. We start with a simple task (the STORE-RECALL task), since this task makes the analysis of neural coding especially transparent. We use it here to demonstrate a novel principle that is used by neurons with SFA to implement working memory – "the negative imprinting principle". That is, firing of a neuron leaves a negative imprint of its activity, because its excitability is reduced due to SFA. Such negative imprinting has previously been utilized in (Gutierrez & Denève, 2019). We then show that the working memory capability of SNNs with SFA scales up to much more demanding, and ecologically more realistic working memory tasks, where not just a single but numerous features, e.g., features that characterize a previously encountered image, movie scene, or sentence, have to be stored simultaneously.

However, these working memory tasks capture just a small fragment of temporal computing capabilities of brains. Substantially more common and more difficult are tasks where information is temporally dispersed over a continuous input stream, say in a sentence or a video clip, and has to be integrated over time in order to solve a task. This requires that the information that is stored in working memory has to be continuously updated. We tested temporal computing capabilities of SNNs with SFA for two types of such tasks. First we consider standard benchmark tasks for temporal computing capabilities: keyword spotting, time series classification (sequential MNIST), and delayed XOR task. Then we consider two tasks that are arguably at the heart of higher-level cognitive brain processing (Lashley, 1951): processing and manipulations of sequences of symbols according to dynamic rules. We also analyze the neural codes that emerge in SNNs with SFA for such tasks, and compare them with neural codes for corresponding tasks in the brain (Barone & Joseph, 1989; Carpenter et al., 2018; Y. Liu et al., 2019). Since our focus is on computing, rather than learning capabilities, we use a powerful tool for optimizing network parameters for task performance: Backpropagation Through Time (BPTT). While this method is not assumed

to be biologically realistic, it has recently been shown that almost the same task performance can, in general, be achieved for SNNs — with and without SFA — through training with a biologically more realistic learning method: *e-prop* (Bellec et al., 2020). We demonstrate this here for the case of the 12AX task.

2.2. Results

2.2.1. Network model

We employ a standard simple model for neurons with SFA, the generalized leaky integrate-and-fire model $GLIF_2$ from (Allen Institute, 2017; Teeter et al., 2018). A practical advantage of this simple model is that it can be efficiently simulated and that it is amenable to gradient descent training (Bellec, Salaj, et al., 2018). It is based on a standard leaky integrate-and-fire (LIF) neuron model. In a LIF neuron, inputs are temporally integrated, giving rise to its membrane potential. The neuron produces a spike when its membrane potential is above a threshold v_{th}. After the spike, the membrane potential is reset, and the neuron enters a refractory period during which it cannot spike again. The precise dynamics of the LIF model is given in equation (2.2) in Methods. The $GLIF_2$ model extends the LIF model by adding to the membrane voltage a second hidden variable, a variable component $a(t)$ of the firing threshold $A(t)$ that increases by a fixed amount after each spike $z(t)$ of the neuron, and then decays exponentially back to 0, see Figure 2.1B, E. This variable threshold models the inactivation of voltage-dependent sodium channels in a qualitative manner. We write $z_j(t)$ for the spike output of neuron j, which switches from 0 to 1 at time t when the neuron fires at time t, and otherwise has value 0. With this notation one can define the SFA model by the equations:

$$\begin{aligned} A_j(t) &= v_{\text{th}} + \beta a_j(t), \\ a_j(t+1) &= \rho_j a_j(t) + (1-\rho_j) z_j(t), \end{aligned} \tag{2.1}$$

where v_{th} is the constant baseline of the firing threshold $A_j(t)$, and $\beta > 0$ scales the amplitude of the activity-dependent component. The parameter

$\rho_j = \exp\left(\frac{-1}{\tau_{a,j}}\right)$ controls the speed by which $a_j(t)$ decays back to 0, where $\tau_{a,j}$ is the adaptation time constant of neuron j. For simplicity, we used a discrete-time model with a time step of $\delta t = 1\,\text{ms}$ (see Methods for further details). We will in the following also refer to this model as "LIF with SFA". Consistent with the experimental data (Allen Institute, 2017), we consider recurrent networks of LIF neurons, SNNs, of which some fraction is equipped with SFA. It turns out that the precise fraction of neurons with SFA does not matter for most tasks, especially if it stays somewhere in the biological range of 20% to 40%. We usually consider for simplicity fully connected recurrent networks, but most tasks can also be solved with sparser connectivity. Neurons in the recurrent network project to readout neurons which produce the output of the network, see Figure 2.1C. The final output was either the maximum value after applying the softmax, or thresholded values after applying the sigmoid on each readout neuron.

In order to analyze the potential contribution of SFA to temporal computing capabilities of SNNs, we optimized the weights of the SNN for each task. We used for this stochastic gradient descent in the form of Backpropagation Through Time (BPTT) (Mozer, 1989; Robinson & Fallside, 1987; Werbos, 1988), which is, to the best of our knowledge, the best performing optimization method. Although this method performs best for differentiable neural network models, it turns out that the non-differentiable output of a spiking neuron can be overcome quite well with the help of a suitably scaled pseudo-derivative (Bellec, Salaj, et al., 2018). In general, similar task performance can also be achieved with a biologically plausible learning method for SNNs *e-prop* (Bellec et al., 2020). Although computing rather than learning capabilities are in the focus of this paper, we demonstrate for one of the most demanding tasks that we consider, the 12AX task, that almost the same task performance as with BPTT can be achieved with *e-prop*.

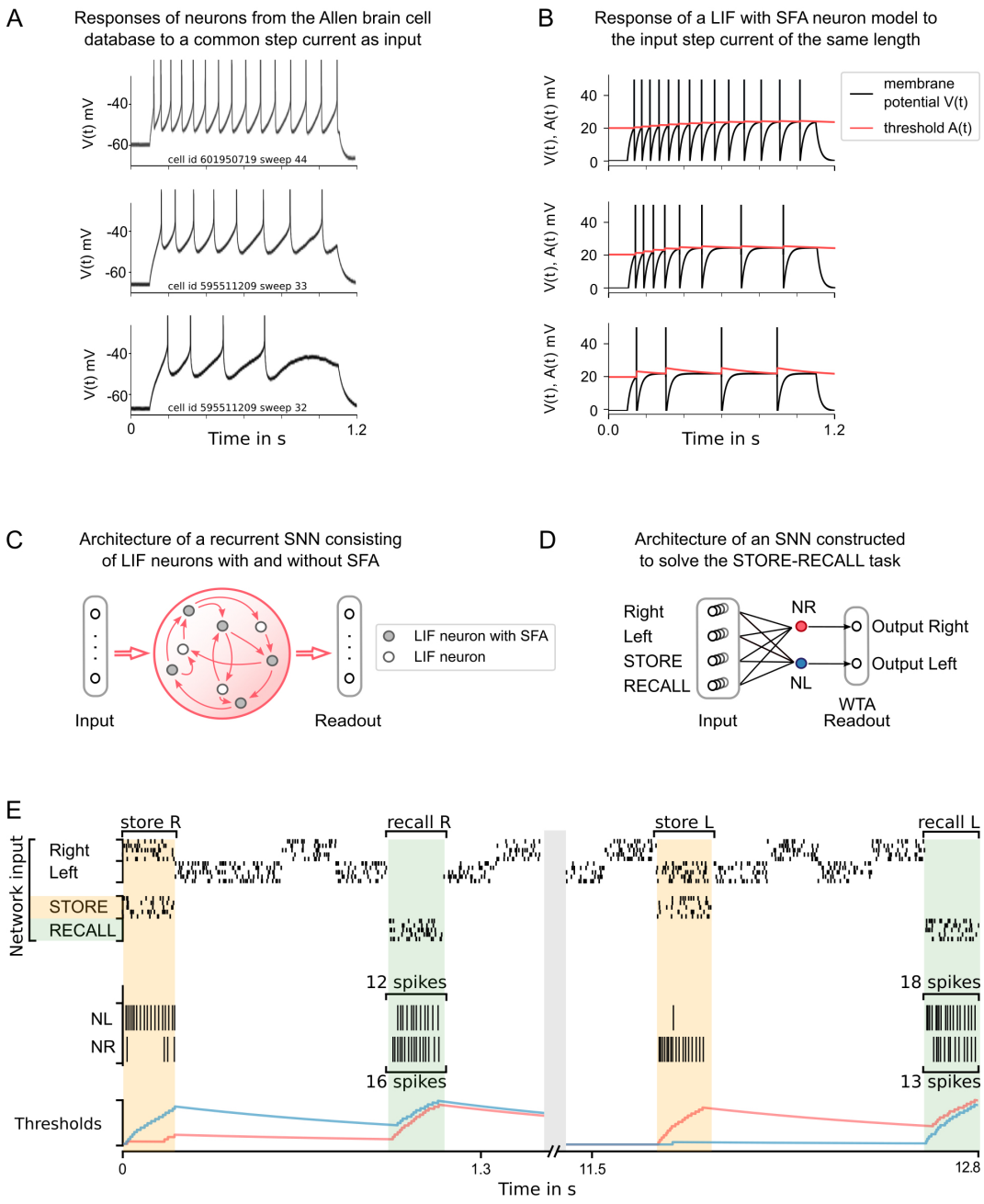


Figure 2.1.: (Caption on the next page.)

Figure 2.1.: **Experimental data on neurons with SFA, and a simple model for SFA.** (**A**) The response to a 1-second long step current is displayed for three sample neurons from the Allen brain cell database (Allen Institute, 2018). The cell id and sweep number identify the exact cell recording in the Allen brain cell database. (**B**) The response of a simple LIF neuron model with SFA to the 1-second long step current. Neuron parameters used: top row $\beta = 0.5\,\mathrm{mV}$, $\tau_a = 1\,\mathrm{s}$, $I_{\mathrm{input}} = 0.024\,\mathrm{A}$; middle row $\beta = 1\,\mathrm{mV}$, $\tau_a = 1\,\mathrm{s}$, $I_{\mathrm{input}} = 0.024\,\mathrm{A}$; bottom row $\beta = 1\,\mathrm{mV}$, $\tau_a = 300\,\mathrm{ms}$, $I_{\mathrm{input}} = 0.022\,\mathrm{A}$. (**C**) Symbolic architecture of recurrent SNN consisting of LIF neurons with and without SFA. (**D**) Minimal SNN architecture for solving simple instances of STORE-RECALL tasks that we used to illustrate the negative imprinting principle. It consists of 4 subpopulations of input neurons and 2 LIF neurons with SFA, labeled NR and NL, that project to two output neurons (of which the stronger firing one provides the answer). (**E**) Sample trial of the network from D for two instances of the STORE-RECALL task. The input "Right" is routed to the neuron NL, which fires strongly during the first STORE signal (indicated by a yellow shading of the time segment), which causes its firing threshold (shown at the bottom in blue) to strongly increase. The subsequent RECALL signal (green shading) excites both NL and NR, but NL fires less, i.e., the storing of the working memory content "Right" has left a "negative imprint" on its excitability. Hence, NR fires stronger during recall, thereby triggering the answer "Right" in the Readout. After a longer pause, which allows the firing thresholds of NR and NL to reset, a trial is shown where the value "Left" is stored and recalled.

2.2.2. SFA provides working memory simultaneously for many pieces of information, and yields powerful generalization capability

To elucidate the mechanism by which SFA supports temporal computing capabilities of SNNs we first consider classical working memory tasks, where information just has to be temporally stored by the neural network, without the need for frequent updates of this working memory during an instance of the task.

Negative imprinting principle. To demonstrate how neurons with SFA can contribute to solving working memory tasks, we first consider the standard case where just a single value, e.g., the position left or right of a prior stimulus, has to be stored during a delay. The simple network shown in

Figure 2.1D, consisting of two neurons with SFA (NL and NR), can already solve this task if there are long gaps between different instances of the task. We assume that these two neurons receive spike inputs from four populations of neurons. Two of them encode the value that is to be stored, and the other two convey the commands STORE and RECALL through high firing activity in these populations of input neurons, see Figure 2.1E for an illustration. The neuron NL (NR) fires when the population that encodes the STORE command fires (yellow shading in Figure 2.1D) and simultaneously the input population for value "Right" ("Left") is active. Furthermore, we assume that the input population that encodes the RECALL command (green shading in Figure 2.1E) causes both NL and NR to fire. However, the firing threshold of that one of them that had fired during the preceding STORE command is higher (see the blue threshold in the left half and the red threshold in the right half of Figure 2.1E), causing a weaker response to this RECALL command. Hence, if the spikes of NL and NR are each routed to one of the two neurons in a subsequent winner-take-all (WTA) circuit, the resulting winner encodes the value that has been stored during the preceding STORE command. The time courses of the firing thresholds of neurons NL and NR in Figure 2.1E clearly indicate the negative imprinting principle that underlies this working memory mechanism: The neuron that fires less during RECALL was the one that had responded the strongest during the preceding STORE phase, and this firing left a stronger "negative imprint" on this neuron. Note that this hand-constructed circuit does not work for large ranges of time differences between STORE and RECALL, and more neurons with SFA are needed to solve the subsequently discussed full versions of the task.

Scaling the negative imprinting principle up to more realistic working memory tasks

We wondered whether SNNs with SFA can also solve more realistic working memory tasks, where not just a single bit, but a higher level code of a preceding image, sentence, or movie clip needs to be stored. Obviously, brains are able to do that, but this has rarely been addressed in models. In addition, brains are generally exposed to ongoing input streams also during the delay between STORE and RECALL, but need to ignore these irrelevant input segments. Both of these more demanding aspects are present in the

more demanding version of the STORE-RECALL task that is considered in Figure 2.2A. Here the values of 20, instead of just 1, input bits need to be stored during a STORE command, and recalled during a RECALL command. More precisely, a 20-dimensional stream of input bits is given to the network, whose values during each 200 ms time segment are visualized as 4×5 image in the top row of Figure 2.2A. Occasionally, a pattern in the input stream is marked as being salient through simultaneous activation of a STORE command in a separate input channel, corresponding for example to an attentional signal from a higher brain area (see yellow shading in Figure 2.2A). The task is to reproduce during a RECALL command the pattern that had been presented during the most recent STORE command. Delays between STORE and RECALL ranged from 200 to 1600 ms. 20 binary values were simultaneously extracted as network outputs during RECALL by thresholding the output values of 20 linear readout neurons. We found that an SNN consisting of 500 neurons with SFA, whose adaptive firing threshold had a time constant of $\tau_a = 800$ ms, was able to solve this task with an accuracy above 99% and average firing activity of 13.90 ± 8.76 Hz (mean $\pm$ standard deviation). SFA was essential for this behavior, because the recall performance of a recurrent network of LIF neurons without SFA, trained in exactly the same way, stayed at chance level (see Methods). In Figure 2.2A, one sees that those neurons with SFA that fired stronger during STORE fire less during the subsequent RECALL, indicating a use of the negative imprinting principle also for this substantially more complex working memory task.

Interestingly, this type of working memory in an SNN with SFA shares an important feature with the activity-silent form of working memory in the human brain that had been examined in the experiments of (Wolff et al., 2017). It had been shown there that the representation of working memory content changes significantly between memory encoding and subsequent network reactivation during the delay by an "impulse stimulus": A classifier trained on the network activity during encoding was not able to classify the memory content during a network reactivation in the delay, and vice versa. Obviously, this experimental result from the human brain is consistent with the negative imprinting principle. We also tested directly whether the experimentally observed change of neural codes in the human brain also occurs in our model. We trained a classifier for decoding the content of

working memory during STORE, and found that this classifier was not able to decode this content during RECALL, and vice versa (see Methods). Hence, our model is in this regard consistent with the experimental data of (Wolff et al., 2017).

We also found that it was not possible to decode the stored information from the firing activity between STORE and RECALL, as one would expect if the network would store the information through persistent firing. Actually, the firing activity was quite low during this time period. Hence, this demo shows that SNNs with SFA have in addition to persistent firing, a quite different method for transient storage of information.

Generalization of SFA-enhanced temporal computations to unseen inputs. In contrast to the brain, many neural network models for working memory can store only information on which they have been trained. In fact, this tends to be unavoidable if a model can only store a single bit. In contrast, the human brain is also able to retain new information in its working memory. The SNN with SFA that we used for the 20-dimensional working memory task also had this capability. It achieved a performance of 99.09%, i.e., 99.09% of the stored 20-dimensional bit vectors were accurately reproduced during recall, on bit vectors that had never occurred during training. In fact, we made sure that all bit vectors that had to be stored during testing had a Hamming distance of at least 5 bits to all bit vectors used during training. A sample segment of a test trial is shown in Figure 2.2A, with the activity of input neurons at the top and the activation of readout neurons at the bottom.

No precise alignment between time constants of SFA and working memory duration is needed. Experimental data from the Allen Institute database suggest that different neurons exhibit a diversity of SFA properties. We show that correspondingly a diversity of time constants of SFA in different neurons provides high performance for temporal computing. We consider for simplicity the one-dimensional version of the task of Figure 2.2A, where just a single bit needs to be stored in working memory between STORE and RECALL commands. The expected delay between STORE and RECALL (see the header row of Table 2.1) scales the working memory time span that is

required to solve this task. Five fixed time constants were tested for SFA ($\tau_a = 200\,\text{ms}$, 2 s, 4 s, 8 s, see top 5 rows of Table 2.1). Also, a power-law distribution of these time constants, as well as a uniform distribution, were considered (see last two rows of Table 2.1). One sees that the resulting diversity of time constants for SFA yields about the same performance as a fixed choice of the time constant that is aligned with the required memory span of the task. However, a much larger time constant (see the row with $\tau_a = 8\,\text{s}$ in the column with an expected memory span of 200 ms or 2 s for the task) or a substantially smaller time constant (see the row with $\tau_a = 2\,\text{s}$ in the column with an expected memory span of 8 s) tends to work well.

Expected delay between STORE and RECALL	200 ms	2 s	4 s	8 s	16 s
without SFA ($\tau_a = 0\,\text{ms}$)	96.7	51	50	49	51
$\tau_a = 200\,\text{ms}$	99.92	73.6	58	51	51
$\tau_a = 2$ s	99.0	99.6	98.8	92.2	75.2
$\tau_a = 4$ s	99.1	99.7	99.7	97.8	90.5
$\tau_a = 8$ s	99.6	99.8	99.7	97.7	97.1
τ_a power-law dist. in $[0, 8]$ s	99.6	99.7	98.4	96.3	83.6
τ_a uniform dist. in $[0, 8]$ s	96.2	99.9	98.6	92.1	92.6

Table 2.1.: **Recall accuracy (in %) of SNN models with different time constants of SFA (rows) for variants of the STORE-RECALL task with different required memory time spans (columns).** Good task performance does not require good alignment of SFA time constants with the required time span for working memory. An SNN consisting of 60 LIF neurons with SFA was trained for many different choices of SFA time constants for variations of the one-dimensional STORE-RECALL task with different required time spans for working memory. A network of 60 LIF neurons without SFA trained under the same parameters did not improve beyond chance level ($\sim 50\%$ accuracy), except for the task instance with an expected delay of 200 ms where the LIF network reached 96.7% accuracy (see top row).

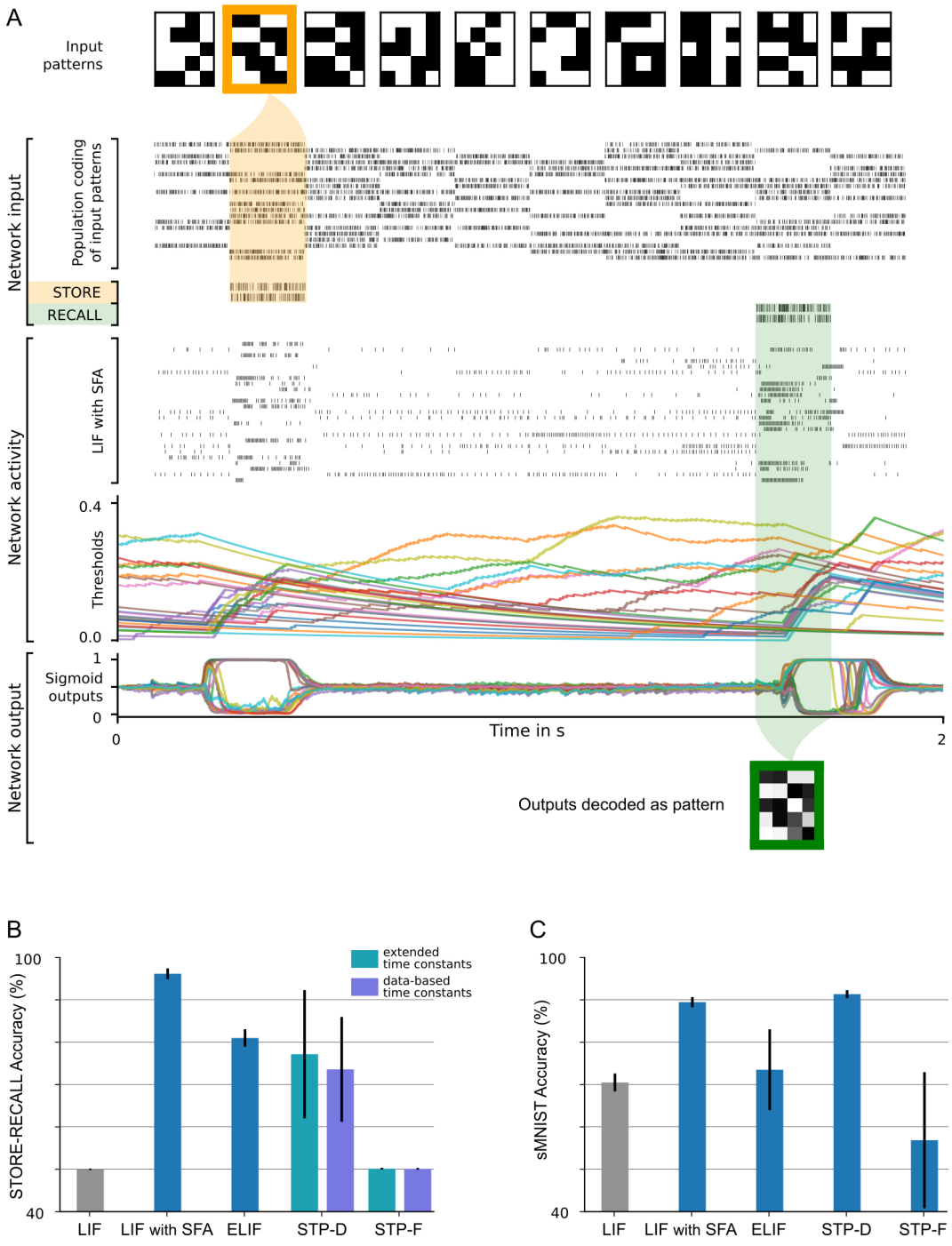


Figure 2.2.: (Caption on the next page.)

Figure 2.2.: **20-dimensional STORE-RECALL and sMNIST task.** (**A**) Sample trial of the 20-dimensional STORE-RECALL task where a trained SNN of LIF neurons with SFA correctly stores (yellow shading) and recalls (green shading) a pattern. (**B-C**) Test accuracy comparison of recurrent SNNs with different slow mechanisms: dual version of SFA where the threshold is decreased and causes enhanced excitability (ELIF), predominantly depressing (STP-D) and predominantly facilitating short-term plasticity (STP-F). (**B**) Test set accuracy of five variants of the SNN model on the one-dimensional STORE-RECALL task. Bars represent the mean accuracy of 10 runs with different network initializations. (**C**) Test set accuracy of the same five variants of the SNN model for the sMNIST time series classification task. Bars represent the mean accuracy of 4 runs with different network initializations. Error bars in (B) and (C) indicate standard deviation.

2.2.3. SFA improves the performance of SNNs for common benchmark tasks that require nonlinear computational operations on temporally dispersed information

We now turn to more demanding temporal computing tasks, where temporally dispersed information not only needs to be stored, but continuously updated in view of new information. We start out in this section with three frequently considered benchmark tasks of this type: sequential MNIST, Google Speech Commands, and delayed XOR.

Sequential MNIST (sMNIST) is a standard benchmark task for time series classification. In this variant of the well-known handwritten digit recognition data set MNIST, the pixel values of each sample of a handwritten digit are presented to the network in the form of a time series, one pixel in each ms, as they arise from a row-wise scanning pattern of the handwritten digit. This task also requires very good generalization capability, since the resulting sequence of pixel values for different handwriting styles of the same digit may be very different, and the network is tested on samples that had never before been presented to the network.

An SNN with SFA was able to solve this task with a test accuracy of 93.7%, whereas an SNN without SFA only reached an accuracy of 51.8%. We refer

to Section 2 of Appendix A for further details.

We also compared the performance of SNNs with and without SFA on the keyword spotting task Google Speech Commands Dataset (Warden, 2018) (v0.02). To solve this task the network needs to correctly classify audio recordings of silence, spoken unknown words, and utterings of one of ten keywords by different speakers. On this task, the performance of SNNs increases with the inclusion of SFA (from 89.04% to 91.21%) and approaches the state-of-the-art artificial recurrent model (93.18%), see Section 3 of Appendix A, and Appendix table A.1.

Finally, we tested the performance of SNNs with SFA on the delayed-memory XOR task, a task which had previously already been used as a benchmark task for SNNs in (Huh & Sejnowski, 2018). In this task, the network is required to compute the exclusive-or operation on a time series of binary input pulses, and provide the answer when prompted by a go-cue signal. Across 10 different runs, an SNN with SFA solved the task with $95.19 \pm 0.014\%$ accuracy, whereas the SNN without SFA just achieved $61.30 \pm 0.029\%$, see Section 4 of Appendix A, and Appendix figure A.2.

The good performance of SNNs with SFA on all three tasks demonstrates that SFA provides computational benefits to SNNs also for substantially more demanding temporal computing tasks in comparison with standard working memory tasks. Before we turn to further temporal computing tasks that are of particular interest from the perspective of neuroscience and cognitive science, we first analyze the contribution of other slow mechanisms in biological neurons and synapses on the basic working memory task and on sMNIST.

2.2.4. Comparing the contribution of SFA to temporal computing with that of other slow processes in neurons and synapses

Facilitating short-term plasticity (STP-F) and depressing short-term plasticity (STP-D) are the most frequently discussed slower dynamic processes in biological synapses. Facilitating short-term plasticity of synapses, also

referred to as paired-pulse facilitation, increases the amplitudes of post-synaptic potentials for the later spikes in a spike train. Whereas synaptic connections between pyramidal cells in the neocortex are usually depressing (Markram et al., 2015), it was shown in (Y. Wang et al., 2006) that there are facilitating synaptic connections between pyramidal cells in the medial prefrontal cortex of rodents, with a mean time constant of 507 ms (standard deviation 37 ms) for facilitation. It was shown in (Mongillo et al., 2008) that if one triples the experimentally found mean time constant for facilitation, then this mechanism supports basic working memory tasks.

Depressing short-term plasticity of synapses, also referred to as paired-pulse depression, reduces the amplitude of postsynaptic potentials for later spikes in a spike train. The impact of this mechanism on simple temporal computing tasks had been examined in a number of publications (Buonomano & Maass, 2009; Hu et al., 2020; Maass et al., 2002; Masse et al., 2019).

In addition, we consider a dual version of SFA: a neuron model where each firing of the neuron causes its firing threshold to decrease — rather than increase as in SFA — which then returns exponentially to its resting value. We call this neuron model the enhanced-excitability LIF (ELIF) model. Such neural mechanisms had been considered for example in (Fransén et al., 2006). Note that a transient increase in the excitability of a neuron can also be caused by depolarization-mediated suppression of inhibition, a mechanism that has been observed in many brain areas (Kullmann et al., 2012). The dynamics of the salient hidden variables in all three models are described in Methods and illustrated in Appendix figure A.7.

We tested the resulting five different types of SNNs, each consisting of 60 neurons, first on the simple 1D working memory task. The results in Figure 2.2B show that SNNs with SFA provide by far the best performance on this task.

Figure 2.2C shows that for sMNIST both SNNs with SFA and SNNs with depressing short-term synaptic plasticity (STP-D) achieve high performance. Surprisingly, the performance of SNNs with facilitating synapses is much worse, both for sMNIST and for the working memory task.

2.2.5. SFA supports demanding cognitive computations on sequences with dynamic rules

Complex cognitive tasks often contain a significant temporal processing component, including the requirement to flexibly incorporate task context and rule changes. To test whether SFA can support such cognitive processing, we consider the 12AX task (Frank et al., 2001). This task is an extension of the A-X version of the continuous performance task (CPT-AX) which has been extensively studied in humans (Barch et al., 2009). It tests the ability of subjects to apply dynamic rules when detecting specific subsequences in a long sequence of symbols while ignoring irrelevant inputs (MacDonald III, 2008; O'Reilly & Frank, 2006). It also probes the capability to maintain and update a hierarchical working memory, since the currently active rule. i.e., the context, stays valid for a longer period of time, and governs what other symbols should be stored in working memory.

More precisely, in the 12AX task, the subject is shown a sequence of symbols from the set {1, 2, A, X, B, Y, C, Z}. After processing any symbol in the sequence, the network should output "R" if this symbol terminates a context-dependent target sequence and "L" otherwise. The current target sequence depends on the current context, which is defined through the symbols "1" and "2". If the most recently received digit was a "1", the subject should output "R" only when it encounters a symbol "X" that terminates a subsequence A...X. This occurs, for example, for the 7th symbol in the trial shown in Figure 2.3. In case the most recent input digit was a "2", the subject should instead respond "R" only after the symbol "Y" in a subsequent subsequence B...Y (see the 20th symbol in Figure 2.3). In addition, the processed sequence contains letters "C" and "Z" that are irrelevant and serve as distractors. This task requires a hierarchical working memory because the most recently occurring digit determines whether subsequent occurrences of "A" or "B" should be placed into working memory. Note also that neither the content of the higher-level working memory, i.e., the digit, nor the content of the lower-level working memory, i.e., the letter A or B, are simply recalled. Instead, they affect the target outputs of the network in a more indirect way. Furthermore, the higher-level processing rule affects what is to be remembered at the lower level.

A simpler version of this task, where X and Y were relevant only if they directly followed A or B respectively, and where fewer irrelevant letters occurred in the input, was solved in (Kruijne et al., 2020; Martinolli et al., 2018; O'Reilly & Frank, 2006) through biologically inspired artificial neural network models that were endowed with special working memory modules. Note that for this simpler version no lower-order working memory is needed, because one just has to wait for an immediate transition from A to X in the input sequence, or for an immediate transition from B to Y. But neither the simpler nor the more complex version, that is considered here, of the 12AX task has previously been solved by a network of spiking neurons.

In the version of the task that we consider, the distractor symbols between relevant symbols occur rather frequently. Hence robust maintenance of relevant symbols in the hierarchical working memory becomes crucial, because time spans between relevant symbols become longer, and hence the task is more demanding — especially for a neural network implementation.

Overall, the network received during each trial (episode) sequences of 90 symbols from the set {1, 2, A, B, C, X, Y, Z}, with repetitions as described in Methods. See the top of Figure 2.3 for an example (the context-relevant symbols are marked in bold for visual ease).

We show in Figure 2.3 that a generic SNN with SFA can solve this quite demanding version of the 12AX task. The network consisted of 200 recurrently connected spiking neurons (100 with and 100 without SFA), with all-to-all connections between them. After training, for new symbol sequences that had never occurred during training, the network produced an output string with all correct symbols in 97.79% of episodes.

