Modeling and Optimization in Science and Technologies

Alex Pappachen James *Editor*

Deep Learning Classifiers with Memristive Networks

Theory and Applications

Springer

Modeling and Optimization in Science and Technologies

Volume 14

The book series Modeling and Optimization in Science and Technologies (MOST) publishes basic principles as well as novel theories and methods in the fast-evolving field of modeling and optimization. Topics of interest include, but are not limited to: methods for analysis, design and control of complex systems, networks and machines; methods for analysis, visualization and management of large data sets; use of supercomputers for modeling complex systems; digital signal processing; molecular modeling; and tools and software solutions for different scientific and technological purposes. Special emphasis is given to publications discussing novel theories and practical solutions that, by overcoming the limitations of traditional methods, may successfully address modern scientific challenges, thus promoting scientific and technological progress. The series publishes monographs, contributed volumes and conference proceedings, as well as advanced textbooks. The main targets of the series are graduate students, researchers and professionals working at the forefront of their fields.

Indexed by SCOPUS. The books of the series are submitted for indexing to Web of Science.

More information about this series at http://www.springer.com/series/10577

Alex Pappachen James
Editor

Deep Learning Classifiers with Memristive Networks

Theory and Applications

Springer

Editor
Alex Pappachen James
School of Engineering
Nazarbayev University
Astana, Kazakhstan

ISSN 2196-7326 ISSN 2196-7334 (electronic)
Modeling and Optimization in Science and Technologies
ISBN 978-3-030-14522-4 ISBN 978-3-030-14524-8 (eBook)
https://doi.org/10.1007/978-3-030-14524-8

Library of Congress Control Number: 2019932719

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Dedicated to the aspiring deep learning neuro-memristive systems engineers

Preface

This book focuses on providing insights into the emerging area of deep neural network classifiers and their implementations with memristive accelerators. The emerging memristor arrays and their implementations for analog computations such as for dot-product operations form the foundations for building analog neural network accelerators. The increase in many layers of the neural networks and the number of neurons pose the need to accelerate the computations. In this book, several chapters are included that provide a background on the deep learning concepts and how it can be build using memristive systems.

The book is suitable for MS, Ph.D. students, and practitioners such as those specifically interested in deep learning neuro-memristive systems. Emulating the neural network classifiers in hardware requires a cross-disciplinary understanding of computing and electronic circuits. In essence, this book provides a bridge between the architecture of different neural networks and its implementations in analog memristive circuits.

Part I of the book provides a fundamental overview of deep learning, memristors, and examples of system-level application. Part II of the book focuses on the neuro-memristive hardware implementations. Several foundational neural networks such as convolutional neural networks, long short-term memory, hierarchical temporal memory, and neuro-fuzzy architectures are introduced. These lay foundations to building larger deep neural network architectures such as that useful for generative adversarial networks, which are growing popularity in neural networks community.

I would like to acknowledge the support of several graduate students and researchers from Neuromorphic Circuits and System group, Nazarbayev University, in the development of this book.

Astana, Kazakhstan
Decemeber 2018

Alex Pappachen James

Contents

Contributors

Kazybek Adam Nazarbayev University, Astana, Kazakhstan

Adilya Bakambekova Nazarbayev University, Astana, Kazakhstan

Yeldos Dauletkhanuly Nazarbayev University, Astana, Kazakhstan

Irina Dolzhikova Nazarbayev University, Astana, Kazakhstan

Anuar Dorzhigulov School of Engineering, Nazarbayev University, Astana, Kazakhstan

Aziza Duisembay Nazarbayev University, Astana, Kazakhstan

Aidana Irmanova Nazarbayev University, Astana, Kazakhstan

Akzharkyn Izbassarova Nazarbayev University, Astana, Kazakhstan

Alex Pappachen James School of Engineering, Nazarbayev University, Astana, Kazakhstan

Olga Krestinskaya Nazarbayev University, Astana, Kazakhstan

Akshay Kumar Maan Nazarbayev University, Astana, Kazakhstan

Serikbolsyn Myrzakhmet Nazabayev University, Astana, Kazakhstan

Damira Pernebayeva Nazarbayev University, Astana, Kazakhstan

Kamilya Smagulova Nazarbayev University, Astana, Kazakhstan

Yeldar Toleubay Nazarbayev University, Astana, Kazakhstan

Acronyms

ANN	Artificial neural network
ASIC	Application-specific integrated circuits
CMOS	Complementary metal–oxide–semiconductor
CNN	Convolutional neural network
CPU	Central processing unit
DNN	Deep learning neural network
DRM	Dynamic route map
FPGA	Field-programmable gate array
GPU	Graphical processing unit
HTM	Hierarchical temporal memory
LSTM	Long short-term memory
MSE	Mean square error
MTL	Memristive threshold logic
NN	Neural network
POP	Power-off plot
RMSE	Root mean square error
RNN	Recurrent neural network
STDP	Spike timing-dependent plasticity
TLG	Threshold logic gates
VLSI	Very large scale integration
VMM	Vector matrix multiplication

Part I
Foundations and System Applications

Chapter 1
Introduction to Neuro-Memristive Systems

Alex Pappachen James

Abstract This chapter provides with an overview of the motivation and direction for neuro-memristive computing hardware. The emergence of deep learning technologies has been largely attributed to the convergence in the growth on computational capabilities, and that of the large availability of the data resulting from Internet of things applications. The need to have higher computational capabilities enforces the need to have low power solutions and smaller devices. However, the physical limits of CMOS device and process technologies pushed us in the recent years to think beyond CMOS era computing. A promising solutions is a class of emerging devices called memristors, that can naturally blend as a viable computing device to implement neural computations that extend the capabilities of exiting computing hardware. The full potential of neuro-memristive systems is yet to be completely realised and could provide ways to develop higher level of socially engineered machine cognition.

1.1 Information Processing in Human Brain

Brain consists of a complex recurrent neural network [22]. The ability of human brain to process complex information and make decisions has been the inspiration of developing modern deep learning algorithms [7]. The quest for modelling neural networks for information processing is very old, and largely the models have been derived from approximations of dynamic systems of human brain [18]. Recurrent neural networks are more likely to reflect this dynamic systems behaviour of human brain. A neural chip that can incorporate these neural networks to create an intelligent silicon brain has been a dream set forward from science fiction movies to the neuromorphic scientist for long.

Biological neural networks are extremely complex to mimic in the entirety of the human brain [3, 19]. The term neural network in the computational context is referred to as the networked connection of computational units that are most

A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_1

simplistic abstraction of biological neuron. It is also the class of models that can perform parallel information processing, taking advantage of data for training these non-linear functions. The simplistic models often ignore the dendritic computations and other complex computations which are common to a real biological neuron [33]. Even with such simplicity, these models are used to build large deep learning neural networks that has a growing appeal and use in modern artificial intelligence applications [28].

1.1.1 Basic Models

Figure 1.1 shows the basic model transition from biological neuron to an artificial neural network build with artificial neurons, that can be realised with memristor [5] arrays. The early forms of neuron models heavily relied on threshold logic [21], where the firing of each neuron is translated as weighted summation of inputs followed by a hard threshold activation function. The hard threshold function roughly correlate with the spiking nature of biological neurons, which is a signature characteristic for cognitive processing. They also find application in generalised design of logic gates, which is relevant even today [10, 20]. Several combinations of such neurons are interconnected between layers to form a neural network of threshold logic neurons, which can be trained using a backpropagation algorithm.

The core ideas of modelling neural networks have drawn inspirations from the structural and functional modularity and hierarchy of information processing mechanisms of human brain. The ability of brain to process sparse information is also seen as a key aspect of information processing that leads to efficient and reliable computing in natural settings. While, the modularity and hierarchy has be questioned in neuroscience on several accounts, largely this assumption holds true for information processing and modelling practical real-world scenarios and problems.

Learning such as using backpropagation in the early neural network models did not perform well in high dimensional data processing such as in imaging and speech recognition problems. This has been largely attributed to the limitations on testing for optimisation of the hyperparameters and the neural network size, both in terms of number of layers and number neurons within a layer. The performance failures

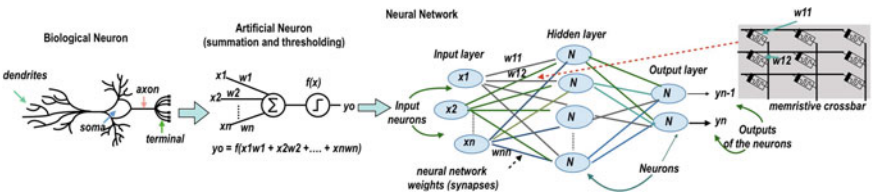

Fig. 1.1 Illustration shows the evolution of biological inspiration from neuron to artificial neuron network

of neural networks effectively shifted the machine learning community to drift the focus from neural networks to more hand-engineered models for solving speech and imaging problems such as support vector machines [27] and hidden markov models [34]. Even though, the larger machine learning community did shift the attention from neural networks, there was still a small and growing community that keep the field alive exploring several architectures both from computational neuroscience perspective and that from computer applications perspective. The quest for building a neuro-chip was even further limited to much smaller community of analog circuit designers and neuromorphic engineers. However, in the last decade, the fast paced progressive developments in electronics, computing and communication technologies have triggered the growth of data driven information processing, where neural networks due to its simplicity in comparison with hand-engineered models stands out as a winner. This has further triggered the development of emerging hardware and computing technologies for implementing larger, energy efficient and faster neural networks for solving complex practical problems related to artificial intelligence.

1.1.2 Generalisation Abilities

The generalised neural network in its simplest form has a input layer, a hidden layer and an output layer. Any neural network with more than one hidden layer is generally known as deep neural network. We can call the three layer feedforward neural network as a shallow neural network. These shallow networks with non-linear activation functions are generalised universal function approximations, that can be used to solve complex classification task. Increasing the number of neurons in hidden layer, increases the number of interconnections between the neurons in the network allowing for highly complex function approximation.

The number of neurons required to approximate a given function in a shallow neural networks can be redistributed across several layers to form the deep neural network. It can be shown that for the same number of neurons, a deep network can approximate more number of functions, making deep neural networks computationally more efficient than shallow neural networks. The expressive power of deep neural networks are much more than shallow neural networks [23, 25].

In computer vision problems, the deep learning neural networks such as convolution neural networks correlate with the deep nature of the visual cortex [17]. This similarity draws further intuitions on the organisation of the neural networks, and how such abstractions can help attain efficient information processing similar to human brain. It can be noted that fine tuning on the number of layers, number of neurons per layer, activation functions and correct selection of weights are required is dependent on the application at hand. Along with providing realistic solutions to practical artificial intelligence problems such abstractions could be also loosely connected to the functional properties of human brain.

As opposed to deep neural networks that are largely feedforward neural networks that often works on static inputs, the deep recurrent neural networks is more closer representative of dynamic behaviour of human brain functions that takes into account the information from past and present inputs. Human brain can be also seen as a complex memory system, where the separation between the computation and memory is often non-existent. In such networks, feedback is essential, where the outputs of the neural networks are fed back as inputs in a given time. The inputs of such neural networks have the inputs in the present time, and the neural networks outputs generated from the inputs in the past times. It is hence a system with temporal evolution of states similar to biological neural networks in the brain.

The recurrent neural networks are useful to capture time dependent information. In fact, almost all the sensory information received by the human brain is time dependent, and hence all the real-time systems of artificial intelligence can make use of the dynamic behaviour approximations of the recurrent neural networks. The recurrent neural networks combines the feedforward neural networks with that of the feedback, and can be implemented with different architectures. In essence, these recurrent neural networks are also deep neural networks, with the additional abilities for dynamic system approximations using data patterns across time and memory. The most common architecture variants of recurrent neural networks include gated recurrent neural network, and long short term memory.

The feedforward neural networks and recurrent neural networks can be used to build higher order functions. Abstraction is the key concept in computational neuroscience and that in computer engineering for building higher order logic and architectures resembling brain functions. One such interesting set of abstraction is the generative and adversarial networks [6], that consists of competing neural networks for revealing idiosyncrasies properties of neural networks. Such higher order networks are complex to train and optimisation of such networks a challenging task.

1.2 Hardware Driven AI

Edge devices such as mobile phones are the primary drivers of change that engages and enlarges the diversity of the users, making the applications of deep learning and artificial intelligence more widespread and popular. The trust in the technologies is vital as its application area to ensure the user acceptability. Over the last decades, the the trust in internet and allied technologies [8, 30] have gradually increased from an application point of view, reflected by increase in technology users and governments around the world, penetrating slowly into every aspect of our lives.

The advances in computing hardware opened up several new doors for the neural networks community. In addition, stretching the limits of Moore's law, the electronic chips have become smaller, energy efficient and significantly more powerful in the last decade. This elevated the capabilities of processing and testing large scale neural networks with large scale data, outperforming every other known machine learning techniques offering the advantage of no need for hand-engineering.

The resurgence of multi-layered deep neural networks post 2015 can be attributed to the exponential increase in sensory data due to growth in internet of things markets. The more number of edge devices such as mobiles, drones etc, the more data is generated. Millions of data points are captured from these devices every second with relatively very little volume of intelligent information processing. The potential of extracting meaningful information from connected data from the sensors can lead to applications that can change the progress and history of humans.

The integration of memories with computing is a key concept of neural networks [13]. This principle has been explored for in memory computing and in the hardware implementations of neural networks. The smaller the size of memory cells, larger the memory capacity per unit area on chip, increasing the computing capabilities of the neural network implementations. There are different ways to implement a neural network on chip. The most obvious approach today is to convert the signals to digital domain and perform the weight summation of inputs using a set of digital logic gates, and digital memories for storing weights. Given that each weight of the neural networks are often in essential analog, the digital implementations are always approximations of the analog, and require larger number of electronic components (transistors) in comparison to an analog memory implementation.

The weight summation of inputs is a dot product operation that is natural computational model in the crossbar memory. There are several memories that can be used for building a crossbar, one of the most emerging set of memories are non-volatile memristors. There are numerous implementations of memristors and as of yet there are no industry standard that is reliable for a widespread use. However, with the recent success of 3DXpoint memory[4], PCM devices [9] and ReRAM [2], it is not very far that family of devices in memristor class, will be matured enough to the produced in mass, and possible will be available for use in industry fabrication settings to be integrated with CMOS and FINFET devices.

1.3 Towards Neuro-Memristor Networks

Figure 1.2 shows an abstract high level view of the applications that use neuro-memristive networks. The crossbar architecture that is useful as a memory array is at the core of many of the neural network circuit designs, as its able to emulate dot-product computations. There are several reasons that makes memristors interesting for building neural networks. On hardware perspective, memristors [16] can be scaled down to less than 10nm making it area efficient two terminal device connected between two metal wires. Ideally, they can be used as non-volatile analog memory suitable to store weights in a neural network. they are also energy efficient as compared with other CMOS based memories due to the possibility to use low voltage and currents. They have high read speeds, although the write speeds can be much slower.

The main characteristic of a memristor can be represented with its unique pinched IV hysteresis characteristics, power off plot and dynamic route maps. The IV charac-

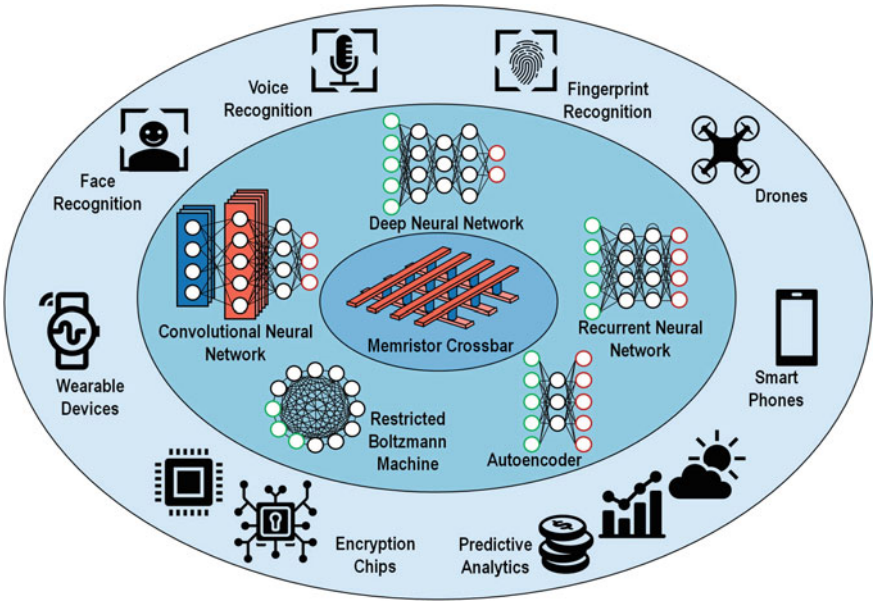

Fig. 1.2 Illustration shows the landscape of applications using memristive networks such as using crossbar array

teristics of the memristors are obtained by applying AC voltage of a given frequency across the memristor and observing the current signal outputs flowing through them. One has to be very careful with this, that if a memristor is reported to have a DC pinched hysteresis IV they are wrong and falsely classified as memristor. Along with the IV characteristic its also important to look at power off plot, where if the rate of change relative to a state variable is zero, it represents a non-volatile memristor. The dynamic route maps provides a simplistic way to look at the switching behaviour of memristor for various frequency and amplitude of voltage pulses applied to the memristor.

There are a large class of devices that fall under the category of memristors [11] such as magnetic or MRAM, phase-change or PCRAM, and resistive or ReRAM that can be used for building neural networks. The memristive crossbar arrays has been at the heart of such neural network designs, where the output currents in each column read from the crossbar can be represented as the weighted summation of input voltages and conductance of connected memristors. This can be mapped as the weighted summation of inputs, or dot-product operations that is fundamental computational process in any neural networks. This idea of replacing dot-product computations with crossbar opens area and energy efficient way for building neural network layer in analog domain. Several neural networks build with such arrays are discussed in the this book, such as deep neural network [14], convolutional neural network [15], long short term memory [31], deep neuro-fuzzy networks, and

hierarchical temporal memory [12]. Often, in the design of analog system, it is also required to use digital logic such as for programming the memristors. An alternative approach to CMOS logic is to use memristor threshold logic gates, that are much more area efficient and generalised structures of universal following the early models of neurons. Array of programmable memristive gates can be used to build any sequential or combinational logic as required for preprocessing and control circuits in the neural network system.

There is a growing set of applications of using memristors in neuromorphic systems. The asynchronous Spike-Timing-Dependent-Plasticity (STDP) using memristors as synapses is one of closest to spike based biologically plausible neural network learning models [1, 29]. The fully synchronous learning memristor circuits performs the weight update and processing happens at the same time. Some of the other models that incline strongly towards mimicking biology with a more practical focus include long short term memory [31], and hierarchical temporal memory [12]. Since these techniques are variants of approximating dynamic systems they can find variety of applications such as not limited to predictive analysis, control systems, speech recognition and natural language processing.

The neuro-memristive systems can find applications in a range of edge AI computing devices. The ability to locally process the information from the sensors, without consuming large communication bandwidth can increase the processing speeds and accuracy of edge computing applications [13]. The neuro-memristive chips can be seen as hardware accelerators in these applications. The memristors in crossbar array can be used as a memory [32], and also for applications such as for security [24, 26]. These crossbar arrays can be also used for imaging and signal processing applications.

1.4 Grand Future of Neurochips

It will be incomplete not to discuss ethical issues in hardware driven AI chips. The ultimate goal of these chips is to mimic all the functionalities of the human brain, including but not limited to pattern matching, contextual intelligence, consciousness, and sub-conscious thinking. The sensors along with communication and computing technologies can provide a vast amount of information to the AI chips at an extremely high speed unseen to the human brain. The machine learning algorithms that work with such neural chips would require to be energy efficient and be able to extract new information from the vast amount of unseen data.

A fully functional brain-like AI chips require us to uncover the brain circuits through computational neuroscience and neuromorphic engineering. The ability to think is ingrained into the function of a brain, and such chips can develop machine memories that reflect the rights of morality and equality. This also could significantly change the laws and policy frameworks that we have in using intelligent devices. Privacy and identity issues of devices, coupled with self-awareness can drive a new

era of electronic evolution, including developing a social framework around artificial life.

Social networks are widespread and primarily driven by human intelligence, and some point in the history ahead, it is likely that machine intelligence due to the internet of everything can outweigh the abilities exhibited by human interactions. The neurochips make the devices talk to each other in more intelligent ways, and can provide a collaborative environment for the machines to digitally evolve. Games and competitive learning strategies can make machines complete to excel in a specific task, making some machines smarter and better than others on a given task. This can also impact the behavior of the machines, and we might have to develop a socially engineered machine psychologist and therapy algorithms to reconfigure and rewrite the hardware circuits.

Human culture and history are incomplete without developments in art and philosophy. The nature and forms of arts derive from the intricate interactions between different levels of cognitive processes, which could be eventually mimicked with neurochips. The ability of these neurochips along with autonomous robotic systems that can manipulate physical space can further create expressive art forms such as dances, drama, and paintings. The neurochips can also manipulate and excel possibly in poetry and storytelling, beating human intelligence in pace and rigor. All these exciting possibilities and beyond can be anticipated, and time will only tell when they occur. Given that many of these forms of machine intelligence already exists in simple form, it is only fair to say that—"the future is happening."

1.5 Conclusion

Neuro-memristive systems offer the possibility to scale the implementation of high density large scale neural networks in integrated chips. The emulation of neural networks with memristor circuits offers several hardware advantages such as area, low power and near-sensor analog computations over the conventional CMOS only logic. The analog dot-product computations with memristive crossbar arrays can be mapped to the neural network layers, which in turn can be used for building a range of deep neural networks architectures. The very large scale implementation of neuro-memristor systems offers a range of challenges and opportunities to build several data driven practical neural network applications that can be integrated into wearable, mobiles, edge and analog computing devices.

Chapter Highlights

- Generalization and expressive power of deep neural networks outweigh the simplicity of shallow neural networks.
- The crossbar memristive arrays function as a dot product operator useful for performing weight summation of inputs in a neural network layer.
- Implementations of memristor based neural computing is area and power efficient, and provides the possibility to perform analog computations of large scale neural networks on chip.
- The stability of memristor devices, process reliability and maturity for industrial manufacturing, packaging issues and accuracy of very large scale circuit simulations will be the key factors for the success of neuro-memristive chips.

References

1. Agrawal A, Roy K (2019) Mimicking leaky-integrate-fire spiking neuron using automotion of domain walls for energy-efficient brain-inspired computing. IEEE Trans Magn 55(1):1–7
2. Akinaga H, Shima H (2010) Resistive random access memory (reram) based on metal oxides. Proc IEEE 98(12):2237–2251
3. Amit DJ, Amit DJ (1992) Modeling brain function: the world of attractor neural networks. Cambridge University Press, Cambridge
4. Bourzac K (2017) Has intel created a universal memory technology?[news]. IEEE Spectr 54(5):9–10
5. Chua L (2018) Five non-volatile memristor enigmas solved. Appl Phys A 124(8):563
6. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680
7. Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning, vol 1. MIT Press, Cambridge
8. Grandison T, Sloman M (2000) A survey of trust in internet applications. IEEE Commun Surv Tutor 3(4):2–16
9. Guo X, Ipek E, Soyata T (2010) Resistive computation: avoiding the power wall with low-leakage, STT-MRAM based computing. In: ACM SIGARCH computer architecture news, vol 38. ACM, pp 371–382
10. Hurst S (1969) An introduction to threshold logic: a survey of present theory and practice. Radio Electron Eng 37(6):339–351
11. Jeong H, Shi L (2018) Memristor devices for neural networks. J Phys D: Appl Phys 52(2):023003
12. Krestinskaya O, Dolzhikova I, James AP (2018) Hierarchical temporal memory using memristor networks: a survey. IEEE Trans Emerg Top Comput Intell 2(5):380–395. https://doi.org/10.1109/TETCI.2018.2838124
13. Krestinskaya O, James AP, Chua LO (2019) Neuro-memristive circuits for edge computing: a review. IEEE Trans Neural Networks Learn Syst (https://doi.org/10.1109/TNNLS.2019.2899262). arXiv:1807.00962

14. Krestinskaya O, Salama KN, James AP (2018) Learning in memristive neural network architectures using analog backpropagation circuits. IEEE Trans Circuits Syst I: Regul Pap 1–14. https://doi.org/10.1109/TCSI.2018.2866510
15. Krestinskaya O, Bakambekova A, James AP (2019) Amsnet: analog memristive system architecture for mean-pooling with dropout convolutional neural network. In: IEEE international conference on artificial intelligence circuits and systems
16. Li Y, Wang Z, Midya R, Xia Q, Yang JJ (2018) Review of memristor devices in neuromorphic computing: materials sciences and device challenges. J Phys D: Appl Phys 51(50):503002
17. Liao Q, Poggio T (2016) Bridging the gaps between residual learning, recurrent neural networks and visual cortex. arXiv:1604.03640
18. Lippmann R (1987) An introduction to computing with neural nets. IEEE ASSP Mag 4(2):4–22
19. Ma J, Tang J (2017) A review for dynamics in neuron and neuronal network. Nonlinear Dyn 89(3):1569–1578
20. Maan AK, Jayadevi DA, James AP (2017) A survey of memristive threshold logic circuits. IEEE Trans Neural Netw Learn Syst 28(8):1734–1746
21. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133
22. Medsker L, Jain LC (1999) Recurrent neural networks: design and applications. CRC Press, Boca Raton
23. Mhaskar H, Liao Q, Poggio T (2016) Learning functions: when is deep better than shallow. arXiv:1603.00988
24. Nili H, Adam GC, Hoskins B, Prezioso M, Kim J, Mahmoodi MR, Bayat FM, Kavehei O, Strukov DB (2018) Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors. Nat Electron 1(3):197
25. Raghu M, Poole B, Kleinberg J, Ganguli S, Dickstein JS (2017) On the expressive power of deep neural networks. In: Proceedings of the 34th international conference on machine learning-volume 70. pp 2847–2854. https://www.JMLR.org
26. Rajendran J, Rose GS, Karri R, Potkonjak M (2012) Nano-ppuf: a memristor-based security primitive. In: IEEE computer society annual symposium on VLSI (ISVLSI). IEEE, pp 84–87
27. Schlkopf B, Smola AJ, Bach F (2018) Learning with kernels: support vector machines, regularization, optimization, and beyond
28. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
29. Serrano-Gotarredona T, Masquelier T, Prodromakis T, Indiveri G, Linares-Barranco B (2013) STDP and STDP variations with memristors for spiking neuromorphic learning systems. Front Neurosci 7:2
30. Sicari S, Rizzardi A, Grieco LA, Coen-Porisini A (2015) Security, privacy and trust in internet of things: the road ahead. Comput Netw 76:146–164
31. Smagulova K, Krestinskaya O, James AP (2018) A memristor-based long short term memory circuit. Analog Integr Circuits Signal Process 95(3):467–472
32. Talati N, Ha H, Perach B, Ronen R, Kvatinsky S (2019) Concept: a column oriented memory controller for efficient memory and PIM operations in RRAM. IEEE Micro
33. Ujfalussy BB, Makara JK, Branco T, Lengyel M (2015) Dendritic nonlinearities are tuned for efficient spike-based computations in cortical circuits. Elife 4:e10056
34. Varga A, Moore R (1990) Hidden Markov model decomposition of speech and noise. In: 1990 International conference on acoustics, speech, and signal processing, ICASSP-90. IEEE, pp 845–848

Chapter 2
Memristors: Properties, Models, Materials

Olga Krestinskaya, Aidana Irmanova and Alex Pappachen James

Abstract The practical realization of neuro-memristive systems requires highly accurate simulation models, robust devices and validations on device characteristics. This chapter covers the basics of memristor characteristics, models and a succinct review of practically realized memristive devices. Memristors represent a class of two terminal resistive switching multi-state memory devices that can be compatible with existing integrated circuit technologies. The modeling of memristors for very large scale simulations requires to accurately capture process variations and other non-idealities from real devices for ensuring the validity of deep neural network architecture designs with memristors.

2.1 Introduction

Memristor is the fourth fundamental element proposed by Leon Chua in 1971 [25], in addition to the resistor, capacitor, and inductor. The main characteristic of memristive devices is a pinched hysteresis loop. Earlier in 1939 and 1962, the hysteresis behavior has been noticed in Metal-insulator-Metal structured devices [18, 44], and the first pinched hysteresis loop was shown in 1971 [60]. After this, the research about memristive devices was stopped and the concept of memristors and memristive material was not developing till early 2000s [10]. Memristors are known for their non-volatility property, the ability to switch and store the resistive state, even when the power is removed [26]. One of the earliest memristive devices have been fabricated in HP Lab using titanium dioxide [110, 117]. The theory about memristors relies on the dependency between flux and charge, which was not covered by any

O. Krestinskaya · A. Irmanova · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

O. Krestinskaya
e-mail: okrestinskaya@nu.edu.kz

A. Irmanova
e-mail: aidana.irmanova@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_2

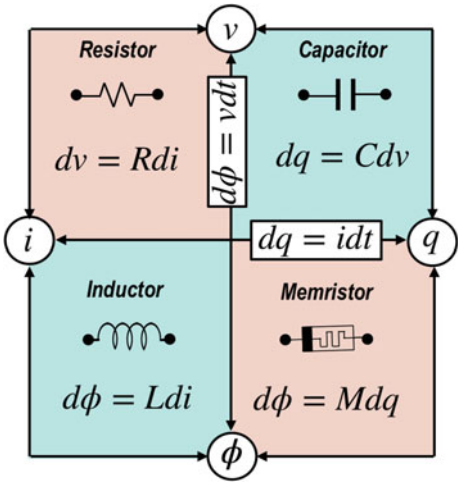

Fig. 2.1 Four fundamental elements representing the dependency between current, voltage, charge and flux [110]

other fundamental element. Four fundamental elements and dependency between current, voltage, flux and charge are illustrated in Fig. 2.1.

Memristive devices have broad application scope from memory arrays [31, 54] and logic gates [70, 80] to neural networks [53] and complex learning systems [66–68]. Memristor has a small on-chip area and low power dissipation; therefore, the memristor is a promising solution for scalability problems of large and complex systems, like neural networks. As the research on memristive devices and their applications gained much attention in recent years, there are different memristor models and materials proposed in recent years. In this chapter, Sect. 2.2 presents the leading theory about memristive devices. There are several memristor models, which emulate the behavior of ideal and non-ideal devices. These models are covered in Sect. 2.3. In addition, there is many different materials exhibiting memristive behavior, discussed in Sect. 2.4.

2.2 Memristor Theory

Memristor is a relatively novel element, and the theory about memristive devices is still evolving. Recently, L. Chua illustrated the solution for five non-volatile memristor enigmas [27]. These five enigmas formulated by L. Chua are the following [27]:

1. "*All non-volatile memristors have continuum memories.*"
2. "*Conductance of all non-volatile memristors can be tuned by applying single voltage pulses.*"
3. "*Faster switching can always be achieved by increasing the pulse amplitude.*"

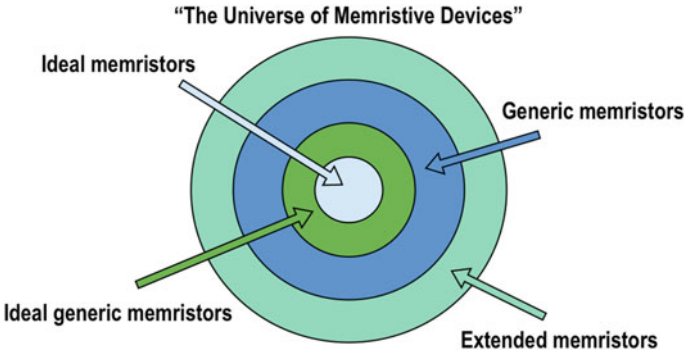

Fig. 2.2 Venn diagram of memristors [27]

4. "*Periodic unipolar input gives non-periodic finger-like multi-prong-pinched hysteresis loops.*"
5. "*DC VI curves of non-volatile memristors are fakes.*"

The Venn diagram for generalization of all memristive devices shown by L. Chua in [27] is illustrated in Fig. 2.2. According to L. Chua, there are four different classes of memristors: ideal, generics ideal, generic and extended memristors. The conductance and memristance values of a memristor depend on one or more state variables. The equations for ideal current and voltage controlled memristors are shown in Eqs. 2.1 and 2.2.

$$v = R(q)i, \ \ where \ \frac{dq}{dt} = i \tag{2.1}$$

$$i = G(\phi)v, \ \ where \ \frac{d\phi}{dt} = v \tag{2.2}$$

The equations for ideal generic current and voltage controlled memristors are shown in Eqs. 2.3 and 2.4. The ideal and ideal generic memristors are shown in HP memristor models [27].

$$v = R(x)i, \ \ where \ \frac{dx}{dt} = \hat{f}(x)i \tag{2.3}$$

$$i = G(x)v, \ \ where \ \frac{dx}{dt} = g\hat{(x)}v \tag{2.4}$$

However, most of the real memristive devices are generic and extended memristors, having a number of limited stable states. Equations 2.5 and 2.6 show the equations for generic memristors. These are state-dependent memristors depending, where state variable depends on two variables.

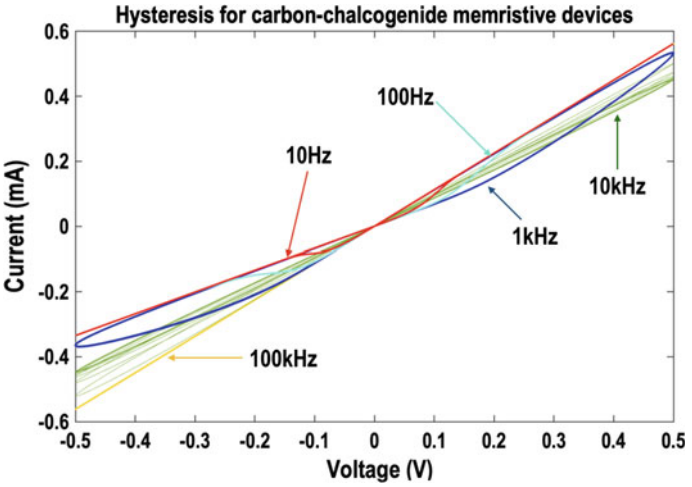

Fig. 2.3 IV characteristics of carbon-chalcogenide memristor model

$$v = R(x)i, \ \ where \ \frac{dx}{dt} = f(x, i) \tag{2.5}$$

$$i = G(x)v, \ \ where \ \frac{dx}{dt} = g(x, v) \tag{2.6}$$

In the extended memristors, the conductance depends on current. Equations 2.7 and 2.8 show the equations for extended memristors.

$$v = R(x, i)i, \ \ where \ R(x, 0) \neq \infty \ and \ \frac{dx}{dt} = f(x, i) \tag{2.7}$$

$$i = G(x, v)v, \ \ where \ G(x, 0) \neq \infty \ and \ \frac{dx}{dt} = g(x, v) \tag{2.8}$$

The main characteristics of a memristive device is a periodic-pinched hysteresis loop [21]. The example of which is shown in Fig. 2.3. Hysteresis curves with zero crossing is an IV characteristic of a memristor, showing the dependency of the current flowing through the device on the applied voltage. The example of the hysteresis loop for the carbon-chalcogenide memristive device is illustrated in Fig. 2.3. This hysteresis is simulated by applying a periodic sinusoidal signal to the device and measuring the current flowing through it. The frequency of the signal affects the conductance levels of the device. Usually, the number of these stable conductance levels is limited. The maximum number of stable states achieved in a single device is 64 levels presented in [75] for Ta/HfO_2 memristor.

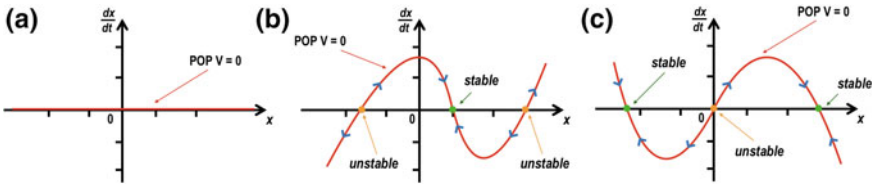

Fig. 2.4 POP for volatile and non-volatile memristors: **a** POP of non-volatile memristive device, **b** POP of volatile memristor and **c** POP of bistable memristor (binary memory) [27]

In the memristive circuits and architectures, the memristive state (conductance) of a memristor can be changed by applying positive and negative voltage pulses. These pulses can be of different width and amplitude depending on the properties of the memristor. In general, the signal with higher applied amplitude leads to faster switching between memristive states.

Most of the time in literature, when we mention the memristive device, we refer to non-volatile memristors. However, devices exhibiting memristive behavior can be divided into volatile and non-volatile [108]. The example of the non-volatile device is Resistive Random Access Memory (ReRAM), while an example of the volatile memristor is a Nanoparticle Organic Memory Field-Effect Transistor (NOMFET) [108].

The other important way to characterize the memristive behavior are Power-Of Plot (POP) and Dynamic Route Map (DRM) [27]. POP and DRM are the plots of derivative of memristor state variable versus state variable. Equation 2.9 is a "power-off" state equation.

$$\frac{dx}{dt} = f(x, 0) \tag{2.9}$$

POP is a continuous single-valued curve [27]. POP illustrates the non-volatility property of the memristor. The POP of non-volatile memristor is flat and for all state variables x, $dx/dt = 0$. Figure 2.4a illustrate POP for ideal non-volatile memristive device. Figure 2.4b illustrates POP of volatile memristor, and Fig. 2.4c shows POP of binary memory with only two stable states [27].

The concept of DRM of the memristive device is similar to POP but applies for the non-zero state variable. DRM contains an infinite number of dynamic routes; however, only a few routes are displayed to illustrate meaningful information. For memristive devices, DRM shows how fast the memristive state can be changed applying a certain current or voltage pulses. The derivative of state variable $\frac{dx}{dt} = f(x, v)$ is a continuous function. Figure 2.5 illustrates DRM for ideal and non-ideal memristors. DRM of ideal memristor in Fig. 2.5a consists of flat horizontal lines. However, all real devices have non-ideal DRM shown in Fig. 2.5b. DRM in Fig. 2.5 illustrates that if $v(t)$ is positive, $x(t)$ increases, and vice versa.

The state variable in POP and DRM plots can be different. For example, Fig. 2.6 illustrates POP and DRM for carbon-chalcogenide voltage-controlled memristor,

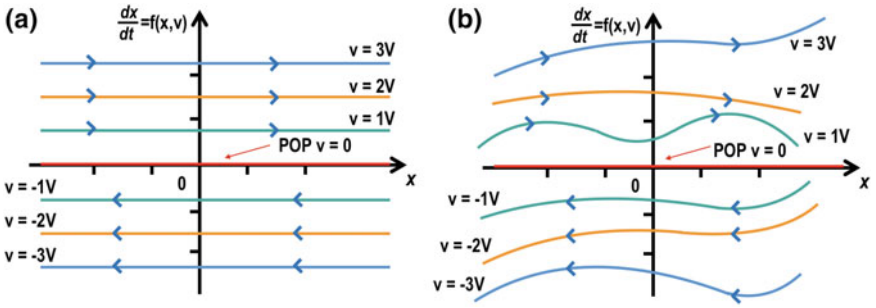

Fig. 2.5 DRM: **a** DRM of ideal memristor and **b** DRM of non-ideal memristor [27]

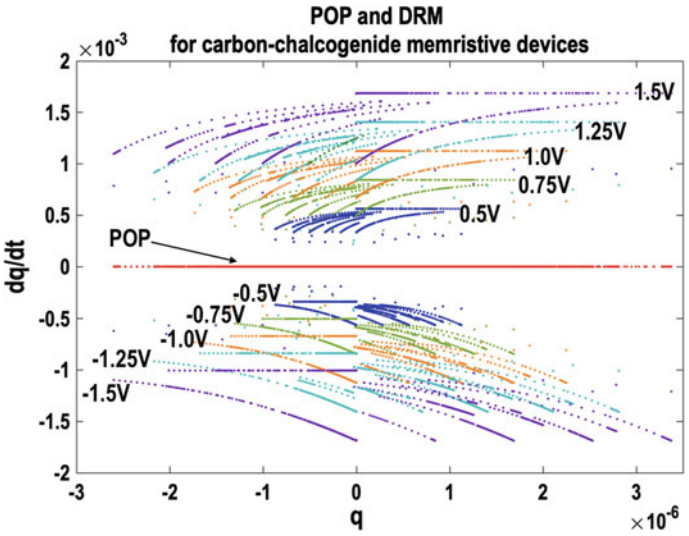

Fig. 2.6 POP and DRM of carbon-chalcogenide memristor model

where state variable is a charge on the x-axis, while the derivative of charge is on the y-axis. The POP and DRM is plotted applying voltage pulses of different frequency and amplitude to the memristor.

2.3 Memristor Models

Memristor model is one of the most essential and critical components for the simulation of memristive circuits and systems. The modeling of memristive devices is still a developing field, as new memristive devices are developed. Memristor models vary from ideal models for emulating the behavior of the ideal memristive device to

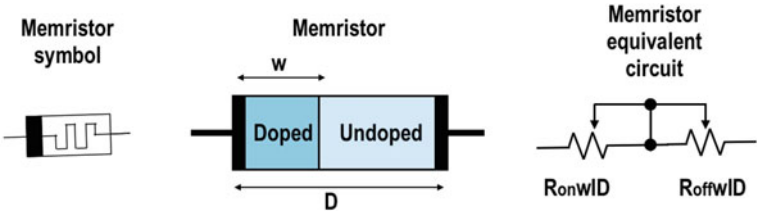

Fig. 2.7 Simplified equivalent memristor circuit [110]

the non-linear models allowing to fit parameters of different memristive devices [9, 16, 109]. This section covers different memristor models that can be used for the simulation of memristive architectures.

2.3.1 Linear Memristor Model

Linear memristor model is illustrated in [110]. The main equation in the model is based on the physical structure of the device [59] and width of the doped and undoped regions in the memristive device at a particular moment. The doped region is a region in the device with oxygen vacancies, which is doped with positive oxygen ions. Undoped region is a region without oxygen vacancies. The most simple model represents these two regions are separate resistors of high and low value. Depending on the width of the doped and undoped regions, the resistive state of a memristive device changes [59]. The main equation for this model is shown in Eq. 2.10, where $x(t) = w(t)/D$, D is a device width and $w(t)$ is a doped region width (conductive width [109]), R_{OFF} and R_{ON} are a high and low resistance levels of a memristor, respectively.

$$v(t) = (R_{ON} \times x(t) + R_{OFF}(1 - x(t))) \times i(t) \tag{2.10}$$

Each memristor model can be presented as an equivalent circuit. Figure 2.7 illustrates the linear memristor symbol, structure and equivalent circuit model of a memristor. Linear memristor models are not very accurate and do not consider the limitations and nonlinearity properties of real devices. These models are mostly used to simulate idealized memristor behavior to support theoretical expectations and to illustrate small signal response of ideal device [16].

2.3.2 Memristor Models Based on a Window Function

The memristor models with a window function in most of the cases emulate the non-linear behavior of the memristive device. The primary purpose of a window function is to restrict the bounds of memristor state. Same as for linear memristor model, the

models with window function do not show the non-idealities of real devices. These models can be used for the proof of concept in the ideal simulations and illustrating the behavior of ideal memristive devices in a design [16]. More recent work on the modification of window function is shown in [84].

2.3.2.1 Strukov's Linear Window

The linear window is proposed in Eq. 2.11 [110], where the boundary conditions for this function are $f(0) = 0$ and $f(D) = \frac{1-D}{D} \simeq 0$. Window function forces the derivative of state variable to reach 0, when this variable is at the bounds [59].

$$f(w) = w(D - w)/D \tag{2.11}$$

2.3.2.2 Joglekar's Model

In [55], the non-linear parameter is introduced to the window function. This window function is shown in Eq. 2.12, where p is a parameter that allows modeling nonlinear boundary conditions in a window function [67]. The increase of p leads to an increase of non-linearity.

$$f(x, p) = 1 - (2x - 1)^{2\times p} \tag{2.12}$$

However, this non-linear model is not accurate and cannot fully illustrate the non-linear behavior of memristive devices. Also, the main issue in this window function is when the model is at the boundaries, the state variable is difficult to be changed, which does not represent the behavior of real memristive devices [59].

2.3.2.3 Biolek's Memristor Model with Nonlinear Dopant Drift

Memristor model with non-linear dopant drift allowing to modify memristor boundary conditions is shown in Eq. 2.13, where $step(-i) = 0$ for $i < 0$ and $step(-i) = 1$ for $i \geq 0$ [17]. Comparing to Joglekar's model [55], Biolek's model includes current i directly into a state variable x [109].

$$f(x, i, p) = 1 - (x - step(-i))^{2\times p} \tag{2.13}$$

Biolek's model solved the problem of Joglekar's model is that the current is a state variable that reacts to the polarity of the applied voltage. State variable of a memristor can be changed by applying the voltage of opposite polarity, therefore, does not get stuck in a particular boundary, when it reaches it [109]. In other words, this window function controls the movement between doped and undoped regions at any boundary and is responsible for the speed of change of the memristive state when applying voltage pulses [67].

2.3.2.4 Prodromakis's Model

Prodromakis's window function for memristor model is proposed in [96]. In comparison to the previous function, this window function is scalable, and the height of a function can be adjusted [59]. This window function is presented in Eq. 2.14, where j is a parameter for controlling the height of the function. Also, the window function is represented as a quadratic equation, which allows better approximation of non-linearity of memristor behavior at the boundaries.

$$f(w) = j(1 - [(w - 0.5)^2 + 0.75]^p) \tag{2.14}$$

2.3.3 Memristor Models for Parameter Fitting

Memristor models, where the parameters of real devices can be fit are considered to be one of the accurate ways to simulate memristive circuits. In most of the cases, the linear models of particular window functions are not enough to show the imperfections of real devices. In most of the memristive devices, the hysteresis is not symmetrical, and the switching is non-linear. To emulate such behavior of a memristor, it is important to use the models, which show all the imperfections in the behavior of memristive devices. According to [16], parameter fitting models are presented in [33, 35, 71, 72, 74, 118]. This section illustrates the two most frequently used models for fitting device characterization data.

2.3.3.1 Yakopcic's Model

One of the generalized models that can be used for fitting parameters of memristors is presented in [118, 119]. Equation 2.15 represents the I-V relationship for Yakopchic's model, where a_1, a_2 and b are the parameters to fit different memristor characterization data.

$$i(t) = \begin{cases} a_1 x(t) sinh(bv(t)) & \text{if} \quad V(t) \geq 0 \\ a_2 x(t) sinh(bv(t)) & \text{if} \quad V(t) < 0 \end{cases} \tag{2.15}$$

State variable in the model is based on Eqs. 2.16–2.18. The parameter g(v(t)) in Eq. 2.16 implements the threshold voltage, where A_p and A_n are magnitudes of exponential responsible for the rate of change of a state, and V_p and V_n are positive and negative threshold voltages of a memristor.

$$g(v(t)) = \begin{cases} A_p(e^{v(t)} - e^{V_p}) & \text{if } V(t) > V_p \\ -A_n(e^{-v(t)} - e^{V_n}) & \text{if } V(t) < -V_n \\ 0 & \text{if } -V_n \leq v(t) \leq v_p \end{cases} \tag{2.16}$$

Equations 2.17 and 2.18 are used to model the change of the state variable of memristive device, depending on the current state, where x_p and x_n represent the thresholds until which the state variable is constant, α_n and α_p are rates of change of state variable. Parameters w_p and w_n represent the window function for $f(x)$, where $w_p(x, x_p) = \frac{x_p - x}{1 - x_p} + 1$ and $w_n(x, x_n) = \frac{x}{1 - x_n}$.

$$f(x) = \begin{cases} e^{-\alpha_p (x - x_p)} w_p(x, x_p) & \text{if } x \geq x_p \\ 1 & \text{if } x < x_p \end{cases} \tag{2.17}$$

$$f(x) = \begin{cases} e^{\alpha_n (x + x_n - 1)} w_n(x, x_n) & \text{if } x \leq 1 - x_n \\ 1 & \text{if } x > 1 - x_n \end{cases} \tag{2.18}$$

2.3.3.2 Eshraghian's Model

The other generalized parameter fitting model for memristive devices is shown in [33]. Equation 2.19 illustrates the state equation for this model, where parameters f_{off}, f_{on} are fitting parameters and the I-V relationship is shown in Eq. 2.20 [121], where ϑ is a fitting parameter to characterize ON state and χ and γ are the fitting parameters to characterize OFF state.

$$\frac{dw}{dt} = \begin{cases} -f_{off}\left(1 + \frac{v}{2\phi_0}\right) e^{f(w)\phi_0\left(1 - \sqrt{1 + \frac{v}{2\phi_0}}\right)} & \text{if } v < 0, w > w_{min} \\ f_{on}\left(1 - \frac{v}{2\phi_0}\right) e^{f(w)\phi_0\left(1 - \sqrt{1 - \frac{v}{2\phi_0}}\right)} & \text{if } v > 0, w, w_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

$$i = w^n sinh(\vartheta V) + \chi\left(e^{(\gamma V)} - 1\right) \tag{2.20}$$

2.3.3.3 Threshold Adaptive Memristor Model

One of such model is ThrEshold Adaptive Memristor (TEAM) model proposed in [71]. This model is similar to Simmons tunnel memristor model for TiO_{2x} devices [95], but more simplified and generalized. The model equations include two parts for linear and non-linear drift [109]. The equations for linear part is illustrated in Eq. 2.21 and for non-linear part in Eq. 2.22, where $R_{OFF}/R_{ON} = e^{\lambda}$ and $w \in [w_{on}, w_{off}]$ [71, 109].

$$v(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{w_{off} - w_{on}}(w - w_{on})\right] i(t) \tag{2.21}$$

$$v(t) = R_{ON} e^{\left(\frac{\lambda}{w_{off} - w_{on}}\right)(w - w_{on})} \tag{2.22}$$

The derivative of state variable in TEAM model is illustrated as in Eq. 2.23, where i_{off} and i_{on} are threshold values for currents, k_{off} and k_{on}, f_{off} and f_{on} are the window function variables.

$$\frac{dw(t)}{dt} = \begin{cases} k_{off}\left(\frac{i(t)}{i_{off}} - 1\right)^{\alpha_{off}} f_{off}(w) & \text{if} \quad 0 < i_{off} < i \\ 0 & \text{if} \quad i_{off} < i < i_{on} \\ k_{on}\left(\frac{i(t)}{i_{on}} - 1\right)^{\alpha_{on}} f_{on}(w) & \text{if} \quad i < i_{on} < 0 \end{cases} \tag{2.23}$$

TEAM model can be used with all previously mentioned window functions, which allows more accurate curve fitting and adaptation of the model to the behavior of the real memristive device. General model for voltage controlled memristor is proposed in [72]. In addition, the research work [72] contains specific parameters for fitting particular memristive devices, such as Pt-Hf-Ti [120], ferroelectric memristor [23] and metallic nanowire memristor [56].

2.3.3.4 Generalized Metastable Switch Memristor Model

This model is developed as a generalized framework for different memristive devices and can be used to fit the parameters of various devices [85]. One of such devices is $W - Ag - chalcogenide$ device, which is commercially available [85]. This memristor model consists of two current components: memory-dependent current I_m and Schottky diode current I_s [91]. Equation 2.24, where $\phi \in [0, 1]$ and how much the Schottky current has a effect on the device. The Schottky current is induced by the metal-semiconductor junction in memristive device and can be represented as $I_s = \alpha_f e^{b_f V} - \alpha_r e^{-b_r V}$.

$$I = \phi I_m(V, t) + (1 - \phi) I_s(V) \tag{2.24}$$

The memory-dependent current I_m is induced from the representation of the conductance of memristive device as a set of conducting channels that change between high and low resistances, therefore, each channel is represented as metastable switch [85, 86, 91]. Equations 2.25 and 2.26 are the probability of transition between states A and B, where $\beta = V_T^{-1}$ is a thermal parameter depending on the temperature, and α ia time step ratio.

$$P_A = \alpha \frac{1}{1 + e^{\beta(V - V_A)}} = \alpha \Gamma(V, V_A) \tag{2.25}$$

$$P_B = \alpha(1 - \Gamma(V, -V_B))) \tag{2.26}$$

The memory-dependent current I_m is calculated as $I_m = V(G_m + \Delta G_m)$, where G_m is a total memristor conductance over N metastable switches and ΔG_m is a change in conductance, which relies on the approximate the binomial distribution with a normal distribution, P_A and P_B [85, 86, 91].

2.3.4 Memristor Models for Particular Devices

Comparing to the generalized memristor models and application of generalized window functions, memristor models created for particular memristive devices are more accurate. The accurate memristor models should include non-volatile and volatile switching mechanisms, which are more computationally complex and include higher order differential equations [9, 16]. Such models for recently proposed memristive devices are illustrated in [11, 40, 41, 73]. More recent work on modelling of HfO_2 is shown in [5].

2.3.4.1 Pickett's Model

The model shown in [95] emulates the behavior of TiO_2 devices. The model is also known as Simmon's Tunnel Barrier model, as the model implies that the tunnel barrier width is modulated by the current flowing through the device [16]. Equation 2.27 represents the state equation, where $i = G(w, u)v$.

$$\frac{dw}{dt} = f(w, i_M) \tag{2.27}$$

Equation 2.28 illustrates how state equation is calculated, where parameters f_{off}, f_{on}, a_{off}, a_{on}, b, w_c, i_{on} and i_{off} are constants shown in [95] determined for particular device.

$$\frac{dw(t)}{dt} = \begin{cases} f_{off} sinh\left(\frac{i}{i_{off}}\right) e^{-e^{\frac{w-a_{off}}{w_c}} - \frac{|i|}{b} - \frac{w}{w_c}} & \text{if} \quad i > 0 \\ f_{on} sinh\left(\frac{i}{i_{on}}\right) e^{-e^{-\frac{w-a_{on}}{w_c}} - \frac{|i|}{b} - \frac{w}{w_c}} & \text{if} \quad i < 0 \end{cases} \tag{2.28}$$

2.3.4.2 Bayat's Model

The other model for $Pt/TiO_{2-x}/Pt$ memristive devices is shown in [9]. This model aims to improve the accuracy and simplify computation. The main equation for the model for $v = 0.5V$ is shown in Eq. 2.29, where ζ, χ, α and θ are fitting parameters shown in [9]. This model can have a problem of overflow [16].

$$\frac{dR_{0.5}}{dt} = \alpha \frac{sinh(v)}{1 + e^{\chi v + \zeta}} \frac{R_{0.5} e^{\lambda R_{0.5}}}{1 + e^{\delta R_{0.5} + \theta}} \tag{2.29}$$

2.3.5 Stochastic Memristor Model

Due to the imperfection of memristive technology and certain characteristic of switching behavior of memristive material, some of the devices can exhibit stochas-

tic behavior. Depending on the application of these devices, the stochastic behavior of memristor can have both positive and negative effects on the system. Stochastic memristors can be applied to model a spiking neuron and neural networks which can perform better than their deterministic analogies [4].

Stochastic behavior can be introduced to different threshold-based memristor models. One of the main works in the modeling of memristor stochasticity is introduced in [89], where the main principle of adding stochasticity to the device and examples of the modification of existing models are illustrated. This subsection covers two different stochastic models and illustrates how stochasticity can be introduced.

2.3.5.1 Stochastic Modification of Yakopcic's Model

The stochasticity to Yakopcic's model from Eqs. 2.15 and 2.16 for parameter fitting is introduced to the $g(v(t))$ corresponding to the threshold voltage. The stochastic parameters in Eqs. 2.15 and 2.16 are V_p and V_n. The endorsed stochasticity on V_p and V_n is shown in Eq. 2.30, where $\alpha\theta(v_{p/n_0} - v_{p/n})dt$ is a deterministic term and $(V - \delta V - v_{p/n})dN(\tau)$ is a stochastic term.

$$dv_{p/n} = \alpha\theta(v_{p/n_0} - v_{p/n})dt + (V - \delta V - v_{p/n})dN(\tau) \tag{2.30}$$

The function θ in the model is a step function, function $N()$ is a Poissonian or log-normal distribution, and parameter α is a fitting parameter [88].

2.3.5.2 Stochastic Modification of Pickett's Model

The stochasticity to Pickett's model for TiO_2 memristors from Eqs. 2.27 and 2.28 is introduced to the threshold parameters a_{off} and a_{on}, which are responsible for the switching points of a model. The method to induce stochasticity into a_{on} and a_{off} is shown in Eq. 2.31, where term $(w - \delta w - a_0)dN(\tau)$ is stochastic term and term $\alpha\theta(a_0 - a_{on/off})dt$ is deterministic.

$$da_{on/off} = \alpha\theta(a_0 - a_{on/off})dt + (w - \delta w - a_0)dN(\tau) \tag{2.31}$$

2.3.6 Memristor Model for Large Scale Simulations

One of the main problems of most of the complex memristor models containing nonlinearity of the device and non-ideal behavior is the computational complexity, errors, and overflow, especially for large scale simulations. One of the most frequent cases, when such problems occur is the simulation of large crossbars in neural network [67]. To avoid such problems, several modified memristor models with approxima-

tions that reduce the computational complexity has been recently proposed [16]. The methodologies for reduction of a computational cost in the model and convergence issues are proposed in [7, 36]. While this section focuses on the modified Biolek's models for large scale simulation are proposed in [16] and derived from the existing memristor models to solve behavioral issues of memristor models in large scale simulations, and recently proposed the model of Messaris et al. in [83].

2.3.6.1 Modified S Model (MS Model)

The MS model is the model proposed in [16] derived from the simple memristor models from [15]. MS model is simple and can be easily used for large scale simulations in SPICE. The equations for the model is presented in Eq. 2.32, where y is a new state variable $y = k \int idt, x = f_s(y)$, k is a switching rate and $R_M(x) = R_{ON}x + R_{OFF}(1-x)$.

$$f_s(y) = \frac{1}{a} ln \frac{1 + e^{a(y+0.5)}}{1 + e^{a(y-0.5)}} \tag{2.32}$$

2.3.6.2 Modified Pickett's Model (MP Model)

In the large scale simulations, Pickett's model can exhibit several problems, such as multiple solutions, the absence of solutions, the equations in the models may not fit into the operating range. These issues are addressed in the modified Pickett's model from Eq. 2.28 is illustrated in Eq. 2.33, where expl(x) function is represented in Eq. 2.34. All fitting parameters for the model are given in [16].

$$\frac{dw(t)}{dt} = \begin{cases} \frac{f_{off}}{2} expl\left(\frac{|i|}{i_{off}} - e^{\frac{w-a_{off}}{w_c} - \frac{|i|}{b}} - \frac{w}{w_c}\right)[1 - e^{\frac{-2|i|}{i_{off}}}] & \text{if} \quad i \geq 0 \\ \frac{-f_{on}}{2} expl\left(\frac{|i|}{i_{on}} - e^{\frac{a_{on}-w}{w_c} - \frac{|i|}{b}} - \frac{w}{w_c}\right)[1 - e^{\frac{-2|i|}{i_{on}}}] & \text{if} \quad i < 0 \end{cases} \tag{2.33}$$

$$expl(x) = \begin{cases} e^x & \text{if} \quad x < x_L \\ e^{x_L}(1 + x - x_L) & \text{if} \quad x \geq x_L \end{cases} \tag{2.34}$$

2.3.6.3 Modified Bayat's Model (MB Model)

In [16], the modification of Bayat's model is also proposed. As the model may have an exponential overflow at circuit memristor voltage v, the expression of the Bayat's model from Eq. 2.29 is modified applying tangential approximation [16]. The original equation is replaces by Eq. 2.35, where g_1 and g_2 are the approximation functions from [9], and the function $h(v)$ is shown in Eq. 2.36 [16].

$$i = v10^{g_1(R_{0.5})h(|v|)+g_2(R_{0.5})} \tag{2.35}$$

$$h(v) = \frac{v^{3/log(10)} - 1}{v^{3/log(10)} + 1} \tag{2.36}$$

This approximation allows to smooth the state equation from [9] and avoid the exponential overflow, which allows avoiding the errors during large scale simulations.

2.3.6.4 Memristor Model of Messaris et al.

One of the models appropriate for large scale simulations in presented in [83]. The model does not contain the integration of state variable, which makes it computationally less complex, and it is based on physical device response data, which is more realistic than most of the generalized memristor models. The I-V relationship for the model is shown in Eq. 2.37, where a_p and a_n are fitting parameters.

$$i(R, v) = \begin{cases} a_p \frac{1}{R} sinh(b_p v) & \text{if} \quad v > 0 \\ a_n \frac{1}{R} sinh(b_n v) & \text{if} \quad v \leq 0 \end{cases} \tag{2.37}$$

The state variable $\frac{dR}{dt} = g(R, v) = s(v) \times f(R, v)$, where $s(v)$ is a switching sensitivity function in Eq. 2.38 with fitting parameters A_p, A_n, t_p and t_n, and $f(R, v)$ is a window function shown in Eq. 2.39 with fitting parameters k_p, k_n, and resistive boundaries $r_n(v)$ and $r_p(v)$.

$$s(v) = \begin{cases} A_p(-1 + e^{t_p|v|}) & \text{if} \quad v > 0 \\ A_n(-1 + e^{t_n|v|}) & \text{if} \quad v < 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.38}$$

$$f(R, v) = \begin{cases} -1 + e^{\eta k_p(r_p(v))-R} & \text{if} \quad R < \eta r_p(v) \\ -1 + e^{\eta k_p(-r_n(v))+R} & \text{if} \quad R > \eta r_n(v) \\ 0 & \text{otherwise} \end{cases} \tag{2.39}$$

This model contains the fitting parameters for TiO_x and TaO_x devices in [83], which allows obtaining a more realistic response of a memristor. The analytical time response expressions for fitting of the characterization data from the real device is shown in Eq. 2.40, where R_0 is an initial resistive state [83].

$$R(t)_{|V_b} = \begin{cases} \frac{ln(e^{\eta k_p r_p(V_b)} + e^{-\eta k_p s_p(V_b)t}(e^{\eta k_p R_0} - e^{\eta k_p r_p(V_b)}))}{k_p} & \text{if} \quad V_b > 0, R < \eta r_p(V_b) \\ \frac{ln(e^{-\eta k_n R_0 + \eta k_n s_n(V_b)t} + e^{-\eta k_n r_n(V_b)}(-1 + e^{\eta k_n s_n(V_b)r}))}{k_n} & \text{if} \quad V_b < 0, R > \eta r_n(V_b) \\ R_0 & \text{otherwise} \end{cases} \tag{2.40}$$

2.3.7 Other Memristor Models and Recent Works

There are several other recently proposed memristor models, which have been shown in different research works. The model based on the complementary resistive switching mechanism is introduced in [116]. The model for drift and diffusion memristors is shown in [124]. The trade-off between simulation cost and accuracy of the memristor model is addressed in [79].

In most of the cases, the models based on IV relationship are used for the system simulations. However, one of the types of memristor models is a flux-charge based model. The model based on the flux-charge relationship is illustrated in [106]. The investigation and modeling of the dynamic behavior of memristive devices in parallel and series connection and flux coupling is shown in [32].

One of the useful tools in the simulation of memristive devices in a system are emulator circuits [62, 122]. The emulator circuit is used for the emulation of the behavior of memristor in large systems, where memristor models cannot be integrated. The generalized framework for memristor emulator circuits can be used not only to emulate the behavior of memristive devices but also to imitate the behavior of meminductors and memcapacitors [103].

2.4 Memristive Devices and Materials

There have been significant research efforts to develop alternative technologies based on memristive materials in the anticipation of scaling limits of today's industry technologies. Emerging memory technologies include Resistive Random Access Memory (RAM) or ReRam, magnetic RAM, (MRAM), ferroelectric RAM (FeRAM), Phase Change Memory (PCM). Resistive switching materials were used as nonvolatile memory devices before memristor was founded as a fundamental circuit element by L. Chua in 1971 [8, 90]. Magnetic and ferroelectric junction tunneling processes were common switching materials reported in 1950–1970 [23, 104]. The use of phase changing materials for memory technologies was first proposed [92] and reproducible resistive change in oxide MIM structures became subject of investigations [44] in early 1960. However, most of the physical implementations of memristors are still in the experimental stage of development that delays massive adoption of the technology.

Typical architecture of the memristor consists of a resistive switching (RS) layer interfaced to the metal electrodes. An insulator sandwiched by two metals, is well known as metal-insulator-metal—MIM, is the most pursued memristor type structure. Hewlett Packard's Titanium Dioxide is also MIM, transition metal oxide memristor [110]. Nevertheless, there is a wide range of material structures that exhibit memristive properties including chalcogenides [51, 52, 76, 98], polymers [24, 57], carbon nanotubes [2, 97], manganite [38, 100], graphene [42, 93], organic materials [14]. It was reported that even human skin show memristive behaviour [81]. Switching time (ST) and switching voltage (V_{th}) of such memristors differ from device to

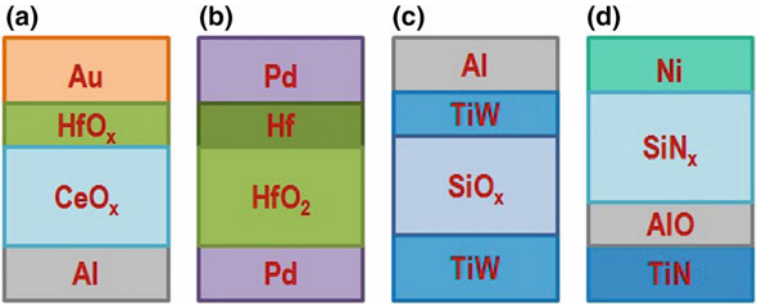

Fig. 2.8 **a** Cerium oxide memristor capped with Hafnium Oxide [47] **b** Hafnium Oxide memristor capped with Hafnium [1] **c** Silicon oxide capped with Tungsten [22] **d** Silicon Nitride capped with Aluminum Oxide [65]

device as well as the ratio of high (R_on) to low resistance (R_on) values of memristors. Table 2.1 provides different implementations of recently reported memristor materials. Resistive switching layers (RSL) consists of titanium, hafnium, silicon, tantalum, germanium, and nickel-based oxides, while electrodes are mainly made from platinum, silicon, silver, nickel, indium, and tantalum. The devices also include capping layers between the electrodes to reduce current overshoots. Figure 2.8 shows several MIM transition metal oxide structures proposed for ReRAM fabrication.

2.4.1 Non-volatile Memristors

The operation of non-volatile memristor based memory devices is based on reproducible resistive switching in memristive materials. Despite the fact that the operation of PCM, ReRAM, FeRAM and MRAM devices is the same, the process of their resistive switching itself differs, which involves different prospects of their further applications.

To describe the setting and resetting processes of these devices, the most generally accepted RS mechanism, the electric pulse induced resistance switching (EPIR) effect [78] will be used.

2.4.1.1 ReRAM

Although any of the above among MRAM, FeRAM, PCM are based on resistive switching, most of the oxide-based memristors are commonly addressed as ReRAM. Oxide-based memristors are usually divided into two types: Filament-Type (FT) and Barrier-Type (BT) Memristors (Fig. 2.9). The switching in filament-type memristors is the process of the formation and rupture of filaments responsible for conductance magnitude. FT memristors can be fabricated to behave as bipolar or unipolar [49]

Table 2.1 Memristor characteristics for different type of materials

Material	RSL	Application	SL thickness	ST	V_{th}(V)	Levels	$R(\Omega)$
$Au/HfO_x/CeO_x/Al$ [47]	Filament	ReRAM	20 nm	1 us	∼ |0.8|	2	600–2.8 M
$Pt/HfO_2/Pt$ [77]	Filament	ReRAM	10 nm	–	0–3	2	∼100 − 1M
$Pt/HfO_2/Ti/Pt$ [37]	Filament	ReRAM	20 nm	–	−3;7	2	10 k–10 M
$Pd/Hf/HfO_2/Pd$ [1]	Filament	ReRAM	10 nm	–	−5;5	2	100–100 M
$Ni/SiN_x/AlO_y/TiN$ [65]	Barrier	ReRAM	100 um	200 ns	−5.5;6.5	multi	–
$Al/TiW/SiO_x/TiW$ [22]	Filament	ReRAM	50 nm	100 ns	|4|;|8|	2	∼100 k–100 M
$Ti/SiN_x/p+-Si$ [64]	Filament	ReRAM	5nm	100 ns	−5;10	2	∼1 –100 M
$Ni/SiN_x/p+(+)Si$ [63]	Barrier	ReRAM	4 nm	–	–	2	–
$TaN/SiO_2/Si$ [125]	Filament	ReRAM	51 nm	–	|5|;|10|	2	–
$Ag/TiO_2/ITO$ [58]	Barrier	ReRAM	20 nm	–	±2	2	1.5–83 k
$Ti/poly-TiO_{2x}/Ti$ [45]	Barrier	ReRAM	350–500 nm	–	±4	2	0.5 M–1 G
$TiOx/TiO2/Au/Si$ [82]	Barrier	ReRAM	30 nm	–	±2.3	2	90–900
$ITO/TiO_2/Ag$ [102]	Filament	ReRAM	400 nm	–	±10	2	0.39–15 k
$Nb_2O_5/NbO_2/TiNO$ 0.5 [69]	–	ReRAM	∼15 nm	20 ns	±1.5	2	<60 k–1 M
$Ti/TiO_2-NT/Au$ [28]	Barrier	ReRAM	80 nm	–	±1.5	2	505–4.7 k
$Pt/HfO_x/Ti$ [43]	Barrier	ReRAM	20 nm	sub 100 ns	−3.5;2	12	1 –200 k
$Pt/PZT/Pt$ and $Ag/PZT/Pt$[46]	Filament	FeRAM	30 nm	5 µs	∼ −1.6; 1.3	2	∼100–1M
$Ag/N-GST/Pt$[98]	–	PCM	30 nm	–	±0.2	2	∼1 k–10 M
$TiW/GST/TiW$ [76]	–	PCM	150 nm	–	±1	2	∼0.94–18.7 k
$La_{2/3}Ca_{1/3}MnO_3/n-Si$ [100]	Filament	ReRAM	100 nm	>1 ms	2.5;−2	2	∼30–10k
$Ge_2Se_3/SnSe/Ag$ [20]	Filament	ReRAM	–	>70ns	−3;1	2	<2k-1M

memristors depending on the chemical composition of insulators. The operation of the barrier-type memristor is based on the oxygen exchange in the metal oxide layer sandwiched between two electrodes, that increases or decreases the Shottky barrier. Metal oxide layer consists of conducting (CL) and insulating layers (IL) placed at the top and bottom electrodes respectively. BT memristors exhibit polarity dependent switching, with the lowest resistance, in case of accumulation of oxygen at the insulator interface: applied current from the top electrode moves vacancies to the IL decreasing the resistance by shrinking the barrier height, while opposite direction of the current brings back the vacancies to CL and increases the resistance of metal oxide [29].

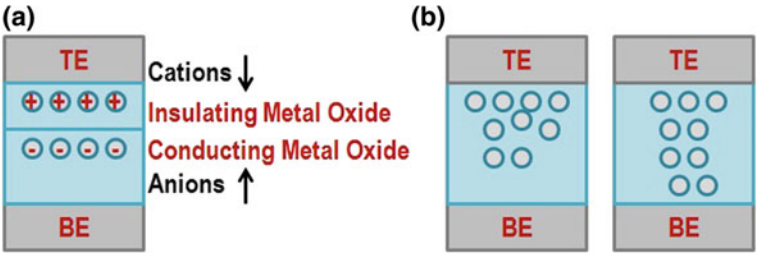

Fig. 2.9 Resistive switching: **a** Barrier type memristors change the resistance exchanging the oxygen vacancies between insulating and conducting layers **b** Filament type memristors switch on forming the filament from top to bottom electrode. Rupture of the filament switches off the memristor

2.4.1.2 FeRAM

FeRAM [39, 50, 61] also shows such polarity dependent switching based polarization or depolarization of ferroelectric material sandwiched between metal electrodes [107]. A current direction during write operation determines the direction of the atom's shift within the FeRAM electrodes, while the displacement of the atom within ferroelectric junction determines whether the current will flow during the read operation (Fig. 2.10). If the current flow is blocked, which corresponds to the high resistive state, the atom is already at that end of the cell. After reading, the atom displacement has to be restored which makes FeRAM a memory with a destructive read [112]. The example of FeRAM structure is given in Fig. 2.11.

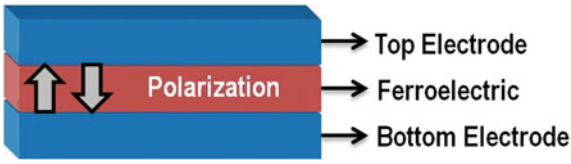

Fig. 2.10 Ferroelectric memristor: Resistance of the material depends on atom displacement within top and bottom electrodes

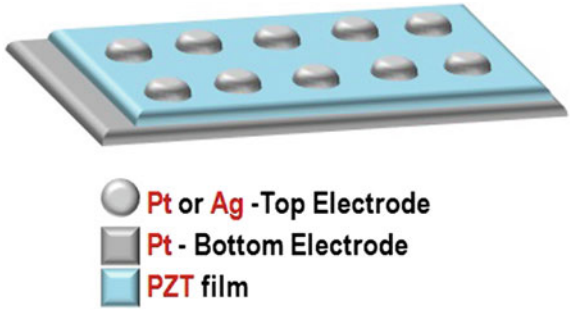

Fig. 2.11 FeRAM: Structure of Pt/PZT/Pt and Ag/PZT/Pt [46]

The main limitation of FeRAM memories is scaling property of PZT and SBT materials ferroelectrics are made of [107]. Recently, a general material in semiconductor devices, a hafnium oxide was reported to be used in FeRAM fabrications. This promises higher data density and scaling possibility which remains the main issues of FeRAMs [105].