The average firing activity of LIF neurons with SFA, and LIF neurons without SFA was (12.37 ± 2.90) Hz, and (10.65 ± 1.63) Hz (mean $\pm$ standard deviation), respectively (the average was calculated over 2000 test episodes for one random initialization of the network). Hence the network operated in a physiologically meaningful regime.

These results were obtained after optimizing synaptic weights via BPTT. However, training with a recently published biologically plausible learn-

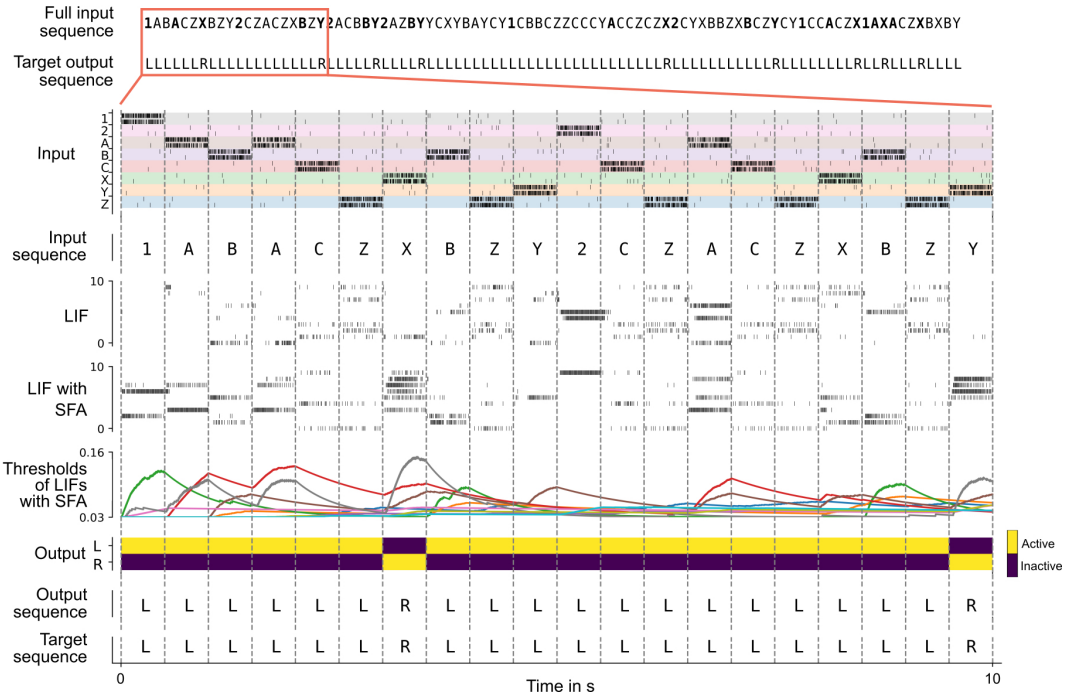


Figure 2.3.: **Solving the 12AX task by a network of spiking neurons with SFA.** A sample trial of the trained network is shown. From top to bottom: Full input and target output sequence for a trial, consisting of 90 symbols each, blow-up for a subsequence – spiking input for the subsequence, the considered subsequence, firing activity of 10 sample LIF neurons without and 10 sample LIF neurons with SFA from the network, time course of the firing thresholds of these neurons with SFA, output activation of the two readout neurons, the resulting sequence of output symbols that the network produced, and the target output sequence.

ing method called *random e-prop* (Bellec et al., 2020) produced a similar performance of 92.89% (averaged over 5 different network initializations).

We next asked how the fraction of neurons with SFA affects SNN performance, in the case of the usual parameter optimization via BPTT. When all, rather than just half of the 200 LIF neurons were endowed with SFA, a much lower accuracy of just 72.01% was achieved. On the other hand, if just 10% of the neurons had SFA a performance of 95.39% was achieved. In contrast, a recurrent SNN with the same architecture but no neurons with SFA only achieved a performance of 0.39% (each success rate was averaged over 5 network initializations). Hence a few neurons with SFA suffice for good performance, and it is important to also have neurons without SFA for this task.

Neuronal networks in the brain are subject to various sources of noise. A highly optimized SNN model with sparse firing activity might utilize brittle spike-time correlations. Such a network would therefore be highly susceptible to noise. To test whether this was the case in our model, we tested how the performance of the above network changed when various levels of noise were added to all network neurons during testing. We found that although the spike responses of the neurons become quite different, see Appendix figures A.4 and A.5, the performance of the SNN model is little affected by low noise, and decays gracefully for higher levels of noise. For details, see Section 5 of Appendix A.

Surprisingly, it was not necessary to create a special network architecture for the two levels of working memory that our more complex version of the 12AX task requires: A near perfectly performing network emerged from training a generic fully connected SNN with SFA.

2.2.6. SFA enables SNNs to carry out complex operations on sequences of symbols

Learning to carry out operations on sequences of symbols in such a way that they generalize to new sequences is a fundamental capability of the human brain, but a generic difficulty for neural networks (Marcus, 2003). Not only humans, but also non-human primates are able to carry out operations on

sequences of items, and numerous neural recordings starting with (Barone & Joseph, 1989) up to recent results such as (Carpenter et al., 2018; Y. Liu et al., 2019) provide information about the neural codes for sequences that accompany such operations in the brain. The fundamental question of how serial order of items is encoded in working memory emerges from the more basic question of how the serial position of an item is combined with the content information about its identity (Lashley, 1951). The experimental data both of (Barone & Joseph, 1989) and (Y. Liu et al., 2019) suggest that the brain uses a factorial code where position and identity of an item in a sequence are encoded separately by some neurons, thereby facilitating flexible generalization of learned experience to new sequences.

We show here that SNNs with SFA can be trained to carry out complex operations on sequences, are able to generalize such capabilities to new sequences, and produce spiking activity and neural codes that can be compared with neural recordings from the brain. In particular, they also produce factorial codes, where separate neurons encode the position and identity of a symbol in a sequence. One basic operation on sequences of symbols is remembering and reproducing a given sequence (Y. Liu et al., 2019). This task had been proposed by (Marcus, 2003) to be a symbolic computation task that is fundamental for symbol processing capabilities of the human brain. But non-human primates can also learn simpler versions of this task, and hence it was possible to analyze how neurons in the brain encode the position and identity of symbols in a sequence (Barone & Joseph, 1989; Carpenter et al., 2018). Humans can also reverse sequences, a task that is more difficult for artificial networks to solve (Y. Liu et al., 2019; Marcus, 2003). We show that an SNN with SFA can carry out both of these operations, and is able to apply them to new sequences of symbols that did not occur during the training of the network.

We trained an SNN consisting of 320 recurrently connected LIF neurons (192 with and 128 without SFA) to carry out these two operations on sequences of 5 symbols from a repertoire of 31 symbols. Once trained, the SNN with SFA could duplicate and reverse sequences that it had not seen previously, with a success rate of 95.88% (average over 5 different network initializations). The "success rate" was defined as the fraction of test episodes (trials) where the full output sequence was generated correctly. Sample episodes of the trained SNN are shown in Figure 2.4, and a zoom-in of the same spike rasters are

provided in Appendix figure A.6. For comparison, we also trained a LIF network without SFA in exactly the same way with the same number of neurons. It achieved a performance of 0.0%.

The average firing activity of LIF neurons without SFA, and LIF neurons with SFA was (19.88 ± 2.68) Hz, and (21.51 ± 2.95) Hz (mean $\pm$ standard deviation), respectively. The average was calculated over 50000 test episodes for one random initialization of the network.

A diversity of neural codes emerge in SNNs with SFA trained to carry out operations on sequences. Emergent coding properties of neurons in the SNN are analyzed in Figure 2.5A-C, and two sample neurons are shown in Figure 2.5D-E. Neurons are sorted in Figure 2.5A,B according to the time of their peak activity (averaged over 1000 episodes), like in (Harvey et al., 2012). The neurons have learned to abstract the overall timing of the tasks (Figure 2.5A). A number of network neurons (about one-third) participate in sequential firing activity independent of the type of task and the symbols involved (see the lower part of Figure 2.5A and the trace for the average activity of neurons left of the marker for the start of duplication or reversal). This kind of activity is reminiscent of the neural activity relative to the start of a trial that was recorded in rodents after they had learned to solve tasks that had a similar duration (Tsao et al., 2018).

The time of peak activity of other neurons in Figure 2.5A depended on the task and the concrete content, indicated by a weak activation during the loading of the sequence (left of the marker), but stronger activation after the start of duplication or reversal (right of the marker). The dependence on the concrete content and task is shown in Figure 2.5B. Interestingly, these neurons change their activation order already during the loading of the input sequence in dependence of the task (duplication or reversal). Using 3-way ANOVA, we were able to categorize each neuron as selective to a specific condition (symbol identity, serial position in the sequence, and type of task) or a non-linear combination of conditions based on the effect size ω^2. Each neuron could belong to more than one category if the effect size was above the threshold of 0.14 (as suggested by (Field, 2013)). Similar to recordings from the brain (Carpenter et al., 2018), a diversity of neural codes emerged that encode one variable or a combination of variables. In other

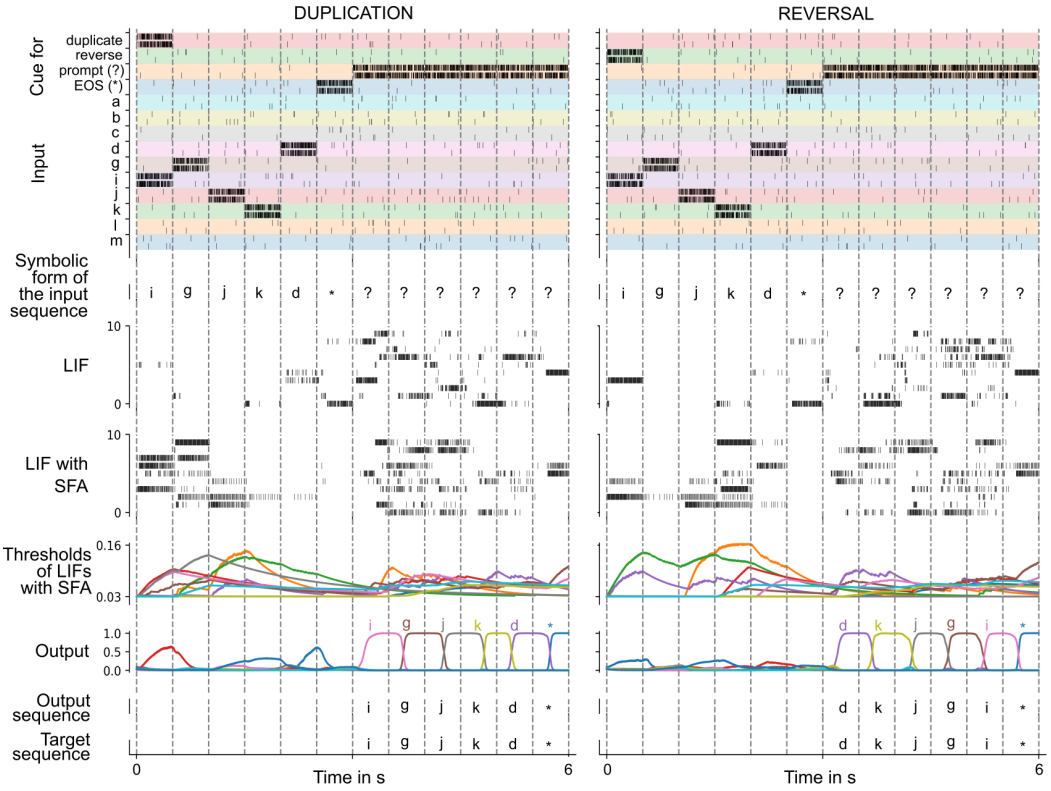


Figure 2.4.: **Spiking activity of an SNN with SFA trained to carry out operations on sequences.** Two sample episodes where the network carried out sequence duplication (left) and reversal (right). Top to bottom: Spike inputs to the network (subset), sequence of symbols they encode, spike activity of 10 sample LIF neurons (without and with SFA) in the SNN, firing threshold dynamics for these 10 LIF neurons with SFA, activation of linear readout neurons, output sequence produced by applying argmax to them, target output sequence.

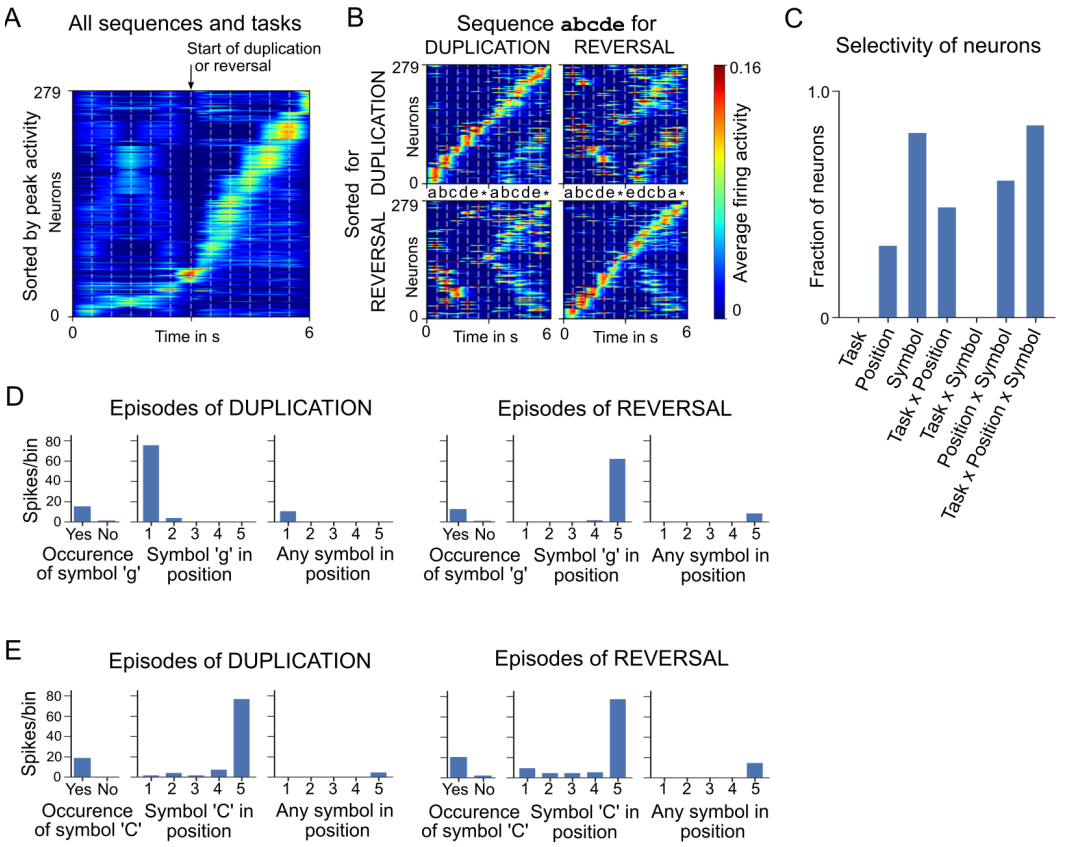


Figure 2.5.: **Analysis of an SNN with SFA trained to carry out operations on sequences.** Emergent neural coding of 279 neurons in the SNN (after removal of neurons detected as outliers), and Peri-Condition Time Histogram (PCTH) plots of two sample neurons. Neurons are sorted by time of peak activity. (**A**) A substantial number of neurons were sensitive to the overall timing of the tasks, especially for the second half of trials when the output sequence is produced. (**B**) Neurons separately sorted for duplication episodes (top row) and reversal episodes (bottom row). Many neurons responded to input symbols according to their serial position, but differently for different tasks. (**C**) Histogram of neurons categorized according to conditions with statistically significant effect (3-way ANOVA). Firing activity of a sample neuron that fired primarily when: (**D**) the symbol "g" was to be written at the beginning of the output sequence. The activity of this neuron depended on the task context during the input period; (**E**) the symbol "C" occurred in position 5 in the input, irrespective of the task context.

words, a large fraction of neurons encoded non-linear combinations of all three variables, see Figure 2.5C. Peri-Condition Time Histogram (PCTH) plots of two sample neurons are shown in Figure 2.5D,E: One neuron is selective to symbol "g" but at different positions depending on task context; The other neuron is selective to the symbol "C" occurring at position 5 in the input, independent of task context. Thus one sees that a realization of this task by an SNN, which was previously not available, provides rich opportunities for a comparison of emergent spike codes in the model and neuronal recordings from the brain. For more details, see the last section of Materials and Methods.

2.3. Discussion

Brains are able to carry out complex computations on temporally dispersed information, for example, on visual input streams, or on sequences of words or symbols. We have addressed the question of how the computational machinery of the brain, recurrently connected networks of spiking neurons, can accomplish that.

The simplest type of temporal computing task just requires holding one item, which typically can be characterized by a single bit, during a delay in working memory, until it is needed for a behavioral response. This can be modeled in neural networks by creating an attractor in the network dynamics that retains this bit of information through persistent firing during the delay. But this implementation is inherently brittle, especially for SNNs, and it is not clear whether it can be scaled up to more ecological working memory tasks where multiple features, e.g., the main features that characterize an image or a story, are kept in working memory, even in the presence of a continuous stream of distracting network inputs. We have shown that SFA enables SNNs to solve this task, even for feature vectors that the network had never encountered before, see Figure 2.2A.

This model for working memory shares many properties with how the human brain stores content that is not attended (Wolff et al., 2017):

1. The data of (Wolff et al., 2017) suggest that in order to understand such working memory mechanisms it is necessary to "look beyond simple measures of neural activity and consider a richer diversity of neural states that underpin content-dependent behavior". We propose that the current excitability of neurons with SFA is an example for such hidden neural state that is highly relevant in this context. This provides a concrete experimentally testable hypothesis.
2. They proposed more specifically that "activity silent neural states are sufficient to bridge memory delays". We have shown this in Figure 2.2A for a quite realistic working memory task, where a complex features vector has to be kept in memory, also in the presence of continuous distractor inputs.
3. They found that an unspecific network input, corresponding to the activation of a population of input neurons for RECALL in our model, is able to recover in the human brain an item that has been stored in working memory, but that storing and recalling of an unattended item by "pinging" the brain generates very different network activity. This is exactly what happens in our model. We have shown that a classifier that was trained to decode the stored item from the neural activity during encoding (STORE) was not able to decode the working memory content during RECALL, and vice versa. Furthermore, we have elucidated a particular neural coding principle, the negative imprinting principle, that is consistent with this effect, see the illustration in Figure 2.1E. An immediate experimentally testable consequence of the negative encoding principle is that the same network responds with reduced firing to repeated representations of an unattended item. This has in fact already been demonstrated for several brain areas, such as sensory cortices (Kok & de Lange, 2015) and perirhinal cortex (Winters et al., 2008).
4. They found that decoding of an unattended working memory item without "pinging" the network "dropped to chance relatively quickly after item presentation". We found that also in our model the content of working memory could not be decoded during the delay between STORE and RECALL.

But obviously ecologically relevant temporal computing tasks that the brain routinely solves are much more complex and demanding than such

standard working memory tasks. The 12AX is a prime example for such more demanding temporally computing task, where two different types of memories need to be continuously updated: memory about the currently valid rule, and memory about data. The currently valid rule determines which item needs to be extracted from the continuous stream of input symbols and remembered: the symbol A or the symbol B. We have shown that SFA enables an SNN to solve this task, without requiring a specific architecture corresponding to the two types of working memory that it requires. This result suggests that it is also unlikely that such two-tiered architecture can be found in the brain, rather that both types of working memory are intertwined in the neural circuitry.

Our result suggests that most other temporal computing tasks that brains are able to solve can also be reproduced by such simple models. We have tested this hypothesis for another cognitively demanding task on temporally dispersed information that has been argued to represent an important "atom of neural computation" in the brain, i.e., an elementary reusable computing primitive on which the astounding cognitive capabilities of the human brain rely (Marcus, 2003; Marcus et al., 2014): the capability to reproduce or invert a given sequence of symbols, even if this sequence has never been encountered before. We have shown in Figures 2.4 and 2.5 that SFA enables SNNs to solve this task. Since also monkeys can be trained to carry out simple operations on sequences of symbols, there are in this case experimental data available on neural codes that are used by the primate brain to encode serial order and identity of sequence items. We found that like in the brain, a diversity of neural codes emerge: Neurons encode one or several of the relevant variables – symbol identity, serial position of a symbol, and type of task. Such comparison of neural coding properties of brains and neural network models is only possible if the model employs — like the brain — spiking neurons, and if the firing rates of these neurons remain in a physiological range of sparse activity, as the presented SNN models tend to provide. Hence, the capability to produce such brain-like computational capabilities in SNNs is likely to enhance the convergence of further biological experiments, models, and theory for uncovering the computational primitives of the primate brain.

Since there is a lack of further concrete benchmark tasks from neuroscience and cognitive science for temporal computing capabilities of brains, we

have also tested the performance of SNNs with SFA on some benchmark tasks that are commonly used in neuromorphic engineering and AI, such as sequential MNIST and the Google Speech Commands Dataset. We found that SNNs with SFA can solve also these tasks very well, almost as well as the state-of-the-art models in machine learning and AI: artificial neural networks with special — unfortunately biologically implausible — units for temporal computing such as Long Short-Term Memory (LSTM) units.

Besides SFA, there are several other candidates for hidden states of biological neurons and synapses that may support brain computations on temporally dispersed information. We examined three prominent candidates for other hidden states, and analyzed how well they support these computations in comparison with SFA: depressing and facilitating short-term plasticity of synapses, as well as an activity-triggered increase in the excitability of neurons (ELIF neurons). We have shown in Figure 2.2B that these other types of hidden states provide lower performance than SFA for the simple working memory task. However, for a more demanding time series classification task short-term depression of synapses provides about the same performance as SFA, see Figure 2.2C. An important contribution of depressing synapses for temporal computing has already previously been proposed (Hu et al., 2020). This is at first sight counter-intuitive, just as the fact that spike-triggered reduction rather than increase of neural excitability provides better support for temporal computing. But a closer look shows that it just requires a "sign-inversion" of readout units to extract information from reduced firing activity. On the other hand, reduced excitability of neurons has the advantage that this hidden state is better protected against perturbations by ongoing network activity: A neuron that is in an adapted state where it is more reluctant to fire is likely to respond less to noise inputs, thereby protecting its hidden state from such noise. In contrast, a more excitable neuron is likely to respond also to weaker noise inputs, thereby diluting its hidden state. These observations go in a similar direction as the results of (Kim & Sejnowski, 2021; Mongillo et al., 2018), which suggest that inhibition, rather than excitation, is critical for robust memory mechanisms in the volatile cortex.

Finally, it should be pointed out that there are numerous other hidden states in biological neurons and synapses that change on slower time scales. One prominent example are metabotropic glutamate receptors, which are

present in a large fraction of synapses throughout the thalamus and cortex. Metabotropic receptors engage a complex molecular machinery inside the neuron which integrates signals also over long time scales from seconds to hours and days (Sherman, 2014). However, at present, we are missing mathematical models for these processes, and hence it is hard to evaluate their contribution to temporal computing.

We have analyzed here the capabilities and limitations of various types of SNNs, and have not addressed the question of how such capabilities could be induced in SNNs of the brain. Hence, we have used the most powerful optimization method for inducing computational capabilities in SNNs: a spike-oriented adaptation of BPTT. It was previously shown in (Bellec et al., 2020) that in general almost the same performance can be achieved in SNNs with SFA when BPTT is replaced by *e-prop*, a more biologically plausible network gradient descent algorithm. We have tested this also for the arguably most difficult temporal computing task that was examined in this paper, 12AX, and found that *e-prop* provides almost the same performance. However temporal computing capabilities are likely to arise in brains through a combination of nature and nurture, and it remains to be examined to what extent the genetic code endows SNNs of the brain with temporal computing capabilities. In one of the first approaches for estimating the impact of genetically encoded connection probabilities on computational capabilities it was already shown that connection probabilities can provide already some type of working memory without any need for learning (Stöckl et al., 2021), but this approach has not yet been applied to SNNs with SFA or other slowly changing hidden states.

Finally, our results raise the question of whether the distribution of time constants of SFA in a cortical area is related to the intrinsic time scale of that cortical area, as measured for example via intrinsic fluctuations of spiking activity (J. D. Murray et al., 2014; Wasmuht et al., 2018). Unfortunately, we are lacking experimental data on the time constants of SFA in different brain areas. We tested the relation between time constants of SFA and the intrinsic time scale of neurons according to (Wasmuht et al., 2018) for the case of the STORE-RECALL task (see Section 1 of Appendix A and Appendix figure A.1). We found that the time constants of neurons with SFA had little impact on their intrinsic time scale for this task, in particular much less than the network input. We have also shown in control experiments that

the alignment between the time scale of SFA and the time scale of working memory duration can be rather loose. Even a random distribution of time constants for SFA works well.

Altogether, we have shown that SFA, a well-known feature of a substantial fraction of neurons in the neocortex, provides an important new facet for our understanding of computations in SNNs: It enables SNNs to integrate temporally dispersed information seamlessly into ongoing network computations. This paves the way for reaching a key goal of modeling: to combine detailed experimental data from neurophysiology on the level of neurons and synapses with the brain-like high computational performance of the network.

2.4. Materials and Methods

In this section, we first describe the details of the network models that we employ, and then we continue with the description of the training methods. After that, we give details about all the tasks and analyses performed.

2.4.1. Network models

Leaky integrate and fire (LIF) neurons. A LIF neuron j spikes as soon as its membrane potential $V_j(t)$ is above its threshold v_{th}. At each spike time t, the membrane potential $V_j(t)$ is reset by subtracting the threshold value v_{th} and the neuron enters a strict refractory period for 3 to 5 ms (depending on the experiment) where it cannot spike again. Between spikes, the membrane voltage $V_j(t)$ is following the dynamics:

$$\tau_m \dot{V}_j(t) \quad = \quad -V_j(t) + R_m I_j(t), \tag{2.2}$$

where τ_m is the membrane constant of neuron j, R_m is the resistance of the cell membrane, and I_j the input current.

Our simulations were performed in discrete time with a time step $\delta t = 1\,\text{ms}$. In discrete time, the input and output spike trains are modeled as binary

sequences $x_i(t), z_j(t) \in \{0, \frac{1}{\delta t}\}$ respectively. Neuron j emits a spike at time t if it is currently not in a refractory period, and its membrane potential $V_j(t)$ is above its threshold. During the refractory period following a spike, $z_j(t)$ is fixed to 0. The neural dynamics in discrete time reads as follows:

$$V_j(t+\delta t) \quad = \quad \alpha V_j(t) + (1-\alpha) R_m I_j(t) - v_{\text{th}} z_j(t) \delta t, \tag{2.3}$$

where $\alpha = \exp(-\frac{\delta t}{\tau_m})$, with τ_m being the membrane constant of the neuron j. The spike of neuron j is defined by $z_j(t) = H\left(\frac{V_j(t)-v_{\text{th}}}{v_{\text{th}}}\right)\frac{1}{\delta t}$, with $H(x) = 0$ if $x < 0$ and 1 otherwise. The term $-v_{\text{th}} z_j(t)\delta t$ implements the reset of the membrane voltage after each spike.

In all simulations, the R_m was set to 1 GΩ. The input current $I_j(t)$ is defined as the weighted sum of spikes from external inputs and other neurons in the network:

$$I_j(t) \quad = \quad \sum_i W_{ji}^{\text{in}} x_i(t - d_{ji}^{\text{in}}) + \sum_i W_{ji}^{\text{rec}} z_i(t - d_{ji}^{\text{rec}}), \tag{2.4}$$

where W_{ji}^{in} and W_{ji}^{rec} denote respectively the input and the recurrent synaptic weights and d_{ji}^{in} and d_{ji}^{rec} the corresponding synaptic delays.

LIF neurons with SFA. The SFA is realized by replacing the fixed threshold v_{th} with the adaptive threshold $A_j(t)$ which follows the dynamics (reproducing equation (2.1) for arbitrary δt):

$$\begin{aligned} A_j(t) &= v_{\text{th}} + \beta a_j(t), \\ a_j(t+\delta t) &= \rho_j a_j(t) + (1-\rho_j) z_j(t) \delta t. \end{aligned} \tag{2.5}$$

Now, the parameter ρ_j is given by $\rho_j = \exp\left(\frac{-\delta t}{\tau_{a,j}}\right)$. In all our simulations, δt was set to 1 ms.

The spiking output of LIF neuron with SFA j is then defined by $z_j(t) = H\left(\frac{V_j(t)-A_j(t)}{A_j(t)}\right)\frac{1}{\delta t}$.

Adaptation time constants of neurons with SFA were chosen to match the task requirements while still conforming to the experimental data from

rodents (Allen Institute, 2018; Mensi et al., 2012; Pozzorini et al., 2015; Pozzorini et al., 2013). For an analysis of the impact of the adaptation time constants on the performance see Table 2.1.

LIF neurons with activity-dependent increase in excitability: ELIF neurons. There exists experimental evidence that some neurons fire for the same stimulus more for a repetition of the same sensory stimulus. We refer to such neurons as ELIF neurons, since they are becoming more excitable. Such repetition enhancement was discussed for example in (Tartaglia et al., 2015). But to the best of our knowledge, it has remained open whether repetition enhancement is a network effect, resulting for example from a transient depression of inhibitory synapses onto the cell that is caused by postsynaptic firing (Kullmann et al., 2012), or a result of an intrinsic firing property of some neurons. We used a simple model for ELIF neurons that is dual to the above-described LIF neuron model with SFA: The threshold is lowered through each spike of the neuron, and then decays exponentially back to its resting value. This can be achieved by using a negative value for β in equation (2.1).

Models for Short-Term Plasticity (STP) of synapses. We modeled the STP dynamic according to the classical model of STP in (Mongillo et al., 2008). The STP dynamics in discrete time, derived from the equations in (Mongillo et al., 2008), are as follows:

$$u'_{ji}(t+\delta t) = \exp\left(\frac{-\delta t}{F}\right) u'_{ji}(t) + U_{ji}(1-u_{ji}(t))z_i(t)\delta t, \quad (2.6)$$

$$u_{ji}(t+\delta t) = U_{ji} + u'_{ji}(t), \quad (2.7)$$

$$r'_{ji}(t+\delta t) = \exp\left(\frac{-\delta t}{D}\right) r'_{ji}(t) + u_{ji}(t)(1-r'_{ji}(t))z_i(t)\delta t, \quad (2.8)$$

$$r_{ji}(t+\delta t) = 1 - r'_{ji}(t), \quad (2.9)$$

$$W^{STP}_{ji}(t+\delta t) = W^{\text{rec}}_{ji} u_{ji}(t) r_{ji}(t), \quad (2.10)$$

where $z_i(t)$ is the spike train of the pre-synaptic neuron and W^{rec}_{ji} scales the synaptic efficacy of synapses from neuron i to neuron j. Networks with STP

were constructed from LIF neurons with the weight W_{ji}^{rec} in equation (2.4) replaced by the time-dependent weight $W_{ji}^{STP}(t)$.

STP time constants of facilitation-dominant and depression-dominant network models were based on values of experimental recordings in (Y. Wang et al., 2006) of PFC-E1 ($D = 194 \pm 18$, $F = 507 \pm 37$, $U = 0.28 \pm 0.02$) and PFC-E2 ($D = 671 \pm 17$, $F = 17 \pm 5$, $U = 0.25 \pm 0.02$) synapse types respectively. Recordings in (Y. Wang et al., 2006) were performed in the medial prefrontal cortex of young adult ferrets. In the sMNIST task for the depression-dominant network model (STP-D) we used values based on PFC-E2, and for facilitation-dominant network model (STP-F) we used values based on PFC-E1, see sMNIST task section below. For the STORE-RECALL task, we trained the network with the data-based time constants based on PFC-E2 and PFC-E1 and also an extended time constants variant where both facilitation and depression time constants were equally scaled up until the larger time constant matched the requirement of the task, see one-dimensional STORE-RECALL task section below.

Weight initialization. Initial input and recurrent weights were drawn from a Gaussian distribution $W_{ji} \sim \frac{w_0}{\sqrt{n_{\text{in}}}}\mathcal{N}(0,1)$, where n_{in} is the number of afferent neurons and $\mathcal{N}(0,1)$ is the zero-mean unit-variance Gaussian distribution and $w_0 = \frac{1\,\text{Volt}}{R_m}\delta t$ is a normalization constant (Bellec, Salaj, et al., 2018). In the default setting, it is possible for neurons to have both positive and negative outgoing weights, also to change their sign during the optimization process. See Section 2 of Appendix A for more results with sparse connectivity and enforcement of Dale's Law.

Sigmoid and softmax functions. In the STORE-RECALL task (one and 20-dimensional), the sigmoid function was applied to the neurons in the output layer. The sigmoid function is given by:

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}, \tag{2.11}$$

where x represents a real-valued variable. The result, bounded to $[0,1]$ range, is then thresholded at the value of 0.5 to obtain the final predictions

– neuron active or not. More precisely, the neuron is active if $\sigma(x) \geq 0.5$, otherwise it is not.

The softmax function (used in tasks sMNIST, 12AX, Duplication/Reversal) is given by:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}, \tag{2.12}$$

where x_i is a real-valued output of neuron i in the output layer with m neurons. The final prediction after applying the softmax is then obtained by taking the maximum of all values calculated for each neuron in the output layer.

2.4.2. Training methods

BPTT. In artificial recurrent neural networks, gradients can be computed with Backpropagation Through Time (*BPTT*) (Mozer, 1989; Robinson & Fallside, 1987; Werbos, 1988). In spiking neural networks, complications arise from the non-differentiability of the output of spiking neurons. In our discrete-time simulation, this is formalized by the discontinuous step function H arising in the definition of the spike variable $z_j(t)$. All other operations can be differentiated exactly with *BPTT*. For feedforward artificial neural networks using step functions, a solution was to use a pseudo derivative $H'(x) := \max\{0, 1 - |x|\}$ (Esser et al., 2016), but this method is unstable with recurrently connected neurons. It was found in (Bellec, Salaj, et al., 2018) that dampening this pseudo-derivative with a factor $\gamma < 1$ (typically $\gamma = 0.3$) solves that issue. Hence we use the pseudo-derivative:

$$\frac{dz_j(t)}{dv_j(t)} := \gamma \max\{0, 1 - |v_j(t)|\}, \tag{2.13}$$

where $v_j(t)$ denotes the normalized membrane potential $v_j(t) = \frac{V_j(t) - A_j(t)}{A_j(t)}$. Importantly, gradients can propagate in adaptive neurons through many time steps in the dynamic threshold without being affected by the dampening.

Unless stated otherwise, the input, the recurrent, and the readout layers were fully connected and the weights were trained simultaneously.

e-prop. In the 12AX task, the networks were trained using the biologically plausible learning method *random e-prop* (Bellec et al., 2020) in addition to *BPTT*.

2.4.3. Tasks

One-dimensional STORE-RECALL task. The input to the network consisted of 40 input neurons: 10 for STORE, 10 for RECALL, and 20 for population coding of a binary feature. Whenever a subpopulation was active, it would exhibit a Poisson firing with a frequency of 50 Hz. For experiments reported in Figure 2.2D each input sequence consisted of 20 steps (200 ms each) where the STORE or the RECALL populations were activated with probability 0.09 interchangeably which resulted in delays between the STORE-RECALL pairs to be in the range [200, 3600] ms. For experiments reported in Table 2.1, the input sequences of experiments with the expected delay of 2, 4, 8, and 16 s were constructed as a sequence of 20, 40, 80, 120 steps respectively, with each step lasting for 200 ms. For the experiment with an expected delay of 200 ms, the input sequence consisted of 12 steps of 50 ms.

Networks were trained for 400 iterations with a batch size of 64 in Table 2.1, and 128 in Figure 2.2B. We used the Adam optimizer with default parameters and an initial learning rate of 0.01 which was decayed every 100 iterations by a factor of 0.3. To avoid unrealistically high firing rates, the loss function contained a regularization term (scaled with coefficient 0.001) that minimized the squared difference of the average firing rate of individual neurons from a target firing rate of 10 Hz. In Figure 2.1D,E the weights were chosen by hand and not trained. The test performance was computed as the batch average over 2048 random input sequences.

Networks consisted of 60 recurrently connected neurons in all experiments except in Figure 2.1D,E where only two neurons were used without recurrent connections. The membrane time constant was $\tau_m = 20$ ms, the refractory period 3 ms. In Figure 2.1D,E, the two LIF neurons with SFA had $\beta = 3$ mV and $\tau_a = 1200$ ms. In Figure 2.2D, for LIF with SFA and ELIF networks, we

used $\beta = 1\,\text{mV}$ and $\beta = -0.5\,\text{mV}$ respectively, with $\tau_a = 2000\,\text{ms}$. Table 2.1 defines the adaptation time constants and expected delay of the experiments in that section. To provide a fair comparison between STP and SFA models in Figure 2.2D, we train two variants of the STP model: one with the original parameters from (Y. Wang et al., 2006) and another where we scaled up both F and D until the larger one reached 2000 ms, the same time constant used in the SFA model. The scaled up synapse parameters of STP-D network were $F = 51 \pm 15\,\text{ms}$, $D = 2000 \pm 51\,\text{ms}$ and $U = 0.25$, and of STP-F network $F = 2000 \pm 146\,\text{ms}$, $D = 765 \pm 71\,\text{ms}$ and $U = 0.28$. The data-based synapse parameters are described in the STP synapse dynamics section above. The baseline threshold voltage was 10 mV for all models except ELIF for which it was 20 mV and the two neurons in Figure 2.1D,E for which it was 5 mV. The synaptic delay was 1 ms. The input to the sigmoidal readout neurons were the neuron traces that were calculated by passing all the network spikes through a low-pass filter with a time constant of 20 ms.

20-dimensional STORE-RECALL task. The input to the network consisted of commands STORE and RECALL, and 20 bits which were represented by subpopulations of spiking input neurons. STORE and RECALL commands were represented by 4 neurons each. The 20 bits were represented by population coding where each bit was assigned 4 input neurons (2 for value zero, and 2 for value one). When a subpopulation was active, it would exhibit a Poisson firing with a frequency of 400 Hz. Each input sequence consisted of 10 steps (200 ms each) where a different population encoded bit string was shown during every step. Only during the RECALL period, the input populations, representing the 20 bits, were silent. At every step, the STORE or the RECALL populations were activated interchangeably with probability 0.2 which resulted in the distribution of delays between the STORE-RECALL pairs in the range [200, 1600] ms.

To measure the generalization capability of a trained network, we first generated a test set dictionary of 20 unique feature vectors (random bit strings of length 20) that had at least a Hamming distance of 5 bits among each other. For every training batch, a new dictionary of 40 random bit strings (of length 20) was generated, where each string had a Hamming distance of at least 5 bits from any of the bit strings in the test set dictionary.

This way we ensured that, during training, the network never encountered any bit string similar to one from the test set.

Networks were trained for 4000 iterations with a batch size of 256 and stopped if the error on the training batch was below 1%. We used the Adam optimizer (Kingma & Ba, 2015) with default parameters and an initial learning rate of 0.01 which is decayed every 200 iterations by a factor of 0.8. We also used learning rate ramping, which, for the first 200 iterations, monotonically increased the learning rate from 0.00001 to 0.01. The same firing rate regularization term was added to the loss as in the one-dimensional STORE-RECALL setup (see above). To improve convergence, we also included an entropy component to the loss (scaled with coefficient 0.3) which was computed as the mean of the entropies of the outputs of the sigmoid neurons. The test performance was computed as average over 512 random input sequences.

We trained SNNs with and without SFA, consisting of 500 recurrently connected neurons. The membrane time constant was $\tau_m = 20\,\text{ms}$ and the refractory period was 3 ms. Adaptation parameters were $\beta = 4\,\text{mV}$ and $\tau_a = 800\,\text{ms}$ with baseline threshold voltage 10 mV. The synaptic delay was 1 ms. The same sigmoidal readout neuron setup was used as in the one-dimensional STORE-RECALL setup (see above).

We ran 5 training runs with different random seeds (initializations) for both SNNs with and without SFA. All runs of the SNN with SFA network converged after $\sim$ 3600 iterations to a training error below 1%. At that point, we measured the accuracy on 512 test sequences generated using the previously unseen test bit strings which resulted in test accuracy of 99.09% with a standard deviation of 0.17%. The LIF network was not able to solve the task in any of the runs (all runs resulted in 0% training and test accuracy with zero standard deviation). On the level of individual feature recall accuracy, the best one out of 5 training runs of the LIF network was able to achieve 49% accuracy which is the chance level since individual features are binary bits. In contrast, all SNNs with SFA runs had individual feature level accuracy of above 99.99%.

Decoding memory from the network activity. We trained a Support Vector Machine (SVM) to classify the stored memory content from the network

spiking activity in the step before the RECALL (200 ms before the start of RECALL command). We performed a cross-validated grid-search to find the best hyperparameters for the SVM which included kernel type {linear, polynomial, RBF} and penalty parameter C of the error term {0.1, 1, 10, 100, 1000}. We trained SVMs on test batches of the 5 different training runs of the 20-dimensional STORE-RECALL task. SVMs trained on the period preceding the RECALL command of a test batch achieved an average of 4.38% accuracy with a standard deviation of 1.29%. In contrast, SVMs trained on a period during the RECALL command achieved an accuracy of 100%. This demonstrates that the memory stored in the network is not decodable from the network firing activity before the RECALL input command.

Additionally, analogous to the experiments of (Wolff et al., 2017), we trained SVMs on network activity during the encoding (STORE) period and evaluated them on the network activity during reactivation (RECALL), and vice versa. In both scenarios, the classifiers were not able to classify the memory content of the evaluation period (0.0% accuracy).

sMNIST task. The input consisted of sequences of 784 pixel values created by unrolling the handwritten digits of the MNIST dataset, one pixel after the other in a scanline manner as indicated in Appendix figure A.2A. We used 1 ms presentation time for each pixel gray value. Each of the 80 input neurons was associated with a particular threshold for the grey value, and this input neuron fired whenever the grey value crossed its threshold in the transition from the previous to the current pixel.

Networks were trained for 36000 iterations using the Adam optimizer with batch size 256. The initial learning rate was 0.01 and every 2500 iterations the learning rate was decayed by a factor of 0.8. The same firing rate regularization term was added to the loss as in the STORE-RECALL setup (see above) but with the scaling coefficient of 0.1.

All networks consisted of 220 neurons. Network models labeled LIF with SFA and ELIF in Figure 2.2C had 100 neurons out of 220 with SFA or transient excitability respectively. The network with SFA had 100 neurons out of 220 with SFA and the rest without. The neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, a baseline threshold of $v_{\text{th}} = 10\,\text{mV}$, and a

refractory period of 5 ms. LIF neurons with SFA and ELIF neurons had the adaptation time constant $\tau_a = 700$ ms with adaptation strength $\beta = 1.8$ mV and -0.9 mV respectively. The synaptic delay was 1 ms. Synapse parameters were $F = 20$ ms, $D = 700$ ms and $U = 0.2$ for STP-D model, and $F = 500$ ms, $D = 200$ ms and $U = 0.2$ for STP-F model. The output of the SNN was produced by the softmax of 10 linear output neurons that received the low-pass filtered version of the spikes from all neurons in the network, as shown in the bottom row of Appendix figure A.2B. The low-pass filter had a time constant of 20 ms. For training the network to classify into one of the ten classes we used cross-entropy loss computed between the labels and the softmax of output neurons.

The 12AX task. The input for each training and testing episode consisted of a sequence of 90 symbols from the set {1,2,A,B,C,X,Y,Z}. A single episode could contain multiple occurrences of digits 1 or 2 (up to 23), each time changing the target sequence (A...X or B...Y) after which the network was supposed to output R. Each digit could be followed by up to 26 letters before the next digit appeared. More precisely, the following regular expression describes the string that was produced:

`[12][ABCXYZ]{1,10}((A[CZ]{0,6}X|B[CZ]{0,6}Y)|([ABC][XYZ])){1,2}.`

Each choice in this regular expression was made randomly.

The network received spike trains from the input population of spiking neurons, producing Poisson spike trains. Possible input symbols were encoded using "one-hot encoding" scheme. Each input symbol was signaled through a high firing rate of a separate subset of 5 input neurons for 500 ms. The output consisted of two readout neurons, one for L, and one for the R response. During each 500 ms time window, the input to these readouts was the average activity of neurons in the SNN during that time window. The final output symbol was based on which of the two readouts had the maximum value.

The neurons had a membrane time constant of $\tau_m = 20$ ms, a baseline threshold $v_{\text{th}} = 30$ mV, a refractory period of 5 ms, and synaptic delays of 1 ms. LIF neurons with SFA had an adaptation strength of $\beta = 1.7$ mV, and adaptation time constants were chosen uniformly from $[1, 13500]$ ms.

A cross-entropy loss function was used to minimize the error between the softmax applied to the output layer and targets, along with a regularization term (scaled with coefficient 15) that minimized the squared difference of average firing rate between individual neurons and a target firing rate of 10 Hz. The SNN was trained using the Adam optimizer for 10000 iterations with a batch size of 20 episodes and a fixed learning rate of 0.001. An episode consisted of 90 steps, with between 4 to 23 tasks generated according to the task generation procedure described previously. We trained the network consisting of 200 LIF neurons (100 with and 100 without SFA) with *BPTT* using 5 different network initializations, which resulted in an average test success rate of 97.79% with a standard deviation of 0.42%.

In the experiments where the fraction of neurons with SFA varied, the network with 200 LIF neurons with SFA (i.e., all LIF neurons with SFA) achieved a success rate of 72.01% with a standard deviation of 36.15%, whereas the network with only 20 LIF neurons with SFA and 180 LIF neurons without SFA achieved a success rate of 95.39% with a standard deviation of 1.55%. The network consisting of 200 LIF neurons without SFA (i.e., all neurons without SFA) was not able to solve the task and it achieved a success rate of 0.39% with a standard deviation of 0.037%. Each success rate reported is an average calculated over 5 different network initializations.

The network consisting of 100 LIF neurons with and 100 LIF neurons without SFA, trained with *random e-prop*, resulted in an average test success rate of 92.89% with a standard deviation of 0.75% (average over 5 different network initializations).

Symbolic computation on strings of symbols (Duplication/Reversal task). The input to the network consisted of 35 symbols: 31 symbols represented symbols from the English alphabet {a, b, c, d, ... x, y, z, A, B, C, D, E}, one symbol was for "end-of-string" (EOS) '*', one for cue for the output prompt '?', and two symbols to denote whether the task command was duplication or reversal. Each of the altogether 35 input symbols were given to the network in the form of higher firing activity of a dedicated population of 5 input neurons outside of the SNN ("one-hot encoding"). This population of input neurons fired at a "high" rate (200 Hz) to encode 1, and at a "low" rate (2 Hz) otherwise. The network output was produced by linear readouts

(one per potential output symbol, each with a low pass filter with a time constant of 250 ms) that received spikes from neurons in the SNN, see the row "Output" in Figure 2.4. The final output symbol was selected using the readout which had the maximum value at the end of each 500 ms time window (a softmax instead of the hard argmax was used during training), mimicking winner-take-all computations in neural circuits of the brain (Chettih & Harvey, 2019) in a qualitative manner.

The network was trained to minimize the cross-entropy error between the softmax applied to the output layer and targets. The loss function contained a regularization term (scaled with coefficient 5) that minimized the squared difference of average firing rate between individual neurons and a target firing rate of 20 Hz.

The training was performed for 50000 iterations, with a batch size of 50 episodes. We used Adam optimizer with default parameters and a fixed learning rate of 0.001. Each symbol was presented to the network for a duration of 500 ms. The primary metric we used for measuring the performance of the network was success rate, which was defined as the percentage of episodes where the network produced the full correct output for a given string, i.e., all the output symbols in the episode had to be correct. The network was tested on 50000 previously unseen strings.

The network consisted of 192 LIF neurons with SFA and 128 LIF neurons without SFA. All the neurons had a membrane time constant of $\tau_m = 20$ ms, a baseline threshold $v_{\text{th}} = 30$ mV, a refractory period of 5 ms, and a synaptic delay of 1 ms. LIF neurons with SFA in the network had an adaptation strength of $\beta = 1.7$ mV. It was not necessary to assign particular values to adaptation time constants of firing thresholds of neurons with SFA; we simply chose them uniformly randomly to be between 1 ms and 6000 ms, mimicking the diversity of SFA effects found in the neocortex (Allen Institute, 2018) in a qualitative manner. All other parameters were the same as in the other experiments. We trained the network using 5 different network initializations (seeds) and tested it on previously unseen strings. The average test success rate was 95.88% with a standard deviation of 1.39%.

Analysis of spiking data for Duplication/Reversal task. We used 3-way ANOVA to analyze if a neuron's firing rate is significantly affected by *task*,

serial position in the sequence, *symbol identity*, or combination of these (similar to (Lindsay et al., 2017)). In such a multifactorial experiment, factors are crossed with each other, and we refer to these factors as "conditions". For 2 possible tasks, 5 possible positions in the input sequence, and 31 possible symbols, there are $2 \cdot 5 \cdot 31 = 310$ different conditions. The analysis was performed on the activity of the neurons of the trained SNN during 50000 test episodes. From each episode, a serial position from the input period was chosen randomly, and hence each episode could be used only once, i.e., as one data point. This was to make sure that each entry in the 3-way ANOVA was completely independent of other entries, since the neuron's activity within an episode is highly correlated. Each data point was labeled with the corresponding triple of (task type, serial position, and symbol identity). To ensure that the dataset was balanced, the same number of data points per particular combination of conditions was used, discarding all the excess data points, resulting in a total of 41850 data points – 135 data points per condition, i.e., 135 repeated measurements for each condition and per neuron, but with no carryover effects for repetitions per neuron since the internal state variables of a neuron are reset between episodes. In such a scenario, neurons can be seen as technical replicates. For the analysis, neurons whose average firing rate over all episodes (for the input period) was lower than 2 Hz or greater than 60 Hz were discarded from the analysis to remove large outliers. This left 279 out of the 320 neurons. To categorize a neuron as selective to one or more conditions, or a combination of conditions, we observed p-values obtained from the $3-$way ANOVA and calculated the effect size ω^2 for each combination of conditions. If the p-value was smaller than 0.001 and ω^2 greater than 0.14 for a particular combination of conditions, the neuron was categorized as selective to that combination of conditions. The ω^2 threshold of 0.14 was suggested by (Field, 2013) to select large effect sizes. Each neuron can have a large effect size for more than one combination of conditions. Thus the values shown in Figure 2.5C sum to > 1. The neuron shown in Figure 2.5D had the most prominent selectivity for the combination of Task $\times$ Position $\times$ Symbol, with $\omega^2 = 0.394$ and $p < 0.001$. The neuron shown in Figure 2.5E was categorized as selective to a combination of Position $\times$ Symbol category, with $\omega^2 = 0.467$ and $p < 0.001$. While the 3-way ANOVA tells us if a neuron is selective to a particular combination of conditions, it does not give us the exact task/symbol/position that the neuron is selective to. To

find the specific task/symbol/position that the neuron was selective to, Welch's t-test was performed, and a particular combination with maximum t-statistic and $p < 0.001$ was chosen to be shown in Figure 2.5D,E.

3. Spike-based symbolic computation on bit strings and numbers

Abstract The brain uses recurrent spiking neural networks for higher cognitive functions such as symbolic computations, in particular, mathematical computations. We review the current state of research on spike-based symbolic computations of this type. In addition, we present new results which show that surprisingly small spiking neural networks can perform symbolic computations on bit sequences and numbers and even learn such computations using a biologically plausible learning rule. The resulting networks operate in a rather low firing rate regime, where they could not simply emulate artificial neural networks by encoding continuous values through firing rates. Thus, we propose here a new paradigm for symbolic computation in neural networks that provides concrete hypotheses about the organization of symbolic computations in the brain. The employed spike-based network models are the basis for drastically more energy-efficient computer hardware – neuromorphic hardware. Hence, our results can be seen as creating a bridge from symbolic artificial intelligence to energy-efficient implementation in spike-based neuromorphic hardware.

This chapter was published as:

Kraišniković, C., Maass, W., Legenstein, R. (2021). Spike-Based Symbolic Computations on Bit Strings and Numbers. In Neuro-Symbolic Artificial Intelligence: The State of the Art (pp. 214-234). IOS Press.

Contributions Conceptualization, Data curation, Software, Formal analysis, Validation, Investigation, Visualization, Writing - original draft, Writing

- review and editing.

Acknowledgements This research was partially supported by the Human Brain Project (Grant Agreement number 785907 and 945539), the SYNCH project (Grant Agreement number 824162) of the European Union, and under partial support by the Austrian Science Fund (FWF) within the ERA-NET CHIST-ERA programme (project SMALL, project number I 4670-N).

3.1. Introduction

Spiking neural networks (SNNs) constitute the third generation of neural network models (Maass, 1997). The most salient difference to standard artificial neural networks (ANNs) that are usually employed in deep learning is the means of communication between neurons in the network. While ANNs communicate real numbers in each computational step, spiking neurons in SNNs communicate events in an asynchronous manner. The output of a spiking neuron is zero most of the time, only occasionally turning to "1", which is referred to as spiking or sending a spike. This communication scheme is much closer to the way neurons communicate in the brain and it has some advantages that we will discuss below. In this chapter, we discuss symbolic computations with SNNs. In particular, since we concentrate on tasks that involve temporally sequential input which necessitates recurrent connections in order to combine temporally dispersed information, we consider here networks of recurrently connected spiking neurons (RSNNs). This also reflects the fact that neuronal networks in the brain are highly recurrent. We use the term RSNNs whenever we want to emphasize the recurrent architecture of the considered network. We review the current state of this research direction and in addition, present new results which show that surprisingly small SNNs can perform symbolic computations on bit sequences and numbers.

Why should we bother about RSNNs when thinking about neuro-symbolic computation? There are at least two answers to this question. First, the brain is very likely a computing system that merges sub-symbolic connectionist computation with symbolic components. It has been argued many times

that a pure subsymbolic system cannot achieve the generalization abilities of symbolic systems where variables can be used in various contexts (Marcus, 2003, 2018; Marcus et al., 2014). Hence, the superb generalization capabilities of the brain likely result from symbolic computations. On the other hand, judged purely from the hardware, it is undisputed that the brain is a neural system. However, if we want to learn from the brain, we should take its basic computational units seriously. The event-based nature of spiking communication in the brain is in many aspects different from communication in ANNs with real-valued neurons. One can in principle quite easily emulate every ANN with an SNN since each real-valued output can be encoded by a spike rate, and several such and more advanced conversion methods exist (Kim et al., 2019; Rueckauer & Liu, 2018). On the other hand, due to the asynchronous and spike-timing based communication in SNNs, the emulation of an SNN with an ANN is much harder (Maass, 1997). Hence, it might not be straightforward to map the solution that evolution has found for symbolic computation to ANNs. Second, the brain is a network of a gigantic number of neurons, and the experience with deep learning has shown that bigger is usually better. Hence, we can expect that neuro-symbolic systems that approach at least some aspects of human intelligence will be huge. To make them practically useful, one should consider energy-efficient hardware solutions. One reason why the brain is extremely energy-efficient is its spike-based communication, where spikes are transmitted only when necessary. Energy-efficient neuromorphic hardware is based on this basic principle utilizing RSNNs (Davies et al., 2021; Furber, 2016; Maass, 2016).

Another lesson we learned from recent artificial intelligence (AI) research is that learning-based approaches are the most promising path to AI systems. Hence, we should not try to construct symbolic systems but rather equip neural networks with inductive biases and then train them to perform symbolic reasoning tasks. Since SNNs can exploit precise spike timing, some functions can be implemented with fewer resources by SNNs as compared to ANNs (Maass, 1997). In practice however, training of SNNs was problematic as the standard training algorithms for ANNs — Backprop for feed-forward networks and Backprop-through-time (BPTT) for recurrent networks — were not applicable due to the non-differentiability of the spiking output of the neurons. Recently, however, several solutions to this problem have been proposed (Bellec, Salaj, et al., 2018; Bellec et al., 2020)

and it turned out that SNNs can also compete with ANNs in practice. Therefore, there is no longer a reason to resort to ANNs when studying neuro-symbolic computation. Here, we will consider a training algorithm for RSNNs called e-prop (Bellec et al., 2020). This algorithm does not only produce competitive results for RSNNs, it is also biologically plausible and can — as opposed to BPTT — easily be implemented in neuromorphic hardware.

In the following section, we will briefly discuss the basic principles of spike-based computations in the brain and introduce spiking neural network models. We will then touch on the main theories of symbolic computation in the brain. In the main part of this chapter, we will give an overview of the state-of-the-art of neuro-symbolic computation with RSNNs. We will show in particular, through a few examples, that biologically realistic RSNNs can perform symbolic tasks on bit strings and numbers. These results show that tasks in which symbols are manipulated can be learned by models of spiking neural networks and even trained using the biologically plausible learning rule e-prop (Bellec et al., 2020). These results are not only of interest from a biological perspective to understand the mechanisms used by our brain, but also as a computing paradigm for energy-efficient neuromorphic hardware. In that light, our results can be seen as creating a bridge from symbolic artificial intelligence to energy-efficient implementation in spike-based neuromorphic hardware.

3.2. Neural networks of the brain and spiking neural networks

Neurons are the basic computational units of the brain. They are cells specialized for information processing and information transmission via brief (approximately 1 ms) stereotyped electrical pulses, so-called action potentials or spikes. A spike — a short, sudden increase in voltage — has an amplitude of about 100 mV and lasts $1 - 2$ ms (Gerstner & Kistler, 2002). Since the spikes of a neuron are all alike, a spike itself (the form of the pulse) does not convey much information. Instead, information can be carried by the number of spikes in a certain time window, the precise timing of

a spike in relation to the spikes of other neurons, and the temporal spike pattern. As the shape of the action potential is considered irrelevant, one can abstract over it and define the output of a neuron as a sequence of time points $t^{(1)}, t^{(2)}, t^{(3)}, \ldots$ where $t^{(i)}$ is the i-th spike of the neuron. Such a sequence of spike times from one neuron is also called a spike train.

The brain consists of large networks of such neurons, connected via synapses. Incoming spikes to a neuron initiate changes in the membrane potential of the cell body (soma), and occasionally, if a certain voltage threshold is exceeded, the neuron generates a spike that is transmitted via synapses to other neurons.

This process is formalized mathematically in spiking neuron models. These models come at various granularities and abstraction levels, with the common characteristic that their outputs are spike trains.

3.2.1. The leaky integrate and fire (LIF) neuron model

One of the simplest spiking neuron models is the leaky integrate and fire (LIF) neuron model, see Figure 3.1 (left).

It captures the essential temporal dynamics of biological neurons, while abstracting over many — potentially important — aspects of its biological counterparts. The dynamics of the membrane potential V_j of a LIF neuron j is described by a first-order linear differential equation:

$$\tau_m \dot{V}_j(t) \quad = \quad -V_j(t) + R_m I_j(t), \tag{3.1}$$

where τ_m is the membrane time constant (on the order of 10 to 30 ms), R_m is the resistance of the cell membrane, and I_j the input current. This equation describes a leaky integrator. The current I_j is integrated over time, however, the integration is leaky, i.e., if there is no input current, the membrane potential decays to zero with time constant τ_m. The neuron outputs a spike at the time when its membrane potential V rises above the threshold v_{th}. After a spike is generated, the neuron dynamics is reset: the threshold value v_{th} is subtracted from the neuron's membrane potential, and the neuron enters a refractory period. During the refractory period, the neuron cannot spike.

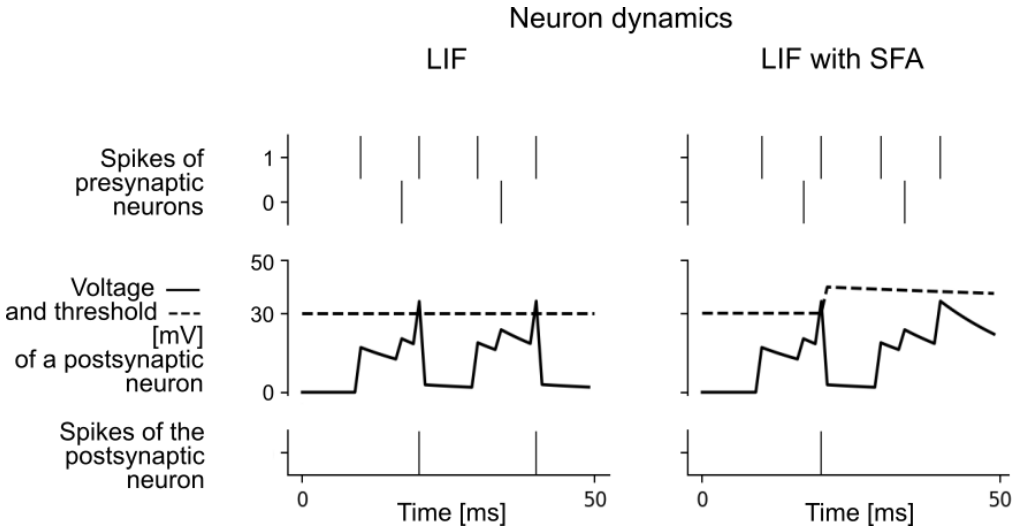


Figure 3.1.: **Neuron dynamics for LIF neuron without (left) and with SFA mechanism (right).** Top to bottom: The same spike trains of two presynaptic neurons that were connected to a postsynaptic neuron, voltage traces (solid line) and threshold (dashed line) for the postsynaptic neuron without (left), and with spike frequency adaptation (SFA) mechanism (right), output spikes of the postsynaptic neuron. Due to the SFA, the postsynaptic neuron in the right panel outputs only one spike. Neuron parameters used: $\tau_m = 20$ ms, threshold $v_{\text{th}} = 30$ mV, $\beta = 1.7$ mV, $\tau_a = 100$ ms.

We consider networks of such neurons where network neurons $1, \ldots, N$ are recurrently connected. We term these networks recurrent spiking neural networks (RSNNs). RSNNs reflect two important biological facts: first, that neurons in the brain are heavily recurrently connected (in contrast to the often used feed-forward architectures in artificial neural networks), and second, that these neurons communicate with spikes. The input current $I_j(t)$ to neuron j in the network is defined as the weighted sum of spikes from external inputs x and other neurons in the network z:

$$I_j(t) = \sum_i W_{ji}^{\text{in}} x_i(t - d_{ji}^{\text{in}}) + \sum_i W_{ji}^{\text{rec}} z_i(t - d_{ji}^{\text{rec}}), \tag{3.2}$$

where W_{ji}^{in} and W_{ji}^{rec} denote the input and the recurrent synaptic weights respectively, and d_{ji}^{in} and d_{ji}^{rec} the corresponding synaptic delays. Here, the weights model the efficacies of biological synapses and are similar to the connection weights in standard artificial neural networks. The delay d_{ji} models the time that passes between an action potential of the presynaptic neuron i and the voltage response at the soma of the postsynaptic neuron j.