2.4.1.3 PCM

Also known as PRAM, Phase-Change Memory technology is based on a material, well known as GST memristors [51, 76, 98] which changes from amorphous to crystalline states (Fig. 2.12), corresponding to high and low resistance levels [101]. The bit cell is melted with current passed through it at different rates, independently of its direction: a higher current in short periods causes the glass to change into a non-conductive amorphous state, while slow heating at low temperatures changes the glass into a conductive crystalline structure (Fig. 2.13). The main limitation of PCMs are considered to be performance limited timing of memory accesses and over time degradation of the material due to the write-erase cycles [34].

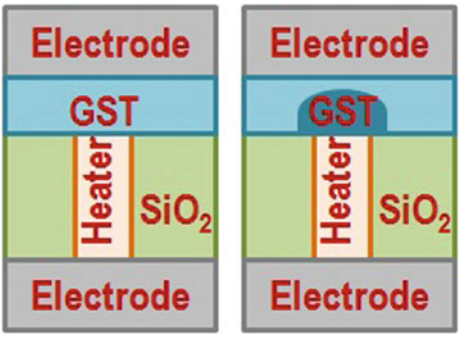

Fig. 2.12 PCM: crystalline state vs amorphous state of GST

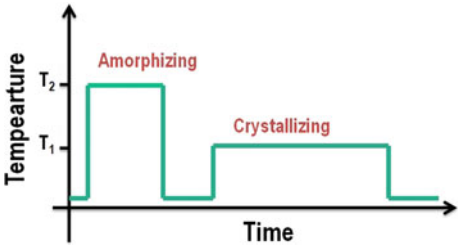

Fig. 2.13 Switching condition of PCM memristors

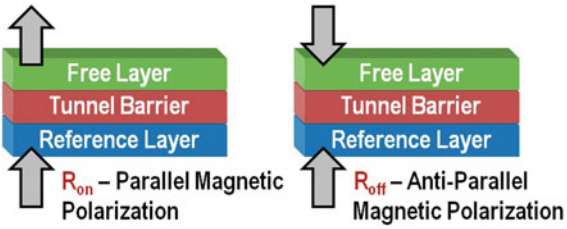

Fig. 2.14 Resistive states of MRAM devices

2.4.1.4 MRAM

Magnetic RAM [113, 114] operation is based on magnetic tunnel junction (MTJ)[99, 123] between ferromagnetic layers. As shown in Fig. 2.14 magnetization in the same direction results in low resistance (Parallel Magnetization), while opposite directions of magnetization cause MTJ has a high resistance(Antiparallel Magnetization). Magnetization of ferromagnetic layers depends on the current polarity and requires the specific amount of energy to flip the spin opposite to the reference layer. This mechanism works stable in earlier memory structures [30]. However, the scaling of this technology is limited due to the high density of write currents that can cause wire melting in shrunk structures[3]. To solve this problem, it was proposed to pass the current through the magnetic layer instead of the electrode wire. The technique is called perpendicular spin torque transfer which allows scaling down to nano-sized structures. Current-perpendicular-to-plane (CPP) geometry of memritors was investigated in Miao Hu et al. in [48]. While current-in-plane (CIP) geometry of the device is studied in Wang et al. [115] where the current flow parallel to the memristor's interface between different layers. The resistance of the magnetic layer is modeled with the domain wall width ΔW and motion that define the parallel or antiparallel magnetization of the material [87] (Fig. 2.15).

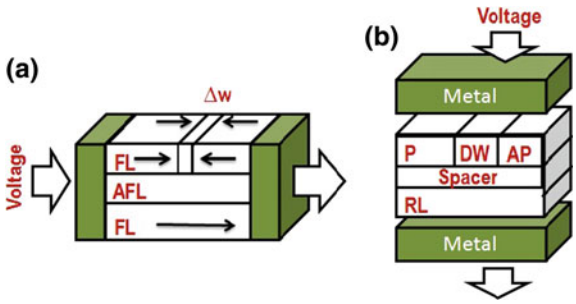

Fig. 2.15 **a** A spintronic memristor with CIP structure [48]:FL—ferromagnetic layer; AFL—antiferromagnetic layer; ΔW- domain wall width; **b** Spintronic memristor with CPP structure [115]:P—Parallel magnetization; AP—Antiparallel magnetization; RL—Reference layer; DW-Domain wall [87]

2.4.2 Volatile Memristors

Majority of the recent works focus on the demonstration of non-volatility property of a memristor [6, 13], as it can benefit scalability of the systems and is a promising solution for many applications. Nevertheless some semi-volatile behaviour of memristors are also reported [19]. It is of note that the volatility of memristive devices can be a useful feature for some of the applications [12]. Volatile memristors can be used in neuromorphic applications besides its conventional application as in volatile memory design. Memristive devices can exhibit rate-limiting volatility property due to irreversible electrodynamic and thermodynamic changes [13, 111], which can be useful for emulating short-term synaptic dynamics [13]. It was reported that introducing a volatile memristor in artificial synapse design improves the overall learning in Neural Networks [108]. Another application is discussed in [94], reporting the possibility of application of volatile memristors for implementation of material implication based logic gates.

2.5 Conclusion

Memristor is an emerging device that has demonstrated the potential to be used in several deep learning neural network applications. This chapter presented the main features and characteristics of memristive devices and introduced the main principles and behavior of memristors. The stability and precision of existing memristive devices is still an open problem, as well as accuracy, generalization issues and computational complexity of memristor models introduced in this chapter. The main issues include a limited number of memristive states, device stability and issues of compatibility with other technologies.

Highlights

- Memristor is a low power device with a small on-chip area and variable resistance.
- Hysteresis, POP and DRM are methods to characterize the behavior of memristor.
- There are several types of memristor models. Some of them are memristor models with window functions, models with fitting parameters, models for particular devices and models suitable for large scale simulations.
- Different materials exhibit the memristive behavior. Each material or combination of materials possesses unique characteristics suitable for certain applications and compatible with particular devices.

References

1. Abunahla H, Jaoude MA, O'Kelly CJ, Halawani Y, Al-Qutayri M, Al-Sarawi SF, Mohammad B (2018) Switching characteristics of microscale unipolar pd/hf/hfo2/pd memristors. Microelectron Eng 185:35–42
2. Ageev O, Blinov YF, Ilin O, Kolomiitsev A, Konoplev B, Rubashkina M, Smirnov V, Fedotov A (2013) Memristor effect on bundles of vertically aligned carbon nanotubes tested by scanning tunnel microscopy. Tech Phys 58(12):1831–1836
3. Åkerman J (2005) Toward a universal memory. Science 308(5721):508–510
4. Al-Shedivat M, Naous R, Cauwenberghs G, Salama KN (2015) Memristors empower spiking neurons with stochasticity. IEEE J Emerg Sel Top Circuits Syst 5(2):242–253
5. Amer S, Sayyaparaju S, Rose GS, Beckmann K, Cady NC (2017) A practical hafnium-oxide memristor model suitable for circuit design and simulation. In: 2017 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 1–4
6. Ascoli A, Corinto F, Tetzlaff R (2016) A class of versatile circuits, made up of standard electrical components, are memristors. Int J Circuit Theory Appl 44(1):127–146
7. Ascoli A, Tetzlaff R, Biolek Z, Kolka Z, Biolkova V, Biolek D (2015) The art of finding accurate memristor model solutions. IEEE J Emerg Sel Top Circuits Syst 5(2):133–142
8. Bashara N, Nielsen P (1963) Memory effects in thin film negative resistance structures. In: Annual report 1963 conference on electrical insulation. IEEE, pp 29–32
9. Bayat FM, Hoskins B, Strukov DB (2015) Phenomenological modeling of memristive devices. Appl Phys A 118(3):779–786
10. Beck A, Bednorz J, Gerber C, Rossel C, Widmer D (2000) Reproducible switching effect in thin oxide films for memory applications. Appl Phys Lett 77(1):139–141
11. Berdan R, Lim C, Khiat A, Papavassiliou C, Prodromakis T (2014) A memristor spice model accounting for volatile characteristics of practical ReRAM. IEEE Electron Device Lett 35(1):135–137
12. Berdan R, Prodromakis T, Khiat A, Salaoru I, Toumazou C, Perez-Diaz F, Vasilaki E (2013) Temporal processing with volatile memristors. In: 2013 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 425–428
13. Berdan R, Vasilaki E, Khiat A, Indiveri G, Serb A, Prodromakis T (2016) Emulating short-term synaptic dynamics with memristive devices. Sci Rep 6:18639
14. Berzina T, Smerieri A, Bernabò M, Pucci A, Ruggeri G, Erokhin V, Fontana M (2009) Optimization of an organic memristor as an adaptive memory element. J Appl Phys 105(12):124515
15. Biolek D, Biolek Z, Biolkova V, Kolka Z (2015) Reliable modeling of ideal generic memristors via state-space transformation. Radioengineering 24(2):393–407
16. Biolek D, Kolka Z, Biolková V, Biolek Z, Potrebić M, Tošić D (2018) Modeling and simulation of large memristive networks. Int J Circuit Theory Appl 46(1):50–65
17. Biolek Z, Biolek D, Biolkova V (2009) SPICE model of memristor with nonlinear dopant drift. Radioengineering 18(2)
18. Bruce J, Hickling A (1939) Electrical conduction of commercial boron crystals. Trans Faraday Soc 35:1436–1439
19. Campbell KA (2015) Carbon-chalcogenide variable resistance memory device. US Patent 9,118,006
20. Campbell KA (2017) Self-directed channel memristor for high temperature operation. Microelectron J 59:10–14
21. Chang T, Jo SH, Kim KH, Sheridan P, Gaba S, Lu W (2011) Synaptic behaviors and modeling of a metal oxide memristive device. Appl Phys A 102(4):857–863
22. Chang YF, Fowler B, Chen YC, Lee JC (2016) Proton exchange reactions in siox-based resistive switching memory: review and insights from impedance spectroscopy. Prog Solid State Chem 44(3):75–85
23. Chanthbouala A, Garcia V, Cherifi RO, Bouzehouane K, Fusil S, Moya X, Xavier S, Yamada H, Deranlot C, Mathur ND et al (2012) A ferroelectric memristor. Nat Mater 11(10):860

24. Chen Y, Liu G, Wang C, Zhang W, Li RW, Wang L (2014) Polymer memristor for information storage and neuromorphic applications. Mater Horiz 1(5):489–506
25. Chua L (1971) Memristor-the missing circuit element. IEEE Trans Circuit Theory 18(5):507–519
26. Chua L (2011) Resistance switching memories are memristors. Appl Phys A 102(4):765–783
27. Chua L (2018) Five non-volatile memristor enigmas solved. Appl Phys A 124(8):563
28. Dorosheva IB, Vokhmintsev AS, Kamalov RV, Gryaznov AO, Weinstein IA (2018) Oxide layer thickness effects on the resistance switching characteristics of ti/tio 2-nt/au structure. In: 2018 Ural symposium on biomedical engineering, radioelectronics and information technology (USBEREIT). IEEE, pp 279–282
29. Edwards AH, Barnaby HJ, Campbell KA, Kozicki MN, Liu W, Marinella MJ (2015) Reconfigurable memristive device technologies. Proc IEEE 103(7):1004–1033
30. Engel B, Akerman J, Butcher B, Dave R, DeHerrera M, Durlam M, Grynkewich G, Janesky J, Pietambaram S, Rizzo N et al (2005) A 4-mb toggle mram based on a novel bit and switching method. IEEE Trans Magn 41(1):132–136
31. Eshraghian JK, Cho K, Iu HHC, Fernando T, Iannella N, Kang S, Eshraghian K (2017) Maximization of crossbar array memory using fundamental memristor theory. IEEE Trans Circuits Syst II: Express Briefs 64(12):1402–1406. https://doi.org/10.1109/TCSII.2017.2767078
32. Eshraghian JK, Iu HH, Fernando T, Yu D, Li Z (2016) Modelling and characterization of dynamic behavior of coupled memristor circuits. In: 2016 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 690–693 (2016)
33. Eshraghian K, Kavehei O, Cho KR, Chappell JM, Iqbal A, Al-Sarawi SF, Abbott D (2012) Memristive device fundamentals and modeling: applications to circuits and systems simulation. Proc IEEE 100(6):1991–2007
34. Ferreira AP, Zhou M, Bock S, Childers B, Melhem R., Mossé, D (2010) Increasing pcm main memory lifetime. In: Proceedings of the conference on design, automation and test in Europe. European Design and Automation Association, pp 914–919
35. García-Redondo F, Gowers RP, Crespo-Yepes A, López-Vallejo M, Jiang L (2016) Spice compact modeling of bipolar/unipolar memristor switching governed by electrical thresholds. IEEE Trans Circuits Syst I: Regul Pap 63(8):1255–1264
36. García-Redondo F, López-Vallejo M, Ituero P (2014) Building memristor applications: From device model to circuit design. IEEE Trans Nanotechnol 13(6):1154–1162
37. Garda B, Ogorzałek M, Kasiliski K, Galias Z (2017) Studies of dynamics of memristor-based memory cells. In: 2017 IEEE 8th Latin American symposium on circuits & systems (LASCAS). IEEE, pp 1–4
38. Ghenzi N, Sánchez M, Gomez-Marlasca F, Levy P, Rozenberg M (2010) Hysteresis switching loops in ag-manganite memristive interfaces. J Appl Phys 107(9):093719
39. Ghoneim MT, Zidan MA, Alnassar MY, Hanna AN, Kosel J, Salama KN, Hussain MM (2015) Thin pzt-based ferroelectric capacitors on flexible silicon for nonvolatile memory applications. Adv Electron Mater 1(6):1500045
40. Guan X, Yu S, Wong HP et al (2012) A spice compact model of metal oxide resistive switching memory with variations. IEEE electron device letters 33(10):1405
41. Hatem FO, Ho PW, Kumar TN, Almurib HA (2015) Modeling of bipolar resistive switching of a nonlinear mism memristor. Semicond Sci Technol 30(11):115009
42. He C, Li J, Wu X, Chen P, Zhao J, Yin K, Cheng M, Yang W, Xie G, Wang D et al (2013) Tunable electroluminescence in planar graphene/sio2 memristors. Adv Mater 25(39):5593–5598
43. He W, Sun H, Zhou Y, Lu K, Xue K, Miao X (2017) Customized binary and multi-level hfo 2- x-based memristors tuned by oxidation conditions. Sci Rep 7(1):10070
44. Hickmott T (1962) Low-frequency negative resistance in thin anodic oxide films. J Appl Phys 33(9):2669–2682
45. Hossein-Babaei F, Alaei-Sheini N (2016) Electronic conduction in ti/poly-tio 2/ti structures. Sci Rep 6:29624

46. Hou P, Wang J, Zhong X, Wu Y (2016) A ferroelectric memristor based on the migration of oxygen vacancies. RSC Adv 6(59):54113–54118
47. Hsieh CC, Roy A, Chang YF, Shahrjerdi D, Banerjee SK (2016) A sub-1-volt analog metal oxide memristive-based synaptic device with large conductance change for energy-efficient spike-based computing systems. Appl Phys Lett 109(22):223501
48. Hu M, Li H, Chen Y, Wang X, Pino RE (2011) Geometry variations analysis of tio 2 thin-film and spintronic memristors. In: 2011 16th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, pp 25–30 (2011)
49. Hu S, Wu S, Jia W, Yu Q, Deng L, Fu YQ, Liu Y, Chen TP (2014) Review of nanostructured resistive switching memristor and its applications. Nanosci Nanotechnol Lett 6(9):729–757
50. Hu Z, Li Q, Li M, Wang Q, Zhu Y, Liu X, Zhao X, Liu Y, Dong S (2013) Ferroelectric memristor based on pt/bifeo3/nb-doped srtio3 heterostructure. Applied Physics Letters 102(10):102901
51. Huang YH, Hsieh TE (2015) Effective thermal parameters of chalcogenide thin films and simulation of phase-change memory. Int J Thermal Sci 87:207–214
52. Ielmini D, Lavizzari S, Sharma D, Lacaita AL (2007) Physical interpretation, modeling and impact on phase change memory (pcm) reliability of resistance drift due to chalcogenide structural relaxation. In: IEEE international electron devices meeting, IEDM 2007. IEEE, pp 939–942 (2007)
53. James AP (2018) Memristor and memristive neural networks
54. Jeon Y, Foltin M (2018) Memristor memory with volatile and non-volatile states. US Patent App. 10/056,140
55. Joglekar YN, Wolf SJ (2009) The elusive memristor: properties of basic electrical circuits. Eur J Phys 30(4):661
56. Johnson S, Sundararajan A, Hunley D, Strachan D (2010) Memristive switching of single-component metallic nanowires. Nanotechnology 21(12):125204
57. Kang E, Neoh K, Tan K (1998) Polyaniline: a polymer with many interesting intrinsic redox states. Prog Polym Sci 23(2):277–324
58. Kavehei, O., Cho, K., Lee, S., Kim, S.J., Al-Sarawi, S., Abbott, D., Eshraghian, K.: Fabrication and modeling of ag/tio 2/ito memristor. In: Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on, pp. 1–4. IEEE (2011)
59. Keshmiri V (2014) A study of the memristor models and applications
60. Kikuchi M, Saito M, Okushi H, Matsuda A (1971) Polarized (letter 8) memory in cdse point contact diodes. Solid State Commun 9(10):705–707
61. Kim D, Lu H, Ryu S, Bark CW, Eom CB, Tsymbal E, Gruverman A (2012) Ferroelectric tunnel memristor. Nano Lett 12(11):5697–5702
62. Kim H, Sah MP, Yang C, Cho S, Chua LO (2012) Memristor emulator for memristor circuit applications. IEEE Trans Circuits Syst I: Regul Pap 59(10):2422–2431
63. Kim S, Chang YF, Park BG (2017) Understanding rectifying and nonlinear bipolar resistive switching characteristics in ni/sinx/p-si memory devices. RSC Adv 7(29):17882–17888
64. Kim S, Jung S, Kim MH, Chen YC, Chang YF, Ryoo KC, Cho S, Lee JH, Park BG (2018) Scaling effect on silicon nitride memristor with highly doped si substrate. Small 14(19):1704062
65. Kim S, Kim H, Hwang S, Kim MH, Chang YF, Park BG (2017) Analog synaptic behavior of a silicon nitride memristor. ACS Appl Mater Interfaces 9(46):40420–40427
66. Krestinskaya O, Dolzhikova I, James AP (2018) Hierarchical temporal memory using memristor networks: a survey. IEEE Trans Emerg Top Comput Intell 2(5):380–395. https://doi.org/10.1109/TETCI.2018.2838124
67. Krestinskaya O, James AP, Chua LO (2018) Neuro-memristive circuits for edge computing: a review. arXiv:1807.00962
68. Krestinskaya O, Salama KN, James AP (2018) Learning in memristive neural network architectures using analog backpropagation circuits. IEEE Trans Circuits Syst I: Regul Pap
69. Kumar S, Davila N, Wang Z, Huang X, Strachan JP, Vine D, Kilcoyne AD, Nishi Y, Williams RS (2017) Spatially uniform resistance switching of low current, high endurance titanium-niobium-oxide memristors. Nanoscale 9(5):1793–1798

70. Kvatinsky S, Belousov D, Liman S, Satat G, Wald N, Friedman EG, Kolodny A, Weiser UC (2014) Magic-memristor-aided logic. IEEE Trans Circuits Syst II: Express Briefs 61(11):895–899
71. Kvatinsky S, Friedman EG, Kolodny A, Weiser UC (2013) Team: threshold adaptive memristor model. IEEE Trans Circuits Syst I: Regul Pap 60(1):211–221. https://doi.org/10.1109/TCSI.2012.2215714
72. Kvatinsky S, Ramadan M, Friedman EG, Kolodny A (2015) Vteam: a general model for voltage-controlled memristors. IEEE Trans Circuits Syst II: Express Briefs 62(8):786–790
73. Laiho M, Hasler JO, Zhou J, Du C, Lu W, Lehtonen E, Poikonen JH (2015) Fpaa/memristor hybrid computing infrastructure. IEEE Trans Circuits Syst I: Regul Pap 62(3):906–915
74. Laiho M, Lehtonen E, Russel A, Dudek P (2010) Memristive synapses are becoming reality. Neuromorphic Eng 10–12 (2010)
75. Li C, Hu M, Li Y, Jiang H, Ge N, Montgomery E, Zhang J, Song W, Dávila N, Graves CE et al (2018) Analogue signal and image processing with large memristor crossbars. Nat Electron 1(1):52
76. Li Y, Zhong Y, Zhang J, Xu X, Wang Q, Xu L, Sun H, Miao X (2013) Intrinsic memristance mechanism of crystalline stoichiometric ge2sb2te5. Appl Phys Lett 103(4):043501
77. Lian X, Gao F, Wan X, Yao J, Gong X, Guo Y, Tong Y (2018) Performance variability, switching mechanism, and physical model for oxide based memristor and rram device. In: 2018 IEEE international symposium on the physical and failure analysis of integrated circuits (IPFA). IEEE, pp 1–4
78. Liu SdQ, Wu N, Ignatiev A (2000) Electric-pulse-induced reversible resistance change effect in magnetoresistive films. Appl Phys Lett 76(19), 2749–2751 (2000)
79. Lupo N, Pérez E, Wenger C, Maloberti F, Bonizzoni E (2018) Analysis of parasitic effects in filamentary-switching memristive memories using an approximated verilog-a memristor model. IEEE Trans Circuits Syst: Regul Pap I
80. Maan AK, Jayadevi DA, James AP (2017) A survey of memristive threshold logic circuits. IEEE transactions on neural networks and learning systems 28(8):1734–1746
81. Martinsen ØG, Grimnes S, Lütken C, Johnsen G (2010) Memristance in human skin. J Phys Conf Ser 224, 012071. IOP Publishing
82. Maslova N, Khrapovitskaya Y, Sokolov I, Grishchenko Y, Mamichev D, Zanaveskin M (2015) Features of titanium oxide memristor fabrication by pulsed laser deposition. Physica Status Solidi (c) **12**(1–2), 242–245 (2015)
83. Messaris I, Serb A, Stathopoulos S, Khiat A, Nikolaidis S, Prodromakis T (2018) A data-driven verilog-a reram model. IEEE Trans Comput-Aided Des Integr Circuits Syst
84. Mladenov, V., Kirilov, S.: A memristor model with a modified window function and activation thresholds. In: 2018 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 1–5 (2018)
85. Model WAM The generalized metastable switch memristor model
86. Molter TW, Nugent MA (2016) The generalized metastable switch memristor model. In: CNNA 2016; Proceedings of 15th international workshop on cellular nanoscale networks and their applications. VDE, pp 1–2 (2016)
87. Nafea SF, Dessouki AA, El-Rabaie S, Elnaghi BE, Ismail Y, Mostafa H (2018) An accurate model of domain-wall-based spintronic memristor. Integration
88. Naous R (2017) Von-neumann and beyond: Memristor architectures. Ph.D. thesis
89. Naous R, Al-Shedivat M, Salama KN (2016) Stochasticity modeling in memristors. IEEE Trans Nanotechnol 15(1):15–28
90. Nielsen P, Bashara N (1964) The reversible voltage-induced initial resistance in the negative resistance sandwich structure. IEEE Trans Electron Dev 11(5):243–244
91. Nugent MA, Molter TW (2014) Ahah computing-from metastable switches to attractors to machine learning. PloS one 9(2):e85175
92. Ovshinsky SR (1968) Reversible electrical switching phenomena in disordered structures. Physical Review Letters 21(20):1450

93. Park WI, Yoon JM, Park M, Lee J, Kim SK, Jeong JW, Kim K, Jeong HY, Jeon S, No KS et al (2012) Self-assembly-induced formation of high-density silicon oxide memristor nanostructures on graphene and metal electrodes. Nano Lett 12(3):1235–1240
94. Pershin Y, Shevchenko S (2017) Computing with volatile memristors: an application of non-pinched hysteresis. Nanotechnology 28(7):075204
95. Pickett MD, Strukov DB, Borghetti JL, Yang JJ, Snider GS, Stewart DR, Williams RS (2009) Switching dynamics in titanium dioxide memristive devices. Journal of Applied Physics 106(7):074508
96. Prodromakis T, Peh BP, Papavassiliou C, Toumazou C (2011) A versatile memristor model with nonlinear dopant kinetics. IEEE Trans Electron Dev 58(9):3099–3105
97. Radoi A, Dragoman M, Dragoman D (2011) Memristor device based on carbon nanotubes decorated with gold nanoislands. Appl Phys Lett 99(9):093102
98. Raeis-Hosseini N, Lim S, Hwang H, Rho J (2018) Reliable ge2sb2te5-integrated high-density nanoscale conductive bridge random access memory using facile nitrogen-doping strategy. Adv Electron Mater 4(11):1800360
99. Raymenants E, Vaysset A, Wan D, Manfrini M, Zografos O, Bultynck O, Doevenspeck J, Heyns M, Radu IP, Devolder T (2018) Chain of magnetic tunnel junctions as a spintronic memristor. Journal of Applied Physics 124(15):152116
100. Rubi D, Kalstein A, Román W, Ghenzi N, Quinteros C, Mangano E, Granell P, Golmar F, Marlasca F, Suarez S et al (2015) Manganite based memristors: Influence of the electroforming polarity on the electrical behavior and radiation hardness. Thin Solid Films 583:76–80
101. Salinga M, Kersting B, Ronneberger I, Jonnalagadda VP, Vu XT, Le Gallo M, Giannopoulos I, Cojocaru-Mirédin O, Mazzarello R, Sebastian A (2018) Monatomic phase change memory. Nat Mater 1
102. Samardzić N, Mionić M, Dakić B, Hofmann H, Dautović S, Stojanović G (2015) Analysis of quantized electrical characteristics of microscale tio 2 ink-jet printed memristor. IEEE Trans Electron Dev 62(6):1898–1904
103. Sánchez-López C, Carro-Pérez I, Carbajal-Gómez VH, Carrasco-Aguilar MA, Morales-López FE (2018) Memristor emulator circuit design and applications. In: Memristor and memristive neural networks. InTech
104. Sbiaa R, Meng H, Piramanayagam S (2011) Materials with perpendicular magnetic anisotropy for magnetic random access memory. Physica Status Solidi (RRL)–Rapid Res Lett 5(12), 413–419 (2011)
105. Schenk T, Mueller S, Schroeder U, Materlik R, Kersch A, Popovici M, Adelmann C, Van Elshocht S, Mikolajick T (2013) Strontium doped hafnium oxide thin films: wide process window for ferroelectric memories. In: 2013 proceedings of the european solid-state device research conference (ESSDERC). IEEE, pp 260–263 (2013)
106. Secco J, Corinto F, Sebastian A (2018) Flux-charge memristor model for phase change memory. IEEE Trans Circuits Syst II: Express Briefs 65(1):111–114
107. Setter N, Damjanovic D, Eng L, Fox G, Gevorgian S, Hong S, Kingon A, Kohlstedt H, Park N, Stephenson G et al (2006) Ferroelectric thin films: Review of materials, properties, and applications. Journal of Applied Physics 100(5):051606
108. Shahsavari M, Falez P, Boulet P (2016) Combining a volatile and nonvolatile memristor in artificial synapse to improve learning in spiking neural networks. In: 2016 IEEE/ACM international symposium on nanoscale architectures (NANOARCH). IEEE, pp 67–72 (2016)
109. Singh J, Raj B (2018) Comparative analysis of memristor models and memories design
110. Strukov DB, Snider GS, Stewart DR, Williams RS (2008) The missing memristor found. Nature 453(7191):80
111. Strukov DB, Williams RS (2009) Exponential ionic drift: fast switching and low volatility ofáthin-film memristors. Appl Phys A 94(3):515–519
112. Sumi T, Moriwaki N, Nakane G, Nakakuma T, Judai Y, Uemoto Y, Nagano Y, Hayashi S, Azuma M, Fujii E et al (1994) A 256 kb nonvolatile ferroelectric memory at 3 v and 100 ns. In: Solid-state circuits conference. 1994 IEEE international digest of technical papers. 41st ISSCC. IEEE, pp 268–269

113. Wang X, Chen Y (2010) Spintronic memristor devices and application. In: Proceedings of the conference on design, automation and test in Europe. European Design and Automation Association, pp 667–672
114. Wang X, Chen Y, Wang A, Xi H, Zhu W, Li H, Liu H (2011) Magnetic tunnel junction and memristor apparatus. US Patent 7,898,844
115. Wang X, Chen Y, Xi H, Li H, Dimitrov D (2009) Spintronic memristor through spin-torque-induced magnetization motion. IEEE Electron Dev Lett 30(3):294–297
116. Wang X, Xu B, Chen L (2017) Efficient memristor model implementation for simulation and application. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36(7):1226–1230
117. Williams RS (2008) How we found the missing memristor. IEEE Spectr 45(12) (2008)
118. Yakopcic C, Taha TM, Subramanyam G, Pino RE (2013) Generalized memristive device spice model and its application in circuit design. IEEE Trans Comput-Aided Des Integr Circuits Syst 32(8):1201–1214
119. Yakopcic C, Taha TM, Subramanyam G, Pino RE, Rogers S (2011) A memristor device model. IEEE electron device letters 32(10):1436–1438
120. Yalon E, Gavrilov A, Cohen S, Mistele D, Meyler B, Salzman J, Ritter D (2012) Resistive switching in \hbox {HfO} _ {2} probed by a metal-insulator-semiconductor bipolar transistor. IEEE Electron Dev Lett 33(1):11–13
121. Yang JJ, Pickett MD, Li X, Ohlberg DA, Stewart DR, Williams RS (2008) Memristive switching mechanism for metal/oxide/metal nanodevices. Nature nanotechnology 3(7):429
122. Yesil A, Gül F, Babacan Y (2018) Emulator circuits and resistive switching parameters of memristor. In: Memristor and memristive neural networks. InTech
123. Yuasa S, Nagahama T, Fukushima A, Suzuki Y, Ando K (2004) Giant room-temperature magnetoresistance in single-crystal fe/mgo/fe magnetic tunnel junctions. Nature materials 3(12):868
124. Zhao Y, Fang C, Zhang X, Xu X, Gong T, Luo Q, Chen C, Liu Q, Lv H, Li Q et al (2018) A compact model for drift and diffusion memristor applied in neuron circuits design. IEEE Trans Electron Dev 99:1–7
125. Zhou F, Chang YF, Fowler B, Byun K.C Lee J (2015) Stabilization of multiple resistance levels by current-sweep in siox-based resistive switching memory 106, 063508

Chapter 3
Deep Learning Theory Simplified

Adilya Bakambekova and Alex Pappachen James

Abstract Deep Learning is a promising field of Artificial Intelligence algorithms that have proven to be capable of solving a wide range of tasks including classification, object detection, regression, face recognition, augmented and virtual reality, self-driving cars and many more. This chapter introduces the reader to Deep Learning, its basic principles, and applications. It covers the essential elements of any Deep Learning system, as well as explains how to connect these elements to form a neural network. The reader will understand the reasoning behind the Deep Learning and why it is so useful nowadays. The training algorithm of the neural network is also covered in this chapter.

3.1 Why Deep Learning: Complex Made Simple

"If you look at the anatomy, the structure, the function, there's nothing in the universe that's more beautiful, that's more complex, than the human brain".Keith BlackDiscover magazine 2004

Deep Learning is a collection of various mathematical algorithms inspired by the human brain. Now, it should be emphasized that there is math behind any Deep Learning architecture—mostly requires simple calculus and linear algebra. These algorithms, or, as they are commonly called—neural networks (NN), learn to perform tasks that can be complicated for a human (for example, cancer prediction based on thousands of genes [2]). The ability of these algorithms to process a large volume of data to extract meaningful information purely using the computational capabilities of existing von Neumann computing architecture makes these attractive for solving various Artificial Intelligence (AI) problems.

A. Bakambekova · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

A. Bakambekova
e-mail: adilya.bakambekova@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_3

With an increase in the demand for processing a large volume of data, alternative computing approaches need to be explored to cope up with higher computational demands and with low power requirements. The need to switch from the conventional von Neumann architecture to the brain-inspired one has been growing stronger in the last few years. "Neuromorphic" systems, named so by Carver Mead in 1990 [15], promise to overcome the stagnation of Moore's law and deliver better solutions for low power computing.

Many engineers and scientists stand by the point that there is nothing more efficient than things created by nature. Thus, it seems natural to try to model the human brain mathematically by adapting its most useful features. This list includes:

- Parallel and asynchronous computation. Parallelism is seen in every level of abstraction of the brain. At the higher level, the brain processes up to 5 input streams (sight, touch, smell, hearing, and taste) simultaneously. If we go deeper and consider the various local neural networks or even cortices, we will observe parallelism as well—there is an evidence [28] that multiple of them are active when the brain makes a decision. And, parallel computing is present at the lowest level as well: many neurons fire at the same time.
- Memory and processing as a single unit. While computers have the data stored in memory and processed in the Central Processing Unit (CPU), the brain performs both functions at the same "place". Biological neural networks are capable of performing both memory and data processing [20], which is one of the reasons why our brain can recognize a banana in a matter of half a second, while it is a not so easy task for a computer.
- Individual connections between units. Computers have a system of buses (or wires) that are shared between different blocks. This imposes a limit on the amount of information transmitted at the same time. For instance, a processor with a bus width of 64 bits can retrieve only 64 bits in one cycle. Neurons, on the other hand, have dedicated connections that are not shared, i.e. it is a point-to-point connection.
- Analog processing. The human brain, unlike a computer, stores data in continuous-amplitude and continuous-time modes. The electrochemical reactions occurring in the brain allow for continuous values characterizing the strength of the connection between different neurons. Also, as we mentioned earlier, the brain operates in asynchronous mode, thus not sampling the input data but processing it constantly, at every single time unit.
- Highly robust to errors and variations. It has been shown that the biological neural network experiences robustness to high noise in input data as well as structural variations in its parameters [3]. It is easily illustrated with the example of face recognition. The human brain can recognize a face it knows even in such conditions as poor lighting or unusual angle, while one flipped bit can cause the computer to give a wrong result.
- Learning by itself. This characteristic is one of the most important ones on the list. Traditional computer architectures are perfect for storing, compiling and executing different software programs created by humans. That is, current von Neumann processors are *programmed* to perform certain tasks. But the human brain is not.

It is capable of *learning* new skills and knowledge. This concept – that computers should no longer be told how to do something, but rather know only what needs to be done – is the driving force of the whole AI community.

- Low power. As transistors get smaller and smaller, fewer electrons are required to drive the circuits, which should make the power consumption lower. However, it is still cannot be compared to the energy efficiency of the human brain. It is estimated that the human brain makes 10^{16} floating point operations per second (flops) [16], while consuming around $20W$ of power. *Summit*, fastest supercomputer [1], operates at a speed of 20×10^{16} flops and consumes $13MW$ of power. It is clear that the operations per second to power consumed ratio is much bigger in the human brain.

All of the features listed above became an inspiration for implementing the deep learning algorithms in hardware. The following sections will slowly build an understanding of the basic buildings blocks of any deep neural network (DNN), as well as explain the main algorithms used for training and optimizing with the help of both graphical and mathematical descriptions.

3.2 Essential Elements of the Architecture

There are three main components that, when joined together, form the artificial neuron: weight, bias and activation function. It is useful to show the similarities between the mathematical and biological neurons to dispel the last doubts regarding the deep learning architectures. Once all building blocks are thoroughly explained, we will connect them into one single neuron.

3.2.1 Weight

There are around 86 billion neurons in our brain [10] and more than 100 trillion synapses (connections between neurons). As it was implied earlier, all synapses are different, so that each pair of neurons is connected differently. Some neurons are connected stronger than others, which means that a weaker signal would be enough to travel between them. Other neurons have weak connections so that the signal needs to be much higher in order to get transmitted.

In artificial neuron, the concept of such variability in synapses is implemented with the help of weights, which are just numbers indicating the strength of the connection. So, if a given neuron has k inputs, there will be k weights, since each input channel will have its strength of connection with the neuron. In practice, these input channels can carry both the data from sensors and the outputs of other neurons.

The total amount of signal received by the neuron would then be the weighted summation of all the signals coming to it. Depending on the weight w_k, the signal will be either amplified or attenuated compared to the other inputs. If the weight of

the particular input is bigger that others, it will have more influence on the output of the neuron. If we denote the input as x_k, and the output as y, then the mathematical representation of what we have described becomes

$$y = \sum_k (w_k \cdot x_k) \tag{3.1}$$

These weights which connect neurons from different layers of the NN can be adjusted to enable the network to learn a certain task. This process will be discussed in Sect. 3.3.