Neurons are usually simulated in discrete time. Writing eq. (3.1) in discrete time with time-step δt and incorporating the reset after a spike, we obtain

$$V_j(t + \delta t) = \alpha V_j(t) + (1 - \alpha) R_m I_j(t) - v_{\text{th}} z_j(t) \delta t, \tag{3.3}$$

where $\alpha = \exp(-\frac{\delta t}{\tau_m})$ and $z_j(t) \in \{0, \frac{1}{\delta t}\}$ represents the spike output of neuron j. The term $-v_{\text{th}} z_j(t) \delta t$ implements the voltage reset after each output spike. The output z_j of the neuron is obtained by comparison with a voltage threshold v_{th}

$$z_j(t) = H\left(V_j(t) - v_{\text{th}}\right) \frac{1}{\delta t}, \tag{3.4}$$

with $H(x) = 0$ if $x < 0$ and 1 otherwise being the Heaviside step function.

3.2.2. Enhancing temporal computing capabilities of RSNNs with Spike Frequency Adaptation (SFA)

Upon inspection of the membrane potential dynamics of a LIF neuron, eq. (3.1), we observe that spiking neurons have some internal memory about previous inputs on the time scale of the membrane time constant τ_m, which is on the order of 10's of milliseconds. This is a difference from standard artificial neurons in deep learning where the output is an instantaneous function of the momentary input. A notable exception are long short-term memory (LSTM) units (Hochreiter & Schmidhuber, 1997), which turned out to be very powerful units in artificial recurrent neural networks. There, so-called memory cells can store their internal state arbitrarily long, which turned out to boost the memory capabilities of recurrent ANNs. The membrane time constant of spiking neurons is however too short to account for short-term memory at behaviorally relevant time scales of seconds in biological organisms. Instead, it has been shown that biological mechanisms at longer time scales can be utilized to boost temporal computing capabilities of RSNNs (Bellec, Salaj, et al., 2018; Salaj et al., 2021). One such mechanism is spike frequency adaptation (SFA), which can easily be incorporated into the standard LIF neuron model.

In contrast to LIF neurons, the threshold of a LIF neuron with SFA is dynamic, see Figure 3.1 (right). After each spike that the neuron produces, its firing threshold is increased by a fixed amount, and then it decays exponentially back to zero. The effective threshold $A_j(t)$ of a LIF neuron j with SFA is given by the equations

$$\begin{aligned} A_j(t) &= v_{\text{th}} + \beta a_j(t), \\ a_j(t+1) &= \rho_j a_j(t) + (1-\rho_j) z_j(t), \end{aligned} \tag{3.5}$$

where v_{th} is the constant baseline of the firing threshold and $\beta > 0$ scales the amplitude of the activity-dependent component $a(t)$. The parameter $\rho_j = \exp\left(-\frac{\delta t}{\tau_{a,j}}\right)$ controls the speed by which $a_j(t)$ decays back to 0, with $\tau_{a,j}$ as the adaptation time constant of neuron j. Experimental data show that many neurons in the brain exhibit SFA with time constants $\tau_{a,j}$ from hundreds of milliseconds up to tens of seconds (Pozzorini et al., 2015; Pozzorini et al., 2013).

3.2.3. Training RSNNs with e-prop

Recurrently connected neural networks are trained on temporal sequences of input-output pairs via Backprop-through-time (BPTT) – an algorithm that unfolds the recurrent network in time and operates in two phases. In the first phase, a forward pass is performed, that is, the network outputs are computed, and a cost function that measures the distance between the network's outputs and desired values is evaluated. In the second phase, by applying the chain rule of calculus, the derivatives of the cost function with respect to the parameters, i.e., synaptic weights, are evaluated for each time step, and the synaptic weights are updated. This approach requires the full history of the network activity to be stored, hence it is quite unlikely that the brain performs such an algorithm for learning. An alternative to the BPTT algorithm that is more biologically realistic is the e-prop learning rule (Bellec et al., 2020). Unlike BPTT, for updating the synaptic weights of the network this learning rule requires only information locally available to each synapse and neuron – an eligibility trace and a learning signal broadcast across the network. Eligibility traces model preceding activity in neurons and synapses at the molecular level, whereas learning signals model top-down signals (e.g., dopamine, acetylcholine, and neural firing related to the error-related negativity) that target different populations of neurons and govern the synaptic plasticity, i.e., the process of adjusting the strengths of synaptic weights.

More precisely, consider a cost function E that should be minimized in the learning process. The gradient $\frac{dE}{dw_{ji}}$ of the cost function with respect to the synaptic weight w_{ji}, needed for the update of this weight can be calculated as follows:

$$\frac{dE}{dw_{ji}} = \sum_t L_j(t) e_{ji}(t), \tag{3.6}$$

with $L_j(t)$ representing the learning signal for neuron j, and $e_{ji}(t)$ the eligibility trace for the synapse from neuron i to neuron j. In e-prop, one uses an approximate learning signal

$$L_j(t) = \sum_k B_{jk}(y_k(t) - y_k^*(t)) \tag{3.7}$$

which captures errors occurring at the output neurons k at the current time step t, that is, deviations of the outputs $y_k(t)$ from the targets $y_k^*(t)$ are calculated, and fed back from output neuron k to every network neuron j via randomly chosen weights B_{jk}. The eligibility trace $e_{ji}(t)$ represents a transient memory for the synapse connecting the presynaptic neuron i and postsynaptic neuron j, that is, it integrates the history of the synapse up to time t based on locally available information. For a LIF neuron, it is based on a low-pass filtered version of the presynaptic spike train and the postsynaptic membrane potential.

The learning signal $L_j(t)$ only takes into account the current time step t, however, the eligibility trace $e_{ji}(t)$ accounts for the past of the neuron. Since they are readily available in every time step t of the forward phase, as soon as the forward phase is completed, the updates of synaptic weights can be performed, without performing the backward phase as required in BPTT. There is no need for unrolling the recurrent network in time for calculating the errors from previous time steps as in BPTT, and as such, e-prop is an online learning method for training models of recurrently connected neurons. For an illustrative, short summary of the e-prop learning rule, see (Manneschi & Vasilaki, 2020).

3.3. Symbolic computation and arithmetics in the brain

3.3.1. Representation of numbers in the brain

There exists experimental evidence for at least two types of neural codes for numbers in the cortex. It has been shown that individual neurons in the lateral intraparietal area (LIP) exhibit a Gaussian tuning for cardinality when macaque monkeys perform a number matching task (Nieder & Miller, 2003). A population of such neurons, each tuned to a different represented value, can collectively encode a number. To make this more precise, we describe here a simple mathematical model for this coding scheme which

will also be used in later sections in our simulations. For a represented number z, each neuron i in such a population spikes with a firing rate of

$$f_i = f_{\max} \exp\Big(- \frac{(m_i - z)^2}{2\sigma^2} \Big), \tag{3.8}$$

where $f_{\max}$ is the maximum firing rate of neurons in the population, m_i is the value to which the neuron is tuned to, and σ is the standard deviation of the Gaussian tuning curve which is chosen here to be equal for all neurons in the population for simplicity. The preferred values m_i for these neurons are typically chosen equally spaced in the range of numbers that can be represented. Hence, each neuron responds with its maximal firing rate $f_{\max}$ if its preferred value m_i is represented. When the represented value is off this preferred value, the firing rate decreases according to a Gaussian profile. Such population coding of numeric values between -1 and 1 is illustrated in Fig. 3.2 for a population of 50 neurons. This coding scheme abstracts over input stimulus properties, as it is preserved over input modalities such as visual or auditory input (Nieder, 2012).

In other tasks, some neurons in LIP fire more strongly for larger numbers and others show the inverse behavior, hence exhibiting a spike frequency code (Roitman et al., 2007). It has been hypothesized that these two types of codes may separate the actual entity represented (i.e., the number represented with a Gaussian tuning) from the structure of the number space (a linear firing rate code which captures similarity between numbers) (Behrens et al., 2018; Summerfield et al., 2020). The authors also state that structure-disentangled codes are not unique to numbers and magnitude, rather this might be a general coding principle for structured information (e.g., space). There exists evidence for similar coding principles in humans. EEG activity varies smoothly with numerical distance when Arabic numbers are presented to humans (Spitzer et al., 2017; Teichmann et al., 2018).

3.3.2. Circuits for arithmetic in the brain

Little is known about the specific circuit mechanisms that implement arithmetic operations in the brain. Several cortical regions are activated during arithmetic tasks. Here, the strength of activation of specific areas often

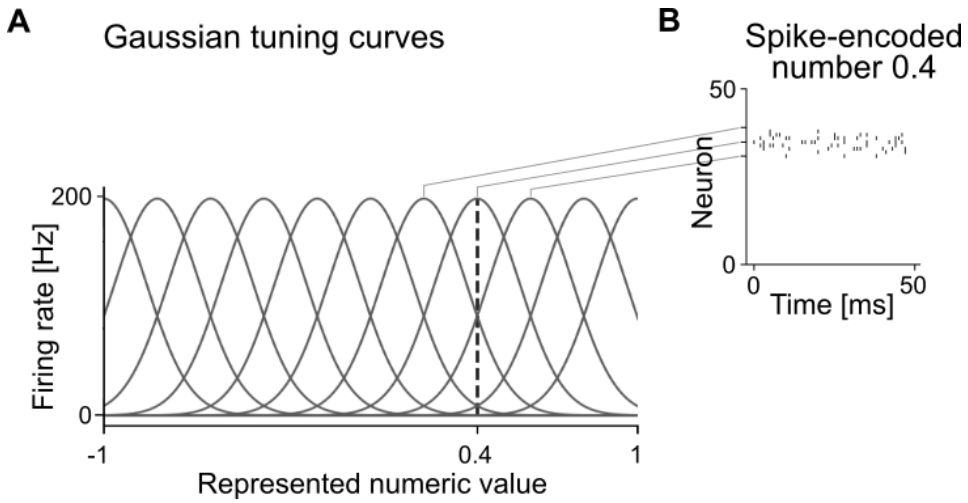


Figure 3.2.: **Population coding of numbers.** This encoding was used in the middle panel of Figure 3.4C to encode numbers. (A) The numeric value from $[-1, 1]$ that is represented by a population of 50 neurons is indicated on the x-axis. The y-axis indicates the firing rate of 11 neurons out of this population in response to the corresponding number. Each neuron responds with its maximal firing rate to its preferred value (at the peak of its Gaussian tuning). When the represented value is off this preferred value, the firing rate decreases according to the indicated Gaussian profile. The dashed vertical line indicates one example value of 0.4. The neuron with its tuning curve centered on this value fires at 200 Hz. The two neurons with the shown neighboring tuning curves fire at approximately 80 Hz. (B) Spiking activity of 50 spiking neurons of this population when the number 0.4 was encoded, as indicated in panel A. Each row corresponds to one neuron of the population, where neurons are sorted by their preferred value from -1 to 1. Each tick indicates one spike. Gray lines connect the centers of tuning curves in A with the corresponding neurons in panel B.

depends on the type of task (e.g. approximate versus exact), task difficulty (e.g. $2 + 3$ versus $27 * 32$), and expert level of the subject (Zamarian et al., 2009). In general, frontal areas are more strongly involved in more complex calculations (Semenza et al., 1997). Interestingly, several lines of research indicate that growing expertise leads to the increased involvement of specific parietal areas whereas frontal areas become less strongly involved (Zamarian et al., 2009). This supports the theory that arithmetic operations can be performed by more general-purpose processing, but practice may train special purpose circuits that can perform the trained tasks with less effort. Our novel results below show that training of complex arithmetic operations can be performed on surprisingly small SNNs.

3.3.3. Symbolic computation

A symbol has meaning and significance. It refers to ideas, objects, relationships and concepts (Gazzaniga et al., 2009). There are a few important aspects to consider when studying the emergence of meaning from symbols, for example, the establishment of links between symbols and objects/concepts that they represent, learning symbolic meaning from context, or generalization over a range of instances of symbols. Also, different brain regions are involved in general meaning processing, but semantic integration mechanisms draw on higher cortical areas of the neocortex, such as the prefrontal, posterior parietal, anterior, inferior, and posterior temporal cortex (Pulvermüller, 2013).

Unlike computations performed directly on values such as the activities of sensory receptors, symbolic computation addresses the question of the manipulation of symbols using rules. A symbolic expression can be expressed using variables that are instantiated by concrete values. In algebra, abstract variables x or y can be substituted by numbers; in language, terms *subject* or *verb* can be replaced by words. The abstract terms act as placeholders, and they are a higher-order representation of knowledge. Through the concept of variable binding — how the placeholders are instantiated by concrete values — one can study how computations performed by neurons give rise to cognitive processes, and eventually elucidate the underlying process in

the brain which contribute to language and abstract thought (Marcus et al., 2014).

Symbols are used to represent categories (e.g., *cat*, *person*) and individuals (e.g., *Garfield*, *James*), variables (e.g., x, y) and computational operations (e.g., '$+$', '$-$', *concatenate*, *compare*) (Marcus, 2003). The mind is a system that represents variables and abstract relationships between variables. It performs operations over variables, uses structured representations, and is capable of distinguishing between categories and individuals. A possible physical implementation of such a system would be a set of buckets, with the bucket's content representing an instantiation of a category represented by the bucket. For example, in digital computers, the utilization of binary registers which can be either empty or full (bits 0 or 1), and the operations over the registers that are performed in parallel reflects the analogy of the buckets. Next, each symbol or variable should correspond to an anatomically defined region or register. For example, the symbol *"black"* in *"black cat"* or *"black Porsche"* should be employed equally, irrespective of the categories of animals and cars that may be represented in widely separated regions of the cortex (Kraisnikovic, 2018; Marcus et al., 2014).

Several neural network models have been proposed for the implementation of symbolic computation via variables and variable-binding mechanisms that bind specific content to a given variable. Pointer-based models (e.g., (Zylberberg et al., 2013)) assume that pointers are implemented by single neurons or co-active populations of neurons that are synaptically linked to content. Another class of models is based on the idea of indirect addressing. These models assume that a neural space for a variable encodes an address to another brain region where the corresponding content is represented (Kriete et al., 2013; O'Reilly, 2006). In anatomical binding models, different brain areas are regarded as distinct placeholders (Hayworth, 2012; Hayworth & Marblestone, 2018). In Dynamically Partionable Autoassociative Neural Networks (DPANN) (Hayworth, 2012), a given symbol corresponds to a global attractor state of the dynamics of a large-scale network that links many regions of the brain. Information flow between particular sets of registers is achieved by turning on and off subsets of synapses, i.e., gating. In neural blackboard architectures, one assumes that besides assemblies for content, there also exist assemblies for structural entities such as phrases, semantic roles, et cetera. Specialized circuits (so-called gating circuits and

memory circuits) are then used to bind content to structure (Van der Velde & De Kamps, 2006). This way, complex syntactic structures can be built. Vector-symbolic architectures (Plate, 1995) use high-dimensional vectors to represent the symbolic concepts encoded by activity patterns of a group of neurons and a defined set of mathematical operations. Both the variables and potential role filters, such as subject, object or verb, are represented as vectors. Compositional operations typical for symbol manipulation are then implemented through a mathematical transformation of these vectors. Activity patterns for binding, for example, a subject and an instance, are achieved through multiplication of two vectors (Eliasmith et al., 2012). Most of these models rely on specifically constructed circuitry that performs the binding operations. In contrast, a rather generic SNN model was proposed in (Müller et al., 2020). In this assembly pointer model, binding emerges through local Hebbian plasticity processes from the activation of assemblies in areas that contain variables to areas that contain content.

3.4. Prior work on symbolic computations on bit strings and numbers with SNNs

Some of the above mentioned neural network models for variable binding have been implemented in SNNs (e.g., (Eliasmith et al., 2012; Müller et al., 2020)). In this review, we concentrate more specifically on symbolic computations on numbers, and in contrast to the above mentioned models, we propose a more generic approach where SNNs are trained to perform specific tasks.

In a recent work (Salaj et al., 2021), a spiking neural network was trained on two tasks based on a rule that was given at the beginning of each trial. For the same input sequence, the task of the network was either to reproduce the sequence in the same order, if the rule was *duplicate*, or the sequence in reversed order if the rule was *reverse*. The SNN had to learn to observe data stimuli (symbol sequences) given as input, and two different rules on how to manipulate them, also given as an input, independent of each other, then to use both data stimuli and the given rule to produce an output. After training, the SNN was able to produce sequences with all correct symbols in

95.88% of test trials. Through an analysis of spiking activity on the test trials, groups of neurons that had specialized in encoding structural information of the task, such as symbol identity, position, and task identity, were identified. The analysis revealed that many neurons specialized in encoding non-linear combinations of conditions, indicating a non-linear mixed selectivity of neurons. In other words, while some neurons showed some preference in their tuning to specific aspects necessary to solve the tasks, there was no clear-cut encoding of symbols by individual neurons in this trained circuit — different from what one would usually do to construct a neural network to perform these operations. Such mixed selectivity of neurons has also been observed in higher cortical areas (Rigotti et al., 2013).

In another experiment of (Salaj et al., 2021), it was shown that SNNs can learn to follow dynamic rules, i.e., make decisions, for example, choose the "Left" or "Right" button, depending on the currently active rule and specific subsequences of symbols relevant for that particular rule. In a long sequence of symbols, the currently active rule, relevant and distractor symbols appeared sequentially. Even for humans, the task is quite demanding, since it requires not only the maintenance and an occasional update of the active rule, but also maintaining relevant symbols and ignoring irrelevant ones until the target subsequence is complete. Such a task requires a hierarchical working memory – higher-level memory for maintenance of the active rule, and lower-level memory for maintenance of the symbols relevant in the context of the active rule, especially because the distractor symbols occur rather frequently. A generic SNN was able to solve this task almost perfectly without requiring a special-purpose architecture. In 97.79% of trials consisting of sequences of 90 symbols, all choices of the button "Left" or "Right" were correct. This result shows that cognitively demanding sequence processing tasks can be solved by generic SNNs.

3.5. New results on spike-based symbolic computation on bit sequences and numbers

In this section, we present new results on how SNNs can perform symbolic computations on bit sequences and numbers. Human infants can perform

mental arithmetic using nonsymbolic, approximate number representations. Even before children learn algorithms for manipulating numerical symbols and formal mathematics, they are able to perform approximate addition and subtraction of large numbers. In such experiments, children performed well above chance – they were able to do so by comparing quantities instead of providing an exact solution (C. Gilmore et al., 2007). Certainly, we use non-symbolic representations before we master symbolic arithmetic, which takes years (C. K. Gilmore et al., 2010). Some authors refer to this capability as "the number sense" (Dehaene, 2011).

There are many open questions: How do humans perform arithmetic? How are the numbers encoded? What are the mental representations behind the comparison of two numbers or quantities? For the case of symbolic arithmetic, we face even more general questions – those of symbol manipulation and sequence processing.

Mastering symbolic representations and symbol processing, in general, helps us perform cognitively demanding tasks. Certainly, the human brain is capable of processing and manipulating symbols, or even sequences of symbols, thereby following the abstract relationships between mental concepts (symbols). These abstract relationships are known as rules, and the brain can process sequences of symbols even when rules dynamically change or when the stimulus is novel (Marcus, 2003).

Given a sequence of items, humans are able to infer the rules that generated the sequence and apply the same rule even on sequences of items that they had never encountered before (Y. Liu et al., 2019). If we revisit the question of numbers, for example, Arabic numbers and their representation, we notice that single-digit numbers are represented by symbols 0 to 9. Multi-digit numbers are represented by sequences of symbols, where the position in the sequence is of particular relevance. Moreover, we learned the meaning of the most significant and least significant digit, and that the length of the sequence representing the number carries another crucial piece of information. To compare two numbers given in a symbolic form, we can first rely on the rule of comparing the lengths of sequences. In case the lengths are the same, we immediately proceed by applying the second rule – comparing the corresponding significant digits, iteratively if necessary, starting from the most significant digits toward the least significant ones,

but we stop when we can unambiguously conclude which number is greater, or after we compared the least significant digit. In fact, we can perform such sophisticated sequence processing during the comparison of two symbolic numbers with little effort. We have mastered this processing during years of practice through schooling, and we are applying the learned rules rather automatically. This subjective observation is backed up by experiments that suggest that specialized cortical circuits learn complex arithmetic operations during training (Zamarian et al., 2009), see Section 3.3.2.

We performed two experiments – both of them show that rather small, generic SNNs can be trained to perform demanding sequence processing tasks. In the first experiment, we considered the number comparison task described above, a task that has often been used to study number processing in cognitive science (Sheahan et al., 2021). For simplicity, we used binary numbers instead of previously described Arabic numbers, since the same sequence processing rules for comparison also apply when comparing numbers in any base. Note that, the rules describing how to compare the numbers were not explicitly given to the network, rather, the network learned to perform the task through training.

In the second experiment, an SNN was trained to perform arithmetic operations (addition, subtraction, and multiplication) on real numbers encoded into spiking activity using population coding with Gaussian tuning, a number encoding that has been observed in monkeys, see Section 3.3.1. Arithmetic expressions — the instructions indicating the rule to be applied on the data stream, i.e., real numbers — were given to the network in a separate stream, and sequentially. The challenge of this task was to incorporate pieces of information processed earlier, that is, to reuse the result of the computation from the previous step while following new instructions. Without explicitly being trained to do so, the network had also to solve the binding problem – between variables that appeared in the expression and the neurons encoding the number. The network learned successfully to process a sequence of arithmetic expressions, and similar to humans, to perform at a level of expertise.

3.5.1. Spike-based symbolic computation on bit strings

The comparison of two numbers is a task often considered in cognitive science (see e.g., (Sheahan et al., 2021) for a recent example). We wondered if an SNN could learn to perform the comparison task on expressions with numbers encoded as bit strings. We refer now to the numbers represented in a binary base as bit strings rather than sequences, since we use the term sequence to denote the (temporal) processing dimension and that symbols appear sequentially, one after the other.

The schematic network architecture is shown in Fig. 3.3A. The input to the network were binary strings of width 10 bits representing numbers in the binary base. Another input stream represented the relational operator to be applied, where we used *equals* '=', *less than* '<', and *greater than* '>'. Variables x_1, x_2, and the relational operator were presented to the network sequentially as Poisson spike trains. A Poisson spike train is a random sequence of spikes with a given rate (i.e., expected number of spikes per second), similar to spike trains observed in the brain. Mathematically, such a spike train is generated by a Poisson point process. Each presentation of a variable or relational operator lasted for 200 ms, with high firing activity coding for 1 and low firing activity coding for 0, see Fig. 3.3C, top. Only after the full sequence indicating the first number x_1, the second number x_2, and the relational operator *rel_op* has been shown, does the network have all the pieces of information required to determine whether the provided relation x_1 *rel_op* x_2 was correct. As the answer, the network was required to output "Yes" or "No".

We trained a recurrently connected spiking neural network consisting of 200 LIF neurons (100 LIF neurons with SFA, 100 without) on this task using the biologically plausible learning algorithm e-prop (Bellec et al., 2020), for which the accuracy over training iteration is shown in Fig. 3.3B. The curve was smoothed using bins of 20 values. Fig. 3.3C illustrates a trial where the network had to answer whether the binary number $x_1 = 1011111111$ was smaller than binary number $x_2 = 1011010100$, for which the final answer is "No". Tested on previously unseen instances x_1 *rel_op* x_2, that is, combinations of bit strings and relational operators that were not used during training, the proportion of correct answers (accuracy) achieved was

0.8723 ± 0.0235 (mean $\pm$ standard deviation), averaged over 5 different network initializations.

This result demonstrates that rather small SNNs can learn to perform arithmetic tasks using a biologically plausible learning algorithm. In this task, numbers were represented in a symbolic manner (i.e., using symbol strings as in Arabic numbers) rather than as magnitudes, and the network had no prior knowledge on the structure of this number encoding or on the structure of the task. Nevertheless, the network was able to generalize to novel task instances, indicating that explicitly designed symbolic computations are not necessary for generalization for this task.

3.5.2. Evaluation of nested arithmetic expressions

One of the most basic arithmetic tasks is the evaluation of an operation applied to two numbers such as $12 + 4$. One can replace numbers in such expressions with variables to obtain an expression such as $x_1 + x_2$. The solution to the expression can be obtained when the values of the variables are defined. For example, defining $x_1 = 12$ and $x_2 = 4$, the result for the above expression is 16. This calculation demands a basic type of computation where values need to be assigned to variables which then have to be combined by some arithmetic operation. We wondered whether such basic symbolic computations on numbers and variables could be learned by SNNs. However, we took a further step by considering long nested expressions built based on this basic template with three basic arithmetic operators addition '$+$', subtraction '$-$', and multiplication '$*$'. An example of such a complex expression is

$$(((((((((\underbrace{\underbrace{(x_2 * x_4)}_{=r_1} * x_5) * x_3) - x_3) + x_5) * x_4) - x_5) * x_4) * x_1) - x_4).$$
$$=r_2$$
$$\cdots$$

The expressions we consider have a nested structure, where the result r_1 of the evaluation of the expression in the innermost nesting level, here, $(x_2 * x_4)$ is used to evaluate the result r_2 for the next nesting level, and so on, until the final result is obtained. We consider nested expressions of 10

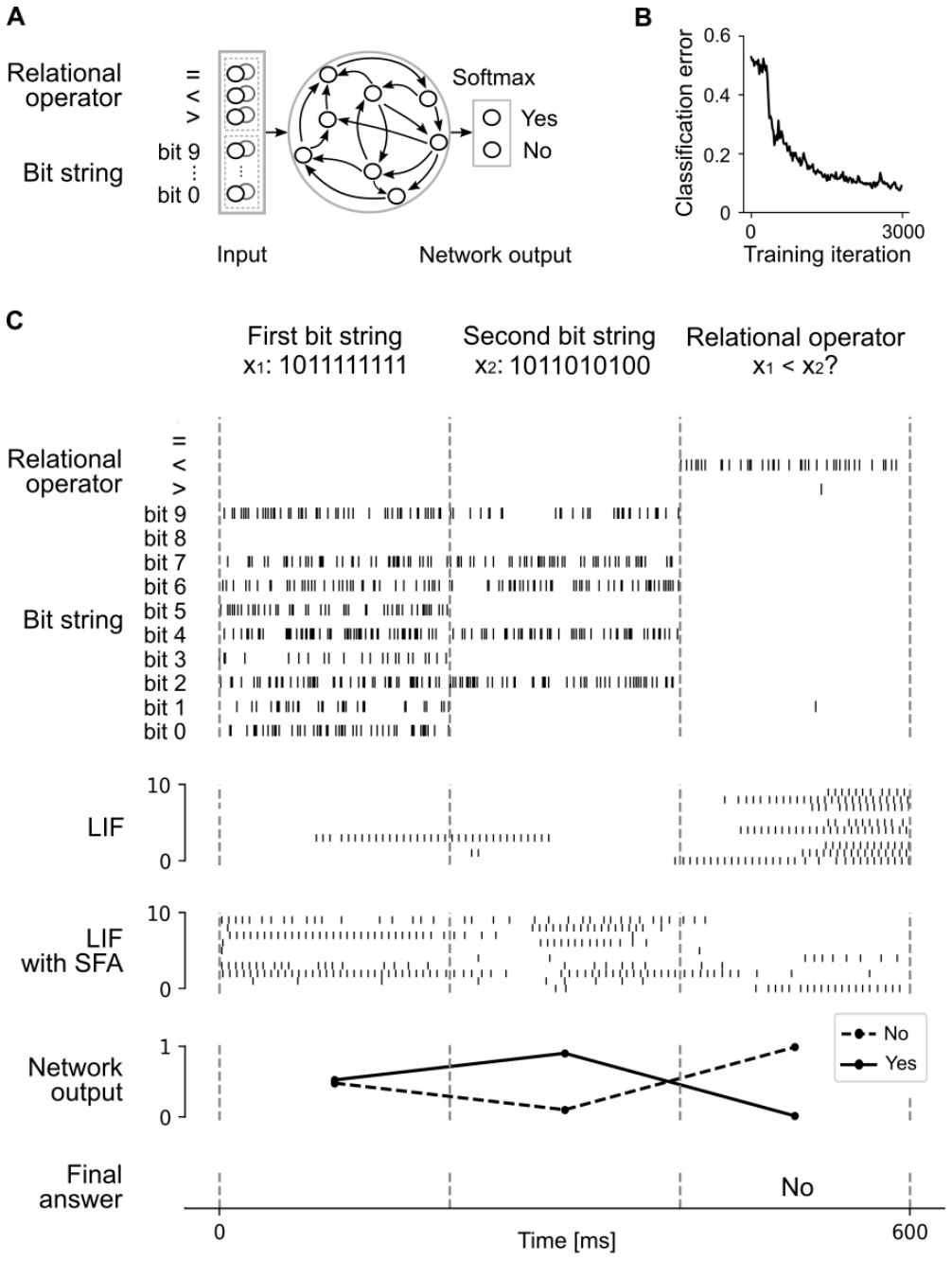


Figure 3.3.: (Caption on the next page.)

Figure 3.3.: **Comparison of two bit strings.** (A) The schematic network architecture: input spiking neurons encoding relational operators and bit strings of width 10 (bit 9 the most significant, and bit 0 the least significant bit), recurrently connected LIF neurons, and two output neurons. The network output is the softmax of the output neurons producing "Yes/No" answers. (B) Training curve: Accuracy over training iteration. Test accuracy for novel trials was 0.8723. (C) Spiking activity of the network for a trial where the network compared if the first bit string $x_1 = 1011111111$ was smaller than the second bit string $x_2 = 1011010100$. Top to bottom: Spiking activity of neurons encoding the relational operator and bit strings (one neuron shown out of 2 for each symbol), 10 sample out of 100 LIF neurons, 10 sample out of 100 LIF neurons with SFA, network output – the softmax applied to readout neurons dedicated to producing the answer, the final (maximum value of those for "Yes" and "No" neurons) and correct answer "No" to the question "$x_1 < x_2$?".

operators combining 5 variables, leading to 9 intermediate results $r_1, \ldots, r_9$ and the final result r_{10} of the evaluation.

More precisely, we define the task as the sequential evaluation of expressions in 10 steps $i = 1, \ldots, 10$ as follows:

$$\begin{aligned}
i = 1: &\quad r_1 = (Operand_1 \;\; operator_1 \;\; Operand_2),\\
i = 2: &\quad r_2 = (r_1 \;\; operator_2 \;\; Operand_3),\\
i = 3: &\quad r_3 = (r_2 \;\; operator_3 \;\; Operand_4),\\
&\quad \vdots\\
i = 10: &\quad r_{10} = (r_9 \;\; operator_{10} \;\; Operand_{11}),
\end{aligned}$$

where in the first step ($i = 1$), $Operand_1$ and $Operand_2$ are drawn uniformly over $\{x_1, \ldots, x_5\}$ without replacement, and $operator_1$ was drawn uniformly over $\{+, *\}$ (the operator '$-$' was omitted in this step to avoid the need of encoding the order of $Operand_1$ and $Operand_2$). For the following steps, each of $operator_2, \ldots, operator_{10}$ was drawn uniformly from $\{+, -, *\}$, and each of $Operand_3, \ldots, Operand_{11}$ uniformly over $\{x_1, \ldots, x_5\}$. This describes the generation procedure for the symbolic part of the task, but numeric values are also required in order to evaluate such nested expressions. We allowed that the values of variables can change after each step i. This was done by drawing in each step i for each of the variables $x_1, \ldots, x_5$ a random value

Step i	**Arith. expression**	**Numeric expression**	**Output r_i**	**Target**	**Abs. error**
1.	x_2 * x_4	$0.4966 * (-0.1113)$	-0.0794	-0.0553	0.0241
2.	r_1 * x_5	$(-0.0794) * (-0.7100)$	0.0510	0.0564	0.0054
3.	r_2 * x_3	$0.0510 * 0.8209$	0.0629	0.0419	0.0210
4.	r_3 - x_3	$0.0629 - 0.3314$	-0.2308	-0.2685	0.0377
5.	r_4 + x_5	$(-0.2308) + 0.4717$	0.2096	0.2409	0.0313
6.	r_5 * x_4	$0.2096 * 0.7014$	0.1304	0.1470	0.0166
7.	r_6 - x_5	$0.1304 - 0.6650$	-0.5417	-0.5346	0.0071
8.	r_7 * x_4	$(-0.5417) * (-0.0165)$	-0.0222	0.0089	0.0311
9.	r_8 * x_1	$(-0.0222) * (-0.8182)$	-0.0128	0.0182	0.0310
10.	r_9 - x_4	$(-0.0128) - (-0.4727)$	0.4739	0.4599	0.0140

Table 3.1.: **Evaluation of nested arithmetic expressions with an SNN.** One example input sequence with network outputs, targets, and errors. In each input step i — each step lasting for 50 ms — an arithmetic expression and numeric values were given to the network, for which the network produced the output r_i. Output values were very close to target values, that is, the absolute (abs.) error was close to zero.

from the uniform distribution over $[-1, 1]$. An example value assignment and target results for the expression given above are shown in Table 3.1.

We show here that an SNN can be trained to evaluate such complex arithmetic expressions. The nested expression was given sequentially, starting from the expression in the innermost nesting level ($x_2 * x_4$ in the above example), together with the numeric assignments to all operands $x_1, \ldots, x_5$. These inputs were encoded by spiking input populations (see below for the encoding) and presented to the network for 50 ms, see Fig. 3.4A for a schematic of the input populations. The network was demanded to produce at its readout the intermediate result r_1. Then, the expression for the next nesting level ($*x_5$ in the above example) was provided for 50 ms and the result r_2 was demanded as the output, and so on, see Fig. 3.4C for the temporal sequence of inputs to the network.

Consequently, the challenges for the network were the processing of symbols with associated numeric values, i.e., it had to learn to bind a concrete value

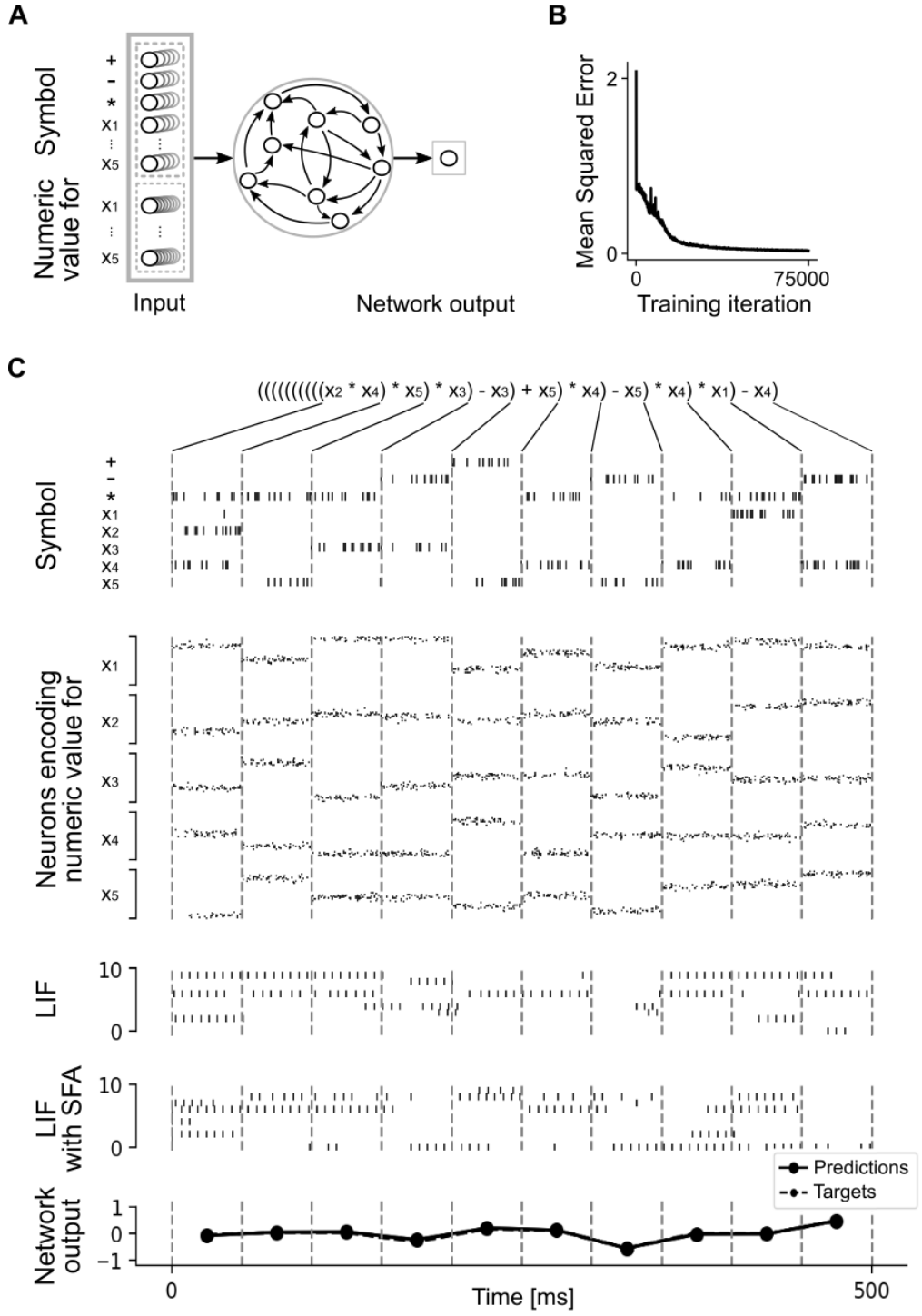


Figure 3.4.: (Caption on the next page.)

Figure 3.4.: **Spike-based arithmetic on numbers and variables.** (A) The schematic network architecture consisting of input spiking neurons that encoded symbols used in arithmetic expressions (operators and variables) and numeric values for variables, recurrently connected LIF neurons, and a linear readout neuron that produced the result for each arithmetic expression. (B) Training curve (Mean Squared Error over training iteration) for the network trained to evaluate nested mathematical expressions. The curve was smoothed using bins of 50 values. (C) Spiking activity of the network for the nested arithmetic expression shown at the top. Solid lines from the expression to dashed lines of the spike raster indicate the time when this subexpression was presented to the network. Spike rasters from top to bottom: Spiking activity of neurons encoding the symbols appearing in the arithmetic expression (one neuron shown out of 5 for each symbol), encoding of numeric values that instantiate variables, 10 sample out of 150 LIF neurons (without SFA), 10 sample out of 150 LIF neurons with SFA, real-valued network outputs (predictions) that almost perfectly match targets (and consequently, the dashed line representing targets is difficult to see).

to a variable, and at the same time, carry out the demanded mathematical operations in an online fashion. An example of computations performed by a trained SNN on the previously mentioned nested expression is illustrated in Table 3.1.

The schematic architecture of a spiking network that we trained is illustrated in Fig. 3.4A. Symbolic inputs to the network were encoded by subpopulations of neurons producing Poisson spike trains, dedicated to each symbol $\{+, -, *, x_1, \ldots, x_5\}$ (5 neurons per symbol) and indicating the presence (absence) of a symbol through a high (low) firing rate. Numeric values for each of variables $x_1, \ldots, x_5$ were encoded by another set of 5 subpopulations of neurons, each consisting of 50 neurons using a population coding scheme. See Fig. 3.2 for an illustration of population coding of a single number with a population of 50 neurons. Each neuron of this population was assigned a preferred value for which it exhibited its maximal firing rate of 200 Hz. When the represented number was off this preferred value, its rate decreased according to a Gaussian profile. The preferred values of the neurons in this population uniformly tiled the whole space of values between -1 and 1 (Fig. 3.2A). Hence, for a specific represented value, neurons with preferred tuning close to the value were active, while others remained mostly silent, see Fig. 3.2B.

As illustrated in Fig. 3.4A, the spiking activity of the input populations was projected to an RSNN. The output of the network was a single linear non-spiking neuron (a readout neuron), where the numeric value of its output (a real number) was interpreted as the result of the computation. We trained an RSNN consisting of 300 LIF neurons (150 with SFA and 150 without SFA) with all-to-all connectivity using e-prop for 75000 iterations on this task. The objective minimized was the Mean Squared Error (MSE) between the network outputs (predictions) and targets for all intermediate and the final result of the arithmetic expression. The evolution of the network error (MSE) over the training iterations is shown in Fig. 3.4B. The network achieved an MSE of 0.0341 and a mean absolute error (MAE) of 0.1307 in the mean over all intermediate results, i.e., all computational steps in all of 5000 test trials.

The column "Abs. error" (Absolute error) of the example illustrated in Table 3.1 confirms that the network learned the task very well. The spiking input (encoding of symbols, and of numeric values for variables) and network spiking activity of LIF neurons for the same trial is shown in Fig. 3.4C, where the bottom panel shows an almost perfect match of outputs (predictions) and targets.

We note that we made sure that the nested symbolic expressions used in the testing trials were different from any nested expression used during training. Also, the used numeric values for the operands were drawn randomly, which, altogether, indicates good generalization capabilities of the trained SNN. This result shows that nested arithmetic expressions can be learned by rather small SNNs to high precision with a biologically plausible learning algorithm. Again, no explicit mechanisms for variable binding were necessary.

3.6. Conclusions

Most studies of computing capabilities of RSNNs have focused on sensory processing tasks. But the brain uses RSNNs also for higher cognitive functions such as symbolic computations, in particular, mathematical computations. We have reviewed recent results, and added two new ones, which

show that RSNNs are very good at these tasks. Moreover, they are able to acquire these capabilities through a biologically plausible learning method – e-prop, and they are able to apply learned computing procedures to new instances of the task that never occurred during training. This is somewhat surprising, since RSNNs do not compute in a clocked mode, like digital computers or ANNs. Rather, they compute in an asynchronous mode where also the timing of spikes carries information. Also, the shown examples for such RSNNs operate in a rather low firing rate regime, where they could not simply emulate ANNs by encoding continuous values through firing rates. Thus we arrive here at a new paradigm for symbolic computation in neural networks that provides concrete hypotheses about the organization of symbolic computations in the brain. Furthermore, since RSNNs can be implemented more energy-efficiently in neuromorphic hardware than ANNs, the results that we have discussed also suggest new methods for improving the energy efficiency of computing hardware for symbolic computations.

3.7. Technical details to new results

In this section, we will describe the technical details of our experiments. First, we will describe the network architecture and neuron model that we used. Next, we will give details about the training method, and finally, describe the tasks that were learned by SNNs.

3.7.1. Network architecture and neuron model

We trained a recurrent spiking neural network of LIF neurons with all-to-all connections. Since in both experiments that we demonstrated temporal integration of pieces of information was crucial for a good performance, a fraction of LIF neurons (one half) was equipped with spike frequency adaptation (SFA). LIF neuron model was described in Section 3.2.1, and LIF neuron model with SFA in Section 3.2.2.

The firing threshold of LIF neurons was $v_{\text{th}} = 30$ mV. The same value of 30 mV was used as a baseline firing threshold v_{th} for LIF neurons with

SFA, with an adaptation strength of $\beta = 1.7$ mV (see eq. (3.5)). All LIF neurons had a synaptic delay of 1 ms, a refractory period of 5 ms, and a membrane time constant of $\tau_m = 20$ ms. The resistance of the cell membrane was $R_m = 1\,\mathrm{G}\Omega$. A population of input neurons encoded inputs to the recurrently connected neurons in the form of spike trains. The network output was produced by linear readout neurons, from the average activity of recurrent neurons in the network during the time window representing one step of a trial.

3.7.2. Training method e-prop

In all tasks, input, recurrent, and readout weights were trained simultaneously. To train the weights of the network by the learning rule random e-prop (Bellec et al., 2020), we used the Adam optimizer with a learning rate of 0.001. The input and recurrent weights were initialized with values from a standard normal distribution, and then scaled with a factor $\frac{1}{\sqrt{n_{\text{in}}}}$, and $\frac{1}{\sqrt{n_{\text{rec}}}}$, respectively, with n_{in} being the total number of input neurons, and n_{rec} the number of recurrent neurons in the network. The readout weights (weights from recurrent to output neurons) were initialized with values from a normal distribution with mean zero, and standard deviation 1.05. The unit for all input and recurrent weights was pAs.

3.7.3. Tasks

Comparison of bit strings. An SNN consisting of 200 recurrently connected neurons was trained for 3000 iterations, with a batch size of 100, to answer the "Yes/No" questions of the form "Is x_1 *Equal to / Smaller than / Greater than* x_2?". Inputs to the network were two binary strings x_1 and x_2 of width 10 bits, and in fact, represented binary encoded numbers. Binary strings x_1 and x_2 were given successively in the first and the second step of each trial, then followed by a relational operator $=, <$ or $>$, given in the third step. In the third, final step, the network had to produce a "Yes/No" answer to the question that got complete, after all three important pieces of information were shown to the network. Each step lasted 200 ms, hence the

total duration of the trial was 600 ms. 10-bit binary strings x_1 and x_2 were encoded into spike trains by a population of 20 spiking neurons – two neurons were dedicated to encoding each bit. Relational operators $\{=, <, >\}$ were encoded using 6 input spiking neurons (2 neurons per operator), in each trial exactly one operator was given in the third step. If a bit represented a value of 1, or if the operator was used in the expression, dedicated neurons fired with a high rate of 200 Hz. To encode a bit value of 0 or that the operator was not used in the expression, neurons fired with a low rate of 2 Hz. Such spike trains encoding the inputs were received by a population of recurrently connected LIF neurons – 100 LIF neurons with SFA and 100 without. Neurons with SFA had an adaptation time constant uniformly distributed from $[1, 600]$. The linear readout consisted of 2 neurons on which we applied softmax – one neuron was dedicated to "Yes" and one for "No" output. The final answer was determined from the maximum value after the softmax – the higher value indicated either "Yes" or "No".

The loss function minimized was a cross-entropy function with an additional regularization term. The regularization term represented the squared difference between the average firing rate of the recurrent neurons and a target firing rate of 20 Hz, scaled by a factor of 10. On average, in 87.23% of, in total, 10000 testing trials, the network produced the correct answer. The average was calculated over 5 different initializations of networks with the same architecture and trained in the same way (the weights and adaptation time constants of LIF neurons with SFA differed in each run). The inputs in each test trial were novel, i.e., not used during training.

Evaluation of nested arithmetic expressions. An SNN consisting of 300 recurrent neurons was trained for 75000 iterations, with a batch size of 50 to perform symbolic computation on nested arithmetic expressions involving numbers and variables. Symbolic input to the network was an arithmetic expression formed by the variables $x_1, \ldots, x_5$ and arithmetic operators $\{+, -, *\}$. In the first step of each trial of length 10, two different variables were drawn randomly to form a valid expression with two operands, and the possible operator was limited to either addition ($+$) or multiplication ($*$), to avoid the problem of encoding the order of operands appearing in the expression, which a noncommutative operator subtraction (-) would require.

In all other steps of a trial, a variable and an operator were drawn randomly over $\{x_1, \ldots, x_5\}$ and $\{+, -, *\}$ respectively – one variable was necessary since the result of the previous step was considered as the value that always instantiated the first operand in the subexpression. If a variable $x_1, \ldots, x_5$ or operator $\{+, -, *\}$ used in an arithmetic expression appeared in a step of the trial, they were encoded by the "high" spiking activity of dedicated 5 neurons (200 Hz), otherwise by their "low" activity (1 Hz). The numeric inputs to the network were analog values (numbers from $[-1, 1] \in \mathbb{R}$) and encoded by a subpopulation of 250 input spiking neurons. In each step, different numeric values that instantiated the variables $x_1, \ldots, x_5$ were given, but the network had to learn the binding ("variable - concrete number"), and also to ignore all unnecessary values if the variable did not appear in the current expression. No special modules that would help the binding of variables with numeric values were included. Each analog value used in arithmetic expressions was encoded using a population coding scheme with a population of 50 input neurons. Neurons in this population were tuned to numbers using Gaussian tuning curves where neuron i responded maximally with a firing rate of $f_{\max} = 200$ Hz if the represented number was equal to its preferred value m_i. The preferred values m_i for these 50 neurons were chosen equally spaced in the input domain $[-1, 1]$. The firing rate f_i of neuron i was given by

$$f_i = f_{\max} \exp\Big(- \frac{(m_i - z)^2}{2\sigma^2} \Big), \tag{3.9}$$

where z was the represented value and the standard deviation of $\sigma = 0.08$ was equal for all input neurons. Each step lasted 50 ms, and consequently, a duration of a trial was 500 ms. From 300 recurrently connected LIF neurons, half (150 neurons) had an SFA mechanism. The adaptation time constant for each of the neurons with SFA was chosen randomly from a uniform distribution $[1, 500]$. The output was a linear readout – in each step, output weights were multiplied by the average firing activity of the recurrently connected neurons. This output per step was a single real value.

The network was trained by minimizing the Mean-Squared-Error (MSE) between the network outputs and targets for each step. The final test MSE of 0.0341, and mean absolute error (MAE) of 0.1307 were achieved, calculated for all computational steps in all of, in total, 5000 trials. All test trials had

novel sequences of computational steps, that is, test trials were different from those used for training.

4. Fault Pruning: Robust Training of Neural Networks with Memristive Weights

Abstract Neural networks with memristive memory for weights have been proposed as an energy-efficient solution for scaling up of neural network implementations. However, training such memristive neural networks is still challenging due to various memristor imperfections and faulty memristive elements. Such imperfections and faults are becoming increasingly severe as the density of memristor arrays increases in order to scale up weight memory. We propose fault pruning, a robust training scheme for memristive neural networks based on the idea to identify faulty memristive behavior on the fly during training and prune corresponding connections. We test this algorithm in simulations of memristive neural networks using both feed-forward and convolutional architectures on standard object recognition data sets. We show its ability to mitigate the detrimental effect of memristor faults on network training.

This chapter was published as:

Kraišniković, C., Stathopoulos, S., Prodromakis, T., Legenstein, R. (2023). Fault Pruning: Robust Training of Neural Networks with Memristive Weights. In 20th International Conference on Unconventional Computation and Natural Computation. (Reproduced with permission from Springer Nature.)

Contributions Data curation, Software, Formal analysis, Validation, Investigation, Visualization, Methodology, Writing - original draft, Writing - review and editing.

Acknowledgements This research was partially supported by SYNCH project funded by the European Commission under the H2020 FET Proactive programme (Grant agreement ID: 824162) and by the CHIST-ERA grant CHIST-ERA-18-ACAI-004, by the Austrian Science Fund (FWF) project number I 4670-N (project SMALL).

4.1. Introduction

Nano-scale electronic elements have recently gained increased attention for machine learning applications and neuromorphic devices (Indiveri et al., 2013; Jeong & Shi, 2018; Yao et al., 2020). In particular, memristive crossbar arrays have been proposed as a replacement for conventional memory technology in hardware implementations of neural networks (Bayat et al., 2018; S. Chen et al., 2020; Q. Xia & Yang, 2019). A memristor is a resistor with memory in the sense that the charge that flows through a memristor changes its resistance. Resistive Random Access Memories (RRAMs) are a common expression of devices that exhibit memristive behavior and can be realized using different architectures ranging from metal-oxides (Ielmini, 2016) and perovskites (John et al., 2021) to fully organic solutions (Valov & Kozicki, 2017). Memristors possess several advantages over conventional memory elements when their resistive state is utilized to store weights of a neural network: First, their resistive state is non-volatile and therefore, only memory changes but not retention consumes energy. Second, memristors can be integrated with ultra-high density, allowing to scale up neural networks. Third, several implementations of memristors have been demonstrated to present many densely packed resistive states (Stathopoulos et al., 2017) allowing them to operate in an analog fashion. This allows for greater flexibility in storing weight values with high resolution. Finally, memristors arranged in a crossbar array architecture are ideally suited to implement the fundamental mathematical operation in neural networks in $O(1)$ time: vector-matrix multiplication. This can significantly speed up computations. These advantages render them the ideal candidate for the realization of synaptic memory in neural networks. However, memristive neural networks still face substantial challenges. The programming of their resistances is noisy and their behavior is faulty. Faults exhibited by the

memristor are primarily associated with issues related to the fabrication and secondarily because of operational constraints. For the former case, these include non-uniformity of the active layer, interface defects, or thermal effects during processing. As a consequence, devices might become inoperable or operate outside expected specifications. However, devices can also fail during operation as, depending on the technology, the endurance of the devices is limited. Yield and repeatability issues also affect the reliability of large crossbars when compared to established memory technologies. In order to scale up such memristive computing systems, it is necessary to increase the integration density of memristive arrays and thus scale down memristor elements. Unfortunately, the aforementioned problems become particularly pronounced in this case.

In this article, we consider the question of memristive neural network training. Such training in particular suffers from faulty memristor behaviors. Typical faulty behavior includes stuck memristors (i.e., devices that do not change their resistance), memristors with an unexpected change rate (i.e., the memristance change is stronger or weaker than expected), or even memristors with an inverted plasticity behavior (i.e., memristors that change their resistance in the wrong direction). We first analyze the impact of such faulty behavior on neural network training. We find that faulty behavior can significantly impact the resulting network performance. Based on recent findings which show that neural network connectivity can be significantly sparsified with minor loss in performance (Bellec, Kappel, et al., 2018; Han et al., 2015; Z. Liu et al., 2019; Srinivas & Babu, 2015), we then propose a novel training strategy (fault pruning) where faulty memristor behavior is detected during training and corresponding devices are pruned on the fly. We evaluate fault pruning on the MNIST and CIFAR-10 data sets and show that this simple strategy is able to recover effective network optimization for both feed-forward and convolutional memristive neural networks. Further analysis reveals that the algorithm can adaptively adjust network sparsity to make use of the functional memristive resources.

4.2. Results

When memristive elements are used to store weight parameters of artificial neural networks, each weight w_i of the network is maintained in the resistance R_i of a corresponding memristor. More precisely, the weight is in the simplest case given by a linear mapping from the conductance $G_i = \frac{1}{R_i}$

$$w_i = \alpha\Big(\frac{1}{R_i} - \frac{1}{R_\mathrm{C}}\Big), \tag{4.1}$$

where α represents the scaling parameter of memristive weight, and R_C represents the bias resistance value. In this simplest case, a single memristor is used to represent both positive and negative weight values in some range $[-w_\mathrm{min}, w_\mathrm{max}]$. More elaborated designs utilize separate memristors for positive and negative weight values, but we adopted here this scheme for simplicity. We assumed that $w_\mathrm{min} = -w_\mathrm{max}$, and let the weight $w_i = 0$ map to $\frac{1}{R_\mathrm{C}}$. Hence, for a given range $[R_\mathrm{min}, R_\mathrm{max}]$ of resistance values, the bias resistance was given by $R_\mathrm{C} = 2R_\mathrm{min}R_\mathrm{max}/(R_\mathrm{min} + R_\mathrm{max})$ and the scaling parameter was $\alpha = \frac{w_\mathrm{max}}{1/R_\mathrm{min} - 1/R_\mathrm{C}}$. Consequently, the inverse mapping from the weight to resistance was given by

$$R_i = \frac{1}{\frac{1}{R_\mathrm{C}} + \frac{w_i}{\alpha}}, \tag{4.2}$$

see Section 4.3.1 for details. We simulated training of memristive neural networks using an in-the-loop training setup (Esser et al., 2016; Schmitt et al., 2017; Woźniak et al., 2020), see Fig. 4.1. Here, the memristive network was simulated using non-ideal noisy memristive updates (see below). In addition, a copy of the network architecture was simulated using high-precision weights (high-precision network). Gradients were computed using backpropagation in the high-precision network and weight updates were accumulated. When significant weight changes were accumulated, memristors in the memristive network were updated (Woźniak et al., 2020; L. Xia et al., 2017). More precisely, a resistance R_i was updated when a weight change $\Delta w_i^\mathrm{ideal}$ resulting in a resistance change of 2% was reached. In addition, an update of all resistances was forced every 100 training batches (see Section 4.3.2 for details). Since such updates were non-ideal due to switching

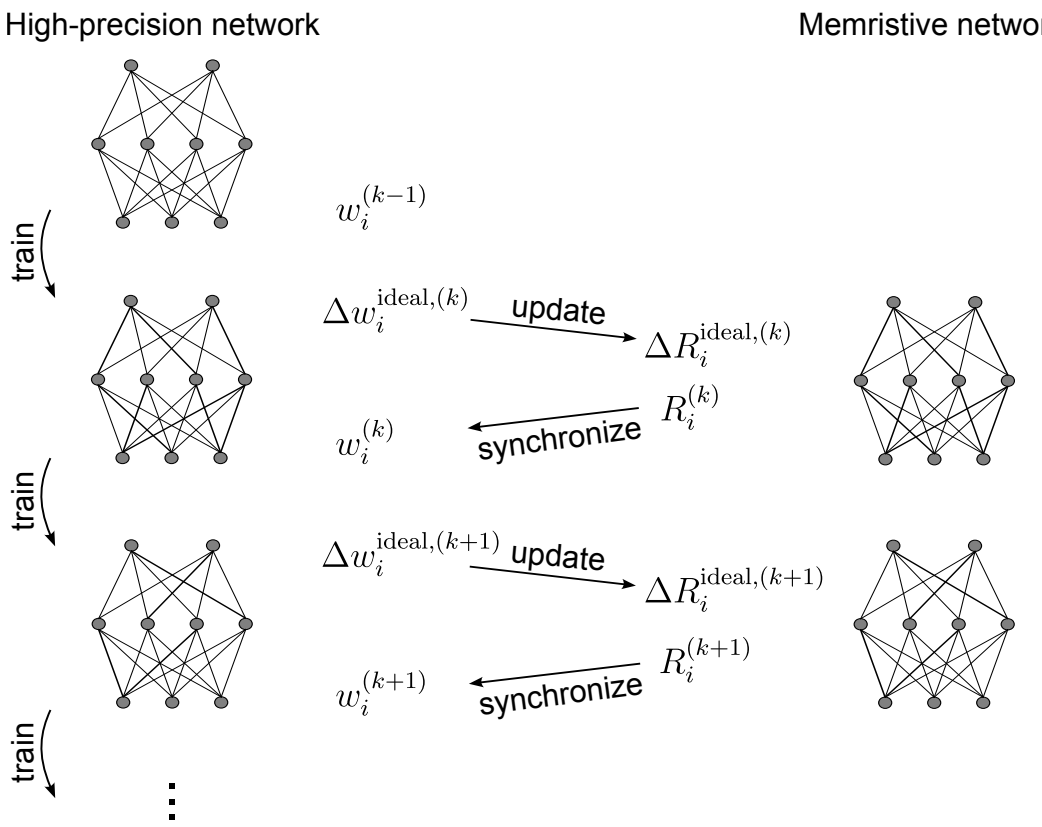


Figure 4.1.: **In-the-loop training setup.** In this setup, training is performed on two networks in parallel. In the memristive network (right), weights are implemented by unreliable and faulty memristors (simulated in our case). The high-precision network (left) has identical architecture, but weights are stored in high precision. The high-precision network is trained until significant weight changes are available. At the kth update step, such weight changes $\Delta w_i^{\mathrm{ideal},(k)}$ are then used to update resistances in the memristive network by the desired amount $\Delta R_i^{\mathrm{ideal},(k)}$. The resulting resistances $R_i^{(k)}$ are read out and used to synchronize the high-precision network. These steps are repeated until training ends.

noise and faulty memristors, resulting resistances were read out, mapped to weight values according to Eq. (4.1), and the high-precision network was updated. This procedure was iterated until the number of target epochs was reached.

Non-ideal memristor updates of the memristive network were modeled as

follows. Let $\Delta w_i^{\text{ideal},(k)}$ denote the proposed update for weight i at update step k, entailing a resistance update $\Delta R_i^{\text{ideal},(k)}$. Due to a number of non-idealities, memristors, when programmed, often show deviations from the intended behavior. Stuck memristors do not change their resistance regardless of the magnitude of the desired resistance change, and we refer to these faults as *stuck faults*. Other memristors underestimate or overestimate the magnitude of resistance change, or even produce updates in the opposite direction of the one predicted by the underlying model of the memristor. We refer to faults where the magnitude of the resistance change is under-/over-estimated as *concordant switching faults* and to faults of memristors that produce resistance change in the opposite direction of the desired one as *discordant switching faults*. In addition, when programming the devices, i.e., switching the devices to different resistive states, the achieved resistance states are noisy due to switching noise.

We modeled faulty memristors by introducing fault factors f_i that modulated the desired (expected) resistance change $\Delta R_i^{\text{ideal},(k)}$ and added a switching noise term

$$\Delta R_i^{(k)} = f_i \cdot \Delta R_i^{\text{ideal},(k)} + \eta_i^{(k)}. \tag{4.3}$$

Fault factors were chosen according to the corresponding memristor fault type: a fault factor $f_i = 0$ for a stuck fault, $f_i < 0$ for a discordant switching fault, and $f_i > 0$ for a concordant switching fault. The switching noise $\eta_i^{(k)}$ was drawn independently for each memristor i and each update step k from a normal distribution with zero mean and a magnitude up to 1% of the current resistance.