3.2.2 Bias

Bias is an additional parameter that sets the threshold of the neuron. This variable helps to regulate the firing conditions of the neuron and can be represented as simply a number added to the weighted summation of inputs. Thus, if a bias is b, then

$$y = \sum_k (w_k \cdot x_k) + b \tag{3.2}$$

Biases help to shift the output of the neuron, thus controlling the amount of input which is enough to activate the neuron.

3.2.3 Activation Function

The activation function is the rule that helps a neuron to decide whether it should fire or not after processing the set of given inputs. Depending on the strength of electrochemical reactions in the biological brain, neurons transmit signals with different strengths.

The simplest activation function is a simple summation that we already have. The rule will be as follows:

$$y = \begin{cases} 1, & \text{if } \sum_k (w_k \cdot x_k) + b > 0 \\ 0, & \text{if } \sum_k (w_k \cdot x_k) + b < 0 \end{cases} \tag{3.3}$$

In Eq. 3.3 if $y = 1$, the neuron will transmit the signal further, and otherwise not. Now it is clearly seen how bias can help shift the level of activation: if we increase the bias it takes less strength of the input signals to fire the neuron.

This activation function, despite having simplicity as its advantage, is quite unpopular due to its inability to give continuous outputs. In many applications it is not enough to know whether the neuron is activated or not, it is also essential to know how reliable this activation is.

Thus, there are other popular functions heavily used by researchers. One of them is a *sigmoid* function, denoted as

$$y = sigmoid(x) = \sigma(x) = \frac{1}{1 + e^{-x}}.$$

Another popular function is the hyperbolic tangent:

$$y = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Both the sigmoid and hyperbolic tangent activation functions have an advantage over other activation functions due to the simplicity of their derivatives (Fig. 3.1).

3.2.4 Artificial Neuron

Now we have everything to build the model of the neuron. Assuming we denote the activation function as a, the final mathematical expression would be

$$y = a\left(\sum_{k}(w_k \cdot x_k) + b\right) \tag{3.4}$$

And graphically it can be represented as:

The illustration shows that each input has its weight that it gets multiplied with. Then, all the resulting products and the bias are summed together and fed into the activation function. The output of this function is the output of the whole neuron.

In matrix notations, if the input is a vector $\mathbf{x}$, $\mathbf{w}$ is a k—length vector containing weights and $\mathbf{y}$ is the output scalar:

$$\mathbf{y} = a(\mathbf{x} \cdot \mathbf{w}^{\mathsf{T}} + b) \tag{3.5}$$

3.3 Neural Network

Now that we have built the necessary background information about the single artificial neuron, it is time to move on and see what happens when we connect a lot of them. As the name suggests, a neural network is a set of neurons interconnected in a specific way.

Definition

Artificial Neural Network (ANN) is a computing system implemented in hardware or software that resembles the features of the human brain and is built from many artificial neurons.

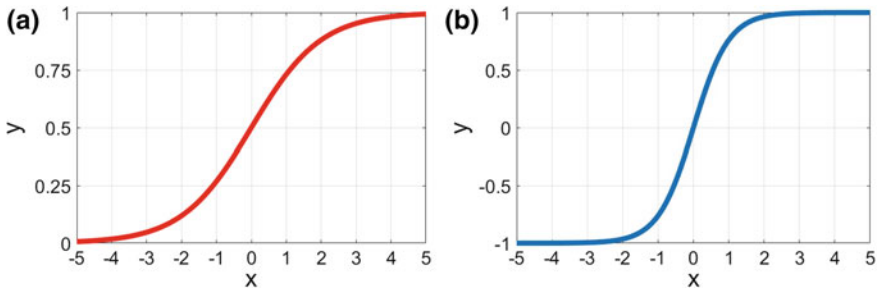

Fig. 3.1 Plots of activation functions: **a** Sigmoid **b** Hyperbolic Tangent

The architecture of the net can is usually chosen based on the task to be solved, but it is convenient to start with the most simple one. Below is a picture of three-layered neural network, which has three input neurons (green: x_1, x_2, x_3), two hidden neurons (blue) and one output neuron (red: y).

Example 2.1

Let us take a specific example to understand the main principles of the network shown in Fig. 3.2. Imagine we have to predict the probability (y) of a student passing the final exam given the information about the amount of hours he/she spent on studying during the semester (x_1), sleeping the night before the exam (x_2) and the grade received on the midterm exam (x_3).

Once we know the data of the student whose final grade we are trying to predict, we *feed* it into the network. The input data is then multiplied by the corresponding weight (shown by solid black lines), which is unique for each pair of neurons located in different layers. Thus, if there are three neurons in the input layer and 2 in the hidden layer, six weights are representing the connections between the two layers. So, following the analogy of a single artificial neuron shown in Fig. 3.3, each hidden neuron takes the weighted summation of the input data, adds the bias b_1 and feeds the result into the activation function. The resulting numbers lie in the range of [0, 1], which is because of the *sigmoid* (Fig. 3.1) activation function.

The output of the hidden layer is fed into the last layer, which has only one neuron with its weights and bias. The procedure repeats and after the activation function is applied we get the output value y which is again in the range of [0, 1]. This y is the probability of passing the final exam that we are trying to predict.

The network is shown in Fig. 3.3 is considered only for educational purposes and is usually too small to solve a real-life problem. When building a neural net, it can control:

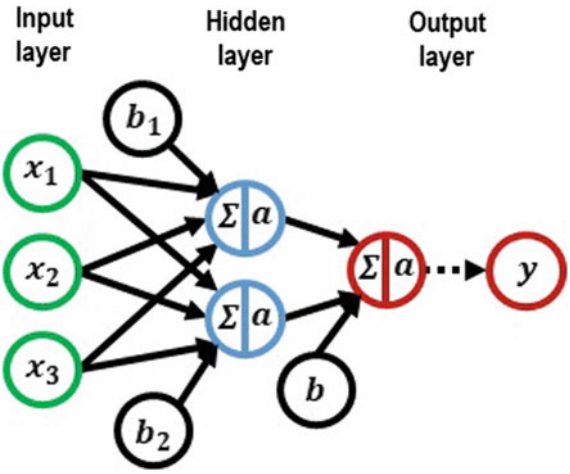

Fig. 3.2 Three-layered artificial neural network

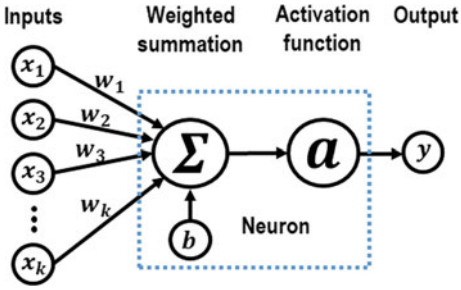

Fig. 3.3 Artificial neuron model

1. Number of input neurons. This parameter depends on the kind of data that is used. If, for instance, consider a case where the output of the networks depends on n different variables, then we will build a network with n input neurons. Whenever we process pictures and use them as the input data, each pixel will be represented by its own input neuron.
2. Number of output neurons. In Example 2.1 we try to figure out only the probability of passing the final exam, but we could increase the number of output neurons if we had more numbers to predict. As a rule of thumb, the more the number of output neurons in a network has, the more complex will be the structure of the hidden layers.
3. Number of hidden layers and the number of neurons in each of them. These parameters influence the accuracy of the network outputs, which means that when the problem to be solved becomes more complex, the size of the hidden layers section increases too. There is not necessarily a linear dependency between these two indicators, but usually a positive correlation.

As the number of layers and neurons grows larger, the resulting network is no longer called a simple ANN. Once it has more than one hidden layer, it is called a Deep

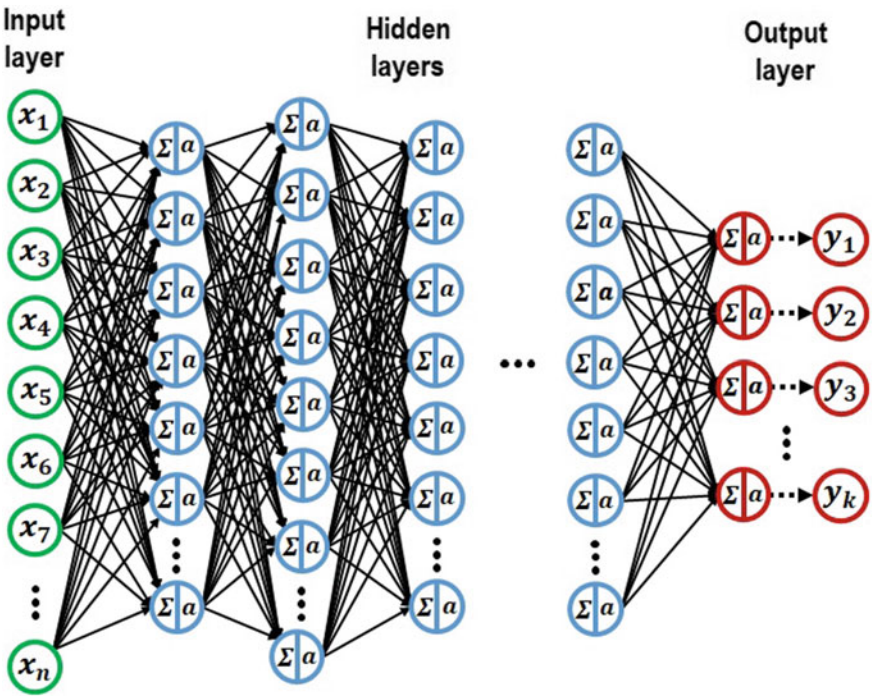

Fig. 3.4 Deep neural network

Neural Network (DNN). The typical configuration of the simple fully-connected DNN is shown in Fig. 3.4.

3.4 Training

Before now it has been assumed that the network is in "good shape". That is, it operates reasonably well, and the predicted output is accurate enough. This might be a good approach to explain the general procedure of how the output gets computed, yet it is too fuzzy for explaining how to get the network into this "good shape". Of course, the network would not give the correct output with random weights and biases. Therefore,

Definition

Training of the neural network is the process of finding the right set of weights and biases, or, trainable parameters, so that if fed with input data the network will give the right output.

Since there are many different neural network architectures besides the staple multi-layered fully-connected DNN, and even a lot of training algorithms, it is justifiable that training of individual network will not be the same. However, there is one thing that makes the training impossible without it: the data set.

Formally, data set is a collection of input and output data that your network is trying to learn. If we continue considering Example 2.1, the data set would be many entries of studying and sleeping hours, midterm exam score and the corresponding Boolean parameter showing whether the student passed the course or not. Then, the training process would consist of showing the data set to the network multiple times, slightly changing the trainable parameters after each iteration. Data set is usually divided into two parts: the bigger one is used for training, and the smaller one is for testing.

3.4.1 Loss Function

In order to efficiently train the neural network, it is essential to set the rules for evaluating its performance. To do so, we introduce the concept of a loss function.

Definition

Loss function is the tool used to measure the difference between the predicted value $\hat{y}$ and the expected value y.

In simple words, our goal is to train the network in a way that its output value is as close to the true one as possible. One of the most popular loss functions used to evaluate the performance of the neural network is Mean-Squared Error (MSE):

$$MSE = \frac{1}{2m}\sum_{m}(y - \hat{y})^2 = \frac{1}{2m}(\mathbf{y} - \hat{\mathbf{y}})^2, \tag{3.6}$$

where m is the total number of data points used for testing.

MSE measures the accuracy of only one output neuron, which is not useful when we have many of them. However, we can define a metric similar to MSE called Sum of Squared Errors (SSE):

$$SSE = \frac{1}{2n}\sum_{n}(y_n - \hat{y}_n)^2 = \frac{1}{2n}(\mathbf{y_n} - \hat{\mathbf{y}}_{\boldsymbol{n}})^2, \tag{3.7}$$

where n is the number of neurons in the output layer.

According to the Eq. 3.7, SSE first computer the MSE for each output neuron and then sums them all together.

There is another loss function commonly used for evaluating and training the neural network, which is called the cross-entropy:

$$H = -\sum_{n} y_n \log \hat{y}_n = -\mathbf{y}^{\mathsf{T}} \cdot \log \hat{\boldsymbol{y}} \tag{3.8}$$

3.4.2 Gradient Descent

Having established the common tools to evaluate the performance of the network, we can begin the training process. Recall that the main goal of training is to find the right weights and biases for a specific data set, which means that we have to minimize the loss function L. This can be achieved with the help of Gradient Descent (GD) optimization algorithm:

Definition

Gradient Descent algorithm is a tool for updating the trainable parameters in the network after each step using the iterative computation of the derivative of the loss function with respect to each weight and bias.

Sometimes the GD algorithm is also called backpropagation since the error propagates from the output layer all the way to the input layer. It is well-known that the gradient of the function defines the direction in which its output increases most rapidly. Since our system requires the opposite—to find how to minimize the cost function (and thus reduce the error)—the gradient should be negative.

In this example we will choose the cross-entropy for our loss function C. From Eqs. 3.5, 3.8 and Table 3.1,

$$C = -\mathbf{y}^{\mathsf{T}} \cdot \log \hat{\boldsymbol{y}}_{\boldsymbol{L-1}} = -\mathbf{y}^{\mathsf{T}} \cdot \log(\mathbf{w}_{\mathbf{L-1}} \cdot \hat{\boldsymbol{y}}_{\boldsymbol{L-2}} + \mathbf{b}_{\mathbf{L-1}}) \tag{3.9}$$

Using the chain rule, one can identify the gradient of the loss function with respect to the weights between the last layer and the one before it.

$$\frac{\partial C}{\partial \mathbf{w}_{\mathbf{L-1}}} = \frac{\partial C}{\partial \mathbf{y}_{\mathbf{L-1}}} \odot \frac{\partial \mathbf{y}_{\mathbf{L-1}}}{\partial \mathbf{w}_{\mathbf{L-1}}} \tag{3.10}$$

With the partial derivative of the cost function with respect to the outputs of the last layer being:

$$\frac{\partial C}{\partial \mathbf{y}_{\mathbf{L-1}}} = -\mathbf{y} \frac{1}{\hat{\boldsymbol{y}}_{\boldsymbol{L-1}}} = -\mathbf{y}(\mathbf{w}_{\mathbf{L-1}} \cdot \hat{\boldsymbol{y}}_{\boldsymbol{L-2}} + \mathbf{b}_{\mathbf{L-1}})^{-\mathbf{1}} \tag{3.11}$$

Table 3.1 Network parameters and notations

Notation	Definition
L	Number of layers
$\mathbf{w_{j-1}}$	Matrix of size $n \times m$ containing the weights of the inputs to the jth layer, where m is the number of neurons in the $(j-1)$th layer and n is the number of neurons in the jth layer
$\mathbf{b_j}$	Vector with biases in the jth layer
$\mathbf{x}$	Input data
$\mathbf{\hat{y}_{j-1}}$	Output of the jth layer
$\mathbf{\hat{y}_{L-1}}$	Predicted output of the network
$\mathbf{y}$	Expected output of the network
$\mathbf{a}$	Activation function
C	Loss function
$\odot$	Piece-wise multiplication
α	Learning rate. Hyper-parameter that controls the influence of the adjustion on the original value

And the derivative of the output with respect to the weights:

$$\frac{\partial \mathbf{y_{L-1}}}{\partial \mathbf{w_{L-1}}} = \frac{\partial (a(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}}))}{\partial \mathbf{w_{L-1}}} =$$

$$= \frac{\partial (a(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}}))}{\partial (\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}})} \odot \frac{\partial (\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}})}{\partial \mathbf{w_{L-1}}} =$$

$$= \frac{\partial (a(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}}))}{\partial (\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}})} \odot \mathbf{y_{L-2}} \tag{3.12}$$

Thus, after each iteration the weight is updated as:

$$\mathbf{w_{j-1}} \leftarrow \mathbf{w_{j-1}} - \alpha \frac{\partial C}{\partial \mathbf{w_{j-1}}} \tag{3.13}$$

Similarly for the biases in the last layer using Eq. 3.11:

$$\frac{\partial C}{\partial \mathbf{b_{L-1}}} = \frac{\partial C}{\partial \mathbf{y_{L-1}}} \odot \frac{\partial \mathbf{y_{L-1}}}{\partial \mathbf{b_{L-1}}} \tag{3.14}$$

The derivative of the output with respect to the biases:

$$\frac{\partial \mathbf{y_{L-1}}}{\partial \mathbf{b_{L-1}}} = \frac{\partial (a(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}}))}{\partial \mathbf{b_{L-1}}} =$$

$$= \frac{\partial(a(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}}))}{\partial(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}})} \odot \frac{\partial(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}})}{\partial \mathbf{w_{L-1}}} =$$

$$= \frac{\partial(a(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}}))}{\partial(\mathbf{w_{L-1}} \cdot \mathbf{y_{L-2}} + \mathbf{b_{L-1}})} \tag{3.15}$$

Then, the bias is updated as:

$$\mathbf{b_{j-1}} \leftarrow \mathbf{b_{j-1}} - \alpha \frac{\partial C}{\partial \mathbf{b_{j-1}}} \tag{3.16}$$

Similar recursive operations are performed for all the layers while going back from the output to the input layer. Then, the derivatives with respect to the weights and biases of the $(j-1)$th layer become:

$$\frac{\partial C}{\partial \mathbf{w_{j-1}}} = \frac{\partial C}{\partial \mathbf{y_{j-1}}} \odot \frac{\partial \mathbf{y_{j-1}}}{\partial \mathbf{w_{j-1}}} \tag{3.17}$$

$$\frac{\partial C}{\partial \mathbf{b_{j-1}}} = \frac{\partial C}{\partial \mathbf{y_{j-1}}} \odot \frac{\partial \mathbf{y_{j-1}}}{\partial \mathbf{b_{j-1}}} \tag{3.18}$$

3.5 Closing Remarks

Deep Learning has been receiving a growing amount of attention in the last decade due to the advancement of computational resources and an increase in both the number and quality of the available data sets. It proves to be an efficient tool capable of solving a variety of classification, regression and generation tasks, using both labeled and unlabeled data.

Even though the focus of this paper has been the fully-connected Deep Neural Network, there are many other architectures with specific neuron interconnections. Major architectures are summarized in Table 3.2.

One of the main advantages of Deep Learning is the fact that it combines feature extraction and data processing in one single algorithm thus simplifying the task for humans. In the future, it might potentially become a universal solution for problems arising in many different areas: business, finance, health, entertainment, education and many more.

This section gave a brief introduction to the basic principles of Deep Learning, as well as built an understanding of how the neural network is built, trained and used. The subsequent chapters of this book will continue exploiting the Deep Learning architectures to build neuromorphic circuits and systems, as well as elaborate on their applications.

Table 3.2 Advanced Neural Network Architectures

Name	Selected Applications	Main Features
Convolutional Neural Network (CNN) [25]	Image processing [17], Video processing [12], 3D Animation [24], Object Detection [7]	Mostly used in image and video analysis. Uses different filters in convolutional layers. Employs pooling layers. Computationally efficient
Recurrent Neural Network (RNN) [13]	Natural Language Processing [4], Predicting stock prices [19]	Learning sequences. Chain-like architecture with feedback loops
Long Short-Term Memory (LSTM) [9]	Natural Language Processing, Text Processing [21], Visual Question Answering [6], Power prediction [26]	Special kind of RNN. Difference in special input and forget gates, which allows for better control of long-term dependencies, and a cell state
Restricted Boltzmann Machine (RBM) [18]	Recommender Systems [27], Voice recognition [31], Emergency detection in health care [11]	RBM has only two layers (one input and one hidden), there is no output layer. Probabilistic energy-based model
Gated Recurrent Unit (GRU) [5]	Acoustic Modeling [30], Sequence Modeling, Question Detection [23]	Similar to RNN and LSTM, but unlike LSTM it has no cell state. There are 3 gates: forget, update, output
Generative Adversarial Network (GAN) [8]	Semantic Segmentation [29], Image Resolution enhancement [14], Image generation [22]	Unsupervised learning. Consists of two models: generative and discriminative. Mainly used to generate new pictures from given samples

Chapter Highlights

- Deep Learning is a collection of various mathematical algorithms inspired by the human brain.
- Neuromorphic circuits and systems implement the Deep Learning algorithms.
- The list of human brain's main features that became the reason to create the Deep Learning algorithms includes parallel and asynchronous computation, combined memory and processing, individual interconnections, analog processing, robustness to errors, low power and ability to learn.
- Weight, bias and activation function constitute the main components of an artificial neuron.
- Weight of the connection between the neurons measures the strength of this connection.

- Bias is a parameter that sets the threshold of the neuron.
- Activation function decides how strong should the signal transmitted to the subsequent layers be.
- Artificial Neural Network is a computing system implemented in hardware or software that resembles the features of the human brain and is built from many artificial neurons.
- When building the neural network you need to choose the number of input and output neurons, as well the number of hidden layers and the number of neurons in them.
- Training of the neural network is the process of finding the right set of weights and biases, or, trainable parameters, so that if fed with input data the network will give the right output.
- Loss function is the tool used to measure the difference between the predicted value $\hat{y}$ and the expected value y.
- Gradient Descent algorithm is a tool for updating the trainable parameters in the network after each step using the iterative computation of the derivative of the loss function with respect to each weight and bias.
- Advanced Deep Learning architectures include the Convolutional Neural Network, Recurrent Neural Network, Long Short-Term Memory, Restricted Boltzmann Machine, Gated Recurrent Unit, Generative Adversarial Networks and many more.

References

1. (2018). https://www.olcf.ornl.gov/summit/
2. Agrawal S, Agrawal J (2015) Neural network techniques for cancer prediction: a survey. Proc Comput Sci 60:769–774. https://doi.org/10.1016/j.procs.2015.08.234
3. Blanchini F, Franco E (2011) Structurally robust biological networks. BMC Syst Biol 5(1):74. https://doi.org/10.1186/1752-0509-5-74
4. Chien JT, Ku YC (2016) Bayesian recurrent neural network for language modeling. IEEE Trans Neural Netw Learn Syst 27(2):361–374
5. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv:1406.1078
6. Chowdhury I, Nguyen K, Fookes C, Sridharan S (2017) A cascaded long short-term memory (lstm) driven generic visual question answering (vqa). In: 2017 IEEE international conference on image processing (ICIP). IEEE, pp 842–1846
7. Deng Z, Lei L, Sun H, Zou H, Zhou S, Zhao J (2017) An enhanced deep convolutional neural network for densely packed objects detection in remote sensing images. In: 2017 international workshop on remote sensing with intelligent processing (RSIP). IEEE, pp 1–4
8. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D,Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680
9. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2017) Lstm: a search space odyssey. IEEE Trans Neural Netw Learn Syst 28(10):2222–2232

10. Herculano-Houzel S (2009) The human brain in numbers: a linearly scaled-up primate brain. Front Human Neurosci **3** (2009). https://doi.org/10.3389/neuro.09.031.2009
11. Kim HG, Han SH, Choi HJ (2017) Discriminative restricted boltzmann machine for emergency detection on healthcare robot. In: 2017 IEEE international conference on big data and smart computing (BigComp). IEEE, pp 407–409
12. Le Callet P, Viard-Gaudin C, Barba D (2006) A convolutional neural network approach for objective video quality assessment. IEEE Trans Neural Netw 17(5):1316–1327
13. Lipton ZC, Berkowitz J, Elkan C (2015) A critical review of recurrent neural networks for sequence learning. arXiv:1506.00019
14. Ma W, Pan Z, Guo J, Lei B (2018) Super-resolution of remote sensing images based on transferred generative adversarial network. In: IGARSS 2018-2018 IEEE international geoscience and remote sensing symposium. IEEE, pp 1148–1151
15. Mead C (1990) Neuromorphic electronic systems. Proc IEEE 78(10):1629–1636. https://doi.org/10.1109/5.58356
16. Mead WR, Kurzweil R (2006) The singularity is near: when humans transcend biology. Foreign Affairs 85(3):160. https://doi.org/10.2307/20031996
17. Rezaee M, Mahdianpari M, Zhang Y, Salehi B (2018) Deep convolutional neural network for complex wetland classification using optical remote sensing imagery. IEEE J Sel Top Appl Earth Obs Remote Sens 11(9):3030–3039
18. Salakhutdinov R, Mnih A, Hinton G (2007) Restricted boltzmann machines for collaborative filtering. In: Proceedings of the 24th international conference on machine learning. ACM, pp 791–798
19. Samarawickrama A, Fernando, T (2017) A recurrent neural network approach in predicting daily stock prices an application to the sri lankan stock market. In: 2017 IEEE international conference on industrial and information systems (ICIIS). IEEE, pp 1–6 (2017)
20. Sangwan VK, Lee HS, Bergeron H, Balla I, Beck ME, Chen KS, Hersam MC (2018) Multi-terminal memtransistors from polycrystalline monolayer molybdenum disulfide. Nature 554(7693):500–504. https://doi.org/10.1038/nature25747
21. Skovajsová L (2017) Long short-term memory description and its application in text processing. In: Communication and information technologies (KIT). IEEE, pp 1–4
22. Tan WR, Chan CS, Aguirre H, Tanaka K (2017) Improved artgan for conditional synthesis of natural image and artwork. arXiv:1708.09533
23. Tang Y, Huang Y, Wu Z, Meng H, Xu M, Cai L (2016) Question detection from acoustic features using recurrent neural network with gated recurrent unit. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 6125–6129 (2016)
24. Weng CY, Curless B, Kemelmacher-Shlizerman I (2018) Photo wake-up: 3d character animation from a single photo. arXiv:1812.02246
25. Wu J (2017) Introduction to convolutional neural networks
26. Xiaoyun Q, Xiaoning K, Chao Z, Shuai J, Xiuda M (2016) Short-term prediction of wind power based on deep long short-term memory. In: 2016 IEEE PES Asia-Pacific power and energy engineering conference (APPEEC). IEEE, pp 1148–1152
27. Yedder HB, Zakia U, Ahmed A, Trajković L (2017) Modeling prediction in recommender systems using restricted boltzmann machine. In: 2017 IEEE international conference on systems, man, and cybernetics (SMC). IEEE, pp 2063–2068
28. Zeki S (2015) A massively asynchronous, parallel brain. Philos Trans Roy Soc B: Biol Sci 370(1668):20140174–20140174. https://doi.org/10.1098/rstb.2014.0174
29. Zhang, X., Zhu, X., Zhang, N., Li, P., Wang, L., et al.: Seggan: Semantic segmentation with generative adversarial network. In: 2018 IEEE fourth international conference on multimedia big data (BigMM). IEEE, pp 1–5 (2018)
30. Zhao Y, Li J, Xu S, Xu B (2016) Investigating gated recurrent neural networks for acoustic modeling. In: 2016 10th international symposium on Chinese spoken language processing (ISCSLP). IEEE, pp 1–5
31. Zhu F, Fan Z, Wu X (2014) Voice conversion using conditional restricted boltzmann machine. In: 2014 IEEE China summit & international conference on signal and information processing (ChinaSIP). IEEE, pp 110–114

Chapter 4
Getting Started with TensorFlow Deep Learning

Yeldar Toleubay and Alex Pappachen James

Abstract TensorFlow is an open-source software Python-based library developed by Google. It has high popularity in machine learning and deep learning area due to its simplicity, flexibility, and compatibility. In this chapter, we introduce the basic syntax of the TensorFlow and its main operations required to construct an artificial neural network. We briefly introduce the codes for building a recurrent neural network and convolutional neural network for example of MNIST based handwritten digits classification problem.

4.1 TensorFlow Basics

> "No programming language is perfect. There is not even a single best language; there are only languages well suited or perhaps poorly suited for particular purposes. Understanding the problem and associated programming requirements is necessary for choosing the language best suited for the solution."
>
> - Herbert G. Mayer Advanced C Programming on the IBM PC

TensorFlow is an open-source software library developed by Google for the production and research. The computational compatibility with different platforms such as CPUs, GPUs, and TPUs and the flexible architecture make TensorFlow extremely popular for machine learning application and neural networks design. The objective of this chapter is to introduce the basic syntax of TensorFlow and the main operations required to construct deep learning neural networks.

Y. Toleubay · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

Y. Toleubay
e-mail: yeldar.toleubay@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_4

4.1.1 Introduction to a Tensor

Scalar, vector, and algebraic matrix elements can be seen as special cases of a tensor. That is 0-dimensional tensor is a scalar, 1-dimensional tensor is a vector, 2-dimensional tensor is a matrix. A tensor is an R-dimensional array. Let's create our first tensor object using TensorFlow library.

Constant Tensor

Initially, we import our TensorFlow library using the `import` command. Then we create a constant tensor and print it. The output of the program will show that the variable `tensor1` is the tensor object of type string.

```
import tensorflow as tf
tensor1=tf.constant('Introduction')
print(tensor1)
#Output: Tensor("Const:0", shape=(), dtype=string)
```

TensorFlow has several built-in tensor generators that can create tensors of different size and type. The tensor's elements can be ones, zeros, any number or obey a distribution. These commands widely used in deep learning are useful during neural network modeling, for example, to assign weight values.

Build-in tensor generators

Let's create two-dimensional tensors consisting of zeros, ones, 1.5s and uniformly distributed numbers.

```
import tensorflow as tf
a = tf.zeros((3,4))
b = tf.ones((3,4))
c = tf.fill((1,3),1.5)
d = tf.random_normal((3,3),mean=0,stddev=0.4)
result1=sess.run(a)
result2=sess.run(b)
result3=sess.run(c)
result4=sess.run(d)
print(result1)
#Output:
[[ 0. 0. 0. 0.]
 [ 0. 0. 0. 0.]
 [ 0. 0. 0. 0.]]
print(result2)
#Output:
```

```
[[ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]]
print(result3)
#Output:
[[1.5 1.5 1.5]
print(result4)
#Output:
[[0.49991614 1.15280306 0.17223553]
 [-0.09865849 -0.15267047 0.28910062]
 [-0.07902408 -0.12034792 0.04202165 ]]
```

4.1.2 Execution of Basic Operations

As mentioned before, the variable `tensor1` is the tensor object. From the previous example, we know that tensors should not be treated as a Python variables. The usual operations such as `print()` cannot be directly applied to print the tensor. Instead, there is a class called `Session` required to run TensorFlow operations [1]. `Session` class allows to evaluate tensors and execute `Operation` objects.

Running a Session

Here, we try to print the content of the `tensor1` object in a session. For that reason create a session called `sess` and run the tensor inside `print` operation.

```
import tensorflow as tf
tensor1=tf.constant('Introduction')
sess=tf.Session()
print(sess.run(tensor1))
#Output: b'Introduction'
```

4.1.2.1 Mathematical Operations

There are several mathematical commands to work with tensors. Here we demonstrate the most important operations such as addition, multiplication and matrix multiplication required to construct neural network models.

Addition

Let's create two constant tensors a=10 and b=20 and find their sum using `add()`. Moreover, we create a session by context manager.

```
import tensorflow as tf
a=tf.constant(10)
b=tf.constant(20)
with tf.Session() as sess:
 addition=sess.run(tf.add(a, b))
print(addition)
#Output: 30
```

Multiplication

Here we calculate multiplication of 2.5 and 0.4 using `multiply()` operation.

```
import tensorflow as tf
a=tf.constant(2.5)
b=tf.constant(0.4)
with tf.Session() as sess:
 multiplication=sess.run(tf.multiply(a, b))
print(multiplication)
#Output: 1.0
```

Matrix multiplication

The `matmul()` operation is used to perform matrix multiplications.

```
import tensorflow as tf
a = tf.constant([ [1,3], [5,7] ]
b = tf.constant([ [9],[11] ])
with tf.Session() as sess:
 result=sess.run(tf.matmul(a,b))
print(result)
#Output:
[[ 42]
[122]]
```

4.1.3 Artificial Neuron Model

In the previous chapter, we got familiar with the artificial neuron model. We know that weights change their values during a network optimization. Therefore, weights and biases are defined as variable tensors, while x_{input} and y_{output} are called placeholder tensors. Initially, placeholders do not possess any value. However, they are filled by a data defined as feed dictionary during the testing and training of a model.

Let's recall the graphical representation of the artificial neuron presented in Fig. 3.3. The inputs and biases represented by separate nodes were connected to a summation node. The connection of a tensor object and an operation object are defined as a graph. The connection of graphs represents a neural network.

Graph example

We are going to create a model with both weight and bias set to 1. The x_{input} of type `float32` have 10 features.

```
import tensorflow as tf
w=tf.Variable(1)
b=tf.Variable(1)
n_features=10
x_ph=tf.placeholder(tf.float32,(None, n_features)
z_graph=tf.add(tf.multiply(w, x_ph), b)
```

Furthermore, you know that in order to get an output we put our `z` into an activation function. TensorFlow has several activation functions easily accessible by following commands:

Basic activation functions

Sigmoid function:

```
y_model=tf.sigmoid(z)
```

Hyperbolic tangent:

```
y_model=tf.tanh(z)
```

Rectified linear unit:

```
y_model=tf.nn.relu(z)
```

As soon as a model is ready, we should introduce a loss function, optimizer, and metrics into the network [2]. The loss function shows the difference between the actual and the predicted values. Our aim is to minimize the difference to get high accuracy. The optimizer is a learning method used to update weights and biases using feed data in order to obtain a low error in the loss function. Metrics are used to measure and display accuracy during the model compilation. Finally, the variables

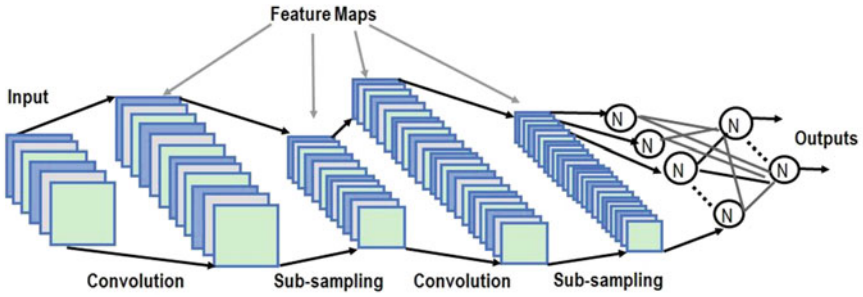

Fig. 4.1 CNN graphical representation

should be initialized using `tf.global_variables_initializer()` before the run of the session.

4.2 Neural Network Models

A human brain can solve efficiently different kind of tasks, for example, image, text and speech recognition or object detection. Unfortunately, such generality does not exist with a single artificial neural network. There are different architectures of neural network that works better for specific purposes. In this section, we are going to present a convolutional neural network and recurrent neural network.

4.2.1 Convolutional Neural Networks

Convolutional neural networks (CNN) are widely used for image classification and recognition. CNN require minimal preprocessing and detect visual patterns at the pixel level [3]. Although, there are different models of the CNN they all have the same architecture attributes initially proposed in LeNet-5. As an example, we present a CNN for handwritten MNIST digits recognition (Fig. 4.1).

CNN example

In this example, we are going to classify MNIST handwritten digits using CNN [4]. First, we create some helper functions, which initialize weights, biases and construct layers.

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
#The MNIST handwritten dataset is uploaded by following command
mnist = input_data.read_data_sets("MNIST_data/",one_hot=True)
#We initialize some functions which will be useful for neural
#network construction
def constr_weights(shape):
  constr_random_dist = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(constr_random_dist)
def constr_bias(shape):
  constr_bias_vals = tf.constant(0.1, shape=shape)
  return tf.Variable(constr_bias_vals)
#The 2-D convolution is created by following function:
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1]
                      padding='SAME')
#A max pooling layer is created by folowing function:
def max_pool_2by2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
            strides=[1, 2, 2, 1],  padding='SAME')
#The convolutional layer is obtained by the next function using
#conv2d:
def convolutional_layer(input_x, shape):
  W = init_weights(shape)
  b = init_bias([shape[3]])
  return tf.nn.relu(conv2d(input_x, W) + b)
#Following function creates a normal fully connected layer:
def normal_full_layer(input_layer, size):
  input_size = int(input_layer.get_shape()[1])
  W = constr_weights([input_size, size])
  b = constr_bias([size])
  return tf.matmul(input_layer, W) + b
```

Then we create placeholders and the network presented in Fig. 4.1. The dropout helps to avoid overfeeding, therefore, we design it using a placeholder.

```
#Placeholders
x = tf.placeholder(tf.float32,shape=[None,784])
y_true = tf.placeholder(tf.float32,shape=[None,10])
Construct layers:
x_image = tf.reshape(x,[-1,28,28,1])
convo_1 = convolutional_layer(x_image,shape=[6,6,1,32])
convo_1_pooling = max_pool_2by2(convo_1)
convo_2 = convolutional_layer(convo_1_pooling,shape=[6,6,32,64])
convo_2_pooling = max_pool_2by2(convo_2)
convo_2_flat = tf.reshape(convo_2_pooling,[-1,7*7*64])
full_layer_one = tf.nn.relu(normal_full_layer(convo_2_flat,1024))
hold_prob = tf.placeholder(tf.float32)
full_one_dropout = tf.nn.dropout(full_layer_one,keep_prob=hold_prob)
y_pred = normal_full_layer(full_one_dropout,10)
```

When our model is ready, we introduce a loss function, optimizer and metrics.