4.2.1. Training of Memristive Neural Networks with Faulty Memristors

We first investigated how faulty memristors impact the training of neural networks. We started with a simple feed-forward architecture trained on the MNIST data set (LeCun et al., 2010). The MNIST data set consists of 70 thousand 28×28 gray-scale images of handwritten digits. The goal is to classify these images into 10 classes according to the written digit, see

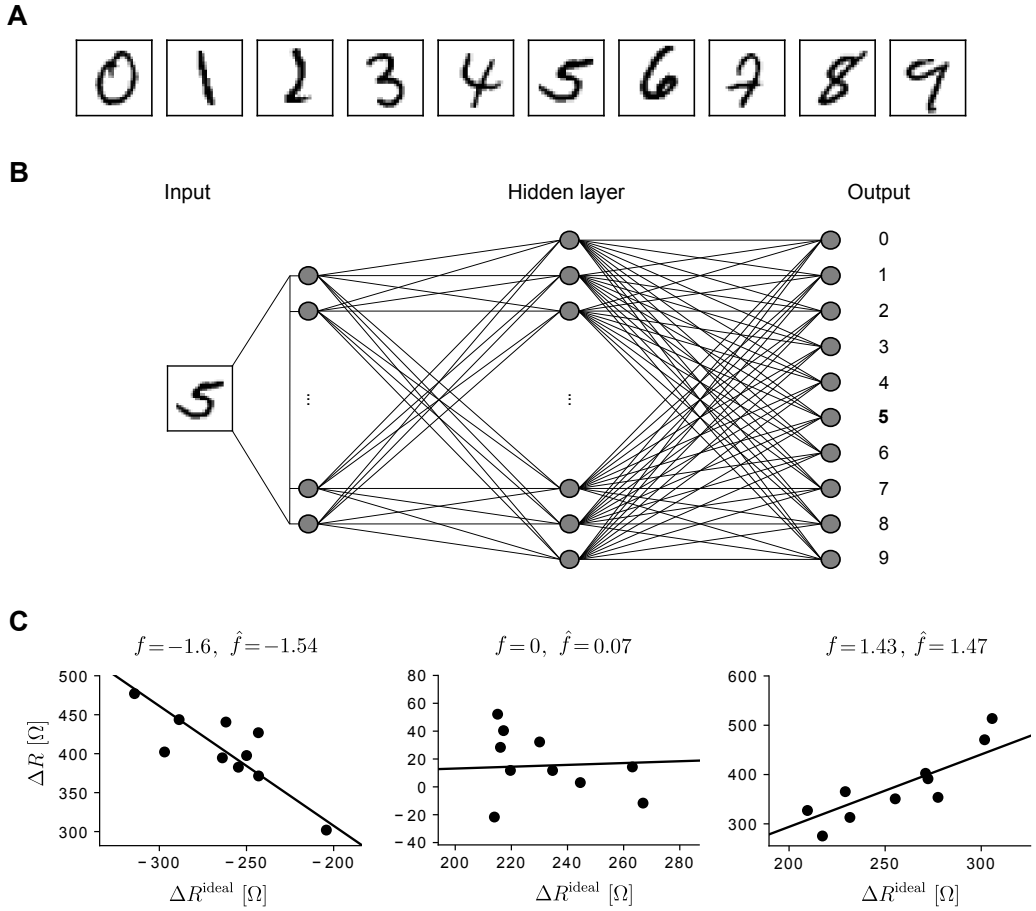


Figure 4.2.: **Feed-forward neural network trained on the MNIST data set.** (A) Example images representing 10 digits. (B) Schematic of the feed-forward architecture used to learn the task. (C) Examples of discordant switching fault (left), stuck fault (middle), and concordant switching fault (right). The noisy achieved resistance change (y-axis) is plotted against the desired change (x-axis). Line indicates linear fit for $\hat{f}$ estimate.

Fig. 4.2A for one example image per class. The neural network architecture is shown in Fig. 4.2B. It consisted of one hidden layer with 128 neurons with a rectified linear (ReLU) activation function and one softmax output layer for classification. We first trained this network in the in-the-loop training setup with switching noise but without faulty memristors. In this case, the network achieved a test classification accuracy of $96.28 \pm 0.32\%$ (percentage

of correctly classified test examples; mean and STD over 10 training runs with random initial conditions). We next performed training with switching noise and faulty memristors. The behavior of three example simulated faulty memristors is shown in Fig. 4.2C, one with a discordant switching fault (left), one with a stuck fault (middle), and one with a concordant switching fault (right). To test the effect of fault type on the network performance, we varied the proportion of fault types, see Fig. 4.3A (left). The figure shows test performance for a relative number of p_{stuck} stuck faults (x-axis) and a relative number of $p_{\text{discordant}}$ discordant switching faults (y-axis). The remaining simulated memristors had concordant switching faults, that is a proportion of $1 - (p_{\text{stuck}} + p_{\text{discordant}})$. We observe that even with 80% concordant switching faults (e.g., cell $(0.1, 0.1)$), the network shows good performance. This is not surprising as the parameter change is still in the correct direction although somewhat distorted. Also, stuck memristors can be tolerated up to some point. Only after 50% stuck faults in the bottom row does the performance fall below 95%. The effect of discordant memristors is more severe. At 30% discordant faults and 10% stuck faults, the performance drops below 94% and then declines rapidly.

The same trend but more strongly pronounced can be observed for the more challenging CIFAR-10 data set (Krizhevsky, 2009) using a convolutional network. This data set consists of color images of size 32×32 from ten different classes representing: airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks (Krizhevsky, 2009), see Fig. 4.4A for example images. The schematic of the architecture used to learn the task is shown in Fig. 4.4B. It consisted of two convolutional-pooling layers, a convolutional layer that after flattening connected to a dense layer, and finally, a softmax output layer with 10 neurons, one per class. Training without memristor faults, we achieved a test classification accuracy of $60.61 \pm 3.15\%$ (mean and STD over 10 training runs). Again, we varied the proportion of fault types as above for the MNIST data set, see Fig. 4.4C (left). When compared to the MNIST results, we can observe a clear performance decrease already for small proportions of stuck- and discordant switching faults (cell $(0.1, 0.1)$), and a more rapid decline of performance for increasing discordant switching faults.

4.2.2. Fault Pruning for Memristive Neural Networks

As the integration density of memristive arrays increases, one can expect more and more unreliable and faulty memristive elements in the array. In principle, one could characterize memristors before training and adapt the training process accordingly. However, memristors can change their characteristics after characterization and in particular memristor faults can appear during training due to limited endurance (L. Xia et al., 2017). Therefore, we propose a robust training scheme (fault pruning) for memristive neural networks that detects unreliable memristors on-line during training using information available in the in-the-loop training setup. We will consider two alternatives to deal with faulty memristors: first, to discard them and set the connection to 0, and second, to continue using the connection but requesting from them no change in resistive states.

In fault pruning, fault factors are estimated for each memristor i during training using the ideal and achieved resistance changes — $\Delta R_i^{\text{ideal},(k)}$ and $\Delta R_i^{(k)}$ — of the N most recent updates of the memristor. A zero-intercept linear regression model is then fitted to these data points to obtain the estimated fault factor

$$\hat{f}_i = \frac{\sum_l \Delta R_i^{\text{ideal},(l)} \Delta R_i^{(l)}}{\sum_l \left(\Delta R_i^{\text{ideal},(l)}\right)^2}. \tag{4.4}$$

A derivation of this estimator is given in Section 4.3.3. Upon the estimation of the fault factor $\hat{f}_i$, the algorithm decides whether or not to prune the weight in the following way: Assuming that memristors with estimated $\hat{f}_i < 0.1$ have stuck or discordant switching faults, the weight w_i is pruned (i.e., set to zero) in both the full-precision and memristive network. Otherwise, $\hat{f}_i \geq 0.1$ indicates a memristor with a concordant switching fault. Such memristors are still useful for training as their weight change goes in the desired direction, thus, further used. The estimated fault factor $\hat{f}_i$ is easily interpretable – it represents the slope of the linear fit, and in principle, any value greater than 0 means that requested and achieved resistance updates have the same trend. For the threshold, we chose the value of 0.1 since for

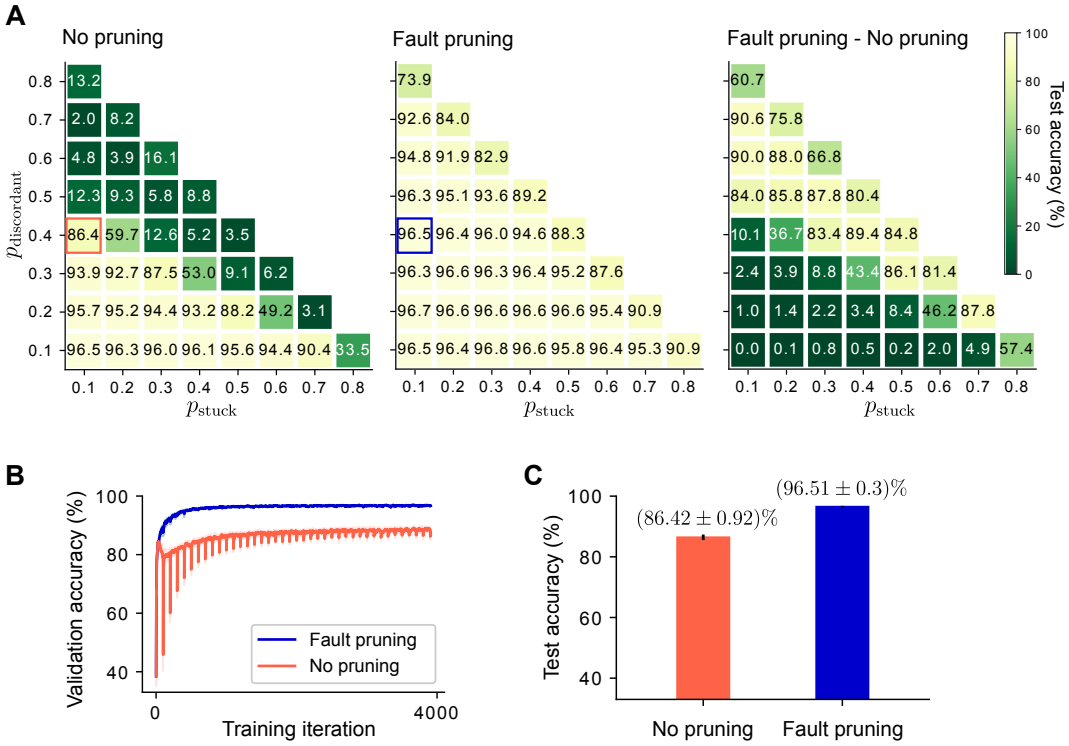


Figure 4.3.: **Performances of feed-forward memristive neural networks trained on the MNIST data set.** (A) Test accuracies for different proportions of memristor faults without fault pruning (left), with fault pruning (middle), and the difference in the accuracies of these scenarios (right). (B) Validation accuracy during training with (blue) and without (red) fault pruning (shading indicates STD over 10 training runs) for the cases indicated by colored boxes in A. (C) Final test accuracy for these training runs (means $\pm$ STD over 10 training runs).

the estimation of $\hat{f}_i$ we used only a few points data points ($N = 10$) that included the switching noise.

As an alternative, we also considered freezing faulty memristors instead of pruning them. In this case, the algorithm keeps the achieved weight in the high-precision network and does not update the faulty memristor anymore.

Fault Pruning for Feed-forward Memristive Neural Networks

We next tested fault pruning in the feed-forward neural network setup on the MNIST data set as described above. During the in-the-loop training, the pruning algorithm estimated fault factor $\hat{f}_i$ for each memristor based on the history of the $N = 10$ most-recent requested and achieved resistance updates in the memristive network. Fig. 4.2C shows a linear fit to the data points $(\Delta R_i^{\text{ideal},(k)}, \Delta R_i^{(k)})$ for three example memristors. Fig. 4.2C (left) illustrates a memristor with a discordant switching fault assigned at the beginning of the training, $f = -1.6$, that the pruning algorithm discarded ($\hat{f} = -1.54 < 0.1$). Similarly, the example memristor in Fig. 4.2C (middle) with a stuck fault ($f = 0$) was discarded. The memristor illustrated in the right panel of Fig. 4.2C, although overestimating the resistance change ($f = 1.43$) was kept for further training.

The pruning algorithm detects faulty memristors and prunes the network connections in an online fashion (during training). This makes it possible to adaptively adjust the network connectivity and avoid the detrimental effects that faulty memristors have on training. We trained feed-forward networks with different proportions of memristor fault types and measured the test accuracy at the end of the training. The fault pruning algorithm preserved the test accuracy even in cases when the proportion of the stuck and discordant switching faults was very significant (see Fig. 4.3A, middle), whereas the test accuracy degraded drastically when there was no pruning (see Fig. 4.3A, left). Note that the memristors that did not have stuck or discordant switching faults had concordant switching faults. The pruning algorithm performed marginally better when the number of memristors with stuck and discordant switching faults was small. As the number of memristors with these fault types increased, the use of fault pruning became essential. This is illustrated in Fig. 4.3A (right), where the differences in test accuracies achieved in both scenarios are shown.

The validation accuracy over memristor updates for the simulations indicated by the blue and red rectangular boxes in Fig. 4.3A are shown in Fig. 4.3B (shaded area indicates STD over 10 simulations with different initial weights). Note that training without pruning was rather unstable,

showing repeated performance decreases due to faulty memristor behavior. The regular negative peaks in the non-pruned case appear since the in-the-loop training setup forces resistance updates for all weights every 100 training batches. Training with pruning on the other hand was stable and reached a much better final accuracy. The final test accuracies for these cases are plotted in Fig. 4.3C. The low standard deviations show that the achieved accuracy (both on validation during training and on the test set) over multiple runs was very consistent.

After the training, connectivity in the network was sparser than initially, and the percentage of weights that remained unpruned is shown in Fig. 4.5A. The zoom-in shows the histogram of fault factors for a single simulation. Here, not all memristors with stuck and discordant switching faults were pruned. The reason for this was the weights that had not changed at all during training (due to low accumulated gradients), hence the fault factors for these memristors were never estimated by the pruning algorithm. Since these memristors did not influence the training, it was not necessary to prune them, which is a positive side effect of the proposed fault pruning algorithm. Note that the connectivity has been adapted by the algorithm in a fault-dependent way, such that networks with more severe faults were pruned more strongly.

Fault Pruning for Convolutional Memristive Neural Networks

Finally, we applied fault pruning on the convolutional architecture, trained on the CIFAR-10 data set (see above). Here, we considered both freezing the weights after the detection of faulty memristors, and discarding them from further use.

When freezing weight values of faulty memristors, fault pruning recovered excellent accuracies even in cases with a significant number of stuck memristors. Also for the cases of larger numbers of discordant switching faults, the algorithm was able to mitigate their detrimental effect on training up to a certain number of faults, see Fig. 4.4C (middle). In each but one of the considered fault distributions, fault pruning was able to improve network performance over plain training, see Fig. 4.4C (right). We examined the single instance (cell $(0.2, 0.1)$) where the fault pruning

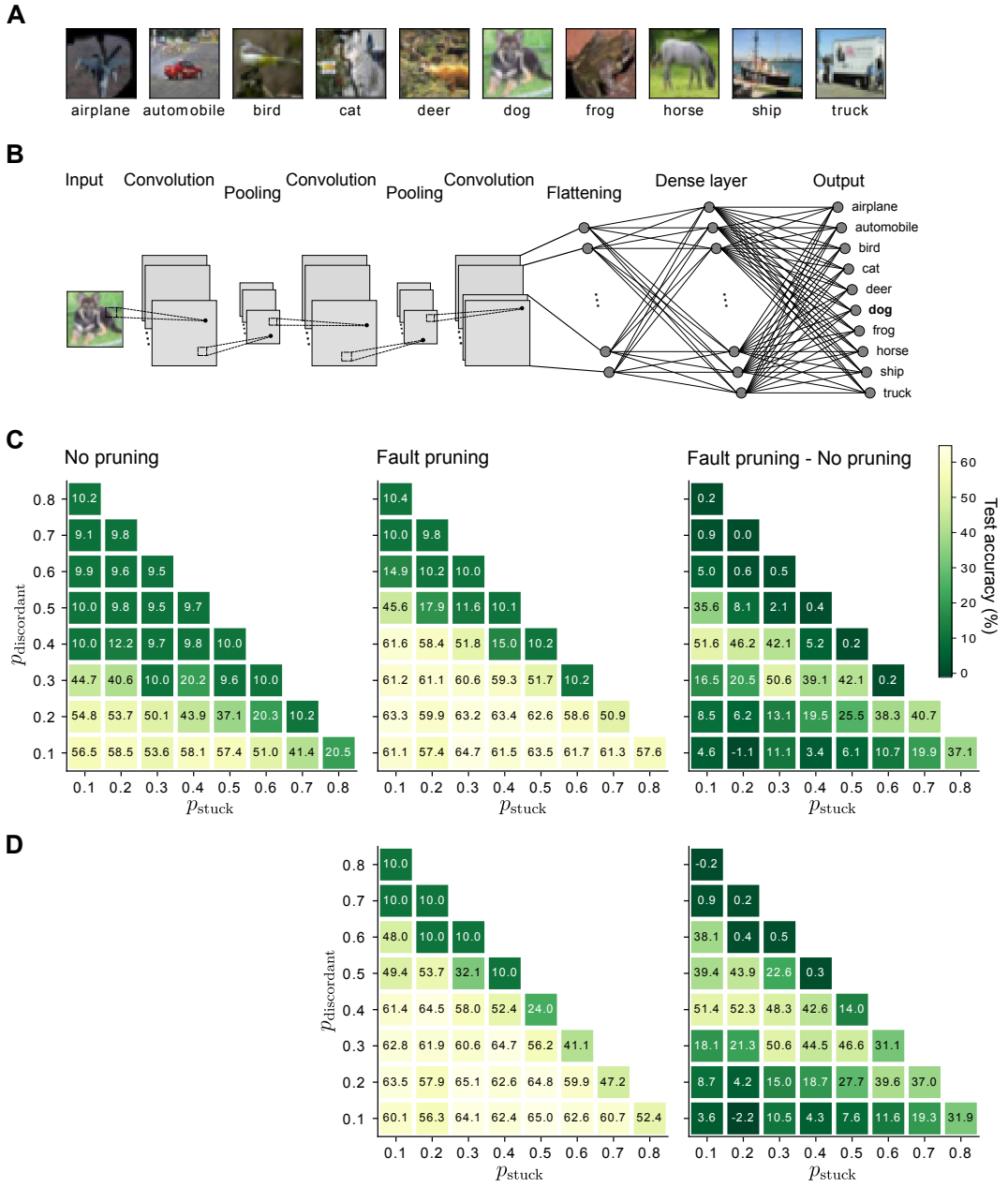


Figure 4.4.: **Performances of memristive CNNs on CIFAR-10.** (A) Example CIFAR-10 images. (B) Convolutional neural network architecture. (C, D) Test accuracies for different proportions of memristor faults without fault pruning (left), with fault pruning (middle), and the difference in the accuracies of these scenarios (right). In (C), pruning was done by freezing weights, in (D), by setting them to zero.

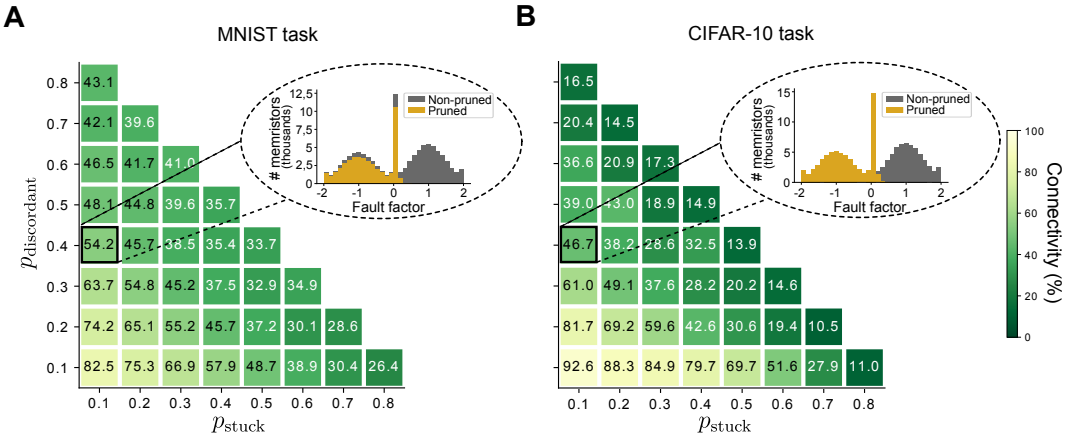


Figure 4.5.: **Connectivity in the network after pruning (in %).** (A) MNIST task. (B) CIFAR-10 task (pruning by freezing weights). Fault pruning adapts network sparsity according to the number of reliable memristive resources. Insets show histograms of fault factors for pruned weights (yellow) and non-pruned weights (gray).

performed slightly worse. There, some connections for which memristors had concordant switching faults f slightly above the threshold value were pruned. When running ten simulations with different initial conditions for this case, pruning performed better on average (pruning $(60.87 \pm 2.48)\%$, no pruning $(58.71 \pm 2.53)\%$ test accuracy). The percentage of unpruned weights/memristors after training (connectivity percentage) is illustrated in Fig. 4.5B. The zoom-in shows a histogram of fault factors assigned to memristors at the beginning of training, with pruned connections shown in yellow and unpruned ones shown in gray.

We also tested performance when faulty memristors were discarded as in the previous section, see Fig. 4.4D. In this case, fault pruning achieved better accuracies for intermediate values of $p_{\text{discordant}}$. Simulation details are given in Section 4.3.4.

4.3. Methods

4.3.1. Memristor Model

In our simulations, we used the following memristor model for all memristive weights. The memristor model defined the resistance range $[R_{\text{min}}, R_{\text{max}}]$ that the state variable R could take. We used $R_{\text{min}} = 6843.97\Omega$, and $R_{\text{max}} = 14109.06\Omega$ – the values calculated for a single device-under-test whose more detailed version of a memristor model is described in (Messaris et al., 2018). Given the maximum value for the weights, w_{max}, for a symmetric range of weights $[-w_{\text{max}}, w_{\text{max}}]$, the bias resistance R_{C} and the scaling parameter α were calculated as $R_{\text{C}} = 2R_{\text{min}}R_{\text{max}}/(R_{\text{min}} + R_{\text{max}})$, and $\alpha = \frac{w_{\text{max}}}{1/R_{\text{min}} - 1/R_{\text{C}}}$, respectively. We used $w_{\text{max}} = 0.5$ in all our simulations.

To model different fault types, fault factors were assigned to memristors as follows: (a) a fault factor $f_i = 0$ for a stuck fault, (b) a fault factor f_i drawn from the normal distribution $\mathcal{N}(-1, 0.5)$ capped within $[-2, -0.1]$ for a discordant switching fault, and (c) a fault factor f_i drawn from $\mathcal{N}(1, 0.5)$ capped within $[0.1, 2]$ for a concordant switching fault. The fault factors were drawn before the training began, assigned randomly to the memristors, and kept constant throughout the training. The switching noise $\eta_i^{(k)}$ in Eq. (4.3) was drawn independently for each memristor i and each update step k from a normal distribution (Stathopoulos et al., 2019). The distribution had zero mean and a standard deviation of $0.01\frac{R_i^{(k)}}{3}$. Resulting resistance changes were then capped at $\pm 1\%$ of the current resistance $R_i^{(k)}$.

4.3.2. Training Schedule

The memristive neural network and its corresponding high-precision network were initially fully connected, and its starting weights were drawn from the Glorot uniform distribution. Fault factors drawn from the distributions described in the previous section were assigned randomly to memristors in the memristive network, simulating in that way the faulty

behaviors of memristors. In each training iteration t, weight updates were computed in the high-precision network using the Adam optimizer (Kingma & Ba, 2015) on one batch of input data (batch size b depended on the task, see below), and the weights in the high-precision network were updated. Since memristance updates are noisy, one usually does not program each individual update in the memristive array, but rather accumulates weight changes until significant updates are available (Woźniak et al., 2020). Hence, memristors were updated typically only when a significant resistance change was available. Therefore, we have to distinguish between training iteration t and update k. To formalize this, denote the achieved resistance for connection i after the most recent update as R_i^{prev} and the proposed resistance after the current training iteration t as R_i^{cur}. Memristor i was updated if $R_i^{\text{cur}} - R_i^{\text{prev}}$ was at least 2% of the current resistance R_i^{prev}. After the update, the achieved resistance value was used to synchronize the corresponding weight w_i in the high-precision network.

In addition to these asynchronous updates, all memristors were updated every 100 training iterations and in the last training iteration. The updates for which the magnitude of the requested change was below the noise level (in our simulations, 1% of R_i^{prev}) were enlarged to $\pm 1\%$ of R_i^{prev}, depending on the sign of the originally requested $\Delta R_i^{\text{ideal},(k)}$. Note that for the memristors with $\Delta R_i^{\text{ideal},(k)} = 0$, the updates were not enforced since the programming of the memristive device was not required. Also here, weights in the high-precision network were synchronized according to the achieved memristance values.

Estimation of the fault factors was implemented as follows: Let $\Delta R_i^{\text{ideal},(k)}$ and $\Delta R_i^{(k)}$ denote the proposed and the achieved resistance change of connection i at the k-th memristor update respectively. The most recent $N = 10$ update pairs $(\Delta R_i^{\text{ideal},(k)}, \Delta R_i^{(k)})$ for memristor i were used to estimate the fault factor $\hat{f}_i$. The connections for which $\hat{f}$ was below the threshold of 0.1 were pruned. We implemented two versions of pruning – setting weights to 0 (being equivalent to removing the weight in the high-precision network, and not using at all memristors in the memristive networks), and keeping the weights/memristors in the network, but without further training. The minimum allowed connectivity was 10%, i.e., at least 10% of the weights

had to be used between any two layers. After estimating fault factors, it was possible to prune up to 20% connections for the MNIST task, and up to 50% of connections for the CIFAR-10 task. Hence the candidate weights for pruning, i.e., the weights for which the estimated fault factor $\hat{f}_i$ was smaller than the threshold value 0.1, were first sorted in the ascending order according to their fault factors, and pruned. This prevented removing many connections at once, and also prioritized removing memristors with discordant switching fault type over stuck faults.

4.3.3. Estimation of the Fault Factors $\hat{f}_i$

For the estimation of the fault factor $\hat{f}_i$, we used a zero-intercept linear regression model

$$\Delta R_i = \hat{f}_i \cdot \Delta R_i^{\text{ideal}} + \epsilon \tag{4.5}$$

that models the linear relation between requested ($\Delta R_i^{\text{ideal}}$) and achieved ($\Delta R_i$) resistance updates under Gaussian noise. It was estimated from $N = 10$ data points $(\Delta R_i^{\text{ideal},(l)}, \Delta R_i^{(l)})$, $l \in \{k-N+1, k-N+2, ..., k-1, k\}$ representing the N most recent updates of memristor R_i. The least-squares estimator of $\hat{f}_i$ minimizes the squared error $\mathcal{L}(\hat{f}_i)$,

$$\mathcal{L}(\hat{f}_i) := \sum_l \left(\Delta R_i^{(l)} - \hat{f}_i \cdot \Delta R_i^{\text{ideal},(l)}\right)^2, \tag{4.6}$$

and a closed-form analytical solution can be found as

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \hat{f}_i} &= 2 \sum_l \left(\Delta R_i^{(l)} - \hat{f}_i \cdot \Delta R_i^{\text{ideal},(l)}\right)\left(-\Delta R_i^{\text{ideal},(l)}\right) \stackrel{!}{=} 0 \\ \hat{f}_i \sum_l \left(\Delta R_i^{\text{ideal},(l)}\right)^2 &= \sum_l \Delta R_i^{(l)} \Delta R_i^{\text{ideal},(l)} \end{aligned} \tag{4.7}$$

Hence, $\hat{f}_i = \frac{\sum_l \Delta R_i^{(l)} \Delta R_i^{\text{ideal},(l)}}{\sum_l \left(\Delta R_i^{\text{ideal},(l)}\right)^2}$ follows. The condition $\sum_l \left(\Delta R_i^{\text{ideal},(l)}\right)^2 \neq 0$ was ensured because the pruning algorithm always enforced updates with magnitudes different than zero.

4.3.4. Details to Computer Simulations

Details to Feed-forward Networks Trained on the MNIST Task

We trained memristive feed-forward networks on the MNIST task consisting of 50000, 10000, and 10000 training, validation, and test images, respectively. The architecture used was $784 - 128 - 10$ neurons per layer. Over training, the connections both in high-precision and memristive networks were pruned. For pruned connections, memristors were discarded from the memristive network, while the weights in the high-precision network were set to zero. In the high-precision network, all neurons in the hidden and output layer had a trainable bias term. For the optimization of the weights and biases of the (high-precision) feed-forward network, the Adam optimizer with an initial learning rate of 0.01 was used. The learning rate decayed exponentially every 1000 iteration by a factor of 0.99. The optimizer used batches of $b = 128$ training images to minimize the cross-entropy error function. The network was trained for 10 epochs (forward and backward propagations of the whole dataset), resulting in a total of 3910 training iterations. The performance numbers reported in Fig. 4.3A represent the test accuracies achieved for a single training run with $p_{\text{discordant}}$ discordant switching and p_{stuck} stuck faults, except for the highlighted ones (the blue and red rectangular boxes) where they are the mean over 10 runs.

Details to Convolutional Networks Trained on the CIFAR-10 Task

We trained memristive convolutional neural networks on the CIFAR-10 task, consisting of 50000 training and 10000 test images.

The architecture that we trained consisted of: (1) an input layer, (2) a $2D$-convolutional layer with 32 output filters, kernels of size (3×3), valid padding, stride $(1, 1)$, with ReLU activation function, (3) a max-pooling layer with a pool size (2×2), (4) a $2D$-convolutional layer with 64 output filters, kernels of size (3×3), valid padding, stride $(1, 1)$, with ReLU activation function, (5) a max-pooling layer with a pool size (2×2), (6) a $2D$-convolutional layer with 64 output filters, kernels of size (3×3), valid padding, stride $(1, 1)$, with ReLU activation function, then flattened which

resulted in a layer of 1024 neurons, (7) a densely connected layer (64 neurons), with ReLU activation function, and (8) an output layer of 10 neurons (one per class), with softmax activation function.

Initially fully connected, the connections in the high-precision and memristive networks were pruned during training. Here we used two approaches for pruning. For pruned connections, either both resistance in the memristive network and weight in the high-precision network were frozen, or the weight was set to 0, and the corresponding memristor was discarded. In the high-precision network, all neurons had biases that were optimized along the weights using the Adam optimizer. The learning rate was set to 0.005, batch size to $b = 256$ training images, and the cross-entropy error function was minimized for 30 epochs. This resulted in a total of 5880 training iterations. The performance numbers in Fig. 4.4C, D report the test accuracies representing a single training run.

4.4. Conclusions

We have proposed fault pruning, a novel training algorithm for robust training of neural networks with memristive weights. We applied this algorithm to both feed-forward and convolutional neural networks. The approach is general, independent of the network structure and the trained task.

Most previously proposed robust-training schemes (e.g., (C.-Y. Chen & Chakrabarty, 2021; Joksas et al., 2020; J. Wang et al., 2020)) are agnostic to the exact location of memristor faults. Their objective is to alleviate the impact of faults when memristors are programmed only once at the end of training. In (C. Liu et al., 2017), a re-training scheme, as well as a re-mapping of weights to memristors, was proposed. Xia et al. (L. Xia et al., 2017) proposed an on-line fault detection method combined with a re-mapping method, but not in the in-the-loop training setup considered here. In contrast to these works, we do not propose re-mapping but assume that arbitrary memristive connections can be pruned.

Our proposed fault pruning algorithm takes advantage of the communication exchange between the two networks in the in-the-loop training scheme. Requested and achieved resistance changes are used to estimate the type of a memristor's fault $\hat{f}_i$, and act accordingly. For the estimation of the faulty behavior, we used a simple linear regression. This can, however, be substituted by more advanced approaches if necessary. In our simulations, the fault type assigned to a memristor was kept constant during training; in practice, the memristor's fault type could change over time. Such a case would not be a problem for fault pruning, because, for the estimation of the fault factor and the fault type, it uses a certain number of most recent memristor updates from which the change could be detected. Our simulations showed that even with a very large percentage of faulty memristors, and in particular with memristors with discordant switching faults, fault pruning managed to preserve very good performance. Another option for handling memristors with discordant fault types not considered in this article could be to adapt the requested update, e.g., by inverting its sign. The advantages of this approach could be investigated in future work.

In summary, we showed in simulations that pruning faulty memristive connections provides a viable strategy for robust training of memristive neural networks. The fault type can be estimated on-line during in-the-loop training, allowing for efficient robust training of performant networks.

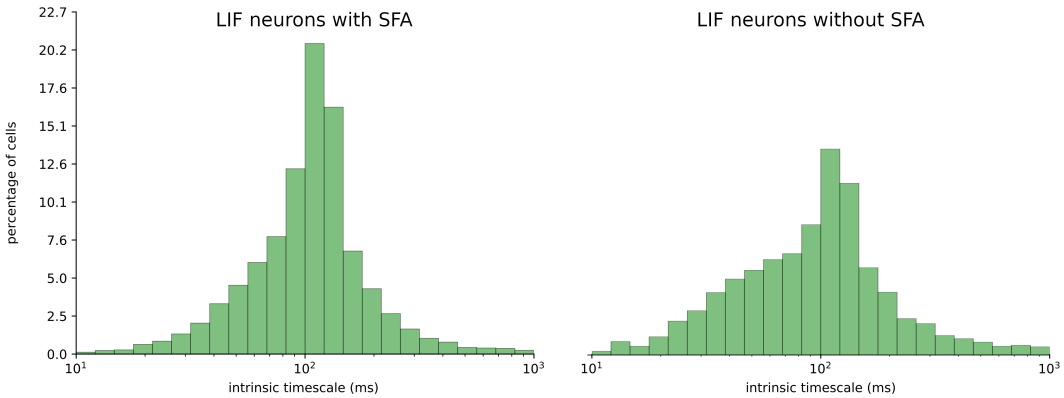
Appendix

Appendix A.

Appendix to Chapter 2: Spike frequency adaptation supports network computations on temporally dispersed information

A.1. Autocorrelation-based intrinsic time scale of neurons trained on STORE-RECALL task.

We wondered whether the adaptive firing threshold of LIF neurons with SFA affects the autocorrelation function of their firing activity — termed intrinsic time scale in (Wasmuht et al., 2018). We tested this for an SNN consisting of 200 LIF neurons without and 200 LIF neurons with SFA that was trained to solve a one-dimensional version of the STORE-RECALL task. It turned out that during the delay between STORE and RECALL these intrinsic time constants were in the same range as those measured in the monkey cortex, see Figure 1C in (Wasmuht et al., 2018). Furthermore, neurons of the trained SNN exhibited very similar distributions of these time constants (see Appendix figure A.1), suggesting that these intrinsic time constants are determined largely by their network inputs, and less by the neuron type.



Appendix figure A.1.: **Histogram of the intrinsic time scale of neurons trained on STORE-RECALL task.** We trained 64 randomly initialized SNNs consisting of 200 LIF neurons with and 200 without SFA on the single-feature STORE-RECALL task. Measurements of the intrinsic time scale were performed according to (Wasmuht et al., 2018) on the spiking data of SNNs solving the task after training. Averaged data of all 64 runs is presented in the histogram. The distribution is very similar for neurons with and without SFA.

A.2. sMNIST task with sparsely connected SNN obeying Dale's law

This task has originally been used as a temporal processing benchmark for ANNs, and has successfully been solved with the Long Short-Term Memory (LSTM) type of ANNs (Hochreiter & Schmidhuber, 1997). LSTM units store information in registers — like a digital computer — so that the stored information cannot be perturbed by ongoing network activity. Networks of LSTM units or variations of such units have been widely successful in temporal processing and reach the level of human performance for many temporal computing tasks.

Since LSTM networks also work well for tasks on larger time-scales, for comparing SNNs with LSTM networks, we used a version of the task with 2 ms presentation time per pixel, thereby doubling the length of sequences to be classified to 1568 ms. Grey values of pixels were presented to the LSTM network simply as analog values. A trial of a trained SNN with SFA (with

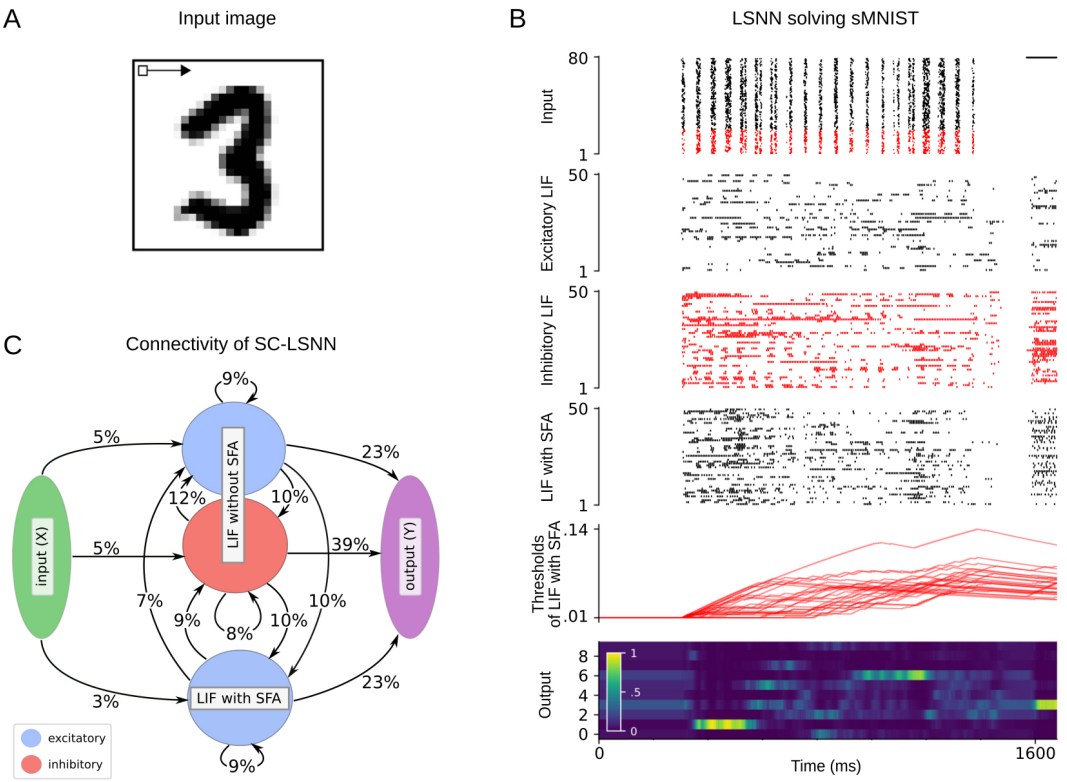
an input sequence that encodes a handwritten digit "3" using population rate coding) is shown in Appendix figure A.2B. The top row of Appendix figure A.2B shows a version where the grey value of the currently presented pixel is encoded by population coding, through the firing probability of 80 input neurons. Somewhat better performance was achieved when each of the 80 input neurons was associated with a particular threshold for the grey value, and this input neuron fired whenever the grey value crossed its threshold in the transition from the previous to the current pixel (this input convention was used to produce the results below).

Besides a fully connected network of LIF neurons with SFA, we also tested the performance of a variant of the model, called SC-SNN, that integrates additional constraints of SNNs in the brain: It is sparsely connected (12% of possible connections are present) and consists of 75% excitatory and 25% inhibitory neurons that adhere to Dale's law. By adapting the sparse connections with the rewiring method in (Bellec, Kappel, et al., 2018) during *BPTT* training, the SC-SNN was able to perform even better than the fully-connected SNN of LIF neurons with SFA. The resulting architecture of the SC-SNN is shown in Appendix figure A.2C. Its activity of excitatory and inhibitory neurons, as well as the time courses of adaptive thresholds for (excitatory) LIF neurons with SFA of the SC-SNN are shown in Appendix figure A.2B. In this setup, the SFA had $\tau_a = 1400\,\text{ms}$. When we used an SNN with SFA, we improved the accuracy on this task to 96.4% which approaches the accuracy of the artificial LSTM model which reached the accuracy of 98.0%.

We also trained a liquid state machine version of the SNN model with SFA where only the readout neurons are trained. This version of the network reached the accuracy of $63.24 \pm 1.48\%$ over 5 independent training runs.

A.3. Google Speech Commands

We trained SNNs with and without SFA on the keyword spotting task with Google Speech Commands Dataset (Warden, 2018) (v0.02). The dataset consists of 105000 audio recordings of people saying thirty different words. Fully connected networks were trained to classify audio recordings, that



Appendix figure A.2.: **sMNIST time series classification benchmark task.** (**A**) Illustration of the pixel-wise input presentation of handwritten digits for sMNIST. (**B**) Rows top to bottom: Input encoding for an instance of the sMNIST task, network activity, and temporal evolution of firing thresholds for randomly chosen subsets of neurons in the SC-SNN, where 25% of the LIF neurons were inhibitory (their spikes are marked in red). The light color of the readout neuron for digit "3" around 1600 ms indicates that this input was correctly classified. (**C**) Resulting connectivity graph between neuron populations of an SC-SNN after BPTT optimization with DEEP R on sMNIST task with 12% global connectivity limit.

were clipped to one second length, into one of the 12 classes (10 keywords, as well as two special classes for silence and unknown words; the remaining 20 words had to be classified as "unknown"). A comparison of the maximum performance of trained spiking networks against state-of-the-art artificial recurrent networks is shown in Appendix table A.1. Averaging over 5 runs, the SNN with SFA reached $90.88 \pm 0.22\%$, and the SNN without SFA reached $88.79 \pm 0.16\%$ accuracy. Thus an SNN without SFA can already solve this task quite well, but the inclusion of SFA halves the performance gap to the published state-of-the-art in machine learning. The only other report on a solution to this task with spiking networks is (Zenke & Vogels, 2020). There the authors train a network of LIF neurons using surrogate gradients with *BPTT* and achieve $85.3 \pm 0.3\%$ accuracy on the full 35 classes setup of the task. In this setup, the SNN with SFA reached $88.5 \pm 0.16\%$ test accuracy.

Features were extracted from the raw audio using the Mel Frequency Cepstral Coefficient (MFCC) method with 30 ms window size, 1 ms stride, and 40 output features. The network models were trained to classify the input features into one of the 10 keywords (yes, no, up, down, left, right, on, off, stop, go) or to two special classes for silence or unknown word (where the remainder of 20 recorded keywords are grouped). The training, validation and test set were assigned 80, 10, and 10 percent of data respectively while making sure that audio clips from the same person stayed in the same set.

All networks were trained for 18000 iterations using the Adam optimizer with batch size 100. The output spikes of the networks were averaged over time, and the linear readout layer was applied to those values. During the first 15000 iterations, we used a learning rate of 0.001 and for the last 3000, we used a learning rate of 0.0001. The loss function contained a regularization term (scaled with coefficient 0.001) that minimized the squared difference of average firing rate between individual neurons and a target firing rate of 10 Hz.

Both SNNs with and without SFA consisted of 2048 fully connected neurons in a single recurrent layer. The neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, the adaptation time constant of SFA was $\tau_a = 100\,\text{ms}$, adaptation strength was $\beta = 2\,\text{mV}$. The baseline threshold was $v_{\text{th}} = 10\,\text{mV}$, and the refractory period was 2 ms. The synaptic delay was 1 ms.

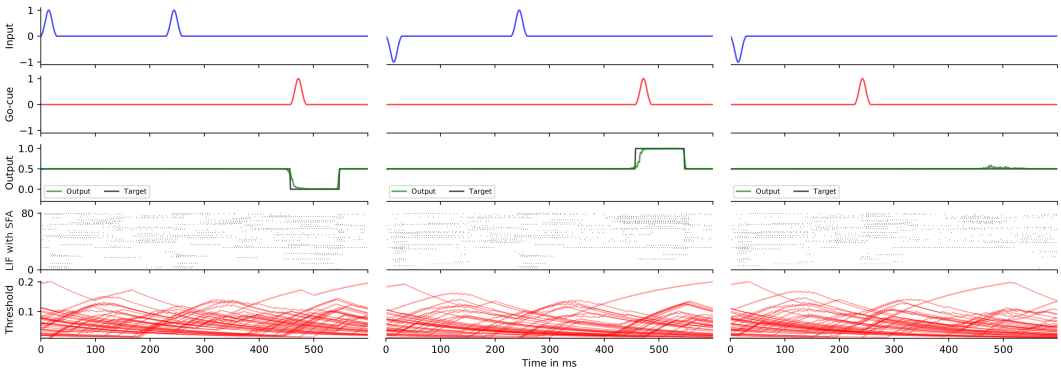
A.4. Delayed-memory XOR

We also tested the performance of SNNs with SFA on a previously considered benchmark task, where two items in the working memory have to be combined non-linearly: the Delayed-memory XOR task (Huh & Sejnowski, 2018). The network is required to compute the exclusive-or operation on the history of input pulses when prompted by a go-cue signal, see Appendix figure A.3.

The network received on one input channel two types of pulses (up or down), and a go-cue on another channel. If the network received two input pulses since the last go-cue signal, it should generate the output "1" during the next go-cue if the input pulses were different or "0" if the input pulses were the same. Otherwise, if the network only received one input pulse since the last go-cue signal, it should generate a null output (no output pulse). Variable time delays are introduced between the input and go-cue pulses. The time scale of the task was 600 ms which limited the delay between input pulses to 200 ms.

This task was solved in (Huh & Sejnowski, 2018), without providing performance statistics, by using a type of neuron that has not been documented in biology — a non-leaky quadratic integrate and fire neuron. We are not aware of previous solutions by networks of LIF neurons. To compare and investigate the impact of SFA on network performance in the delayed-memory XOR task, we trained SNNs, with and without SFA, of the same size as in (Huh & Sejnowski, 2018) – 80 neurons. Across 10 runs, SNNs with SFA solved the task with $95.19 \pm 0.014\%$ accuracy, whereas the SNNs without SFA converged at lower $61.30 \pm 0.029\%$ accuracy.

The pulses on the two input channels were generated with 30 ms duration and the shape of a normal probability density function normalized in the range $[0, 1]$. The pulses were added or subtracted from the baseline zero input current at appropriate delays. The go-cue was always a positive current pulse. The 6 possible configurations of the input pulses ($+$, $-$, $++$, $--$, $+-$, $-+$) were sampled with equal probability during training and testing.



Appendix figure A.3.: **Delayed-memory XOR task.** Rows top to bottom: Input signal, Go-cue signal, network readout, network activity, and temporal evolution of firing thresholds.

Networks were trained for 2000 iterations using the Adam optimizer with batch size 256. The initial learning rate was 0.01 and every 200 iterations the learning rate was decayed by a factor of 0.8. The loss function contained a regularization term (scaled with coefficient 50) that minimized the squared difference of the average firing rate of individual neurons from a target firing rate of 10 Hz. This regularization resulted in networks with a mean firing rate of 10 Hz where firing rates of individual neurons were spread in the range [1, 16] Hz.

Both SNNs with and without SFA consisted of 80 fully connected neurons in a single recurrent layer. The neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, a baseline threshold $v_{\text{th}} = 10\,\text{mV}$, and a refractory period of 3 ms. SFA had an adaptation time constant of $\tau_a = 500\,\text{ms}$ and an adaptation strength of $\beta = 1\,\text{mV}$. The synaptic delay was 1 ms. For training the network to classify the input into one of the three classes, we used the cross-entropy loss between the labels and the softmax of three linear readout neurons. The input to the linear readout neurons were the neuron traces that were calculated by passing all the network spikes through a low-pass filter with a time constant of 20 ms.

A.5. The 12AX task in a noisy network

As a control experiment, aimed at testing the robustness of the solution (performance as a function of the strength of added noise), we simulated the injection of an additional noise current into all LIF neurons (with and without SFA). The previously trained network (trained without noise) was reused and tested on a test set of 2000 episodes. In each discrete time step, the noise was added to the input current $I_j(t)$ (see equation 2.4 in Chapter 2), hence affecting the voltage of the neuron:

$$I_j(t) = \sum_i W_{ji}^{\text{in}} x_i(t - d_{ji}^{\text{in}}) + \sum_i W_{ji}^{\text{rec}} z_i(t - d_{ji}^{\text{rec}}) + I_{\text{noise}}, \qquad \text{(A.1)}$$

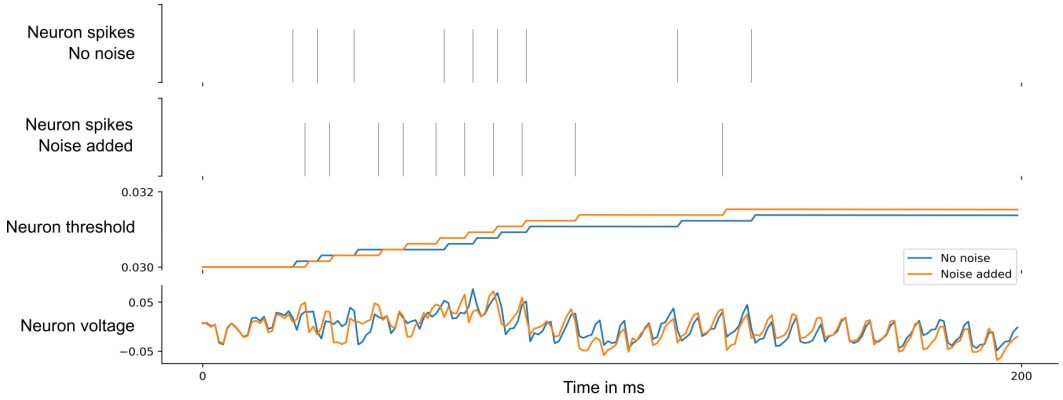
where I_{noise} was drawn from a normal distribution with mean zero, and standard deviation $\sigma \in \{0.05, 0.075, 0.1, 0.2, 0.5\}$.

The performance of the network without noise was 97.85% (performance of one initialization of the network with 100 LIF neurons with SFA and 100 LIF neurons without SFA). During testing, including the noise current of mean zero and standard deviation $\sigma \in \{0.05, 0.075, 0.1, 0.2, 0.5\}$ lead to the performance of 92.65%, 89.05%, 80.25%, 27.25%, 0.25%, respectively. The network performance degraded gracefully up to a current of standard deviation of about 0.1.

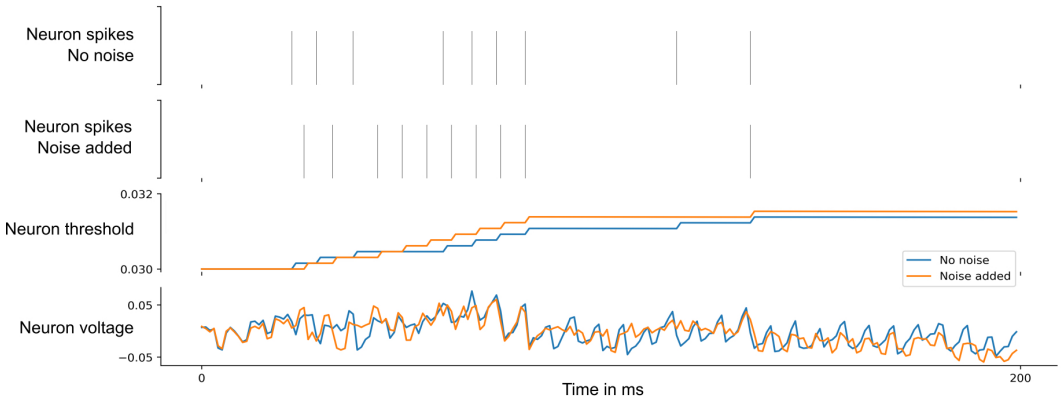
For an illustration of the effect of noise, see Appendix figure A.4 and A.5. There, we compare the output spikes, adaptive threshold, and membrane voltage of one neuron with noise current to the versions without noise. The shown simulations started from exactly the same initial condition and noise with standard deviation 0.05 (0.075) was injected only into the shown neuron (other neurons did not receive any noise current). One sees that even this weak noise current produces a substantial perturbation of the voltage, adaptive threshold, and spiking output of the neuron.

Model	test accuracy (%)
FastGRNN-LSQ (Kusupati et al., 2018)	93.18
SNN with SFA	91.21
SNN	89.04

Appendix table A.1.: **Google Speech Commands.** Accuracy of the spiking network models on the test set compared to the state-of-the-art artificial recurrent model reported in (Kusupati et al., 2018). Accuracy of the best out of 5 simulations for SNNs is reported.



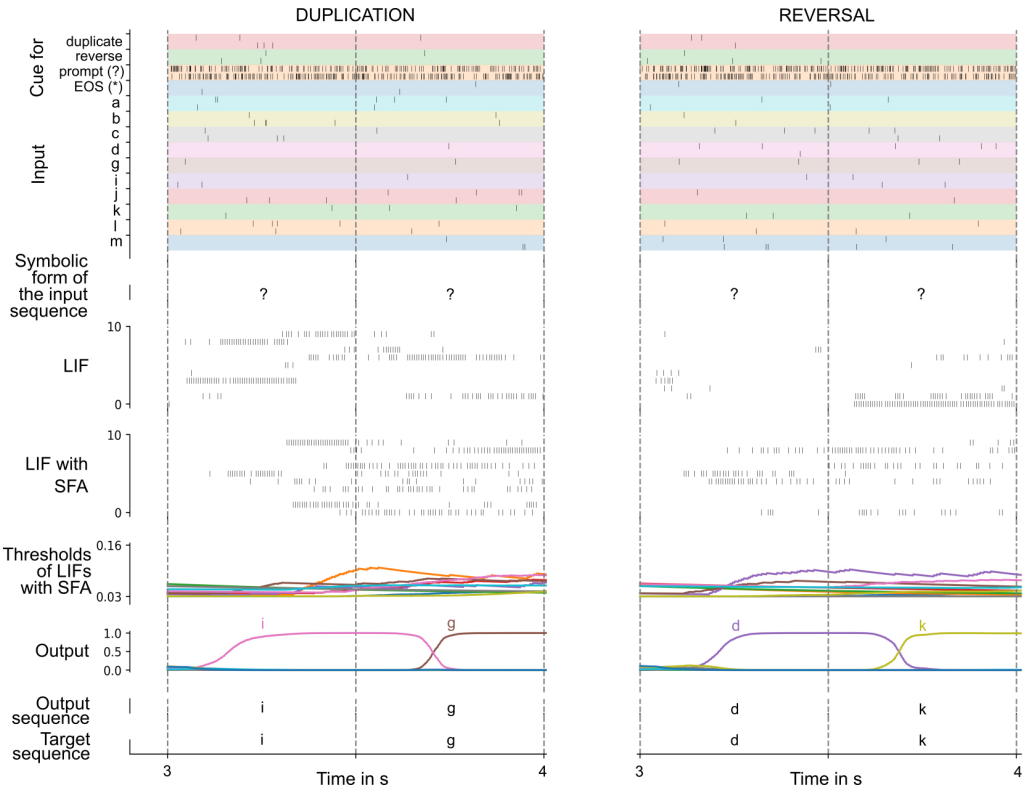
Appendix figure A.4.: **Effect of a noise current with zero mean and standard deviation 0.05 added to a single neuron in the network for the 12AX task.** Spike train of a single neuron without noise, followed by spike train in the presence of the noise, adaptive threshold of the neuron that corresponds to the spike train with no noise (shown in blue), spike train with noise present (shown in orange), and corresponding neuron voltages over the time course of 200 ms.



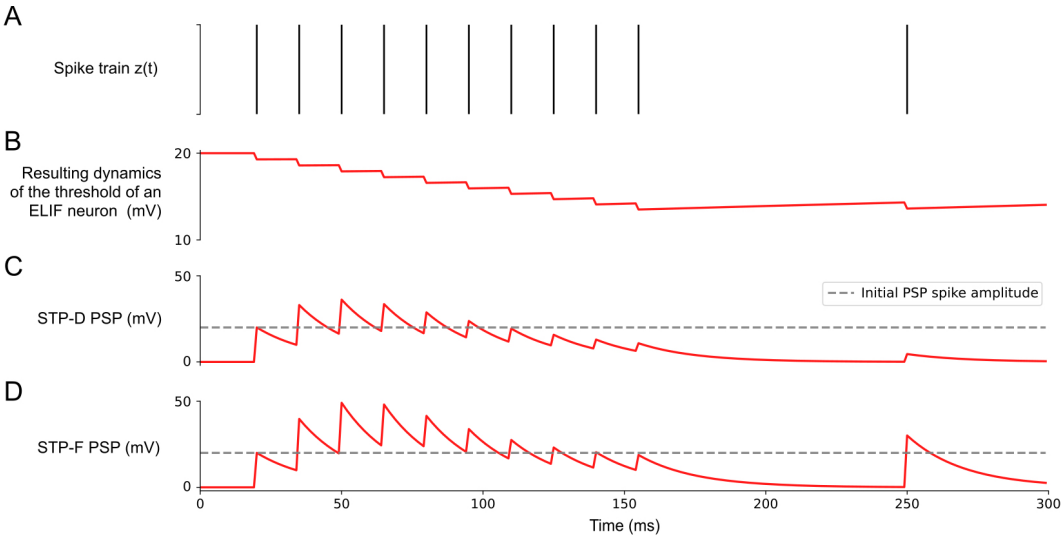
Appendix figure A.5.: **Effect of a noise current with zero mean and standard deviation 0.075 added to a single neuron in the network for the 12AX task.** Spike train of a single neuron without noise, followed by spike train in the presence of the noise, adaptive threshold of the neuron that corresponds to the spike train with no noise (shown in blue), spike train with noise present (shown in orange), and corresponding neuron voltages over the time course of 200 ms.

A.6. Duplication/Reversal task

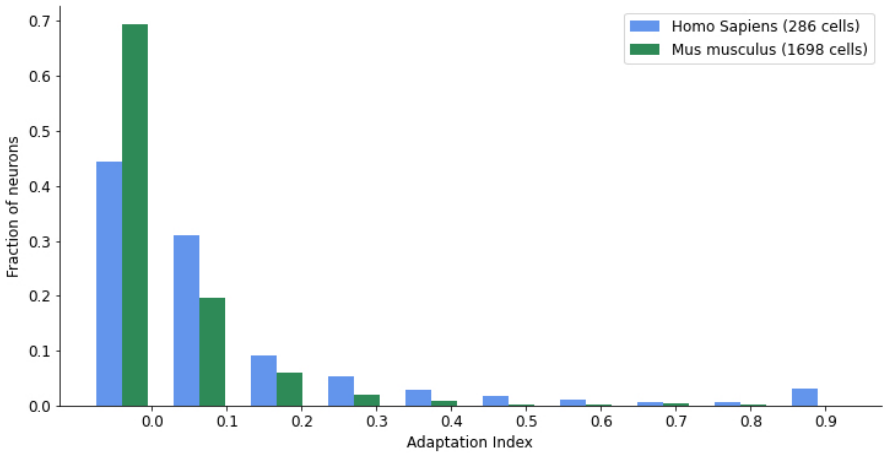
A zoom-in for the rasters shown in Figure 2.4 is shown in Appendix figure A.6, for the time period $3 - 4$ seconds.



Appendix figure A.6.: **A zoom-in of the spike raster for a trial solving Duplication task (left) and Reversal task (right).** A sample episode where the network carried out sequence duplication (left) and sequence reversal (right), shown for the time period of $3 - 4$ ms (2 steps after the start of network output). Top to bottom: Spike inputs to the network (subset), sequence of symbols they encode, spike activity of 10 sample LIF neurons (without and with SFA) in the SNN, firing threshold dynamics for these 10 LIF neurons with SFA, activation of linear readout neurons, output sequence produced by applying argmax to them, target output sequence.



Appendix figure A.7.: **Illustration of models for an inversely adapting ELIF neuron, and for short-term synaptic plasticity.** (**A**) Sample spike train. (**B**) The resulting evolution of firing threshold for an inversely adapting neuron (ELIF neuron). (**C-D**) The resulting evolution of the amplitude of postsynaptic potentials (PSPs) for spikes of the presynaptic neuron for the case of a depression-dominant (STP-D: D$\gg$F) and a facilitation-dominant (STP-F: F$\gg$D) short-term synaptic plasticity.



Appendix figure A.8.: **Distribution of adaptation index from Allen Institute cell measurements (Allen Institute, 2018).**

Bibliography

Allen Institute. (2017). *Allen Cell Types Database Technical white paper: GLIF models*
http://help.brain-map.org/download/attachments/8323525/glifmodels.pdf (tech. rep.) [v4]. v4. (Cit. on pp. 2, 19, 20).

Allen Institute. (2018). 2018 Allen Institute for Brain Science. Allen Cell Types Database, cell feature search. Available from: celltypes.brain-map.org/data (cit. on pp. 17, 22, 46, 55, 125).

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate, In *International conference on learning representations*. (Cit. on p. 7).

Barch, D. M., Berman, M. G., Engle, R., Jones, J. H., Jonides, J., MacDonald III, A., Nee, D. E., Redick, T. S., & Sponheim, S. R. (2009). Cntrics final task selection: Working memory. *Schizophrenia bulletin*, *35*(1), 136–152 (cit. on pp. 5, 31).

Barone, P., & Joseph, J.-P. (1989). Prefrontal cortex and spatial sequencing in macaque monkey. *Experimental brain research*, *78*(3), 447–464 (cit. on pp. 18, 35).

Barrett, D., Hill, F., Santoro, A., Morcos, A., & Lillicrap, T. (2018). Measuring abstract reasoning in neural networks, In *International conference on machine learning*. PMLR. (Cit. on p. 5).

Bayat, F. M., Prezioso, M., Chakrabarti, B., Nili, H., Kataeva, I., & Strukov, D. (2018). Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. *Nature communications*, *9*(1), 1–7 (cit. on p. 92).

Behrens, T. E., Muller, T. H., Whittington, J. C., Mark, S., Baram, A. B., Stachenfeld, K. L., & Kurth-Nelson, Z. (2018). What is a cognitive map? organizing knowledge for flexible behavior. *Neuron*, *100*(2), 490–509 (cit. on p. 69).

Bellec, G., Kappel, D., Maass, W., & Legenstein, R. (2018). Deep rewiring: Training very sparse deep networks, In *International conference on learning representations*. (Cit. on pp. 93, 115).

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., & Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons, In *Advances in neural information processing systems*. (Cit. on pp. 9, 17, 19, 20, 47, 48, 61, 66).

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., & Maass, W. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, *11*(1), 1–15 (cit. on pp. 9, 19, 20, 34, 43, 49, 61, 62, 67, 77, 86).

Benda, J. (2021). Neural adaptation. *Current Biology*, *31*(3), R110–R116 (cit. on p. 7).

Benda, J., & Herz, A. V. (2003). A universal model for spike-frequency adaptation. *Neural computation*, *15*(11), 2523–2564 (cit. on pp. 7, 17).

Benda, J., Maler, L., & Longtin, A. (2010). Linear versus nonlinear signal transmission in neuron models with adaptation currents or dynamic thresholds. *Journal of Neurophysiology*, *104*(5), 2806–2820 (cit. on p. 17).

Bohnstingl, T., Woźniak, S., Pantazi, A., & Eleftheriou, E. (2022). Online spatio-temporal learning in deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (cit. on p. 9).

Bohte, S. M., Kok, J. N., & La Poutré, J. A. (2000). Spikeprop: Backpropagation for networks of spiking neurons., In *Esann*. Bruges. (Cit. on p. 9).

Botvinick, M. M., & Cohen, J. D. (2014). The computational and neural basis of cognitive control: Charted territory and new frontiers. *Cognitive science*, *38*(6), 1249–1285 (cit. on p. 4).

Buonomano, D. V., & Maass, W. (2009). State-dependent computations: Spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience*, *10*(2), 113–125 (cit. on p. 30).

Buschman, T. J., & Miller, E. K. (2022). Working memory is complex and dynamic, like your thoughts. *Journal of cognitive neuroscience*, *35*(1), 17–23 (cit. on p. 7).