```
#Loss function:
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
```

```
              (labels=y_true,logits=y_pred))
#Optimizer:
optimizer = tf.train.AdamOptimizer(learning_rate=0.0001)
train = optimizer.minimize(cross_entropy)
Intialize variables
init = tf.global_variables_initializer()
#Session:
steps = 5000
with tf.Session() as sess:
  sess.run(init)
  for i in range(steps):
    batch_x , batch_y = mnist.train.next_batch(50)
    sess.run(train,feed_dict={x:batch_x,
          y_true:batch_y,hold_prob:0.5})
  #Metrics each 1000 steps:
  if i%1000 == 0:
    print('Currently executing step {}'.format(i))
    print('Accuracy is:')
  matches = tf.equal(tf.argmax(y_pred,1),tf.argmax(y_true,1))
  acc = tf.reduce_mean(tf.cast(matches,tf.float32))
  print(sess.run(acc,feed_dict={x:mnist.test.images,
y_true:mnist.test.labels,hold_prob:1.0}))
   print('\n')
Currently executing step 0
Accuracy is:
0.0539
Currently executing step 1000
Accuracy is:
0.9589
Currently executing step 2000
Accuracy is:
0.9733
Currently on step 3000
Accuracy is:
0.9805
Currently on step 4000
Accuracy is:
0.9834
```

The last iteration of handwritten digit classification resulted in 98.3% accuracy. Accuracy is affected by different learning rate, network size, and its parameters.

4.2.2 Recurrent Neural Networks

A recurrent neural network is the type of neural network used to solve sequence related problems such as a natural language, sound recognition or stock price prediction. The main difference of RNN from artificial neural network presented before, it sends its output back to the input. One of the well-known extended models of recurrent neural networks is the long short-term memory network [5]. The detailed

theory of LSTM networks are going to be discussed more in deep in Chaps. 11 and 12 later on, while, in this chapter, we briefly present the TensorFlow model of LSTM as an example of a recurrent neural network.

LSTM example

We perform the same classification problem of MNIST data-set, however, we will treat images as a sequence of pixels. Since MNIST has shape `(28,28)`, we can think of each image as 28 pieces of the 28 pixels long sequence [6]. Initially, we will define the model's basic parameters and then construct the model.

```
import tensorflow as tf
from tensorflow.contrib import rnn
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
learn_rate = 0.001
train_steps = 10000
batch_size = 128
steps = 2000
n_input = 28
tsteps = 28
n_features = 128
n_labels = 10
#Weights and placeholders
X = tf.placeholder("float", [None, tsteps, n_input])
Y = tf.placeholder("float", [None, n_labels])
weights={'out':tf.Variable(tf.random_normal([n_features,
        n_labels]))}
biases = {'out': tf.Variable(tf.random_normal([n_labels]))}
#The RNN network with LSTM as a basic cell. Unstuck gives a
#list of time-steps.
def RNN(x, weights, biases):
  x = tf.unstack(x, tsteps, 1)
  lstm_cell = rnn.BasicLSTMCell(n_features, forget_bias=1.0)
   outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)
  return tf.matmul(outputs[-1], weights['out']) + biases['out']
  Unstecks
logits = RNN(X, weights, biases)
prediction = tf.nn.softmax(logits)
#Loss function and optimizer
loss_op =tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
          logits=logits, labels=Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learn_rate)
train_op = optimizer.minimize(loss_op)
#Model evaluation:
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
init = tf.global_variables_initializer()
 with tf.Session() as sess:
  sess.run(init)
  for step in range(1, train_steps+1):
    batch_x, batch_y = mnist.train.next_batch(batch_size)
    batch_x = batch_x.reshape((batch_size, tsteps, n_input))
    sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
    if step % steps == 0 or step == 1:
     loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x, Y:batch_y})
  print("Step " + str(step) + ", Minibatch Loss Value= " +"{:.4f}".
  format(loss) + ", Training Accuracy= " + \ "{:.3f}".format(acc))
```

```
  print("Test starts!")
  test_l = 128
  test_data = mnist.test.images[:test_l].reshape((-1, tsteps,n_input))
  test_label = mnist.test.labels[:test_l]
 print("Testing Accuracy:", \ sess.run(accuracy, feed_dict={X:test_data,
                                  Y: test_label}
Step 1, Minibatch Loss Value= 2.6629, Training Accuracy= 0.133
Step 2000, Minibatch Loss Value= 1.3780, Training Accuracy= 0.578
Step 4000, Minibatch Loss Value= 0.9629, Training Accuracy= 0.719
Step 6000, Minibatch Loss Value= 0.7431, Training Accuracy= 0.758
Step 8000, Minibatch Loss Value= 0.6793, Training Accuracy= 0.797
Step 10000, Minibatch Loss Value= 0.3640, Training Accuracy= 0.875
Test starts!

Testing Accuracy: 0.882813
```

We have constructed two different neural networks and applied them to the same dataset. From the above examples, it is clear that although models and their optimization are different, the building approach of the neural networks generally remains the same. In other words, we define model parameters, placeholders, variables, loss function, optimization, metrics and the model itself despite the type of the neural network.

Chapter Highlights

- The TensorFlow is an open-source library that widely used in machine learning and deep learning networks design.
- A tensor is an R-dimensional array.
- The `Session` class allows to evaluate tensors and execute `Operation` objects
- `add()`, `multiply()` and `matmul()` are mathematical operations used to add, multiply and perform matrix multiplication of tensors respectively.
- The connection of a tensor object and an operation object are defined as a graph.
- As soon as a model is ready, a loss function, optimizer, and metrics should be introduced into the network.
- Initialize variables before a compilation of the session.
- Convolutional neural networks (CNN) are widely used in areas related to image classification and recognition.
- A recurrent neural network is the type of neural network used to solve a time sequence related problems such as a natural language, sound recognition or stock price prediction.
- The main difference of RNN from a CNN, it sends its output back to the input.

References

1. (2018). https://www.tensorflow.org/api_docs/java/reference/org/tensorflow/Session
2. (2018). https://www.tensorflow.org/tutorials/keras/basic_classification/
3. LeCun Y (2018) Lenet-5, convolutional neural networks. http://yann.lecun.com/exdb/lenet/
4. Mnist with CNN code (2018). https://github.com/nlintz/TensorFlow-Tutorials
5. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
6. Damien A (2018) A recurrent neural network (LSTM) implementation example using tensorflow library. https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/recurrent_network.py

Chapter 5
Speech Recognition Application Using Deep Learning Neural Network

Akzharkyn Izbassarova, Aziza Duisembay and Alex Pappachen James

Abstract Deep Neural Network (DNN) has demonstrated a great potential in speech recognition systems. This chapter presents two cases with successful implementations of speech recognition based on DNN models. The first example includes a DNN model developed by Apple for its personal assistant Siri. To detect and recognize a "Hey Siri" phrase program runs a detector based on a 5-layer network with 32 and 192 hidden units. To create an acoustic model, sigmoid and softmax activation functions are used together with a recurrent network. The second example is a region-based convolutional recurrent neural network (R-CRNN) designed by Amazon for rare sound detection in home speakers. This system is used in a security package called Alexa Guard. To allow efficient power and memory utilization while running complex machine learning algorithms special hardware is required. This chapter describes hardware solutions used in mobile phones and home speakers to process complex DNN models.

5.1 Introduction

Speech recognition is a technology used to make computers (or other kinds of machines) correctly identify spoken words. One of the types of speech recognition is a "speech-to-text". Such programs convert the spoken words to the text. Another type of speech recognition is an interactive speech recognizer. It is mostly used in personal assistants developed by Amazon (Alexa), Apple (Siri), Google (Google Assistant), etc. In contrast to "speech-to-text" software, they perform commands spoken by the

A. Izbassarova · A. Duisembay · A. P. James (✉)
Nazarbayev University, 53 Kabanbay batyr avenue, Astana 010000, Kazakhstan
e-mail: alex.james@nu.edu.kz

A. Izbassarova
e-mail: aizbassarova@nu.edu.kz

A. Duisembay
e-mail: aziza.duisembay@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_5

user. For example, they can turn the music on, tell us the weather forecast, and answer our questions.

Speech consists of a sequence of words that are, in their turn, are composed of a sequence of phones. Automatic speech recognition is based on two probability functions: an acoustic model that computes the probability of correspondence of an utterance to the input word sequence, and a language model that calculates the prior probability of the meaning of the input. These models are automatically trained with every input they receive. We can train the system using a database with various utterances pronounced by different speakers. Thus, using a wide range of inputs, the automatic speech recognizer improves its models making them more accurate.

In this chapter, the software and hardware designs of the speech recognition systems are described. "Software" section introduces two different DNN models used for a personal assistant and a home speaker. The following section describes the hardware design of Siri and System-On-Chip (SoC) development kit designed for Alexa Voice Service.

5.2 Software

5.2.1 Personal Assistant

One of the successful industry-scale examples of using deep learning for speech recognition is a personal assistant, Siri, introduced by an Apple company in 2011. This application allows users to interact with their devices through natural voice commands [11]. Set of available functions include typing specific instructions, opening required applications, navigating to the menu, browsing the web and scanning results. To accomplish these tasks, Siri must contain two features: understanding commands and converting speech to text. First, to invoke personal assistant, one need to speak out a "Hey Siri" keyword command. At the background, the device runs a speech recognition program all the time to identify these keywords among all surrounding noises and conversations [4]. This section introduces the design and work of this speech recognition program implemented in a Siri to recognize a "Hey Siri" command.

A speech recognizer developed to detect a keyword command and invoke a Siri is based on a DNN. Figure 5.1 shows the flow diagram for "Hey Siri" detection. First, sound signals are preprocessed before inputting them into a neural network. Speech signals caught by the microphone are sampled at a rate of 16,000 samples per second. Then, a sequence of frames is created and about twenty frames at a time are sent to the acoustic model. Acoustic model, in simple words, is our DNN which is trained and uploaded to the device. However, you can get a new version of the model if it is updated on the server. The acoustic model gives a distribution of probabilities of belonging to each of twenty phonetic classes. These classes are different combinations of phoneme "s" surrounded by front vowels. After that, detector calculates

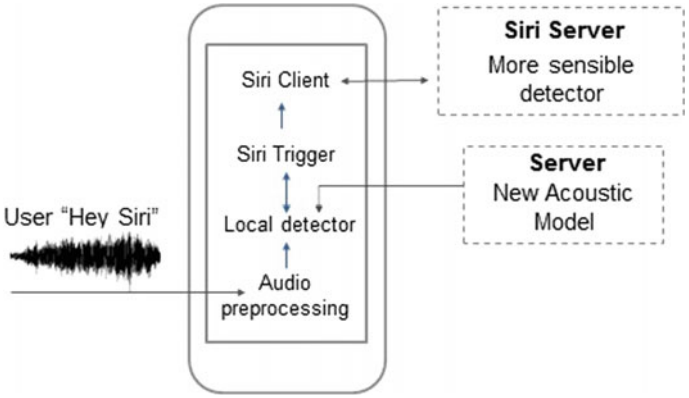

Fig. 5.1 Siri flow diagram [11]

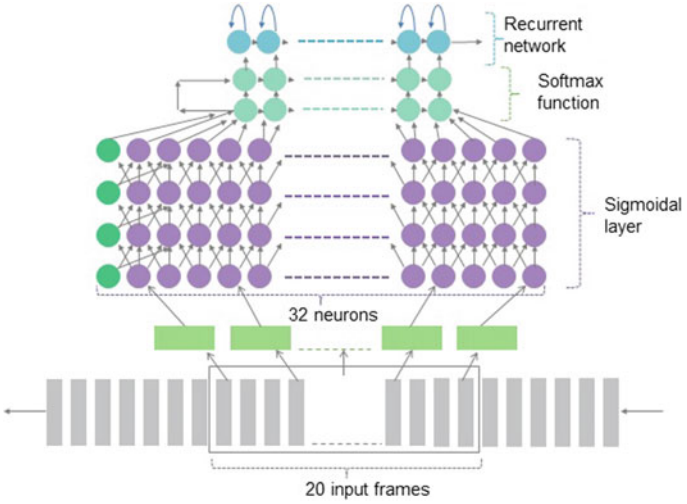

Fig. 5.2 DNN for "Hey Siri" recognition [11]

the score evaluating the chance that the "Hey Siri" phrase was pronounced. If this score is higher than the threshold value, then the program accepts your speech as a keyword command to invoke a Siri. The threshold value is flexible to adapt to different conditions, maximizing true positive detection but minimizing false-positive activation. A score value is compared to a lower and normal threshold value. If it exceeds a normal threshold, Siri is invoked. If the score is in between lower and normal thresholds program runs the second checker for a few seconds. The second checker is more sensible than the primary. This sensibility is achieved by increasing the number of neurons in the acoustic model. Figure 5.2 shows the neural network architecture used to create an acoustic model.

The DNN implemented by Apple has five hidden layers with an equal number of neurons in each layer. A sigmoid activation function is used in each layer except the last one. To get a probability distribution a softmax function is applied in the last layer. Since the network detects a pronounced phonetic class for every 0.2 seconds of the speech, it is essential to recognize the whole speech. For example, network recognizes that you said "Siri". However, to detect a "Hey Siri" phrase, network needs to know whether the previous output was a "Hey". The recurrent network can deal with such sequence related problems by holding the result of the previous input and using it for future input. The final score is obtained using a recurrent network.

5.2.2 Smart Home Speaker

In smart home speakers, speech recognition is performed in a similar way as in the smart assistants. However, smart home speakers are more adapted to home issues, and thus, have some additional functions. For example, Amazon created a security package called Alexa Guard for its Echo smart speaker. Alexa Guard can send an alarm message to a user's phone if it recognizes any extraordinary or suspicious sound, like a signalization from a gas/smoke detector or a sound of broken glass.

The Amazon researchers have developed a Region-based Convolutional Recurrent Neural Network (R-CRNN) for rare sound identification [7].

The model of the R-CRNN is presented in Fig. 5.3. It is composed of three main parts: a convolutional recurrent neural network (CRNN), region proposal network (RPN), and a final classifier. The CRNN module obtains high-level features from the input spectrogram. Based on the obtained features, the RPN calculates possible events by detecting one-dimensional regions of an audio input that possibly contain the searched sounds. Further, the classifier improves the center of event proposals to generate an audio event prediction.

5.2.2.1 Convolutional Recurrent Neural Network

The architecture of the CRNN is presented in Fig. 5.4. It takes 30 s audio streams and obtains a high-level feature map. Each clip is further divided into a sequence of 46 ms frames with a 23 ms shift [8].

The input of the CRNN is generated by combining 64-dimensional log filter bank energies (LFBEs) obtained from each frame. The residual network introduced in [6] is used as a convolutional network in CRNN. The residual network helps to solve the degradation problem, i.e., drop in accuracy as the depth of the network grows. It has two convolutional blocks inside. The first convolutional layer has a kernel size of 7*7 with stride value of 2*1 and 64 channels. Here, the first dimension is time, and the second dimension is a frequency. After that max pooling with a 3*3 kernel and 2*2 stride is applied to reduce the dimension. Then it is followed by several convolutional layers with a 3*3 kernel and 64 output channels. It is continued with

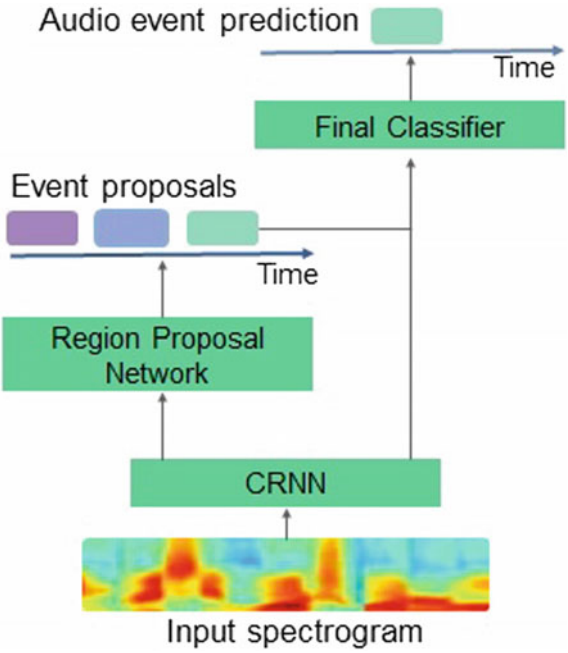

Fig. 5.3 R-CRNN model [7]

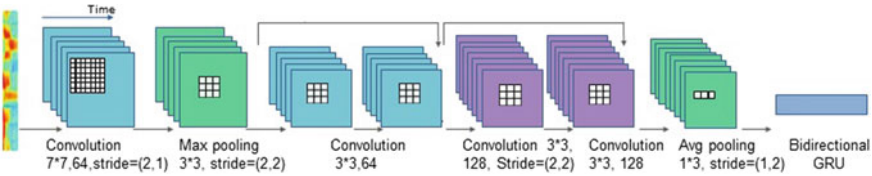

Fig. 5.4 CRNN [7]

convolutional layers of the same kernel size but with an increased number of output channels. These 2D convolutional kernels generate a high-level feature map with a time resolution of 186 ms. After a bidirectional gated recurrent unit (GRU) the final size of the feature map is (162, 2U), where U is the number of units in the last layer.

5.2.2.2 Region Proposal Network

The Amazon researchers use RPN developed in [10] to compute event proposals for the audio event detection (AED). The simplified RPN computes event proposals in a time-dimensional search space. It takes as an input a high-level feature map from the CRNN and produces a set of possible events also called event proposals. Each of the produced event proposals are evaluated for the presence of audio events. The

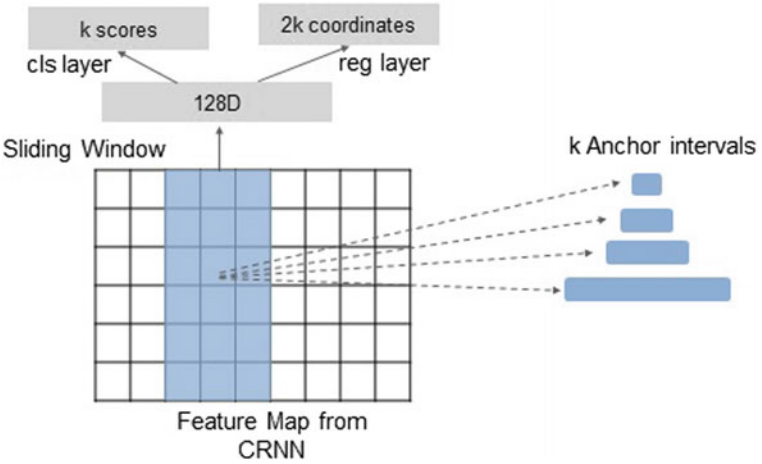

Fig. 5.5 Region proposal network [7]

main purpose of this part is to identify the region of the event in a very short time. This helps to reduce the time intervals that will be evaluated in the final classifier.

As presented in Fig. 5.5, a 3*2U window slides over the high-level feature map to obtain a lower-dimensional feature. The perceptive field of the window is 557 ms.

For the RPN training, all anchor intervals are appointed to a ground-truth binary label. The label indicates whether the interval contains a target event or not. The RPN cost function is the following:

$$L(\{p_i\}, \{t_i\}) = \sum_i L_{cls}(p_i, p_i^*) + \lambda \sum_i p_i^* L_{reg}(p_i, p_i^*), \tag{5.1}$$

where i defines the index of an anchor interval, and p_i is the probability that the anchor i contains a target event. If the anchor interval overlaps with the target events, the ground-truth label (p_i^*) becomes equal to 1. If not, it becomes equal to 0. L_{cls} is a cross entropy of binary classification. t_i is a vector of coordinates of the predicted 2D interval proposal, t_i^* is the vector of the ground-truth event interval. L_reg is a robust loss function extracted from [5]. λ is a balancing coefficient for reduction of classification error and regression error.

5.2.2.3 Final Classifier

Next, the non-maximum suppression (NMS) presented in [9] is used to cut out highly-overlapped interval proposals. In total, 100 proposals with higher probabilities of the audio event are sent to the classifier. The output of the classifier is audio event predictions.

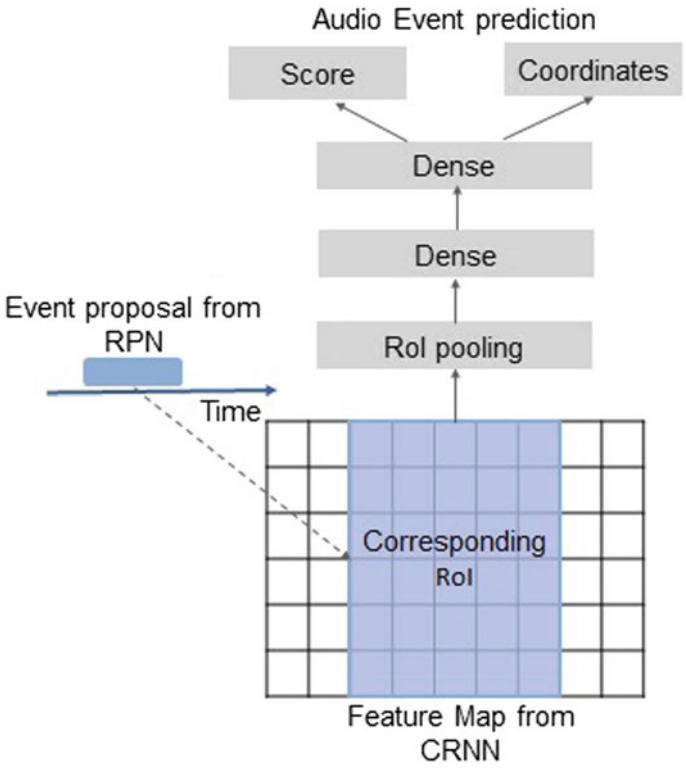

Fig. 5.6 Final classifier [7]

As shown in Fig. 5.6, the regions of interest (RoI) of the high-level feature map are cut for each event proposal correspondingly. The modified region is transmitted to a RoI pooling layer to produce a feature vector. This vector is further sent to the dense layers with a dropout rate of 0.5, followed by two output layers. The output layers compute the probability of the region to contain the target event, and the regression to fix the center of the event. The NMS removes highly-overlapped events in the generated probabilities.

The final classifier is trained based on the loss function defined in (5.1). For final decision, events only with a probability of having an audio event higher than 0.8 are considered.

5.3 Hardware

5.3.1 Apple

A speech recognition program used by a Siri requires a special hardware architecture to perform deep learning algorithms in mobile phones in a fast and energy-efficient manner. To avoid continuous load on a processor and memory, a small low-power auxiliary processor is used to run a Siri trigger program. This processor with limited power runs a "Hey Siri" detector with a small acoustic model. It uses DNN with only five layers and 32 neurons in the hidden layers. If the small acoustic model detects the key phrase a larger DNN with 192 neurons in each layer will be run in the main processor. This helps to save a battery life [4].

In 2018 Apple has introduced an updated A12 Bionic chip during the presentation of iPhone XR and XS models [2]. AX Bionic chip is a family of Systems on a Chip (SoC) designed by Apple for their mobile phones. The architecture of the latest model is shown in Fig. 5.7.

A12 Bionic chip is the first 7-nanometer chip packed with 6.9 billion of transistors. This chip is embedded in Apple's mobile phones to improve the performance of applications and programs using AI. Unlocking phone with a Face ID, creating Animoji and Siri shortcuts, recognizing speech and other features require running of machine learning algorithms. To enable the processing of such complex algorithms A12 Bionic chip utilizes 4-core GPU, 6-core CPU, and 8-core Neural engine [3].

CPU has two high-performance cores for complex computational tasks and four efficiency cores for everyday tasks. To improve graphics processing A12 uses four

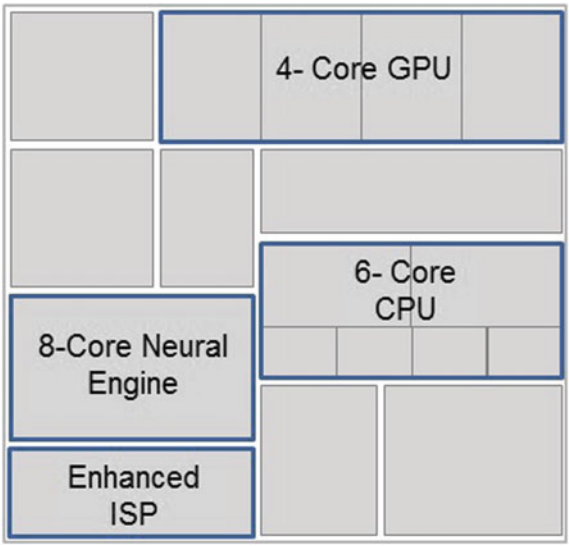

Fig. 5.7 Block diagram of the A12 Bionic chip [3]

core GPU. Bionic chip has a dedicated machine learning engine called Neural engine. This engine decides by analyzing the neural network data whether to run it on a CPU, GPU or a Neural engine. According to Apple, A12 Neural engine can process 5 trillion operations per second.

5.3.2 Alexa Voice Service

At the beginning of 2018, Amazon launched their first SoC development kits designed for Alexa Voice Service (shortly, Alexa). The kits have a simpler architecture and require less material for construction. One of the SoC kits is A113X1 Far-Field Development Kit by Amlogic. It is created for the integration of Alexa into smart home speakers, in-car assistants, and other devices. The SoC has such advanced features as echo removal, noise reduction, and audio post analysis. As will be discussed further, the A113X chip provides high performance along with low power consumption compared to earlier versions of AVS hardware.

Already existing systems designed for Alexa consist of the following processing units: an audio front end (AFE) that filters the input audio, a wake word engine (WWE) used for recognition of the wake word "Alexa" and the Alexa client that sends process outputs to and receives commands from the cloud.

The new kits for Alexa have SoCs performing all three modules' functions on one chip. The A113X1 Far-Field Development Kit comes with the A113X main board. The A113X board is constructed specially for smart home speakers. It is composed of the following units:

- CPU (Quad-core Cortex-A5);
- Audio processing unit;
- Runtime program;
- Peripherals for different smart home devices.

The A113X board can be connected to a microphone board, two speaker boards, two stereo speakers, and a UART debug board. It also comes along with a 12V power adapter. The interface of the A113X board is shown in Fig. 5.8 [1].

The A113X chips are based on the Quad-Core Cortex-A5 CPU. It has 16-channel I2S interface, TDM/PCM input, and output (up to 8 channels), 8 PDM channels, 1Gb Ethernet and LVDS/MIPI-DSI panel output. The RAM is 512 Mbyte DDR3/4.

Features of Cortex-A5 are:

- Single-issue;
- Thumb-2 instruction set encoding;
- Jazelle RCT;
- 1.57 DMIPS/MHz.

Thus, the A113X1 Far-Field Development Kit for AVS is a versatile solution designed by Amazon to make users able to build smart speaking devices with far-field speech recognition performance.

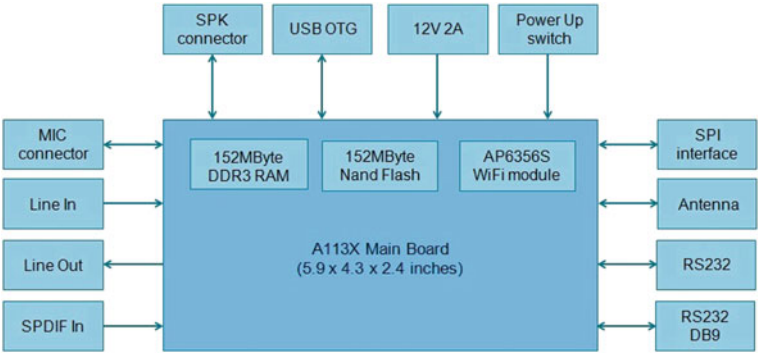

Fig. 5.8 Interface description of A113X board [1]

5.4 Summary

This chapter has described two successful realizations of speech recognition using a DNN. Apple's Siri is a mobile personal assistant that can recognize commands, give an answer to questions and convert a speech to text. In this chapter, we discussed only one function of the Siri program. It can be evoked by a special speech command using a detector based on DNN. To run such a program on a mobile phone Apple uses A12 Bionic chip with a Neural engine.

Next example of the DNN implementation is a smart home speaker developed by Amazon. An Alexa Guard security package can identify a source of different specific sounds detected at home. Such a large neural network requires powerful hardware in smart home speakers. Amazon developed SoC development kit with A113X chip. This SoC kit has 512 Mb of dedicated memory and a Quad-Core Cortex-A5 CPU.

- Siri uses a 5-layer DNN with sigmoidal and softmax activation functions and a recurrent network
- R-CRNN is used for a rare sound detection in Alexa Guard and it consists of CRNN, RPN, and Final Classifier
- CRNN has several convolutional and max-pooling layers to generate a high-level future map of the input audio signal
- A12 bionic chip has 4-core GPU, 6-core CPU and 8-core Neural engine to run machine learning algorithms on iPhones
- Amazon uses A113X chip based on the Quad-Core Cortex-A5 CPU for smart home speaker

References

1. Amlogic, Ltd (2018) A113x1 development kit user guide. http://openlinux2.amlogic.com/download/doc/
2. Apple: September event 2018 - apple. https://www.youtube.com/watch?v=wFTmQ27S7OQ
3. Apple (2018) A12 bionic the smartest, most powerful chip in a smartphone. https://www.apple.com/iphone-xs/a12-bionic/
4. Eckel E (2018) Apple siri: an insider's guide. http://b2b.cbsimg.net
5. Girshick RB (2015) Fast R-CNN. CoRR http://arxiv.org/abs/1504.08083
6. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition, pp 770–778
7. Kao C, Wang W, Sun M, Wang C (2018) R-CRNN: region-based convolutional recurrent neural network for audio event detection, pp 1358–1362
8. Lim H, Park J, Han Y (2017) Rare sound event detection using 1D convolutional recurrent neural networks. Technical report, DCASE2017 Challenge
9. Neubeck A, Van Gool L (2006) Efficient non-maximum suppression, pp 850–855. http://dx.doi.org/10.1109/ICPR.2006.479
10. Ren S, He K, Girshick RB, Sun J (2017) Faster r-cnn: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 39(6):1137–1149
11. Siri Team (2017) Hey Siri: an on-device dnn-powered voice trigger for apple's personal assistant. Machine Learning Journal

Chapter 6
Deep-Learning-Based Approach for Outdoor Electrical Insulator Inspection

Damira Pernebayeva and Alex Pappachen James

Abstract The outdoor electrical insulators are widely used in power transmission and distribution networks. They provide electrical isolation and mechanical support to conductors. Overhead insulators need to be inspected and monitored regularly to prevent faults and provide permanent electricity for consumers. The condition monitoring system for insulators is quite a challenging task due to the harsh operating conditions and the large number of insulators and their wide distribution in power transmission network. Traditional inspection methods for insulator evaluation are time-consuming, labor-intensive and costly. However, the inspection system based on deep learning model jointly with Unnamed Aerial Vehicle (UAV) can provide a remote, real-time condition evaluation in efficient and cost-effective manner. In this chapter, the frameworks of the deep neural network for insulator inspection are presented. The deep architecture including critical tasks such as insulator localization and insulator state evaluation is provided. The performance of existing deep learning models based on different architecture is also given.

6.1 Introduction

The outdoor electrical insulators are essential and widely deployed assets in power transmission and distribution systems. They play an essential role in the reliability of the power system as insulators provide mechanical support and electrical isolation to conductors [6]. However, due to the harsh weather conditions, surface contamination, mechanical cracks, and aging events, the outdoor insulators tend to deteriorate and fail. The failures of insulators may cause serious power outages, interruptions in power supply and economic losses. Traditional inspection methods for overhead electrical insulators include field and airborne surveys, which are labor-intensive,

D. Pernebayeva · A. P. James (✉)
Nazarbayev University, 53 Kabanbay Batyr ave, Astana, Kazakhstan
e-mail: alex.james@nu.edu.kz

D. Pernebayeva
e-mail: damira.pernebayeva@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_6

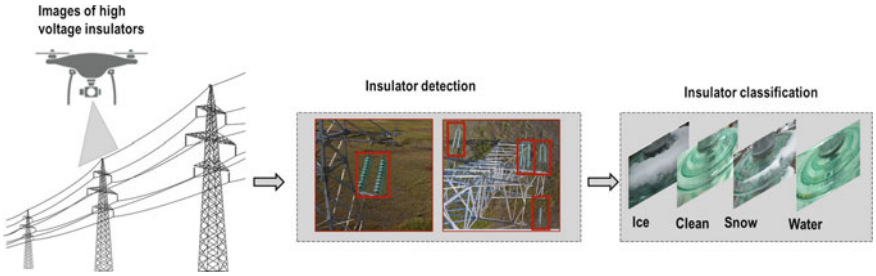

Fig. 6.1 Visual inspection of high voltage overhead insulators using UAV and deep neural network system

time-consuming and costly. Also, continuous monitoring is a challenging event due to changes in weather conditions, geographically isolated locations and terrains, the cost factors such as for experts and measurement setups, and sheer volume of insulators to be analyzed. Therefore, an automated condition monitoring of outdoor transmission line insulators using machine learning (ML) techniques is an essential task in the power industry. The inspection system integrated with deep learning networks can effectively and continuously manage the state of insulators and replace human decisions. Furthermore, an airborne monitoring system using Unnamed Aerial Vehicle (UAV) has become a new trend of power line inspection due to its low cost and reliable operation. In aerial inspection via UAVs, the camera captures the images of insulators followed by image processing and analysis for object position recognition and condition evaluation (Fig. 6.1).

The deep learning is a branch of a family of ML based on data learning and representation algorithms. Due to the improvements in computing power and data accessibility, the deep learning algorithms have achieved great performance in different computer vision tasks such as language processing [21], image classification [15] and face recognition [10]. DL neural network architecture is inspired by the biological neural network of the mammal brain, which is the deep architecture, with input information represented at multiple levels of abstraction. Multilayer neural networks characterize DL. A convolution neural network (CNN) is a representative model of deep learning, which has proven to be a powerful tool in visual recognition problems for different application domains including power line infrastructure monitoring [3]. CNN is utilized to extract mid- and high-level features from input images by shrinking the feature maps. In recent years, deep neural network architectures have been proposed for the remote inspection of outdoor insulators including two main tasks:

- insulator detection: localization the position of the insulators in aerial images
- classification: condition evaluation of insultors

This chapter presents different frameworks based on deep neural networks and UAV system for the insulator monitoring system. The architectures of CNN models, includ-

ing CNN, Faster R-CNN, for insulator detection and localization are provided. Further, the performance of existing defect recognition methods using different architectures are presented.

6.2 Insulator Localization Algorithms

The localization and detection of overhead insulators in aerial images is a prerequisite task to analyze the insulator condition and distinguish normal surface from the abnormal surface. In the last decade, many studies have focused on automatic insulator localization methods in aerial images. The method based on Orientation Angle Detection and Binary Shape Prior Knowledge (OAD-BSPK) [22] was proposed to localize multiple transmission line insulators in complex aerial images. There are several approaches based on features matching for insulator detection using local features [7] and a coarse-to-fine strategy for effective feature matching. The Gabor features [20] and Local Binary Patterns histogram Fourier (LBP-HF) features [13] used for insulator identification and background suppression from aerial images for feature extraction followed by Support Vector Machine (SVM) for classification. The other possibilities include the use of color, shape and texture features of insulator [19], local features and RANSAC-based clustering approach [11], Otsu thresholding algorithm with morphological processing [8], and region-based active contour segmentation framework [4]. In recent years, the application of deep convolutional neural network outperforms the traditional machine learning algorithms in object detection tasks and has shown a promising prospect in terms of accuracy and reproducibility.

6.2.1 CNN

The CNN is a biological inspired neural network widely used for image-related analysis. A typical CNN architecture includes multiple feature extraction layers, followed by fully connected layers and final a classifier layer. The feature extraction layers are composed of three main layers such as convolutional, activation and pooling. The output of the network is calculated by convolution, sub-sampling, non-linear activation function, and fully connection layer. The convolutional layer computes the feature map of an input image by applying and sliding a trainable filter bank with a certain size of l x l x k (k-number of channels, same as an input image) around the image. The convolution is a repeatable multiplication of the pixel values of the input image with the values in the filter for every location on the input image. The following layer is the pooling layer that used to reduce feature dimension by down-sampling to avoid overfitting. Activation layer usually, utilized by rectified linear unit (ReLU), is a simple and fast computing function that has a unilateral asymmetric structure. The fully connected layers normally constitute the last few layers of a CNN. The

structure of the fully connected layers is the same as that of a traditional neural network. The structure of the CNNs can be divided into the two types of networks: feed-forward and back-propagation. A feed-forward CNN model computes the output based on the given input data, whereas the back-propagation network adjusts the network parameters according to the change of loss function value.

6.2.2 CNN Architectures

There are various architectures proposed to use for insulator detection task. The cascading architecture of deep CNN [18] is a good example for both localization and defect detection of insulators (Fig. 6.2). The proposed model combines two networks: a region proposal network (RPN) and defect detector network (DDN). An RPN technique is used to determine the location of the insulator in aerial images, and then DDN is applied to detect any insulator defects. In the model the features extraction is done using the CNN of VGG-16 network [16] and fed to the RPN model. Next, in the RPN with convolution layer and ReLU function, the feature map is converted into 512-dimensional feature and fed to the next fully connection layer followed by pooling layer. Then, insulator image patches are cropped to the smallest rectangular area and provided to the DDN as input. The DDN uses the ResNet101 [2]convolution layers, RPN and region of interest (ROI) Pooling layers between two sets of convolution layers. The output of the DDN provides both the detection and the insulator status.

Another simple example of CNN model [1] consists of three convolution layers, three pooling layers, and one fully connected layer. The classification accuracy for detection of glass insulators from background reaches 89.9% with 12,000 of real aerial insulator images. A similar architecture [17] based on three convolution layers and two sub-sampling layers, followed by fully connection layer was tested on UAV images of insulators containing 1500 as training and 500 as test data. The detection accuracy of correctly positioning of insulators in images is 88.4%.

A Faster R-CNN is a widely proposed model for localization of the electric insulators. It is a typical end-to-end object detection framework that is composed of two main modules: RPN and Fast R-CNN network [14]. The RPN shares full-image con-

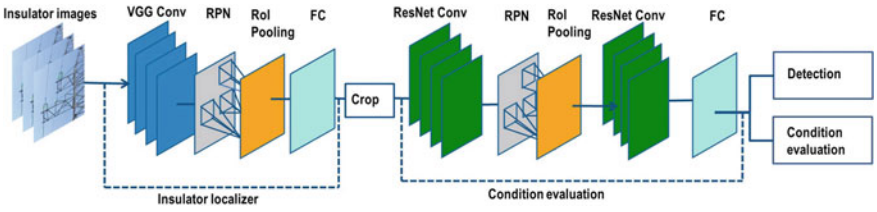

Fig. 6.2 Cascading CNN architecture for insulator localization and defect detection

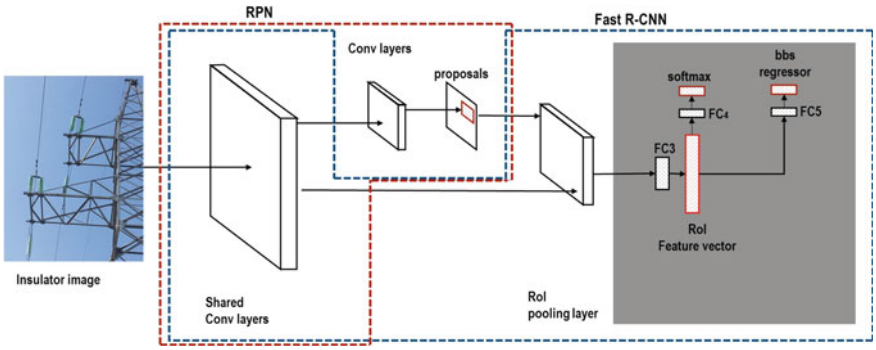

Fig. 6.3 Faster R-CNN architecture

volutional features with the detection network. The RPN takes an input image and generates high-quality rectangular region proposals, that may contain the object to be detected. To generate region proposals, the network slides a spatial window through the feature map of the shared convolutional layers and simultaneously predicts k anchors of different scales and aspect ratios at each sliding position [5]. Then, the region proposals are selected and fed to the region of interest (RoI) pooling layer of the Fast R-CNN. The RoI pooling layer uses max pooling to transform the features of the region proposals into fixed-length feature vectors. Finally, in Fast R-CNN, the obtained feature map is reshaped into high dimensional for further processing. At the final stage, the probability of a class of the candidate is computed (Fig. 6.3).

6.3 Insulator Defect Detection

Once the location of the insulator is determined, insulator state classification can be performed. As it was mentioned earlier, the outdoor high voltage insulators operate in a harsh environment, and they are subjected to electrical and mechanical stresses leading to system failures. The failures are mostly caused due to surface pollution, high humidity and mechanical damages. Moreover, the combination of humidity and contamination increases the probability of flashover phenomena, which is an air-gap breakdown, that occurs at the interference of surface and air.

Recently, many studies have been conducted to evaluate the condition of the insulators. The multi-layer CNN application is proposed to use for flashover evaluation using UV imaging [12]. The framework of a system includes the pre-processing UV images, the CNN training, and the detection output. The structure of the proposed deep neural network includes convolutional layers, followed by rectified linear unit (ReLU) activation function, pooling layer, and normalization layer. To evaluate the condition of the insulator under different operating conditions such as temperature, humidity, pollution level, the framework based on the CNN was applied [9] using

Table 6.1 Performance of different CNN architectures for insulator detection and classification

Input layer	Convolution layer	Pooling layer	Fully connected layer	Accuracy%	Ref
6000 × 4000	1st layer: 28 × 28 × 8 2nd layer: 10 × 10 × 20	1st layer: 14 × 14 2nd layer: 5 × 5	1st layer: 1 × 1 × 120	88.4	[17]
32 × 32	1st layer: 28 × 28 × 20 2nd layer: 10 × 10 × 50	1st layer: 14 × 14 2nd layer 5 × 5	1st layer: 500 2nd layer: 2	93	[21]
227 × 227 × 3	1st layer: 55 × 55 × 96 2nd layer: 27 × 27 × 256 3d layer: 13 × 13 × 384 4th layer: 13 × 13 × 384 5th layer: 13 × 13 × 256	1st layer: 27 × 27 2nd layer: 13 × 13 3d layer: 6 × 6	1st layer: 4096 2nd layer: 4096 3d layer: 3	96.8	[20]
256 × 256 × 3	1st layer: 250 × 250 × 4 2nd layer: 120 × 120 × 6 3d layer: 38 × 38 × 9	1st layer: 125 × 125 2nd layer: 40 × 40 3d layer: 19 × 19	1st layer: 65280	89.9	[16]

infrared imaging of defective insulators. Similarly, the model consists of three main parts as infrared images pre-processing, training the CNN, and the detection result. The architecture is composed of two convolution feature maps followed by two pooling layers, one fully connected layer and the mean square error as the loss function. The summary of recent CNN algorithms with high detection and classification accuracy is given in Table 6.1.

6.4 Summary

The inspection of outdoor high voltage insulators is highly required to maintain safe and reliable performance of power transmission lines. Since traditional inspection approaches are time-consuming and costly, deep learning algorithms shown great advances in the object detection and considered to be a feasible and effective tool for inspection. There are two main tasks for the remote aerial inspection of insulators using UAVs and deep neural networks: insulator detection and condition evaluation. In this chapter, existing approaches based on deep convolutional neural network

architectures are presented for both localization and defect detection of outdoor high voltage insulators. The detection and classification accuracy of different models using multi-layer neural networks is provided.

Chapter Highlights

- CNN is a deep neural network architecture used for image classification
- Faster R-CNN combines RPN and Fast R-CNN for object detection
- Cascading CNN based on RPN and DDN detects and evaluates the object

References

1. Cheng H, Chen R, Wang J, Liu X, Zhang M, Zhai Y (2018) Study on insulator recognition method based on simulated samples expansion. In: 2018 Chinese control and decision conference (CCDC). IEEE, pp 2569–2573
2. Dai J, Li Y, He K, Sun J (2016) R-fcn: object detection via region-based fully convolutional networks. In: Advances in neural information processing systems. pp 379–387
3. Gubbi J, Varghese A, Balamuralidhar P (2017) A new deep learning architecture for detection of long linear infrastructure. In: 2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA). IEEE, pp 207–210
4. Iruansi U, Tapamo J, Davidson I (2015) An active contour approach to insulator segmentation. In: AFRICON, 2015. IEEE (2015), pp 1–5
5. Kang G, Gao S, Yu L, Zhang D (2018) Deep architecture for high-speed railway insulator surface defect detection: Denoising autoencoder with multitask learning. IEEE Trans Instrum Meas
6. Kiessling F, Nefzger P, Nolasco JF, Kaintzyk U (2014) Overhead power lines: planning, design, construction. Springer
7. Liao S, An J (2015) A robust insulator detection algorithm based on local features and spatial orders for aerial images. IEEE Geosci Remote Sens Lett 12(5):963–967
8. Liu, G., Zhu, Z., Jie, X.: Orientation and damage inspection of insulators based on tchebichef moment invariants. In:2008 International conference on neural networks and signal processing. IEEE, pp 48–52 (2008)
9. Liu Y, Pei S, Fu W, Zhang K, Ji X, Yin Z (2017) The discrimination method as applied to a deteriorated porcelain insulator used in transmission lines on the basis of a convolution neural network. IEEE Trans Dielectr Electr Insul 24(6):3559–3566
10. Lu J, Liong VE, Zhou X, Zhou J (2015) Learning compact binary face descriptor for face recognition. IEEE Trans Pattern Anal Mach Intell 37(10):2041–2056
11. Oberweger M, Wendel A, Bischof H (2014) Visual recognition and fault detection for power line insulators. In: 19th computer vision winter workshop
12. Pei S, Liu Y, Ji X, Geng J, Zhou G, Wang S (2018) Uv-flashover evaluation of porcelain insulators based on deep learning. IET Science Measurement & Technology
13. Prasad PS, Rao BP (2016) Lbp-hf features and machine learning applied for automated monitoring of insulators for overhead power distribution lines. In: International conference on wireless communications, signal processing and networking (WiSPNET). IEEE, pp 808–812 (2016)

14. Ren S, He K, Girshick R, Sun J (2017) Faster r-cnn: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 6:1137–1149
15. Romero A, Gatta C, Camps-Valls G (2016) Unsupervised deep feature extraction for remote sensing image classification. IEEE Trans Geosci Remote Sens 54(3):1349–1362
16. Silberman N, Hoiem D, Kohli P, Fergus R (2012) Indoor segmentation and support inference from rgbd images. In: European conference on computer vision. Springer, pp 746–760 (2012)
17. Tao G, Fengxiang C, Wei W, Ping S, Lei S, Tianzhu C (2017) Electric insulator detection of uav images based on depth learning. In: 2017 2nd international conference on power and renewable energy (ICPRE). IEEE, pp 37–41 (2017)
18. Tao X, Zhang D, Wang Z, Liu X, Zhang H, Xu D (2018) Detection of power line insulator defects using aerial images analyzed with convolutional neural networks. IEEE Trans Syst Man Cybernet: Syst
19. Wang W, Wang Y, Han J, Liu Y (2016) Recognition and drop-off detection of insulator based on aerial image. In: 2016 9th international symposium on computational intelligence and design (ISCID), vol 1. IEEE, pp 162–167 (2016)
20. Wang X, Zhang Y (2016) Insulator identification from aerial images using support vector machine with background suppression. In: 2016 international conference on unmanned aircraft systems (ICUAS). IEEE, pp 892–897 (2016)
21. Young T, Hazarika D, Poria S, Cambria E (2017) Recent trends in deep learning based natural language processing. arXiv:1708.02709
22. Zhao Z, Liu N, Wang L (2015) Localization of multiple insulators by orientation angle detection and binary shape prior knowledge. IEEE Trans Dielectr Electr Insul 22(6):3421–3428

Part II
Memristor Logic and Neural Networks

Chapter 7
Learning Algorithms and Implementation

Olga Krestinskaya and Alex Pappachen James

Abstract This chapter provides a brief overview of learning algorithms and their implementations on hardware. We focus on memristor based systems for leaning, as this is one of the most promising solutions to implement deep neural network on hardware, due to the small on-chip area and low power consumption.

7.1 Introduction