Camuñas-Mesa, L. A., Linares-Barranco, B., & Serrano-Gotarredona, T. (2019). Neuromorphic spiking neural networks and their memristor-cmos hardware implementations. *Materials*, *12*(17), 2745 (cit. on p. 10).

Carpenter, A. F., Baud-Bovy, G., Georgopoulos, A. P., & Pellizzer, G. (2018). Encoding of serial order in working memory: Neuronal activity in motor, premotor, and prefrontal cortex during a memory scanning task. *Journal of Neuroscience*, *38*(21), 4912–4933 (cit. on pp. 18, 35, 36).

Chen, C.-Y., & Chakrabarty, K. (2021). Pruning of deep neural networks for fault-tolerant memristor-based accelerators, In *2021 58th acm/ieee design automation conference (dac)*. IEEE. (Cit. on p. 109).

Chen, S., Mahmoodi, M. R., Shi, Y., Mahata, C., Yuan, B., Liang, X., Wen, C., Hui, F., Akinwande, D., Strukov, D. B., Et al. (2020). Wafer-scale integration of two-dimensional materials in high-density memristive crossbar arrays for artificial neural networks. *Nature Electronics*, *3*(10), 638–645 (cit. on p. 92).

Chettih, S. N., & Harvey, C. D. (2019). Single-neuron perturbations reveal feature-specific competition in V1. *Nature*, *567*(7748), 334–340 (cit. on p. 55).

Christensen, D. V., Dittmann, R., Linares-Barranco, B., Sebastian, A., Le Gallo, M., Redaelli, A., Slesazeck, S., Mikolajick, T., Spiga, S., Menzel, S., Et al. (2022). 2022 roadmap on neuromorphic computing and engineering. *Neuromorphic Computing and Engineering*, *2*(2), 022501 (cit. on p. 9).

Chua, L. (1971). Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, *18*(5), 507–519 (cit. on p. 10).

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99 (cit. on pp. 1, 9).

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., Plank, P., & Risbud, S. R. (2021). Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, *109*(5), 911–934 (cit. on pp. 9, 61).

Dehaene, S. (2011). *The number sense: How the mind creates mathematics*. OUP USA. (Cit. on p. 75).

Deneve, S. (2008). Bayesian spiking neurons i: Inference. *Neural computation*, *20*(1), 91–117 (cit. on p. 17).

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., & Rasmussen, D. (2012). A large-scale model of the functioning brain. *science*, *338*(6111), 1202–1205 (cit. on p. 73).

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211 (cit. on p. 7).

Ermentrout, B. (1998). Linearization of fi curves by adaptation. *Neural computation*, *10*(7), 1721–1729 (cit. on p. 17).

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., Berg, D. J., McKinstry, J. L., Melano, T., Barch, D. R., Nolfo, C. d., Datta, P., Amir, A., Taba, B., Flickner, M. D., & Modha, D. S. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, *113*(41), 11441–11446 (cit. on pp. 48, 94).

Field, A. (2013). *Discovering statistics using ibm spss statistics*. Sage. (Cit. on pp. 36, 56).

Fitz, H., Uhlmann, M., Van den Broek, D., Duarte, R., Hagoort, P., & Petersson, K. M. (2020). Neuronal spike-rate adaptation supports working memory in language processing. *Proceedings of the National Academy of Sciences*, *117*(34), 20881–20889 (cit. on p. 17).

Frank, M. J., Loughry, B., & O'Reilly, R. C. (2001). Interactions between frontal cortex and basal ganglia in working memory: A computational model. *Cognitive, Affective, & Behavioral Neuroscience*, *1*(2), 137–160 (cit. on pp. 5, 31).

Fransén, E., Tahvildari, B., Egorov, A. V., Hasselmo, M. E., & Alonso, A. A. (2006). Mechanism of graded persistent cellular activity of entorhinal cortex layer v neurons. *Neuron*, *49*(5), 735–746 (cit. on p. 30).

Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of neural engineering*, *13*(5), 051001 (cit. on pp. 9, 61).

Furber, S., Galluppi, F., Temple, S., & Plana, L. A. (2014). The spinnaker project. *Proceedings of the IEEE*, *102*(5), 652–665 (cit. on pp. 1, 9).

Gazzaniga, M. S., Ivry, R. B., & Mangun, G. (2009). Cognitive neuroscience: The biology of the mind. Norton: New York. (Cit. on pp. 2, 4, 6, 8, 71).

Gerstner, W., & Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press. (Cit. on pp. 2, 62).

Gilmore, C., McCarthy, S. E., & Spelke, E. S. (2007). Symbolic arithmetic knowledge without instruction. Loughborough University. (Cit. on p. 75).

Gilmore, C. K., McCarthy, S. E., & Spelke, E. S. (2010). Non-symbolic arithmetic abilities and mathematics achievement in the first year of formal schooling. *Cognition*, *115*, 394–406 (cit. on p. 75).

Goldman-Rakic, P. S. (1995). Cellular basis of working memory. *Neuron*, *14*(3), 477–485 (cit. on p. 6).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press. (Cit. on pp. 2, 8).

Gutierrez, G. J., & Denève, S. (2019). Population adaptation in efficient balanced networks. *eLife*, *8*, e46926 (cit. on pp. 17, 18).

Gutkin, B., & Zeldenrust, F. (2014). Spike frequency adaptation [revision #143322]. *Scholarpedia*, *9*(2), 30643. https://doi.org/10.4249/scholarpedia.30643 (cit. on pp. 8, 17)

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network, In *Advances in neural information processing systems*. (Cit. on p. 93).

Haque, Z. Z., Samandra, R., & Mansouri, F. A. (2021). Neural substrate and underlying mechanisms of working memory: Insights from brain stimulation studies. *Journal of Neurophysiology*, *125*(6), 2038–2053 (cit. on p. 4).

Harvey, C. D., Coen, P., & Tank, D. W. (2012). Choice-specific sequences in parietal cortex during a virtual-navigation decision task. *Nature*, *484*(7392), 62–68 (cit. on p. 36).

Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, *95*(2), 245–258 (cit. on pp. 1, 7).

Hayworth, K. J. (2012). Dynamically partitionable autoassociative networks as a solution to the neural binding problem. *Frontiers in computational neuroscience*, *6*, 73 (cit. on p. 72).

Hayworth, K. J., & Marblestone, A. H. (2018). How thalamic relays might orchestrate supervised deep training and symbolic computation in the brain. *bioRxiv*, 304980 (cit. on p. 72).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780 (cit. on pp. 7, 66, 114).

Hu, B., Garrett, M. E., Groblewski, P. A., Ollerenshaw, D. R., Shang, J., Roll, K., Manavi, S., Koch, C., Olsen, S. R., & Mihalas, S. (2020). Adaptation supports short-term memory in a visual change detection task. *bioRxiv* (cit. on pp. 30, 42).

Huh, D., & Sejnowski, T. J. (2018). Gradient descent for spiking neural networks, In *Advances in neural information processing systems*. (Cit. on pp. 29, 118).

Ielmini, D. (2016). Resistive switching memories based on metal oxides: Mechanisms, reliability and scaling. *Semiconductor Science and Technology*, *31*(6), 063002 (cit. on p. 92).

Ielmini, D., & Wong, H.-S. P. (2018). In-memory computing with resistive switching devices. *Nature electronics*, *1*(6), 333–343 (cit. on p. 11).

Indiveri, G., Linares-Barranco, B., Legenstein, R., Deligeorgis, G., & Prodromakis, T. (2013). Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology*, *24*(38), 384010 (cit. on pp. 1, 10, 92).

Jeong, H., & Shi, L. (2018). Memristor devices for neural networks. *Journal of Physics D: Applied Physics*, *52*(2), 023003 (cit. on p. 92).

John, R. A., Shah, N., Vishwanath, S. K., Ng, S. E., Febriansyah, B., Jagadeeswararao, M., Chang, C.-H., Basu, A., & Mathews, N. (2021). Halide perovskite memristors as flexible and reconfigurable physical unclonable functions. *Nature Communications*, *12*(1) (cit. on p. 92).

Joksas, D., Freitas, P., Chai, Z., Ng, W. H., Buckwell, M., Li, C., Zhang, W., Xia, Q., Kenyon, A., & Mehonic, A. (2020). Committee machines—a universal method to deal with non-idealities in memristor-based neural networks. *Nature communications*, *11*(1), 1–10 (cit. on p. 109).

Kamiński, J., & Rutishauser, U. (2019). Between persistently active and activity-silent frameworks: Novel vistas on the cellular basis of working memory. *Annals of the New York Academy of Sciences* (cit. on p. 6).

Kilpatrick, Z. P., & Ermentrout, B. (2011). Sparse gamma rhythms arising through clustering in adapting neuronal networks. *PLoS Comput Biol*, *7*(11), e1002281 (cit. on p. 17).

Kim, R., Li, Y., & Sejnowski, T. J. (2019). Simple framework for constructing functional spiking recurrent neural networks. *Proceedings of the national academy of sciences*, *116*(45), 22811–22820 (cit. on p. 61).

Kim, R., & Sejnowski, T. J. (2021). Strong inhibitory signaling underlies stable temporal dynamics and working memory in spiking neural networks. *Nature Neuroscience*, *24*(1), 129–139 (cit. on p. 42).

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization, In *International conference on learning representations*. (Cit. on pp. 51, 106).

Kok, P., & de Lange, F. P. (2015). Predictive coding in sensory cortex. In *An introduction to model-based cognitive neuroscience* (pp. 221–244). Springer. (Cit. on p. 40).

Kraisnikovic, C. (2018). *Symbolic computation in spiking neural networks* (Master's thesis). Graz University of Technology. (Cit. on p. 72).

Krakauer, J. W., Ghazanfar, A. A., Gomez-Marin, A., MacIver, M. A., & Poeppel, D. (2017). Neuroscience needs behavior: Correcting a reductionist bias. *Neuron*, *93*(3), 480–490 (cit. on p. 4).

Kriegeskorte, N. (2015). Deep neural networks: A new framework for modeling biological vision and brain information processing. *Annual review of vision science*, *1*, 417–446 (cit. on p. 2).

Kriegeskorte, N., & Douglas, P. K. (2018). Cognitive computational neuroscience. *Nature neuroscience*, *21*(9), 1148–1160 (cit. on p. 5).

Kriete, T., Noelle, D. C., Cohen, J. D., & O'Reilly, R. C. (2013). Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. *Proceedings of the National Academy of Sciences*, *110*(41), 16390–16395 (cit. on p. 72).

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (tech. rep.). (Cit. on p. 98).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks, In *Advances in neural information processing systems*. (Cit. on p. 2).

Kruijne, W., Bohte, S. M., Roelfsema, P. R., & Olivers, C. N. L. (2020). Flexible working memory through selective gating and attentional tagging. *Neural Computation*, *0*(0), 1–40 (cit. on p. 32).

Kullmann, D. M., Moreau, A. W., Bakiri, Y., & Nicholson, E. (2012). Plasticity of inhibition. *Neuron*, *75*(6), 951–962 (cit. on pp. 30, 46).

Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., & Varma, M. (2018). Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network, In *Advances in neural information processing systems*. (Cit. on p. 121).

Lanza, M., Sebastian, A., Lu, W. D., Le Gallo, M., Chang, M.-F., Akinwande, D., Puglisi, F. M., Alshareef, H. N., Liu, M., & Roldan, J. B. (2022). Memristive technologies for data storage, computation, encryption, and radio-frequency communication. *Science*, *376*(6597), eabj9979 (cit. on p. 11).

Lashley, K. S. (1951). *The problem of serial order in behavior* (Vol. 21). Bobbs-Merrill Oxford, United Kingdom. (Cit. on pp. 5, 18, 35).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444 (cit. on pp. 2, 8).

LeCun, Y., Cortes, C., & Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, *2* (cit. on p. 96).

Legenstein, R. (2015). Nanoscale connections for brain-like circuits. *Nature*, *521*(7550), 37–38 (cit. on pp. 9, 10).

Li, G., Deng, L., Tang, H., Pan, G., Tian, Y., Roy, K., & Maass, W. (2023). Brain inspired computing: A systematic survey and future trends (cit. on p. 1).

Li, Y., Kim, R., & Sejnowski, T. J. (2021). Learning the synaptic and intrinsic membrane dynamics underlying working memory in spiking neural network models. *Neural Computation*, *33*(12), 3264–3287 (cit. on p. 8).

Lillicrap, T. P., Cownden, D., Tweed, D. B., & Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, *7*(1), 13276 (cit. on p. 8).

Lindsay, G. W., Rigotti, M., Warden, M. R., Miller, E. K., & Fusi, S. (2017). Hebbian learning in a random network captures selectivity properties of the prefrontal cortex. *Journal of Neuroscience*, *37*(45), 11021–11036 (cit. on p. 56).

Liu, C., Hu, M., Strachan, J. P., & Li, H. (2017). Rescuing memristor-based neuromorphic design with high defects, In *2017 54th acm/edac/ieee design automation conference (dac)*. IEEE. (Cit. on p. 109).

Liu, Y., Dolan, R. J., Kurth-Nelson, Z., & Behrens, T. E. (2019). Human replay spontaneously reorganizes experience. *Cell*, *178*(3), 640–652 (cit. on pp. 18, 35, 75).

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2019). Rethinking the value of network pruning, In *International conference on learning representations*. (Cit. on p. 93).

Lundqvist, M., Brincat, S. L., Rose, J., Warden, M. R., Buschman, T. J., Miller, E. K., & Herman, P. (2023). Working memory control dynamics follow principles of spatial computing. *Nature Communications*, *14*(1), 1429 (cit. on p. 7).

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671 (cit. on pp. 2, 60, 61).

Maass, W. (2016). Energy-efficient neural network chips approach human recognition capabilities. *Proceedings of the National Academy of Sciences* (cit. on p. 61).

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560 (cit. on p. 30).

MacDonald III, A. W. (2008). Building a clinically relevant cognitive task: Case study of the ax paradigm. *Schizophrenia bulletin*, *34*(4), 619–628 (cit. on p. 31).

Manneschi, L., & Vasilaki, E. (2020). An alternative to backpropagation through time. *Nature Machine Intelligence*, *2*(3), 155–156 (cit. on p. 68).

Marblestone, A. H., Wayne, G., & Kording, K. P. (2016). Toward an integration of deep learning and neuroscience. *Frontiers in computational neuroscience*, 94 (cit. on p. 8).

Marcus, G. (2003). *The algebraic mind: Integrating connectionism and cognitive science*. MIT Press. (Cit. on pp. 4, 5, 34, 35, 41, 61, 72, 75).

Marcus, G. (2018). Deep learning: A critical appraisal. *arXiv preprint arXiv: 1801.00631* (cit. on p. 61).

Marcus, G., Marblestone, A., & Dean, T. (2014). The atoms of neural computation. *Science*, *346*(6209), 551–552 (cit. on pp. 41, 61, 72).

Marder, E., Abbott, L., Turrigiano, G. G., Liu, Z., & Golowasch, J. (1996). Memory from the dynamics of intrinsic membrane currents. *Proceedings of the national academy of sciences*, *93*(24), 13481–13486 (cit. on p. 17).

Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., Ailamaki, A., Alonso-Nanclares, L., Antille, N., Arsever, S., Et al. (2015). Reconstruction and simulation of neocortical microcircuitry. *Cell*, *163*(2), 456–492 (cit. on pp. 1, 30).

Marschall, O., Cho, K., & Savin, C. (2020). A unified framework of online learning algorithms for training recurrent neural networks. *The Journal of Machine Learning Research*, *21*(1), 5320–5353 (cit. on p. 8).

Martinolli, M., Gerstner, W., & Gilra, A. (2018). Multi-timescale memory dynamics extend task repertoire in a reinforcement learning net-

work with attention-gated memory [30061819[pmid]]. *Frontiers in computational neuroscience*, *12*, 50–50 (cit. on p. 32).

Masse, N. Y., Yang, G. R., Song, H. F., Wang, X.-J., & Freedman, D. J. (2019). Circuit mechanisms for the maintenance and manipulation of information in working memory. *Nature Neuroscience*, *22*(7), 1159–1167 (cit. on pp. 6, 7, 30).

Mastrogiuseppe, F., & Ostojic, S. (2018). Linking connectivity, dynamics, and computations in low-rank recurrent neural networks. *Neuron*, *99*(3), 609–623 (cit. on p. 2).

McClelland, J. L., Rumelhart, D. E., Group, P. R., Et al. (1987). *Parallel distributed processing, volume 2: Explorations in the microstructure of cognition: Psychological and biological models* (Vol. 2). MIT press. (Cit. on p. 2).

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, *78*(10), 1629–1636 (cit. on p. 10).

Mead, C. (2020). How we created neuromorphic engineering. *Nature Electronics*, *3*(7), 434–435 (cit. on p. 10).

Mensi, S., Naud, R., Pozzorini, C., Avermann, M., Petersen, C. C., & Gerstner, W. (2012). Parameter extraction and classification of three cortical neuron types reveals two distinct adaptation mechanisms. *Journal of neurophysiology*, *107*(6), 1756–1775 (cit. on p. 46).

Messaris, I., Serb, A., Stathopoulos, S., Khiat, A., Nikolaidis, S., & Prodromakis, T. (2018). A data-driven verilog-a reram model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(12), 3151–3162 (cit. on p. 105).

Miller, E. K., & Cohen, J. D. (2001). An integrative theory of prefrontal cortex function. *Annual review of neuroscience*, *24*(1), 167–202 (cit. on pp. 4, 5).

Mitchell, T. (1997). *Machine learning*. McGraw-Hill Education. (Cit. on p. 8).

Mongillo, G., Barak, O., & Tsodyks, M. (2008). Synaptic theory of working memory. *Science*, *319*(5869), 1543–1546 (cit. on pp. 6, 30, 46).

Mongillo, G., Rumpel, S., & Loewenstein, Y. (2018). Inhibitory connectivity defines the realm of excitatory plasticity. *Nature neuroscience*, *21*(10), 1463–1470 (cit. on p. 42).

Mozer, M. C. (1989). A focused back-propagation algorithm for temporal pattern recognition. *Complex systems*, *3*(4), 349–381 (cit. on pp. 20, 48).

Müller, M. G., Papadimitriou, C. H., Maass, W., & Legenstein, R. (2020). A model for structured information representation in neural networks of the brain. *Eneuro*, *7*(3) (cit. on p. 73).

Murray, J. M. (2019). Local online learning in recurrent networks with random feedback. *Elife*, *8*, e43299 (cit. on p. 9).

Murray, J. D., Bernacchia, A., Freedman, D. J., Romo, R., Wallis, J. D., Cai, X., Padoa-Schioppa, C., Pasternak, T., Seo, H., Lee, D., Et al. (2014). A hierarchy of intrinsic timescales across primate cortex. *Nature neuroscience*, *17*(12), 1661 (cit. on p. 43).

Newell, A., Simon, H. A. Et al. (1972). *Human problem solving* (Vol. 104). Prentice-hall Englewood Cliffs, NJ. (Cit. on p. 4).

Nieder, A. (2012). Supramodal numerosity selectivity of neurons in primate prefrontal and posterior parietal cortices. *Proceedings of the National Academy of Sciences*, *109*(29), 11860–11865 (cit. on p. 69).

Nieder, A., & Miller, E. K. (2003). Coding of cognitive magnitude: Compressed scaling of numerical information in the primate prefrontal cortex. *Neuron*, *37*(1), 149–157 (cit. on p. 68).

O'Reilly, R. C. (2006). Biologically based computational models of high-level cognition. *Science*, *314*(5796), 91–94 (cit. on p. 72).

O'Reilly, R. C., & Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural computation*, *18*(2), 283–328 (cit. on pp. 31, 32).

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks*, *6*(3), 623–641 (cit. on p. 73).

Pozzorini, C., Mensi, S., Hagens, O., Naud, R., Koch, C., & Gerstner, W. (2015). Automated high-throughput characterization of single neurons by means of simplified spiking models. *PLoS computational biology*, *11*(6) (cit. on pp. 17, 46, 66).

Pozzorini, C., Naud, R., Mensi, S., & Gerstner, W. (2013). Temporal whitening by power-law adaptation in neocortical neurons. *Nature neuroscience*, *16*(7), 942–948 (cit. on pp. 17, 46, 66).

Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., & Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, *521*(7550), 61–64 (cit. on p. 10).

Pulvermüller, F. (2013). How neurons make meaning: Brain mechanisms for embodied and abstract-symbolic semantics. *Trends in Cognitive Sciences*, *17*(9), 458–470 (cit. on p. 71).

Raven, J. C., & Court, J. (1938). *Raven's progressive matrices*. Western Psychological Services Los Angeles. (Cit. on p. 5).

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., Clopath, C., Costa, R. P., de Berker, A., Ganguli, S., Et al. (2019). A deep learning framework for neuroscience. *Nature neuroscience*, *22*(11), 1761–1770 (cit. on p. 8).

Rigotti, M., Barak, O., Warden, M. R., Wang, X.-J., Daw, N. D., Miller, E. K., & Fusi, S. (2013). The importance of mixed selectivity in complex cognitive tasks. *Nature*, *497*(7451), 585–590 (cit. on p. 74).

Robinson, A., & Fallside, F. (1987). *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering Cambridge, MA. (Cit. on pp. 8, 20, 48).

Roitman, J. D., Brannon, E. M., & Platt, M. L. (2007). Monotonic coding of numerosity in macaque lateral intraparietal area. *PLoS Biol*, *5*(8), e208 (cit. on p. 69).

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386 (cit. on p. 2).

Rougier, N. P., Noelle, D. C., Braver, T. S., Cohen, J. D., & O'Reilly, R. C. (2005). Prefrontal cortex and flexible cognitive control: Rules without symbols. *Proceedings of the National Academy of Sciences*, *102*(20), 7338–7343 (cit. on pp. 4, 7).

Roy, K., Jaiswal, A., & Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature*, *575*(7784), 607–617 (cit. on p. 1).

Rueckauer, B., & Liu, S.-C. (2018). Conversion of analog to spiking neural networks using sparse temporal coding, In *2018 ieee international symposium on circuits and systems (iscas)*. IEEE. (Cit. on p. 61).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536 (cit. on p. 8).

Salaj, D., Subramoney, A., Kraisnikovic, C., Bellec, G., Legenstein, R., & Maass, W. (2021). Spike frequency adaptation supports network com-

putations on temporally dispersed information. *Elife*, *10*, e65459 (cit. on pp. 7–9, 66, 73, 74).

Schmitt, S., Klähn, J., Bellec, G., Grübl, A., Guettler, M., Hartel, A., Hartmann, S., Husmann, D., Husmann, K., Jeltsch, S., Et al. (2017). Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system, In *2017 international joint conference on neural networks*, IEEE. (Cit. on p. 94).

Semenza, C., Miceli, L., & Girelli, L. (1997). A deficit for arithmetical procedures: Lack of knowledge or lack of monitoring? *Cortex*, *33*(3), 483–498 (cit. on p. 71).

Shanker, S. (1995). Turing and the origins of ai. *Philosophia Mathematica*, *3*(1), 52–85 (cit. on p. 4).

Sheahan, H., Luyckx, F., Nelli, S., Teupe, C., & Summerfield, C. (2021). Neural state space alignment for magnitude generalization in humans and recurrent networks. *Neuron*, *109*(7), 1214–1226 (cit. on pp. 76, 77).

Sherman, S. M. (2014). The function of metabotropic glutamate receptors in thalamus and cortex. *The Neuroscientist*, *20*(2), 136–149 (cit. on p. 43).

Spitzer, B., Waschke, L., & Summerfield, C. (2017). Selective overweighting of larger magnitudes during noisy numerical comparison. *Nature Human Behaviour*, *1*(8), 1–8 (cit. on p. 69).

Srinivas, S., & Babu, R. V. (2015). Data-free parameter pruning for deep neural networks, In *Proceedings of the british machine vision conference*, BMVA Press. (Cit. on p. 93).

Stathopoulos, S., Khiat, A., Trapatseli, M., Cortese, S., Serb, A., Valov, I., & Prodromakis, T. (2017). Multibit memory operation of metal-oxide bi-layer memristors. *Scientific Reports*, *7*(1) (cit. on p. 92).

Stathopoulos, S., Serb, A., Khiat, A., Ogorzałek, M., & Prodromakis, T. (2019). A memristive switching uncertainty model. *IEEE Transactions on electron devices*, *66*(7), 2946–2953 (cit. on p. 105).

Stöckl, C., Lang, D., & Maass, W. (2021). Probabilistic skeletons endow brain-like neural networks with innate computing capabilities. *bioRxiv* (cit. on p. 43).

Stokes, M. G., Kusunoki, M., Sigala, N., Nili, H., Gaffan, D., & Duncan, J. (2013). Dynamic coding for cognitive control in prefrontal cortex. *Neuron*, *78*(2), 364–375 (cit. on p. 6).

Storrs, K. R., & Kriegeskorte, N. (2020). Deep Learning for Cognitive Neuroscience. In *The Cognitive Neurosciences*. The MIT Press. (Cit. on p. 1).

Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). The missing memristor found. *nature*, *453*(7191), 80–83 (cit. on p. 10).

Summerfield, C., Luyckx, F., & Sheahan, H. (2020). Structure learning and the posterior parietal cortex. *Progress in neurobiology*, *184*, 101717 (cit. on p. 69).

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, *27* (cit. on pp. 2, 7).

Tallec, C., & Ollivier, Y. (2018). Unbiased online recurrent optimization, In *International conference on learning representations*. (Cit. on p. 9).

Tartaglia, E. M., Mongillo, G., & Brunel, N. (2015). On the relationship between persistent delay activity, repetition enhancement and priming. *Frontiers in psychology*, *5*, 1590 (cit. on p. 46).

Teeter, C., Iyer, R., Menon, V., Gouwens, N., Feng, D., Berg, J., Szafer, A., Cain, N., Zeng, H., Hawrylycz, M., Et al. (2018). Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature communications*, *9*(1), 1–15 (cit. on p. 19).

Teichmann, L., Grootswagers, T., Carlson, T., & Rich, A. N. (2018). Decoding digits and dice with magnetoencephalography: Evidence for a shared representation of magnitude. *Journal of cognitive neuroscience*, *30*(7), 999–1010 (cit. on p. 69).

Tsao, A., Sugar, J., Lu, L., Wang, C., Knierim, J. J., Moser, M.-B., & Moser, E. I. (2018). Integrating time from experience in the lateral entorhinal cortex. *Nature*, *561*(7721), 57–52 (cit. on p. 36).

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–60 (cit. on pp. 1, 4).

Turrigiano, G. G., Marder, E., & Abbott, L. (1996). Cellular short-term memory from a slow potassium conductance. *Journal of neurophysiology*, *75*(2), 963–966 (cit. on p. 17).

Valov, I., & Kozicki, M. (2017). Organic memristors come of age. *Nature materials*, *16*(12), 1170–1172 (cit. on p. 92).

Van der Velde, F., & De Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, *29*(01), 37–70 (cit. on p. 73).

Wang, J., Xu, Q., Yuan, B., Chen, S., Yu, B., & Wu, F. (2020). Reliability-driven neural network training for memristive crossbar-based neuromorphic computing systems, In *2020 ieee international symposium on circuits and systems (iscas)*. IEEE. (Cit. on p. 109).

Wang, X.-J. (1998). Calcium coding and adaptive temporal computation in cortical pyramidal neurons. *Journal of Neurophysiology* (cit. on p. 17).

Wang, Y., Markram, H., Goodman, P. H., Berger, T. K., Ma, J., & Goldman-Rakic, P. S. (2006). Heterogeneity in the pyramidal network of the medial prefrontal cortex. *Nature neuroscience*, *9*(4), 534 (cit. on pp. 30, 47, 50).

Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (cit. on pp. 29, 115).

Wasmuht, D. F., Spaak, E., Buschman, T. J., Miller, E. K., & Stokes, M. G. (2018). Intrinsic neuronal dynamics predict distinct functional roles during working memory. *Nature communications*, *9*(1), 3499 (cit. on pp. 43, 113, 114).

Weber, A. I., & Fairhall, A. L. (2019). The role of adaptation in neural coding [Computational Neuroscience]. *Current Opinion in Neurobiology*, *58*, 135–140 (cit. on pp. 7, 17).

Weber, A. I., Krishnamurthy, K., & Fairhall, A. L. (2019). Coding principles in adaptation. *Annual Review of Vision Science*, *5*(1), 427–449 (cit. on pp. 7, 17).

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, *1*(4), 339–356 (cit. on pp. 8, 20, 48).

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, *1*(2), 270–280 (cit. on p. 9).

Winters, B. D., Saksida, L. M., & Bussey, T. J. (2008). Object recognition memory: Neurobiological mechanisms of encoding, consolidation and retrieval. *Neuroscience & Biobehavioral Reviews*, *32*(5), 1055–1070 (cit. on p. 40).

Wolff, M. J., Jochim, J., Akyürek, E. G., & Stokes, M. G. (2017). Dynamic hidden states underlying working-memory-guided behavior. *Nature Neuroscience*, *20*(6), 864 (cit. on pp. 6, 24, 25, 39, 40, 52).

Woźniak, S., Pantazi, A., Bohnstingl, T., & Eleftheriou, E. (2020). Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, *2*(6), 325–336 (cit. on pp. 94, 106).

Xia, L., Liu, M., Ning, X., Chakrabarty, K., & Wang, Y. (2017). Fault-tolerant training with on-line fault detection for rram-based neural computing systems, In *Proceedings of the 54th annual design automation conference 2017*. (Cit. on pp. 94, 99, 109).

Xia, Q., & Yang, J. J. (2019). Memristive crossbar arrays for brain-inspired computing. *Nature materials*, *18*(4), 309–323 (cit. on p. 92).

Yamins, D. L., & DiCarlo, J. J. (2016). Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, *19*(3), 356–365 (cit. on p. 2).

Yao, P., Wu, H., Gao, B., Tang, J., Zhang, Q., Zhang, W., Yang, J. J., & Qian, H. (2020). Fully hardware-implemented memristor convolutional neural network. *Nature*, *577*(7792), 641–646 (cit. on p. 92).

Zamarian, L., Ischebeck, A., & Delazer, M. (2009). Neuroscience of learning arithmetic—evidence from brain imaging studies. *Neuroscience & Biobehavioral Reviews*, *33*(6), 909–925 (cit. on pp. 71, 76).

Zenke, F., & Ganguli, S. (2018). Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, *30*(6), 1514–1541 (cit. on p. 9).

Zenke, F., & Vogels, T. P. (2020). The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *BioRxiv* (cit. on p. 117).

Zhang, C., Gao, F., Jia, B., Zhu, Y., & Zhu, S.-C. (2019). Raven: A dataset for relational and analogical visual reasoning, In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. (Cit. on p. 5).

Zhang, W., Gao, B., Tang, J., Yao, P., Yu, S., Chang, M.-F., Yoo, H.-J., Qian, H., & Wu, H. (2020). Neuro-inspired computing chips. *Nature electronics*, *3*(7), 371–382 (cit. on pp. 9–11).

Zhou, G., Wang, Z., Sun, B., Zhou, F., Sun, L., Zhao, H., Hu, X., Peng, X., Yan, J., Wang, H., Et al. (2022). Volatile and nonvolatile memristive devices for neuromorphic computing. *Advanced Electronic Materials*, *8*(7), 2101127 (cit. on p. 10).

Zylberberg, A., Paz, L., Roelfsema, P. R., Dehaene, S., & Sigman, M. (2013). A neuronal device for the control of multi-step computations. *Papers in physics*, *5*(2), 1–15 (cit. on p. 72).