Learning is an essential part in the implementation of different neuromorphic architectures, such as neural networks [12], long short term memory [20], hierarchical temporal memory [10], etc. There are various learning algorithms used in the implementation of the learning architectures. Hebbian learning rule [4] and backpropagation algorithm [2, 15] are the most common in this variety. This chapter provides a brief overview of the learning algorithms for different systems and illustrates the example of the hardware implementation of the backpropagation algorithm. Section 7.2 if this chapter covers the main learning rules and algorithms that can be used to train deep learning systems, and provides a mathematical framework for backpropagation algorithm. Section 7.3.1 introduces the overview of the implementation of learning algorithms in different memristive learning architectures and neural networks. Section 7.3.2 illustrates the example of analog hardware implementation backpropagation algorithm.

O. Krestinskaya · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

O. Krestinskaya
e-mail: ok@ieee.org

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_7

7.2 Learning Rules and Algorithms

This section provides a brief overview of several learning rules and mathematical framework of the backpropagation algorithm, as this is one of the most frequently used algorithms to train deep neural networks.

7.2.1 Learning Rules

Each learning architecture has a particular set of algorithms and learning rules that can be applied to enable the learning. Table 7.1 contains an overview of several learning architectures and corresponding learning rules and algorithms that can be applied there.

The machine learning methods can also be divided into online learning and offline learning methods. Online learning is faster and in most of the cases includes simple algorithms, as complicated learning algorithms cannot be implemented due to computational complexity, especially on hardware. The offline learning can include complicated algorithms, and most of the training processes are performed offline storing training results in a memory. The amount of storage space for online learning algorithms is usually smaller, comparing to offline learning algorithms, which is critical for hardware implementation of learning architectures.

The learning rules can be divided into four main groups:

- Hebb's learning rules. These rules include original Hebb's and Anti-Hebb's rules. Hebbian learning implies that if the synapses are activated synchronously, the strength of the connection between them increases, and vice versa. The anti-Hebbian learning implies that the strength of connection decreases the efficiency of a presynaptic neuron to elicit the response of a postsynaptic neuron [3, 22].
- Error-based learning rules. These rules are derived from Hebb's rules, such as simple perceptron. perceptron learning is supervised learning with reinforcement. The main principle is to minimize the error by adjusting the weights in a series of small updates. The other learning method, which is similar to the perceptron

Table 7.1 Learning rules corresponding to enabling learning architectures

Enabling learning architectures	Learning rules and algorithms
LSTM, recurrent neural network	LSTM algorithm
Convolutional neural network	Backpropagation algorithm
HTM	Hebb's rules
Neuronal boltzmann machines	Contrastive divergence learning rule
Spiking neural network	Hebbian learning, backpropagation, STDP
Belief networks	Parameter estimation (statistical approach)

learning is delta rule, which is a foundation of the backpropagation algorithm. It is based on error calculation. The other rule is a Manhattan rule, which is different from the delta rule by the binary quantization process. The update process in this rule has a constant update value but different signs. One of the other rules derived from Hebb's learning approach is Oja learning rule, which is a Hebb's rule with normalization [19, 25].

- Rate and gradient-based learning rules. These rules are more complex and include backpropagation process. These rules are based on the gradient of change of the error. The backpropagation algorithm with gradient descent is one of the widely used algorithms. Bienenstock-Cooper-Munro (BCM) rule is the other rate-based learning rule, which depends on the rate of pre- and post-synaptic spikes. It is based on the LTD method [1, 17].
- Time-based learning rules. These rules include STDP rules. STDP rule is based on the delay between the presynaptic and postsynaptic firing [24].

7.2.2 *Backpropagation Algorithm*

Backpropagation algorithm is one of the most frequently use algorithms for deep learning systems and deep neural networks. Equations 7.1–7.14 illustrate generalized representation of backpropagation algorithm [14, 23]. Parameters $w_{ji}^{(n)}$ represent weights connecting jth input unit to ith output unit of layer n. Equation 7.1 shows all the inputs to jth unit of the hidden layer n. Parameter X illustrates the input to the neural network, and parameter $Z_j^{(n)}$ shows the overall input to the unit including all the synapses connected to it.

$$Z_j^{(n)} = \sum_i w_{ji}^{(n)} X_i^{(n)} + \theta_j^{(n)} \tag{7.1}$$

Equation 7.2 illustrates the output of nth layer $Y^{(n)}$. Parameter f is an activation function of nth layer .

$$Y_j^{(n)} = f_j^{(n)}(Z_j^{(n)}) \tag{7.2}$$

Equation 7.3 illustrates the calculation of of error E. Parameter Y represents an ideal output.

$$E = \frac{1}{2} \sum_x (Y_x - Y_x^{(n)})^2 \tag{7.3}$$

In most of the neural networks sigmoid activation function σ is used. Also, the bias parameter θ can also be neglected in several approaches, which may reduce the accuracy of the network, as well as its complexity. Equations 7.4–7.14 illustrate backpropagation equations, where $\theta = 0$ and $f = \sigma$. Equation 7.4 illustrates the output of the layer.

$$Y_j^{(n)} = \sigma_j^{(n)} \left(\sum_i w_{ji}^{(n)} X_i^{(n)} \right) \tag{7.4}$$

Equation 7.5 illustrates the error in the output layer N.

$$e^{(N)} = Y_j - Y_j^{(N)} \tag{7.5}$$

Equation 7.6 shows the derivative of error which is propagated back. Parameter $Y_j^{(N-1)}$ represents the input to layer N, which is equivalent to the output of layer $N-1$.

$$\frac{\partial E^{(N)}}{\partial w_{ji}^{(N)}} = (Y_j - Y_j^{(N)}) \cdot \sigma_j'^{(N)} \left(\sum_i Y_i^{(N-1)} w_{ij}^{(N)} \right) \tag{7.6}$$

Equation 7.7 illustrate the calculation of a derivative of error.

$$\frac{\partial Y^{(N)}}{\partial w_{ji}^{(N)}} = (Y_j - Y_j^{(N)}) \cdot Y_j \cdot (1 - Y_j^{(N)}) \tag{7.7}$$

Equation 7.8 shows the simplified version of the error propagated back through the network.

$$\frac{\partial E^{(N)}}{\partial w_{ji}^{(N)}} = -\frac{\partial Y^{(N)}}{\partial w_{ji}^{(N)}} \cdot Y_i^{(N-1)} \tag{7.8}$$

Equation 7.9 shows how the weights in the last layer of neural network are updated. Parameter t is the time variable, parameter $w_{ji}^{(N)}(t+1)$ shows the updated weight.

$$w_{ji}^{(N)}(t+1) = w_{ji}^{(N)}(t) - \frac{\partial E^{(N)}}{\partial w_{ji}^{(N)}} \tag{7.9}$$

Equations 7.10 and 7.11 illustrate the calculation of the derivative of error and backpropagation of error.

$$\frac{\partial E^{(n)}}{\partial w_{ji}^{(n)}} = \left(\sum_j \frac{\partial Y^{(n+1)}}{\partial w_{ji}^{(n+1)}} w_{ij}^{(n+1)} \right) \cdot \sigma_j'^{(n)} \left(\sum_i Y_i^{(n-1)} w_{ij}^{(n)} \right) \tag{7.10}$$

$$\frac{\partial Y^{(n)}}{\partial w_{ji}^{(n)}} = \left(\sum_j \frac{\partial Y^{(n+1)}}{\partial w_{ji}^{(n+1)}} w_{ij}^{(n+1)} \right) \cdot Y_j \cdot (1 - Y_j^{(n)}) \tag{7.11}$$

Equations 7.12 and 7.13 illustrate weight update process in all the layers, except the first and the last layer of synapses. Parameter n is the layer number.

$$\frac{\partial E^{(n)}}{\partial w_{ji}^{(n)}} = -\frac{\partial Y^{(n)}}{\partial w_{ji}^{(n)}} \cdot Y_i^{(n-1)} \tag{7.12}$$

Equation 7.14 is used to update the first layer of weights in the neural network.

$$w_{ji}^{(n)}(t+1) = w_{ji}^{(n)}(t) - \frac{\partial E^{(n)}}{\partial w_{ji}^{(n)}} \tag{7.13}$$

$$\frac{\partial E^{(n)}}{\partial w_{ji}^{(n)}} = -\frac{\partial Y^{(n)}}{\partial w_{ji}^{(n)}} \cdot X_i \tag{7.14}$$

7.3 Learning on Hardware in Memristive Systems

The application of memristive crossbars and integration of memristors to CMOS designs is a recent trend that gained high popularity in recent years. Memristive device is a promising solution to reduce on-chip area and power dissipation in learning architectures, which are especially critical in deep learning systems with multiple layers. Figure 7.1 illustrate how neural network can be translated to the memristive crossbar implementation. Memristive devices represent neural network weights. One column of a crossbar represents the weighted summation of the input voltages, where weights are the conductance values of memristive devices. The outputs from the crossbar are usually fed into the next neuron or activation function of the neural network.

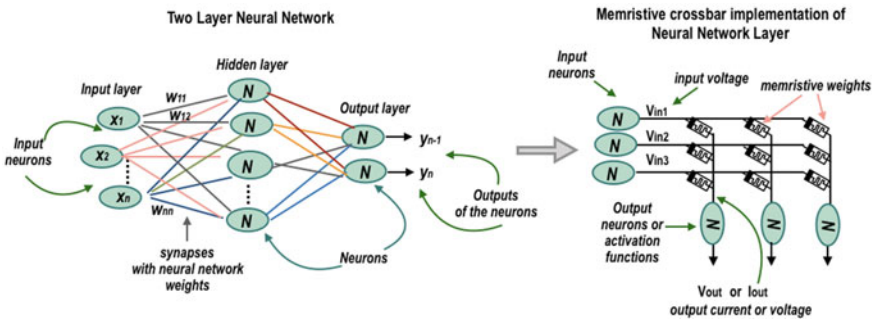

Fig. 7.1 Neural network implementation based on memristive crossbar

7.3.1 Implementation of Learning Algorithms in Memristive Systems

There have been several attempts to implement learning architectures on hardware using memristive crossbar circuits. These include conventional neural networks with backpropagation algorithm, HTM with Hebb's learning and gradient descent rules and spiking neural networks with STDP. The implementation of basic Hebb's learning rules has been performed with a single synapse and on a crossbar for both spike and pulse based approaches. The perceptron learning rules have been implemented for both online and offline pattern classification methods [5, 7–9, 16]. However, most of the implementations of learning methods are performed using the software. There are several implementations of neural networks on FPGA and using digital architectures [16]. In most of the cases, such system focus on the acceleration of the existing architectures. Several digital training circuits for memristive crossbars have been recently proposed in [2, 6, 18, 21]. The architecture is shown in [2] illustrate the training architecture for memristive deep neural networks, which attempts to accelerate the learning and transfer it into hardware. In [6] the neural network with analog neurons and digital on-chip training circuit is illustrated. However, there are only a few approaches illustrating completely analog hardware based implementation of learning algorithms [11, 13, 14, 26], which allows avoiding analog-to-digital conversion. The research works [13, 14] and [26] implement backpropagation algorithm on hardware. While the research work [11] illustrate the analog learning circuit for HTM implementing Hebbian learning, which is shown in Chap. 14. Analog approach for implementation of the learning circuit opens up the opportunity to perform near-sensor processing and integrate the learning circuits directly into the sensors. Analog learning circuits are useful for edge computing applications [12].

7.3.2 Analog Hardware Implementation of Backpropagation Algorithm for Memristive System

The recently proposed implementation of the backpropagation algorithm for two-layer network is illustrated in Fig. 7.2 [26]. The backpropagation algorithm is simplified in this architecture. After the calculation of error, the errors are propagated back, and memristive devices in both layers are updated. Memristive switches allow isolating the memristors in a layer, which is not currently updated. The update value for memristive devices is calculated on FPGA or using Look-Up Table [12].

The other architecture for backpropagation learning is shown in Fig. 7.3 [13, 14]. This architecture implements the backpropagation algorithm with gradient descent without the simplification of the algorithm. The backpropagation architecture consists of several blocks, which can be used for different types of learning systems, including deep neural networks [14]. Figure 7.3 illustrates the overall architecture of

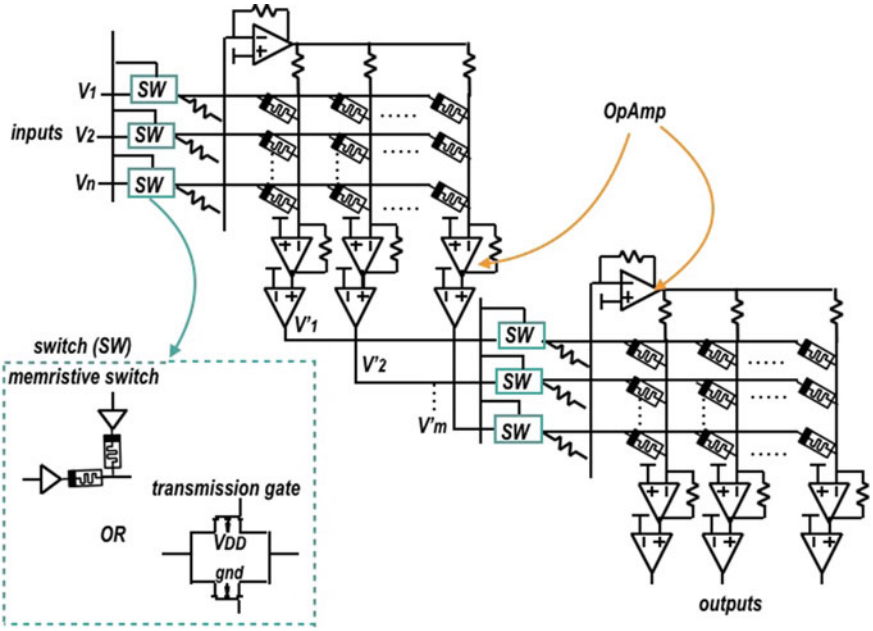

Fig. 7.2 Memristive crossbar architecture with backpropagation learning [26]

the network, while Fig. 7.4 shows each block of the circuit and implementation of the corresponding device used in the design.

In Fig. 7.3 shows a two-layer neural network with backpropagation error calculation circuits, sigmoid activation function, the derivative of sigmoid error, multiplication circuit corresponding to different stages of backpropagation, control transistors and weight update circuits. The input signals to the neural network are represented as V_{in}. Normalization circuit is optional and used to normalize the input signals to the required range, which depends on the threshold voltage memristive devices. Switch transistors are used to control the direction of the current and switch between forward propagation, backpropagation and update processes. The output of the crossbar (dot product) is presented as the current from the crossbar column. Therefore, the parameters of the transistors are adjusted to avoid the distortion of the output current. In this system, the currents from the crossbar are read one at a time to avoid the interference and leakage currents. As negative resistance is memristive device are not possible, the second crossbar and sign control circuit is used when reading the inputs to the crossbar. The sing control circuit changes the sign of the input voltage when the weight saved in the crossbar is negative. The sign of the weight is stored in the sign crossbar, as R_{ON} and R_{OFF} corresponding to a positive and negative sign, respectively.

The backpropagation algorithm is divided into several blocks. The circuit level implementation of these blocks is shown in Fig. 7.4. Block 1 corresponds to the

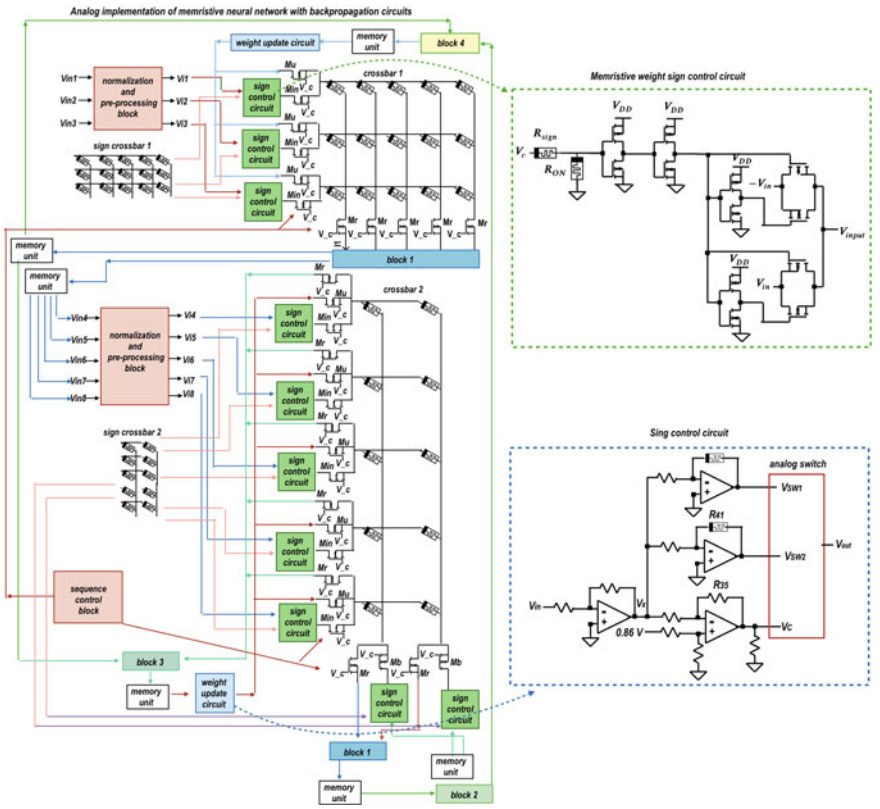

Fig. 7.3 Analog implementation of backpropagation algorithm with gradient descent [13, 14]

forward propagation and activation function. It consists of the current buffer, sigmoid function and derivative of the sigmoid. Current buffer is used to ensure the sigmoid circuit does not affect the output of the crossbar. The derivative of the sigmoid is not used in forward propagation, however, is used in backpropagation stage. The circuit level implementation of derivative of the sigmoid involved the difference amplifier to subtract the sigmoid output from 1, and multiplication circuits to multiply the results with the sigmoid output. The outputs of the sigmoid and derivative of the sigmoid are stored in memory units, which can also be based on memristive devices.

After the forward propagation, the crossbar switches switch to backpropagation mode and revert the direction of the current to propagate it back through the crossbar. Block 2 performs the propagation through the output layer. It involves the calculation of the error, subtracting the sigmoid output from the ideal output. Then, the error is multiplied by the derivative of the output layer. This multiplication circuit is sensitive to the changes in the sign of the signal. Therefore, there are two multiplication circuits to ensure that the positive and negative multiplications are performed correctly, as the positive range and negative range of the multiplier has a different amplitude. The

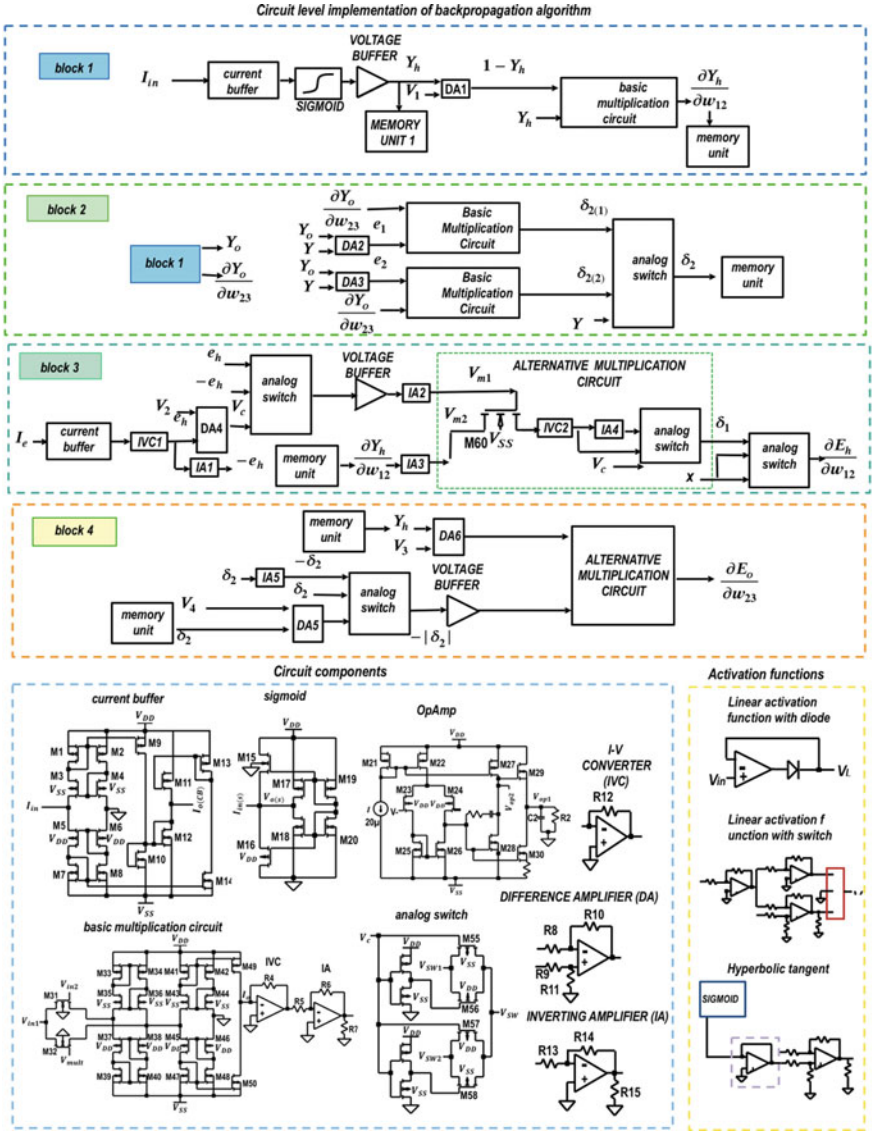

Fig. 7.4 Circuit level implementation of backpropagation algorithm [13, 14]

analog switch selects whether the multiplication is positive or negative, and selects the correct output. The output of block 2 is used later in the update of memristors in the output layer. Block 3 performs the backpropagation thought all other layers, except the output layer. The calculated error is propagated back through the crossbar in the direction opposite to forward propagation, and the output current from the crossbar is read at the same point where the inputs were supplied. Then, this current is converted into a voltage by the current-to-voltage converter. Then, the multiplication of this error to the derivative of the hidden layer is performed. As the multiplier is sensitive, to perform the accurate multiplication, the compensating circuits are added, and the error signal is shifted, as the multiplier is not accurate for small signals in mV. The positive and negative multiplications are performed similarly to block 2. The output of the block 3 is used to update the memristors in the hidden layer. Block 4 is used to perform the multiplication of the output of block 2 to the output of a hidden layer (input to the output layer) to perform the update process of the output layer. The weight update circuit is illustrated in Fig. 7.4, which calculates the amplitude of update pulses from blocks 2 and 4. The sequence control circuit controls the duration of the pulses.

7.4 Conclusion

This chapter briefly covered various learning rules and algorithms and provided the mathematical framework for backpropagation algorithm, which is one of the most frequently used in the deep learning systems. As memristive neural networks is a promising solution for deep learning systems on hardware, due to their small size, low power consumption and ability to change the resistive state, this chapter focuses on the hardware implementations of learning algorithms for memristive systems. Analog hardware implementation of this backpropagation algorithm has been shown for the memristive neural network, which can also be applied for deep memristive neural network. The primary challenge of the implementation of full analog learning systems on hardware is a complexity of the learning algorithms. The full analog on-chip memristive system for training and learning in deep neuromorphic architectures is still an open problem.

Chapter Highlights

- Hebbian learning rule and backpropagation algorithm are the most common learning algorithms in the implementation of various neuromorphic architectures.

- There are a lot of different hardware implementations of learning algorithms on FPGA and using digital and mixed-signal design.
- Even though several implementations of fully analog learning architectures have been proposed in recent years, it is still an open problem.

References

1. Azghadi MR., Al-Sarawi S, Iannella N, Abbott D (2012) Design and implementation of BCM rule based on spike-timing dependent plasticity. In: The 2012 international joint conference on neural networks (IJCNN), IEEE, pp 1–7
2. Cheng M, Xia L, Zhu Z, Cai Y, Xie Y, Wang Y, Yang H (2017) Time: a training-in-memory architecture for memristor-based deep neural networks. In: 2017 54th ACM/EDAC/IEEE design automation conference (DAC), pp 1–6, https://doi.org/10.1145/3061639.3062326
3. Choe Y (2015) Anti-hebbian learning. In: Encyclopedia of computational neuroscience. Springer, pp 191–193
4. Gerstner W, Kistler WM (2002) Mathematical formulations of hebbian learning. Biol Cybern 87(5–6):404–415
5. Hasan R, Taha TM (2014) Enabling back propagation training of memristor crossbar neuromorphic processors. In: 2014 International joint conference on neural networks (IJCNN), IEEE, pp 21–28
6. Hasan R, Taha TM, Yakopcic C (2017) On-chip training of memristor crossbar based multilayer neural networks. Microelectron J 66:31–40
7. Hassan AM, Yang C, Liu C, Li HH, Chen Y (2017) Hybrid spiking-based multi-layered self-learning neuromorphic system based on memristor crossbar arrays. In: 2017 Design, automation & test in Europe conference & exhibition (DATE), IEEE, pp 776–781
8. Hu X, Feng G, Duan S, Liu L (2017) A memristive multilayer cellular neural network with applications to image processing. IEEE Trans Neural Netw Learn Syst
9. Kataeva I, Merrikh-Bayat F, Zamanidoost E, Strukov D (2015) Efficient training algorithms for neural networks based on memristive crossbar circuits. In: 2015 International joint conference on neural networks (IJCNN), IEEE, pp 1–8
10. Krestinskaya O, Dolzhikova I, James AP (2018) Hierarchical temporal memory using memristor networks: a survey. IEEE Trans Emerg Top Comput Intell 2(5):380–395. https://doi.org/10.1109/TETCI.2018.2838124
11. Krestinskaya O, Ibrayev T, James AP (2018) Hierarchical temporal memory features with memristor logic circuits for pattern recognition. IEEE Trans Comput-Aided Des Integr Circuits Syst 37(6):1143–1156
12. Krestinskaya O, James AP, Chua LO (2018) Neuro-memristive circuits for edge computing: a review. arXiv preprint arXiv:1807.00962
13. Krestinskaya O, Salama KN, James AP (2018) Analog backpropagation learning circuits for memristive crossbar neural networks. In: 2018 IEEE international symposium on circuits and systems (ISCAS), IEEE, pp 1–5
14. Krestinskaya O, Salama KN, James AP (2018) Learning in memristive neural network architectures using analog backpropagation circuits. IEEE Trans Circuits Syst I: Regul Pap
15. Leonard J, Kramer M (1990) Improvement of the backpropagation algorithm for training neural networks. Comput Chem Eng 14(3):337–341
16. Li B, Wang Y, Wang Y, Chen Y, Yang H (2014) Training itself: mixed-signal training acceleration for memristor-based neural network. In: 2014 19th Asia and South Pacific design automation conference (ASP-DAC), IEEE, pp 361–366

17. Nair MV, Dudek P (2015) Gradient-descent-based learning in memristive crossbar arrays. In: 2015 International joint conference on neural networks (IJCNN), IEEE, pp 1–7
18. Rosenthal E, Greshnikov S, Soudry D, Kvatinsky S (2016) A fully analog memristor-based neural network with online gradient training. In: 2016 IEEE international symposium on circuits and systems (ISCAS), pp 1394–1397, https://doi.org/10.1109/ISCAS.2016.7527510
19. Sheridan PM, Du C, Lu WD (2016) Feature extraction using memristor networks. IEEE Trans Neural Netw Learn Syst 27(11):2327–2336
20. Smagulova K, Krestinskaya O, James AP (2018) A memristor-based long short term memory circuit. Analog Integr Circuits Signal Process 95(3):467–472
21. Soudry D, Castro DD, Gal A, Kolodny A, Kvatinsky S (2015) Memristor-based multilayer neural networks with online gradient descent training. IEEE Trans Neural Netw Learn Syst 26(10):2408–2421. https://doi.org/10.1109/TNNLS.2014.2383395
22. Soudry D, Di Castro D, Gal A, Kolodny A, Kvatinsky S (2013) Hebbian learning rules with memristors. Israel Institute of Technology, Haifa, Israel
23. Srinivasan N, Ravichandran V, Chan K, Vidhya J, Ramakirishnan S, Krishnan S (2002) Exponentiated backpropagation algorithm for multilayer feedforward neural networks. In: Proceedings of the 9th international conference on neural information processing, 2002. ICONIP'02, vol 1. IEEE, pp 327–331
24. Wu X, Saxena V, Zhu K (2015) A CMOS spiking neuron for dense memristor-synapse connectivity for brain-inspired computing. In: 2015 International joint conference on neural networks (IJCNN), IEEE, pp 1–6
25. Zamanidoost E, Bayat FM, Strukov D, Kataeva I (2015) Manhattan rule training for memristive crossbar circuit pattern classifiers. In: 2015 IEEE 9th international symposium on intelligent signal processing (WISP), IEEE, pp 1–6
26. Zhang Y, Wang X, Friedman EG (2018) Memristor-based circuit design for multilayer neural networks. IEEE Trans Circuits Syst I: Regul Pap 65(2):677–686

Chapter 8
Multi-level Memristive Memory for Neural Networks

Aidana Irmanova, Serikbolsyn Myrzakhmet and Alex Pappachen James

Abstract Analog memory is of great importance in neuromorphic engineering as it enables scalable neural network design and energy efficient implementation of computationally expensive operations. With the advent of memristors, the realization of the analog memory became possible due to the intrinsic properties of memristors such as nanoscale size, non-volatility, and energy efficiency. In hardware implementations of neural networks, memristors store the values of synaptic weights and operate similarly to the synapses that are reinforced with the application of external stimuli. Memristors that are ideally continuum memories, currently are at the early stage of the development, which causes several issues in neuromorphic circuit design. Device level and architecture level issues force memory engineers to approach memristive memory design in different ways. In this chapter device-level problems: restricted number of resistance states, stochastic switching and architecture level problem: sneak paths will be discussed, and their state of the art solutions will be presented.

8.1 Introduction

Most of the recently proposed neural network (NN) designs are based on memristive crossbar architecture [16] as memristive crossbars enable in-memory computing, simultaneously providing both memory resources and vector-matrix multiplication (VMM) computation. VMM is the core operation of NN that perform multiplication of input values to the matrix of synaptic weights. This operation is repeatedly used during inference and training stages and is evaluated to be the most computationally expensive operation of the system [9].

A. Irmanova · S. Myrzakhmet · A. P. James (✉)
Nazabayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

A. Irmanova
e-mail: aidana.irmanova@nu.edu.kz

S. Myrzakhmet
e-mail: serikbolsyn.myrzakhmet@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_8

Currently, several issues hinder the scaling of analog memristive cells in crossbar architecture. First is related to the memristors itself: the pure analog memristors are not fabricated yet, while *binary memristors* exhibit *stochastic switching* behavior [23]. There are also works that report multi-level memristor fabrication that stores discrete analog states [17]. The *repeatability* of resistive switching in such memristors remains the main issue, that results in inaccurate performance of the memory [31].

The second problem, on the other hand, is the architecture level issue. Crossbars that were initially designed to use the switches at each junction and operate within binary logic have to be modified to avoid the *sneak path* leakage current [35] if memristors will be used. Sneak path current affects both programming and the reading process of the system.

In this chapter abovementioned issues of memristive analog memory design for NN implementations are presented. In the background section, history and the operation of crossbars and memristors and memristor-based memory related vital concepts and terminology are given. Further, controlling the resistive switching in memristors: writing, erasing operations are explained in detail. Following on that, designing multi-level memory with binary and multi-level memristors is discussed. In conclusion, the sneak paths solutions for scaling the memristive crossbar architecture are explained.

8.2 Background

Crossbars can be represented as a set of parallel wires with the width of less than 50 nm crossed by another set of wires [8]. At each intersection of wires, there is a device that can change its conductance after application of some voltage signal. There are several advantages crossbars can provide. First is its repetitive structure of intersecting wires, that makes the fabrication process much easier compared to conventional processors based on CMOS technology [19, 24]. Second, there are many options on the material implementation of crossbars, which makes the design parameters flexible for different types of architectures [12]. Also, last but not least, crossbar architecture can be used simultaneously as memory, logic and computation blocks [21]. Before 2008, different CMOS materials were used at the intersection of crossbar wires. Such crossbar circuits were used for digital logic gates implementations and digital memory architectures [27]. After 2008 when the first memristor was fabricated, memristors became common in crossbar architectures [14].

The existence of the memristor was theoretically predicted by an American scientist L. Chua in 1971 [4]. He put forward and mathematically substantiated the hypothesis that there is a fourth basic element of electrical circuits—along with inductor, capacitor, and resistor. The memristor was defined as a passive element of electrical circuits which resistance depends on the charge passing through it. After the shutdown of voltage in the circuit, a memristor "remembers" the last resistance value it had (memristor—short for memory resistor) [4]. Since then memristors remained hypothetical element for more than thirty, but in 2008 a group of researchers directed

by Stanley Williams from Hewlett-Packard, same that patented crossbar technology, have fabricated first physical memristor [29]. Its properties were consistent with the model, proposed by Chua. The memristor exhibited two distinct resistance states: low as R_{on} and high as R_{off}.

Ideally, memristors have all possible analog states between R_{on} and R_{off}. However, current implementations of memristors suffer from severe variability of switching to intermediate states between R_{on} and R_{off} [13]. Most of the research papers assume idealistic behavior of memristors and use resistors for their simulations. The resistors would represent memristors that were preprogrammed to the specific analog states.

Key Concepts

A *memristor*, is a two-terminal electrical component, the value of which is the linear relationship between the charge and flux [4]. Memristor has multilevel resistance property that can be controlled by the voltage or current applied across the device as a function of time and can be programmed to any resistance level between a maximum possible resistance R_{off} and minimum resistance R_{on} [5].

Resistive switching refers to the physical phenomenon whereby the device changes its resistance as the result of applied stimuli. The change of resistance is non-volatile and reversible.

Discrete analog state of the memristor is non-binary logical states stored in the memristor. One logical state can be represented with the range of resistance values that will not match with the resistance values of following logic state.

Binary memristor has only two distinguish states that corresponds to two logic states: $R_{on} \in [R_1, R_2, ..R_n] < R_{th}$, and $R_{off} \in [R_1, R_2, ..R_n] > R_{th}$, where R_{th} is defined as mean of R_{on}^{max} and R_{off}^{min}. Example of binary states of binary memristor can be found in Fig. 8.1a.

Multi-level memristor have multiple distinguish states (discrete analog states) so that, all resistance values $[R_1, R_2...R_n,]$ within the range R_{on} and R_{off} : $R_{all} \in [R_{on}, R_{off}]$. Example of resistance states of 6 level memristor can be found in Fig. 8.1b.

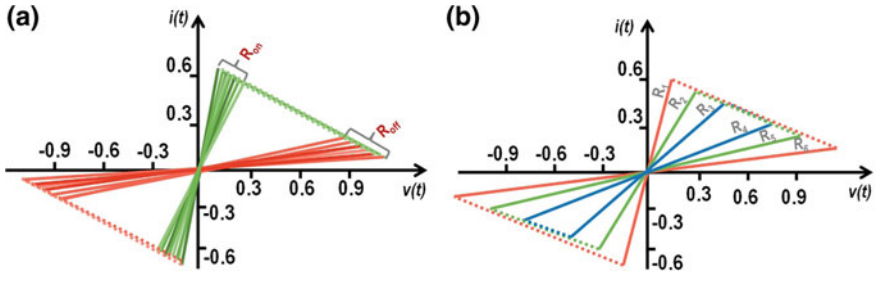

Fig. 8.1 Resistive states of Binary memristor (**a**) Resistive states of Multilevel memristors (**b**)

Stochastic switching refers to the probabilistic resistive switching or/and fluctuation of resistive switching in memristors after application of the particular amount of stimuli [28]. In Fig. 8.1a possible variations of R_{on}, R_{off} states of binary memristor are presented.

Threshold voltage is the minimum magnitude and duration of the stimuli that causes the resistive switching in memristors.

Reading voltage is the voltage that is less than the threshold voltage. It is used for reading operation.

There are three operations performed with memristors (Figs. 8.4 and 8.5):

1. *Reading* operation is used for calculating the current resistance state of the memristor.
2. *Writing* is a programming operation that decreases the resistance level of the memristor to R_{on}.
3. *Erasing* is a programming operation that increases the resistance level of the memristor to R_{off}.

The endurance or life span of the device is measured in the number of successful erasing and writing operations that caused resistive switching of the device before collapsing [34]. The endurance of the devices is dependent on the voltages and currents used. Higher voltage and current will reduce lifetimes and eventually destroy the devices. Results vary depending on the protocol (amplitude and duration of the stimuli) of operations.

Repeatability of resistive switching is the capability of memristors to switch into the range of R_{on} and the range of R_{off} values that preserve their ratio stable throughout the device lifetime.

Switching type of the memristor is classified as *unipolar* and *bipolar* based on the polarity of voltage stimuli required for increasing (erasing) or decreasing (writing) the resistance states [33].

For increasing and decreasing the resistance states of the *unipolar* memristor, applied voltages have to be of the same polarity but different magnitudes; while for the *bipolar* memristor, the voltages have to be in opposite polarities, as depicted in Fig. 8.3.

8.3 Controlling Resistive Switching

One of the main reasons why memristors are good to build memory cells is their non-volatility. There are mathematical characteristics such as *power of plot* (POP) and *dynamic route map* (DRM) that describe this property. Idealistic representation of POP and DRM are presented in Fig. 8.2. Basically, POP is the rate of change of the current state of the memristor when $V = 0$, which is just a curve in the f(x,0) [20]. DRM, on the other hand, describes the response of the memristors to any input at an initial condition of the device [2]. POP and DRM also serve as the basis for

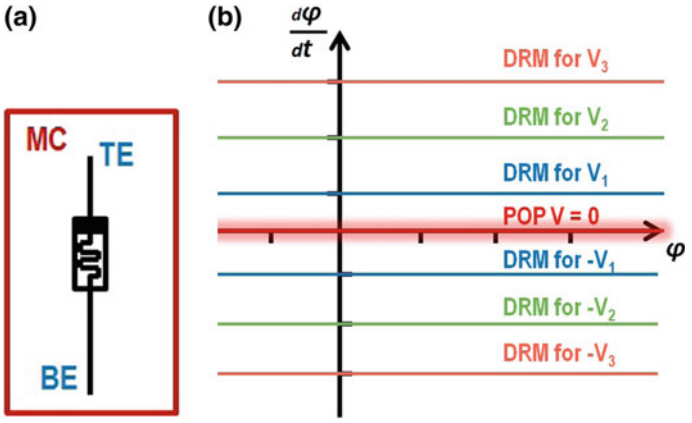

Fig. 8.2 Symbol of the memristor: MC—memristive cell, TE -top electrode, BE bottom electrode (**a**), POP and DRM characteristic of the memristors (**b**)

constructing the protocol of writing, erasing and reading operations, which are the control operations of resistive switching.

The stimuli applied to the memristors control the resistive switching process. The impact of the stimuli in case of *voltage controlled memristors* [20] depends on the voltage amplitude V_w, duration of the stimuli t_s, and its initial state R_i. The amplitude and polarity of the voltage stimuli V_w, and the duration of t_s vary for the different material type of memristors. Extended memristor theory correlates the amplitude and duration of the stimuli showing that the higher the amplitude, the shorter the time for voltage stimuli is required and vice versa [20]. Material implementation and fabrication process also define the switching type of memristors. The type of resistive switching, whether memristor is bipolar or unipolar (Fig. 8.3) is crucial in the overall memory design process as the polarity and magnitude of control signals define the type and size of the CMOS technology will be used for memristive cell design.

8.3.1 Programming and Reading Memristors

In this section writing, erasing, and reading operations are described for voltage controlled *binary* and *multi-level memristors* of *bipolar* type of switching.

All the operations are described for the configuration of the memristor, where the stimuli are applied to the top electrode (TE) of the memristor when the bottom electrode (BE) is grounded.

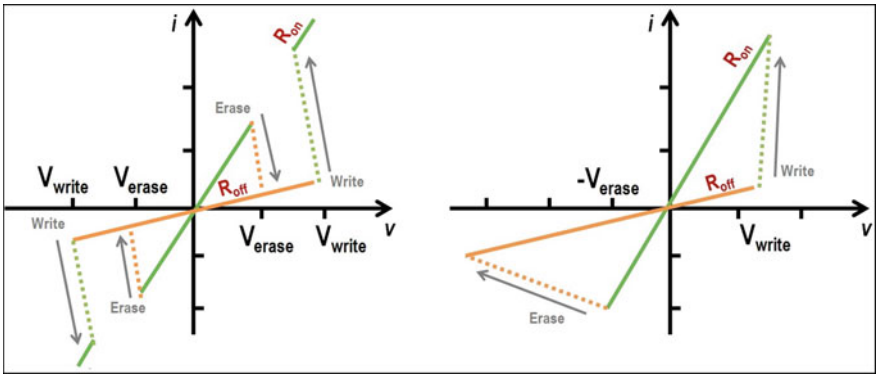

Fig. 8.3 Unipolar switching (left) and bipolar switching (right) [10]

8.3.1.1 Writing—Decreasing Resistance

For n-level bipolar memristor that can store set of resistive states $[R_1, R_2, ..R_n]$,

1. There should be respective set of stimuli V_w of different magnitudes $\forall[V_1, V_2, .. V_n] > V_{th}$ with the same duration $t > t_s$ to switch the memristor. V_{th} is the minimum positive switching voltage of the memristor. The example of three write operations is given in Fig. 8.5a.
2. There should be the respective set of stimuli of different duration $\forall[t_1, t_2, ..t_n] > t_s$ of the same amplitude $V_w > V_{th}$ to switch the memristor. t_s is the minimum switching time of the memristor. The example of three write operations is given in Fig. 8.5b.

In case of binary memristor there should be some V_w so that $V_w > V_{th}, t > t_s$ that will switch R_{ini} into the range of $[R_1, R_2, ..R_n] < R_{th}$. The example of three write operations is given in Fig. 8.4a, b.

8.3.1.2 Erasing—Increasing Resistance

For both binary and multi-level memristor there should be large enough $|V_w|$ so that, $V_w << -V_{th}, t > t_s$ that could switch R_{ini} into the range of $R_{off} \in [R_1, R_2, ..R_n] > R_{th}$. The example of erase operations is given in Fig. 8.4 and 8.5.

8.3.1.3 Reading

For both bipolar and unipolar memristors, there should be the set of V_r of low enough reading voltages : $| V_r | << | V_{th} |$, that will not impact the current state of the memristor. Reading voltage values are used at the inference stage of neural network operation.

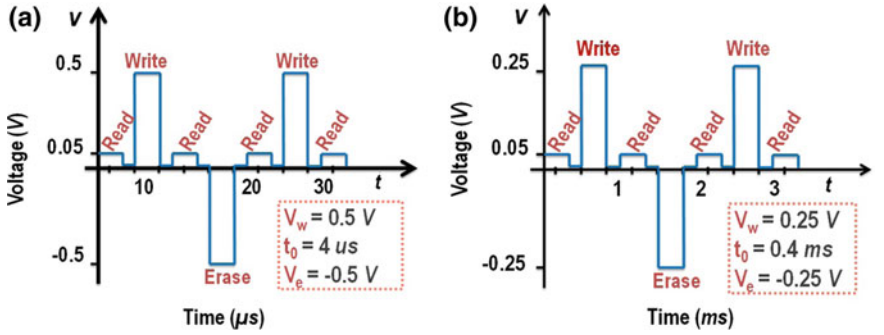

Fig. 8.4 Programming binary memristors with **a** with higher amplitude but shorter duration, and **b** with longer duration lower amplitude

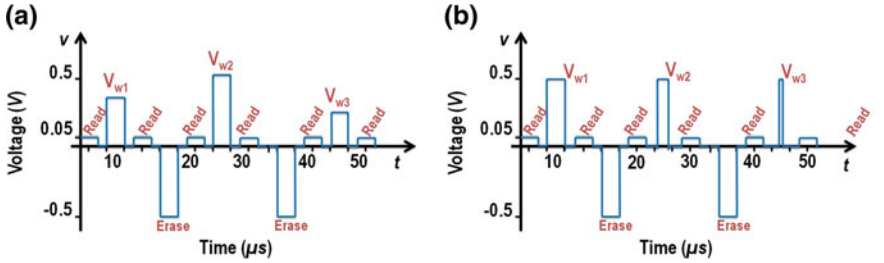

Fig. 8.5 Programming multi-level memristors with **a** changing only amplitude preserving the same duration, and **b** changing the pulse duration preserving the same amplitude

8.4 Designing Multi-level Memory

Programming memristor to analog levels might be challenging due to the immaturity of the technology [30]. Therefore, one of the possible ways of designing a reliable memory cell is to use a combination of discrete state memristors in a cell. Memristors could be combined in series or parallel, and then programmed separately to discrete states, thereby increasing the sufficient number of logic states stored in the cell configuration.

8.4.1 Parallel Combination of Memristors

A memory cell consisting of a combination of parallel memristors enables obtaining multiple resistive levels. For example, a memory cell presented in [32] is based on binary memristors, as using intermediate states of a memristor will most probably require high-precision power sources. Hence, a memory cell is shown in Fig. 8.6a consisting of one transistor and two memristors is proposed to obtain four different

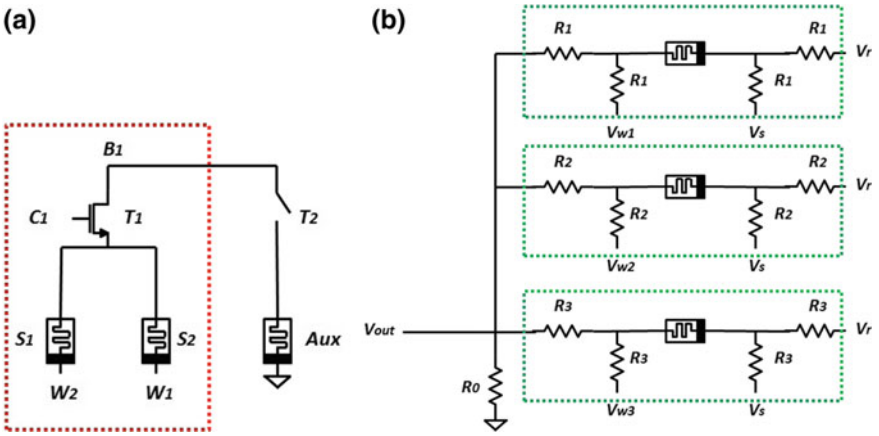

Fig. 8.6 Combination of binary memristors as MLM in [32] (**a**); Combination of ternary memristors as MLM in [11] (**b**)

resistive levels in which only binary states (R_{on} and R_{off}) of each memristor is utilized. As the result, the circuit generates distinct voltage levels respective to the resistive logic states stored in the cell. In general, during designing a memory circuit using memristors, it is necessary to apply effective methods of writing and reading which indeed can simplify a circuitry, i.e., minimizing the number of required voltage sources, avoiding comparator and other reference circuits, simultaneous reading from several memory cells, etc. The memory cell proposed in [32] incorporates those considerations as well, so, there is no need for separate writing and reading circuits. During the reading process, it is required to connect an external auxiliary memristor to the cell. This memristor functions as a potential divider and provides the output voltage.

Similar to [32], a memory cell which presents a simple way of writing and reading is proposed in [11]. The memory cell consisting of three memristors depicted in Fig. 8.6b is designed to store 10 or 27 discrete resistive levels using ternary memristors. The values of resistive networks (four resistors around memristor) at each memristor branch, introduce the order, therefore is decisive whether the cell will store 10 or 27 levels. The design was also proposed with the possibility to remodel this cell to store 9 levels deploying binary memristor or up to 64 levels with quaternary memristors. The number of logic states stored in the cell also depends on the number of memristors in the configuration. Thus, the tradeoff between the cell area (data density as well) and the number of logic states has to be considered during the design of memristive crossbar circuits with such memristive cells.

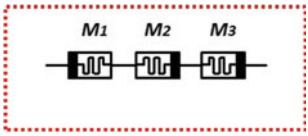

Fig. 8.7 A memory cell proposed in [1]

8.4.2 Series Combination of Memristors

A multilevel memory cell can be designed with the serial combination of several memristors. One of such implementations is presented in [1]. The memory cell designed with serially connected three identical memristors (Fig. 8.7) to provide ternary logic. To program the cell to the first resistive level, two memristors is switched to the R_{on} state, while for the second state all memristors are at the R_{on} state, and to obtain the third resistive level only one memristor is kept at the R_{on} state.

8.5 Crossbar Architecture

During the inference stage of neural networks, reading voltages represent the input vector and conductance values stored in analog memories represent synaptic weights. Feedforward operation is performed via the dot product operation of synaptic weights and input vector followed by activation function at each layer.

The sneak path leakage is one of the critical concerns encountering memristor-based circuits and architectures. The sneak path takes place in large crossbar arrays when the current flows in the wrong direction resulting in error-prone read and write operations. Along with complicating reading and writing processes, it increases the power consumption of the circuit. Therefore, crossbar array sizes are usually limited [6]. Possible sneak path solutions include multistage reading, use of unfolded architecture, diode gating, transistor gating (Fig. 8.8), sensing via AC signals, use of complementary memristors for reducing sneak path currents, etc. [35]. Designing a reliable multilevel memory requires placing the diode or transistor at each memory cell in crossbars [25] as single memristor is insufficient. Hence, several approaches suitable for developing a multilevel memory will be discussed.

8.5.1 Transistor Gating

It is common to use small-sized transistor along with the memristor or combination of memristors within each memory cell for designing a multilevel memory. Such transistor gating allows accessing desired memory cell for reading or writing operations and avoiding the sneak paths. For instance, [26] deploys TSMC 130 nm CMOS

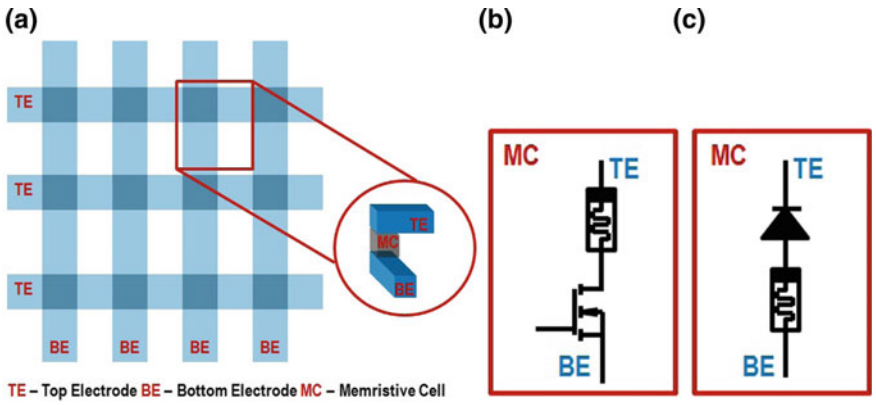

Fig. 8.8 Solving sneak path problem of the crossbar (**a**) with Transistor Gating (**b**) and Diode gating (**c**)

transistors for different array architectures such as 1T1M, 1T2M and 2T2M, while [18] proposes minimum-sized single FET for 1T2M configuration. 1T1M memory cell was proposed in [15] as well as forming NOR type array, which demonstrates reliable resistance switching, good endurance and retention properties. However, using small-sized transistors will not fully eliminate but only reduce the sneak path current [35]. On the other hand, large transistors that demonstrate stronger sneak path is blocking are larger and increases the die area.

8.5.2 Diode Gating

Another way of preventing sneak paths is adding the diode into each memory cell. According to [3], implementation of the 1D1M structure is more straightforward compared to 1T1M design, and it might need less area. A memory cell consisting of one inorganic diode and one organic unipolar memory element is proposed in [3]. A comparative analysis of 1D1M structure with 1M based on endurance cycles and retention time indicates that both of them possess similar characteristics, but, in 1D1M structure, the cross-talk reading interference can be prevented for crossbar-based memories. Hence, integrating a diode with a memristor will not adversely influence on the performance, however crossbar-based memory containing diodes possesses capacitive load introducing undesired delay [7], but it has not been thoroughly investigated yet.

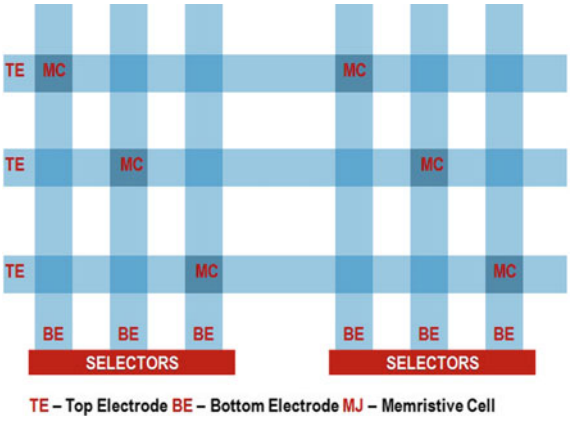

Fig. 8.9 Unfolded architecture proposed in [22]

8.5.3 Unfolded Architecture

Solving the sneak path in crossbars without gating is possible with the unfolded architecture [22]. In this configuration, each memristor is placed in a separate column as it is shown in Fig. 8.9 However, the data density is decreased significantly, as the result, a memory circuit will require very large area. Likewise, an array consisting of a single row merely can eliminate the sneak path current [35], and it will need less area compared to the unfolded architecture proposed in [22].

8.6 Summary

In this chapter key concepts related to the memristive memory design and approaches utilized for increasing the number of logic states stored in the memristive cells are discussed. Basic operations such as writing erasing and reading are explained and mathematical characteristics of memristors such as POP and DRM are introduced. At the current stage, when the design of memristive multi-level memory is bounded with memristor development, proposed memory cells introduce area and energy tradeoffs, nevertheless enabling in-memory computing architectures for neural networks. As memristor fabrication process will mature, most of the proposed designs may become redundant requiring only sneak path prevention for VLSI implementation of crossbars.

Chapter Highlights

- An analog memory cell in crossbar circuits store synaptic weights of NN.
- In the ideal case, an analog memory cell based on single memristor in crossbar circuits store infinite synaptic weight values.
- In practice memristors store discrete resistive states. A memory cell consisting of a combination of parallel or series memristors can be used to increase stored resistive levels.
- During the inference stage of neural networks, reading voltages are used to avoid resistive switching.
- During the training stage of neural networks, programming voltages are used to force the resistive switching.
- The sneak path takes place in memristive crossbar arrays complicating inference and training processes of the system. Possible sneak path solutions suitable for developing a multilevel memory include designing unfolded architecture, diode gating, and transistor gating.

References

1. Amatllé i Llucià E (2017) Design of a multi-level memory cell with new emerging non-volatile memristive technology. B.S. thesis, Universitat Politècnica de Catalunya
2. Ascoli A, Tetzlaff R, Menzal S (2018) Exploring the dynamics of real-world memristors on the basis of circuit theoretic model predictions. IEEE Circuits Syst Mag 18(2):48–76
3. Cho B, Kim TW, Song S, Ji Y, Jo M, Hwang H, Jung GY, Lee T (2010) Rewritable switching of one diode-one resistor nonvolatile organic memory devices. Adv Mater 22(11):1228–1232
4. Chua L (1971) Memristor-the missing circuit element. IEEE Trans. Circuit Theory 18(5):507–519
5. Chua L (2011) Resistance switching memories are memristors. Appl. Phys. A 102(4):765–783
6. Deng Y, Huang P, Chen B, Yang X, Gao B, Wang J, Zeng L, Du G, Kang J, Liu X (2013) Rram crossbar array with cell selection device: A device and circuit interaction study. IEEE Trans. Electron Devices 60(2):719–726
7. Fei W, Yu H, Zhang W, Yeo KS (2012) Design exploration of hybrid cmos and memristor circuit by new modified nodal analysis. IEEE Trans Very Large Scale Integr (VLSI) Syst 20(6), 1012–1025
8. Hasan R, Taha TM, Yakopcic C (2017) On-chip training of memristor crossbar based multi-layer neural networks. Microelectron J 66:31–40
9. Hu M, Strachan JP, Li Z, Grafals EM, Davila N, Graves C, Lam S, Ge N, Yang JJ, Williams RS (2016) Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication. In: Proceedings of the 53rd annual design automation conference. ACM (2016), p. 19
10. Hu S, Wu S, Jia W, Yu Q, Deng L, Fu YQ, Liu Y, Chen TP (2014) Review of nanostructured resistive switching memristor and its applications. Nanosci Nanotechnol Lett 6(9):729–757
11. Irmanova A, James AP (2017) Multi-level memristive memory with resistive networks. In: 2017 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia). IEEE, pp. 69–72
12. Jo SH, Kim KH, Lu W (2009) High-density crossbar arrays based on a si memristive system. Nano Lett 9(2):870–874

13. Kim H, Sah MP, Yang C, Chua LO (2010) Memristor-based multilevel memory. In: 2010 12th international workshop on Cellular nanoscale networks and their applications (CNNA). IEEE, pp 1–6
14. Kim KH, Gaba S, Wheeler D, Cruz-Albrecht JM, Hussain T, Srinivasa N, Lu W (2011) A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. Nano Lett 12(1):389–395
15. Kim S, Jeong HY, Kim SK, Choi SY, Lee KJ (2011) Flexible memristive memory array on plastic substrates. Nano Lett 11(12):5438–5442
16. Krestinskaya O, James AP, Chua LO (2018) Neuro-memristive circuits for edge computing: A review. arXiv:1807.00962
17. Kuzum D, Jeyasingh RG, Lee B, Wong HSP (2011) Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. Nano Lett 12(5):2179–2186
18. Lastras-Montaño MA, Cheng KT (2018) Resistive random-access memory based on ratioed memristors. Nat Electron 1(8):466
19. Lastras-Montano MA, Ghofrani A, Cheng KT (2015) Architecting energy efficient crossbar-based memristive random-access memories. In: 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH). IEEE, pp 1–6
20. Leon C (2015) Everything you wish to know about memristors but are afraid to ask. Radioengineering 24(2):319
21. Li Y, Zhou YX, Xu L, Lu K, Wang ZR, Duan N, Jiang L, Cheng L, Chang TC, Chang KC et al (2016) Realization of functional complete stateful boolean logic in memristive crossbar. ACS Appl Mater Interfaces 8(50):34559–34567
22. Manem H, Rose GS, He X, Wang W (2010) Design considerations for variation tolerant multilevel cmos/nano memristor memory. In: Proceedings of the 20th symposium on Great lakes symposium on VLSI. ACM, pp 287–292
23. Naous R, Al-Shedivat M, Salama KN (2016) Stochasticity modeling in memristors. IEEE Trans Nanotechnol 15(1):15–28
24. Prezioso M, Merrikh-Bayat F, Hoskins B, Adam G, Likharev KK, Strukov DB (2015) Training and operation of an integrated neuromorphic network based on metal-oxide memristors. Nature 521(7550):61
25. Rose, G.S.: Overview: Memristive devices, circuits and systems. In: Proceedings of 2010 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 1955–1958 (2010)
26. Shaarawy N, Emara A, El-Naggar AM, Elbtity ME, Ghoneima M, Radwan AG (2018) Design and analysis of 2t2m hybrid cmos-memristor based rram. Microelectron J 73:75–85
27. Snider G, Kuekes P, Williams RS (2004) Cmos-like logic in defective, nanoscale crossbars. Nanotechnology 15(8):881
28. Soni R, Meuffels P, Staikov G, Weng R, Kügeler C, Petraru, A., Hambe, M., Waser, R., Kohlstedt, H.: On the stochastic nature of resistive switching in cu doped ge0. 3se0. 7 based memory devices. Journal of applied physics **110**(5), 054509 (2011)
29. Strukov DB, Snider GS, Stewart DR, Williams RS (2008) The missing memristor found. Nature 453(7191):80
30. Truong SN, Ham SJ, Min KS (2014) Neuromorphic crossbar circuit with nanoscale filamentary-switching binary memristors for speech recognition. Nanoscale Res Lett 9(1):629
31. Wang C, He W, Tong Y, Zhang Y, Huang K, Song L, Zhong S, Ganeshkumar R, Zhao R (2017) Memristive devices with highly repeatable analog states boosted by graphene quantum dots. Small 13(20):1603435
32. Wang X, Li S, Liu H, Zeng Z (2018) A compact scheme of reading and writing for memristor-based multivalued memory. IEEE Trans Comput-Aided Des Integr Circuits Syst 37(7):1505–1509
33. Yanagida T, Nagashima K, Oka K, Kanai M, Klamchuen A, Park BH, Kawai T (2013) Scaling effect on unipolar and bipolar resistive switching of metal oxides. Sci Rep 3:1657

34. Yang JJ, Zhang MX, Strachan JP, Miao F, Pickett MD, Kelley RD, Medeiros-Ribeiro G, Williams RS (2010) High switching endurance in tao x memristive devices. Appl Phys Lett 97(23):232102
35. Zidan MA, Fahmy HAH, Hussain MM, Salama KN (2013) Memristor-based memory: the sneak paths problem and solutions. Microelectron J 44(2):176–183

Chapter 9
Memristive Threshold Logic Networks

Irina Dolzhikova, Akshay Kumar Maan and Alex Pappachen James

Abstract Threshold logic gates (TLGs) are known for high-speed and low power consumption, which is essential for applications such as real-time processing and recognition of natural signals, as well as on-chip memory architecture and neural network implementation. Integration of memristors into the design allows extending the capabilities of threshold logic circuits. In this chapter, we review the hardware designs of memristive threshold logic (MTL) circuits that are inspired by the principle of neuron firing inside the brain. Variety of threshold architectures, their limitations and possible field of application are discussed.

9.1 Introduction

Threshold logic block is the simplest type of the computation unit that is used in artificial neural networks (ANNs) to model the activation of the neuron mathematically. The threshold processing unit performs decision making at different functional and structural levels in the hierarchy of biological brain. There is a vast variety of decisions that need to be made depending on the type of the problem, as a result, there is a variety of thresholding techniques that could be found.

In signal processing application, thresholding is widely used to remove the noise effects and recover the signal to noise ratio. By setting the threshold, unimportant features in the image could be discarded, and only useful information will be kept. The thresholding techniques are highly relevant in pattern recognition. The selection of the proper set of features is the key to ensure a good recognition accuracy. The

I. Dolzhikova · A. Kumar Maan · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: alex.james@nu.edu.kz

I. Dolzhikova
e-mail: ifedorova@nu.edu.kz

A. Kumar Maan
e-mail: akshay.maan@gmail.com

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_9

accuracy for the image classification could be improved by eliminating specific pixels through the threshold value.

The first attempt to implement the model for the very simplified version of the biological neuron was presented in 1943 by McCulloch and Pitts [8]. It depends on the threshold logic implementation that computes the sign function represented by Eq. 9.1, where x_i is the input, w_i is the weight of the synapse, n is the number of inputs to the threshold gate, and θ is the threshold value.

$$f(x_1, \ldots, x_n) = sgn\left(\sum_{i=1}^{n} w_i x_i - \theta\right) = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} x_i w_i > \theta \\ 0 \text{ if } \sum_{i=1}^{n} x_i w_i = 0 \\ -1 \text{ if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases} \tag{9.1}$$

Considering Eq. 9.1, the general form of any one-sided thresholding technique, which is the one that has a single decision boundary, could be formulated as follows: the output is represented by logic 'high' for the weighted input greater than or equal to the given threshold value, while logic 'low' corresponds to the case when weighted input is less than threshold. Depending on the application, two-sided or multi-sided thresholding might be required [3].

This chapter focuses on the different methods that have been attempted for implementing threshold logic on the hardware using memristors. It is organized as follows. In Sect. 9.2 we present an overview of the existing hardware implementations of MTL circuits by describing the main blocks and circuit elements of the designs and the type of the problems for which these designs could be applied. Comparative analysis and discussion of the open problems, as well as future prospects, of the available designs, are presented in Sect. 9.3.

9.2 Hardware Realization of Memristive Threshold Logic Circuits

Rajendran et al. [11] presented one of the first implementations of the memristive threshold logic based circuit in 2009. Figure 9.1 demonstrates the circuit design of programmable threshold logic array (PTLA) that consists of Goto pairs and memristors. The presented circuit uses three 5-input TL gates (TLGs). Goto pair is formed by a serial connection of two resonant tunneling diodes (RTDs). Multiple inputs that are connected through memristors at the node between two RTDs drive the Goto pair. The input voltage values are converted to the currents as they pass through the memristors. The clocked RTDs make the Goto pair to be latched, and as a result, the output depends on both the present input and previous outputs. In order to ensure necessary operation of the circuit, the process is divided into pre-discharge phase, during which the past outputs are discharged prior the processing of the present input, and evaluation phase, where the output depends only on the current input. The

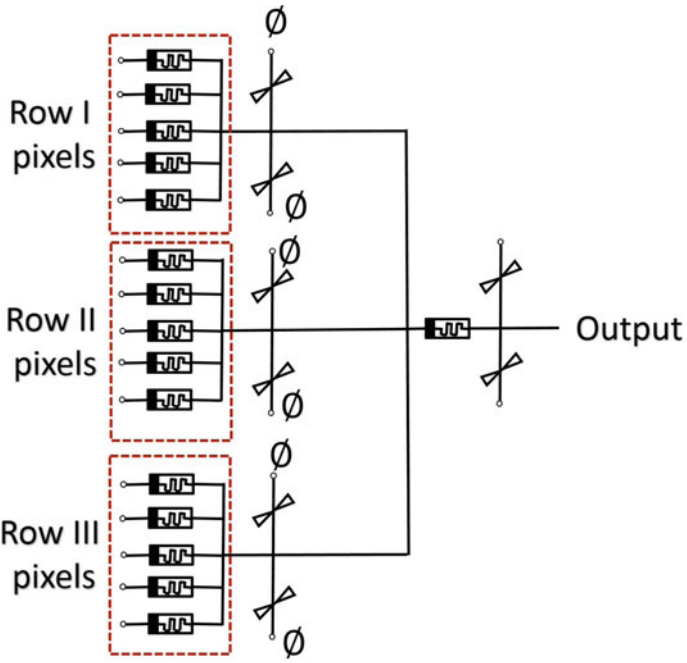

Fig. 9.1 PTLA using memristors and goto pair as proposed in [11]

evaluation phase happens when signal ϕ is high. For every new input, two phases are repeated.

The design presented on Fig. 9.1 was used for 3×5 image classification problem that classifies the input image into a rectangle or a triangle. The design considers the memristors that have 3 distinct resistance levels, i.e. each input could be weighted by 1 of the 3 values that correspond to weight '0' when the memristance is 100 MΩ, '1' for 50 MΩ resistance and weight '2' when resistance is 10 MΩ.The Goto pair produces logic '1' in the output if the sum of the weighted inputs is greater than the threshold (this indicates the rectangle class) and logic '0' otherwise (triangle class of the input image).

Different threshold gate-array architecture was implemented in 2010 [9]. The authors program the memristors to adjust the weights of the inputs. The proposed architecture is shown in Fig. 9.2. The inputs are the voltage values that are converted to the current values after passing through memristors. The weighted input values, i.e. obtained currents, are summed up and the sum is compared with the reference current (I_{ref}). The total number of transistors that is required to implement an n-input threshold gate is $(2 \times n) + 18$. There are two outputs from a memristor-based threshold gate, one of which is the complement of the other.

The additional circuitry on the Fig. 9.2 is used to circumvent specific problems that may arise from the direct realization of memristor-based threshold gate. A cur-

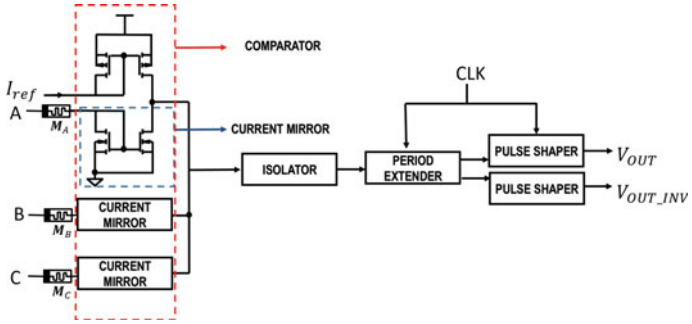

Fig. 9.2 Three-input threshold gate using memristors as proposed in [9]

rent mirror consisting of two transistors is used in each input line to prevent the flow of current from the output of the circuit to the input. If the circuit is used in larger configurations, the next stage can load the current comparator. This could be prevented by adding two transistor based isolation circuit. The proposed circuit express logic '1' as a positive pulse followed by a negative pulse, while the current mirror and isolator operate during a positive period and cut the negative pulse. In order to restore the negative part of the logic '1' signal the circuit for the period extension, which consists of 4 CMOS transistors and is based on CLK signal, is used. One more issue that needs to be considered is that weights should be fixed during the operation of the circuit to ensure proper functionality. The weights might be changed with time even if the constant voltage is applied across memristor. The negative pulse immediately following the positive pulse for representation of logic '1' helps to reverse the changes of the weights that were made during the positive pulse. Logic '0' is represented by 0 V signal. The role of the six transistor pulse shaper in the circuit is to generate a required pattern of the signal for logic '0' and logic '1'. The output from the pulse shaper becomes an input for the next threshold gate.

The proposed n-input threshold gate could be used to construct islands of the gates for implementation of larger cascaded architectures. The crossbar based island with the size of 3×3 is presented on the Fig. 9.3. The number of columns in the crossbar represents the number of threshold gates being used. The number of rows determines the number of inputs to the threshold gate. Each island has input and output interconnection network through which the inputs are supplied and outputs are extracted. Such an interconnection network helps to connect several islands as it is shown in Fig. 9.4. When the cascaded architecture is used the fan-out, which is the maximum number of threshold gates in the island, should be considered. Fan-out of the previous stage determines the acceptable number of columns on the island.

In comparison to the lookup tables (LUTs) that are based on CMOS transistors, the proposed threshold architecture using memristors helps to reduce power consumption and area footprint by approximately 75%. However, this improvement comes at the cost of increased delay (Fig. 9.5).

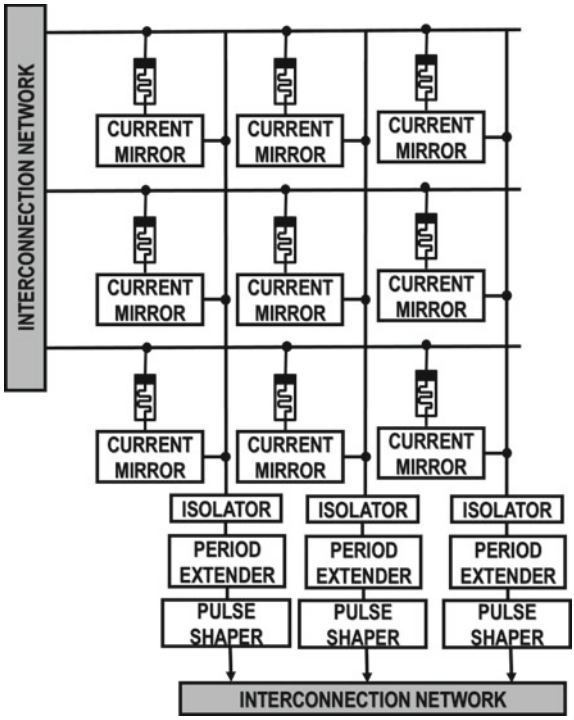

Fig. 9.3 The 3 × 3 crossbar-based island architecture with 3 threshold gates [9]

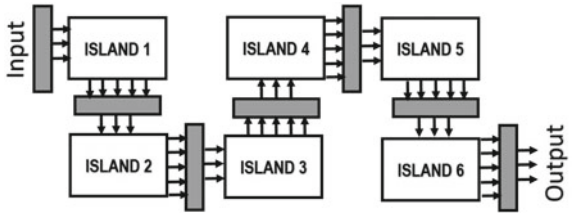

Fig. 9.4 The cascaded architecture using six islands [9]

The training methodology for the MTL gate [10] (see the Fig. 9.6), which is used as a three-input perceptron, is suggested in a research work of Mahem et al. [7]. The training is based on the stochastic gradient descent method. It is used to effectively reconfigure memristor based synapses in the trainable threshold gate array (TTGA). TTGA system consists of an array of threshold gates [10] and CMOS based trainer circuits for local and global training as it is shown in Fig. 9.6. Single perceptron requires one global trainer circuit, while there is a local trainer for each memristor. The proposed design with a single layer based four-input trainable threshold gate (perceptron) is able to realize different linearly separable Boolean functions with up to four inputs to the gate.

In 2012, Tran et al. [13] proposed the reconfigurable design of the threshold logic gates using silver-chalcogenide memristive devices in combination with CMOS

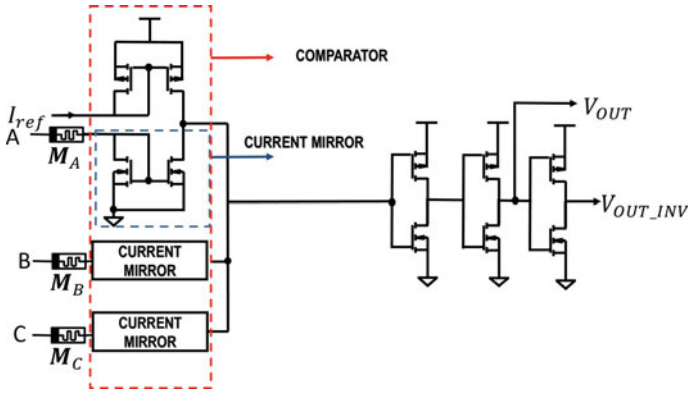

Fig. 9.5 Three-input memristive threshold logic gate [10]

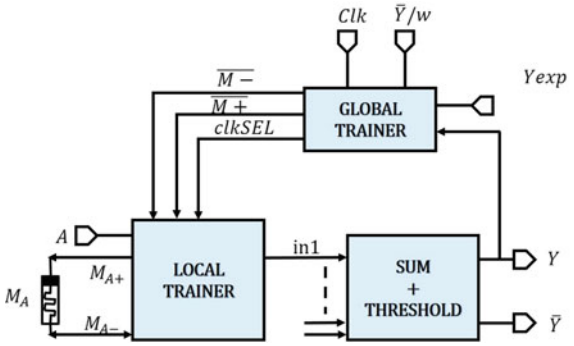

Fig. 9.6 Programming circuit for memristive devices [13]

circuits. By changing the resistance of the memristors, the proposed circuit can produce four separable logic functions (AND, OR, NAND, NOR) using a single layer network. The circuit design is presented in Fig. 9.7. In the proposed design all of the two-input logic gates, except OR gate, require consideration of the negative weights. The negative weights could be obtained by supplemental negative current I_{op} from the output of the operational amplifier (op-amp) that is placed in parallel to the memristor. The summation of the I_{op} and current flowing through the memristor I_{mem} is the synaptic current I_s that can be both positive and negative. The gain of the circuit that determines the range of possible weights is expressed through resistors R_F and R_N. The programming circuit for the memristive devices is illustrated in Fig. 9.8.

Gao et al. [2] proposed another hybrid CMOS/memristor programmable threshold logic gate in 2013. The proposed linear threshold gate (LTG) is demonstrated in Fig. 9.9. Several diode-like memristors are connected in parallel to a pull-down resistor R_L. The behavior of the selected memristive devices depends on the effective threshold V_{TH}. Below this threshold memristors have actively suppressed current, while above it has linear conductance with the slope of $1/R$, which can vary in

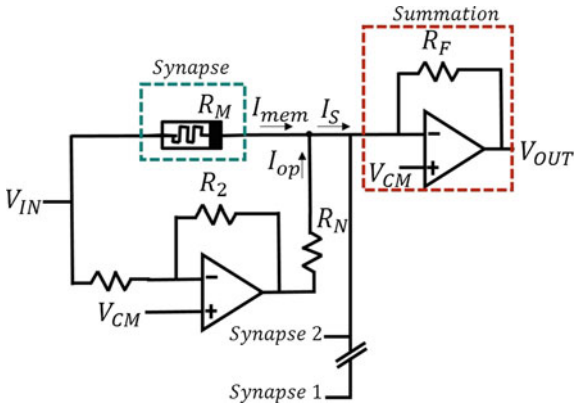

Fig. 9.7 Re-programmable TLG using memristors and CMOS circuits with both positive and negative weights [13]

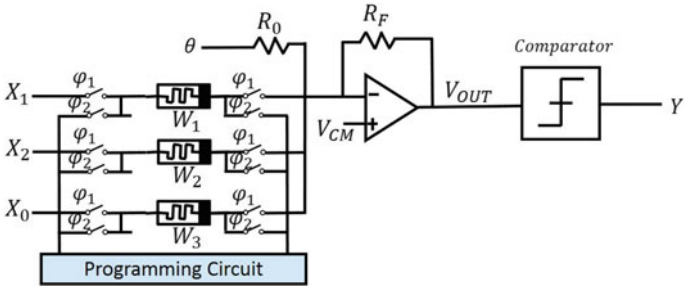

Fig. 9.8 Programming circuit for memristive devices [13]

the range from R_{ON}^{H} to R_{ON}^{L}. The proposed configuration implements the ratioed diode-resistor logic. The variety of Boolean functions that this logic gate can realize is determined by the dynamic range of R_{ON}^{H}/R_{ON}^{L} ratio of memristors. In order to restore the voltage swing and to drive the inputs of the successive logic gates, the diode-resistor logic is usually connected to the CMOS gate. In the suggested design such a CMOS gate is D flip-flop, which in comparison to the CMOS inverter is more efficient for high-throughput pipelined circuits.

The threshold logic circuit implementation of Gao et al. [2] proved to be more robust in comparison to the design that was demonstrated by Rajendran et al. [11] because during logic operation it does not rely on the changes of the memristive states. Another advantage of the proposed design [2] is that unlike methods for threshold logic implementation in [9, 11], there is no need for additional CMOS current mirror circuitry for each memristive device. Due to these additional circuits, the designs of [9, 11] might be unsuitable for a compact the crossbar architectures. In the case of the logic circuit of [2], the area for the design implementation is mostly dependant on the CMOS D flip-flop and small circuitry for programming of memristors.

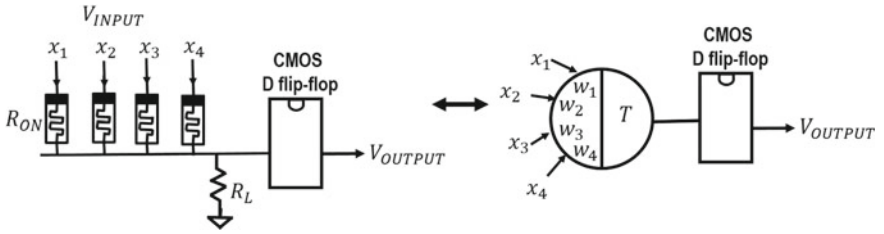

Fig. 9.9 LTG implemented using memristors and CMOS D flip-flop [2]

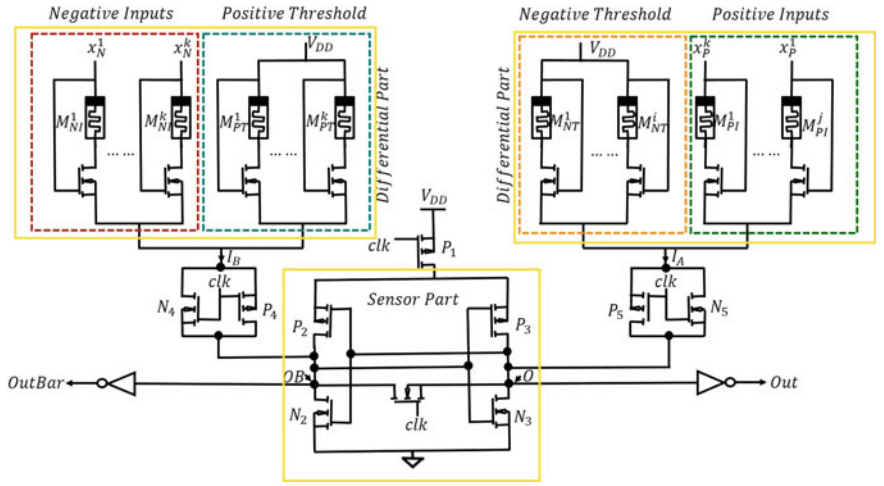

Fig. 9.10 Current-mode memristor based threshold logic as proposed in [1]

A novel Current-Mode Memristor-based Threshold Logic (CMMTL) that was proposed by Dara et al. [1] is shown in Fig. 9.10. The proposed configuration of the logic circuit is composed of two differential blocks and one sensor part. The differential parts consist of sub-blocks for the negative/positive inputs and positive/negative thresholds that are implemented using a serial combination of the memristive devices and NMOS transistors. The thresholding logic of this circuit is as follows: if the current I_A is greater than the current I_B than the voltage at the node O is high and voltage at OB is low, while the reverse situation happens when the current I_B is greater than I_A.

This new clocked design of CMMTL performs better for the three-, four, and five-input threshold logic in comparison to the design of [10]. The 77, 50, and 88% improvements have been noticed for the delay, energy consumption, and energy-delay product, respectively.

Sharad et al. [12] suggested design for dynamic resistive threshold logic for reconfigurable computing. The circuit is presented in Fig. 9.11, where the resistive memory elements are the Ag-Si memristors. The memory elements realize the functions of

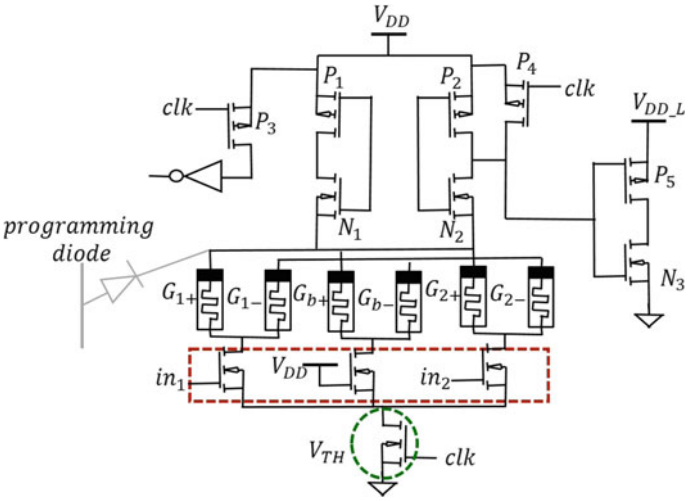

Fig. 9.11 Circuit for 2-fan in DRTL gate [12]

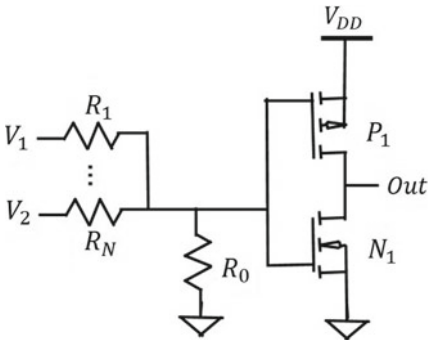

Fig. 9.12 Circuit diagram of the resistive divider boolean logic cell that consists of a two input resistive divider and a variable threshold CMOS inverter [4]

weights and thresholds, and CMOS latch implements comparison operation. Ag-Si memristors were used for the resistive threshold logic gates in the paper of [2], where a CMOS D flip-flop was used for comparison. However, that design required large voltage and static current across memristors that results in large power consumption. The dynamic resistive threshold circuit [12] demonstrated the performance with 96% higher energy efficiency and more than two orders of magnitude lower energy-delay product in comparison to the four-input LUT based CMOS FPGA.

In 2014, James et al. [4] proposed a cell for universal Boolean logic implementation. The design of the logic cell is shown in the Fig. 9.12, where the analog resistive divider part could be implemented using memristors and CMOS inverter is used for the threshold operation that produces the binary output.

By changing the threshold voltage of the inverter, the configuration could be used to implement the NAND and NOR logic. For the $V_{DD} = 1V$, high input $V_H = 1$ and a low input value of V_L the two-input NOR gate requires the threshold voltage in the

range from 0 to 1/3 V, while the inverter threshold of $2/3\,\text{V} < V_{TH} < 1/3\,\text{V}$ will produce the NAND gate. In case if the threshold value of the inverter could be reduced to a very small voltage, the resistive threshold logic with a large number of inputs could be implemented. The authors propose to introduce additional inverters to lower the threshold [4]. This idea is illustrated in Fig. 9.12. The additional inverters have different power supply voltages. The set of switches help to regulate the universal logic configuration to realize AND, NAND, OR, NOR, and NOT gates. The NAND logic is possible when the switches S_1 and S_4 are closed and the output is measured at V_{out} (Fig. 9.13). The switches S_1 and S_3 are closed when the AND logic is required. The output, in this case, is measured at $\overline{V_{out}}$. The switches S_2 and S_3 are enabled together to implement OR logic. The implementation of the NOR gate requires only one inverter that is enabled by closing S_2 and S_4. The design is based on 0.25 μm TSMC process and HP memristor.

James et al. [5] presented Memristive-CMOS circuits with the threshold logic computing for fast Fourier transform and Vedic multiplication. The variety of proposed circuits is illustrated in Fig. 9.14, where combinations of memristor-based averaging circuits, operational amplifiers and/or inverters are used.

The application of the MTL for the detection of moving objects in real-time was noticed in the work of Maan et al. [6]. The idea of the circuit implementation is the same as it was presented in Fig. 9.12 [4]. The memristive resistance based voltage divider and CMOS inverter are used to encode the pixels from the template

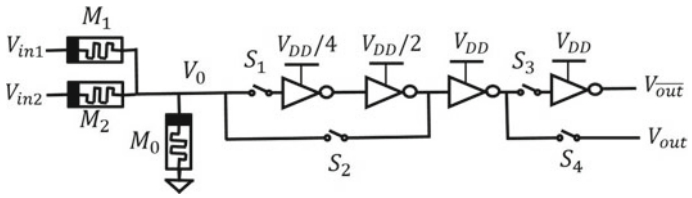

Fig. 9.13 Circuit implementation of NAND, NOR, AND, OR and NOT logic gates that are based on memristive resistance divider and CMOS inverters with three different V_{DD} values [4]

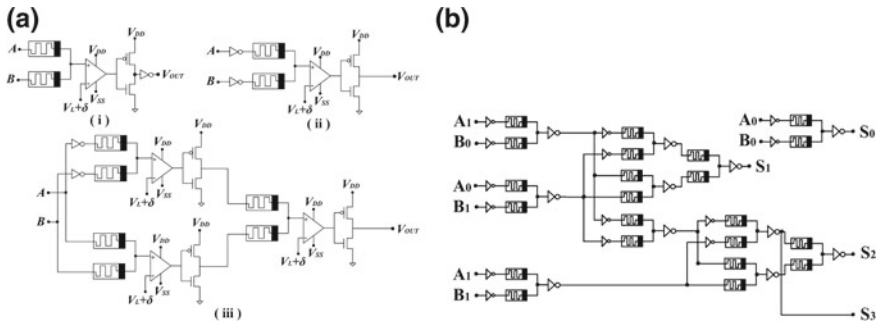

Fig. 9.14 **a** Circuit diagrams of the logic gates using proposed cell (i) OR gate (ii) AND gate (iii) XOR gate **b** 2 bit memristive threshold vedic multiplier [5]

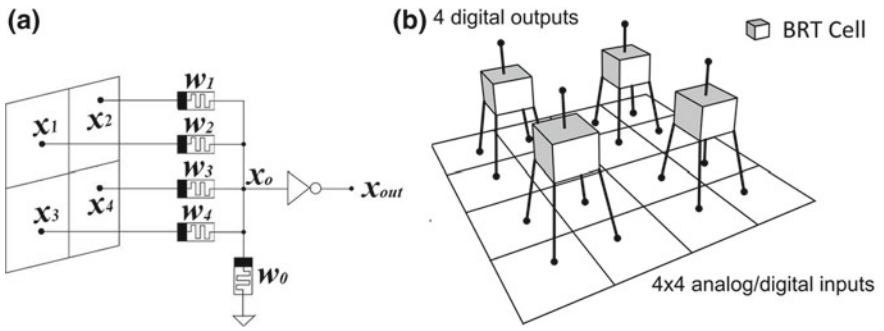

Fig. 9.15 **a** A modular BPRT cell with four input pixels **b** Application of the proposed BPRT cell for image dimensionality reduction [6]

image. The Fig. 9.15a shows a modular Bilevel Programmable Resistance memristive Threshold logic (BPRT) cell with four-input pixel values. This BPRT cell with four analog/digital pixel inputs is used for image dimensionality reduction as illustrated on the Fig. 9.15b. The proposed logic cell when compared with CMOS equivalent designs demonstrate improved performance in the area, leakage power, power dissipation, and delay.

9.3 Discussion

The properties of low leakage currents, resistive switching, and smaller on-chip area make memristor an attractive element as a building block for in-place memory and execution. Threshold logic, being the simplest type of computation unit, lends itself nicely to the synaptic unit required for neuron activation during firing of neurons in the biological brain. Hence, memristive threshold logic circuits have gained significant traction among researchers for most straightforward logic functionalities such as logic gates to more involved cognitive tasks such as face recognition. In this chapter, we have shown practical applications of memristive threshold logic circuits that can be used to mimic biological neuron and perform hardware realizations of memristive synapses, memristor-based programmable threshold logic arrays, memristive threshold logic gates, hybrid CMOS/memristor threshold logic circuits, memristive threshold logic circuits for fast moving object detection and for Fourier transformation and Vedic multiplication.

9.3.1 Open Problems

In comparison with CMOS-only circuits, memristor-based circuits are faced with the complexity of power dissipation reduction and integration with CMOS-memristor hybrid circuits. In a threshold logic configuration, the resistive path inherent in memristive circuits allows for larger currents resulting in higher power dissipation. Also, leakage currents in MOS-gated memristive arrays outweigh the low leakage properties of memristive circuits, requiring compensated read-out circuits. Also, realizing different states in a programmable memristive array is often challenging, requiring add-on circuits and thus increasing the complexity of the circuits. If not designed carefully the area of programming circuits can outweigh the benefits of the lower area offered by the memristors. For further research and development of memristive threshold logic circuits, more accurate models for memristors are needed so they can be incorporated into the prevailing simulation tools. These tools need to be able to simulate the wide range of behaviors exhibited by memristors.

Besides, the physical level design of memristors and Memristor-CMOS hybrid circuits is challenged by the lack of process standards and requires further adoption of these devices in the semiconductor industry. In this sense, memristor devices and circuits are quite early in the stages of commercial implementation.

9.3.2 Future Prospects

The inherent ability of the memristive threshold logic circuits to act as in-place programmable memory in hardware opens up a wide range of applications, such as memory architectures that have ability to learn, remember and forget, design of human like silicon self-learning classifiers, implementation of intelligent modules for programmable and self-learning chips, design of analog sensory processing such as in computer vision, development of very large scale imulation and implementation of such threshold logic systems.

9.4 Summary

The emergence of the new nano devices such as the memristor brings numerous opportunities for the field of bio-inspired computing. Threshold logic implementation is one of such areas that quickly realized the advantages that this small device can bring. However, along with the opportunities, the challenges related to the integration of the memristors into the chip should be considered.

This chapter surveyed the modeling approaches for the most recent implementations of memristor-based threshold logic architectures on the hardware. It also

provided an overview of biggest challenges in the design of memristive circuit configurations and several application areas for MTL.

Chapter Highlights

- Threshold logic gates (TLGs) are known for high-speed and low power consumption performance, which is essential for such applications as real-time processing and recognition of natural signals, as well as on-chip memory architecture and neural network implementation.
- Integration of memristors into the design of TL allows extending the capabilities of TL circuits.
- However, along with the opportunities, the challenges related to the integration of the memristors into the chip should be considered.

References

1. Dara CB, Haniotakis T, Tragoudas S (2013) Low power and high speed current-mode memristor-based tlgs. In: 2013 IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT). IEEE, pp 89–94
2. Gao L, Alibart F, Strukov DB et al (2013) Programmable cmos/memristor threshold logic. IEEE Trans Nanotechnol 12(2):115–119
3. James AP (2014) Threshnomics: an introduction to threshold logic in algorithms and circuits. J Comput Sci Syst Biol 7(6):1
4. James AP, Francis LRV, Kumar DS (2014) Resistive threshold logic. IEEE Trans VLSI Syst 22(1):190–195
5. James AP, Kumar DS, Ajayan A (2015) Threshold logic computing: memristive-CMOS circuits for fast fourier transform and vedic multiplication. IEEE Trans Very Large Scale Integr (VLSI) Syst 23(11):2690–2694
6. Maan AK, Kumar DS, Sugathan S, James AP (2015) Memristive threshold logic circuit design of fast moving object detection. IEEE Trans Very Large Scale Integr (VLSI) Syst 23(10):2337–2341
7. Manem H, Rajendran J, Rose GS (2012) Stochastic gradient descent inspired training technique for a CMOS/nano memristive trainable threshold gate array. IEEE Trans Circuits Syst I: Regul Pap 59(5):1051–1060
8. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133
9. Rajendran J, Manem H, Karri R, Rose GS (2010) Memristor based programmable threshold logic array. In: Proceedings of the 2010 IEEE/ACM international symposium on nanoscale architectures. IEEE Press, pp 5–10
10. Rajendran J, Manem H, Karri R, Rose GS (2012) An energy-efficient memristive threshold logic circuit. IEEE Trans Comput 61(4):474–487

11. Rajendran J, Manem H, Rose GS (2009) NDR based threshold logic fabric with memristive synapses. In: 9th IEEE conference on nanotechnology, 2009, IEEE-NANO 2009. IEEE, pp 725–728
12. Sharad M, Fan D, Roy K (2013) Ultra-low energy, high-performance dynamic resistive threshold logic. arXiv preprint arXiv:1308.4672
13. Tran T, Rothenbuhler A, Smith EHB, Saxena V, Campbell KA (2012) Reconfigurable threshold logic gates using memristive devices. In: 2012 IEEE Subthreshold microelectronics conference (SubVT). IEEE, pp 1–3

Chapter 10
Memristive Deep Convolutional Neural Networks

Olga Krestinskaya and Alex Pappachen James

Abstract This chapter covers the implementation of deep learning neural networks and memristive systems. In particular, deep memristive convolutional neural network (CNN) implementation is illustrated. In addition, the main issues and challenges of deep neural network implementation are discussed.

10.1 Introduction

Deep learning neural networks are networks with a large number of cascaded layers [3, 8, 11]. In the implementation of deep learning neural networks on hardware, power consumption and on-chip area are the critical parameters [5]. Therefore, memristor-based implementation is a promising solution for this due to a small on-chip area and power dissipation. Therefore, this chapter focuses on deep memristive neural networks. Different algorithmic implementations, FPGA-based acceleration, and modifications of deep neural networks have been proposed recently for various applications [1, 2, 4, 10]. However, there are only several existing works showing deep neural networks on hardware with memristive devices [12, 13]. In most of the cases, two-layer neural networks are implemented [6]. Most of the existing implementations of deep memristive neural networks with more than three layers are Convolutional Neural Networks (CNN) [12–14].

O. Krestinskaya · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

O. Krestinskaya
e-mail: ok@ieee.org

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_10

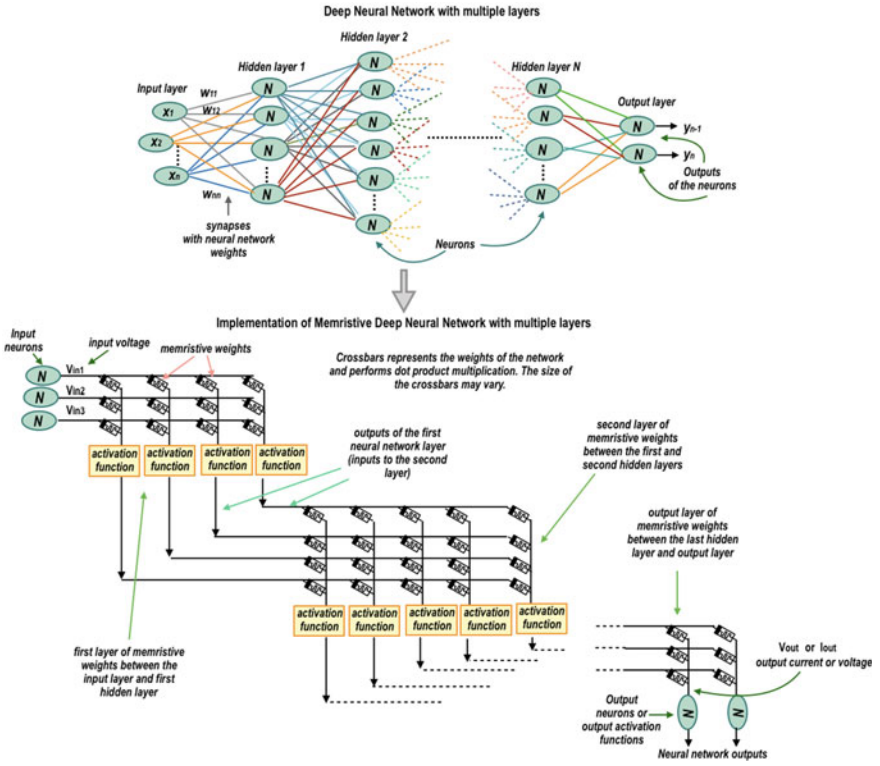

Fig. 10.1 Deep multilayer neural network [7]

10.2 Deep Memristive Neural Network

The deep neural network is a network with a large number of cascaded layers. The activation functions between the layers of the neural network can be different [3]. For example, deep neural networks for the classification can be implemented with sigmoid or binary thresholding function at the output layer and Rectifier Linear Unit (ReLU) in all other layers. As in such systems, each output neuron is mostly responsible for the classification of the input data, presented as 1 or 0 (same or different class).

The implementation of such systems is mostly performed using memristive crossbar architecture. The crossbar is used to perform dot product multiplication of inputs, which are usually presented as input voltages, by the memristive weights, represented by the conductance of the memristor. The output of the crossbar is a weighted summation of the input voltage shown in Fig. 10.1, which is usually represented as current from the crossbar or voltage. The crossbar output is fetched to the activation function or the output neuron. Different activation functions can be used in the memristive neural network. The memristive deep neural network is illustrated in Fig. 10.1.

It consists of the memristive crossbars of the different sizes with the activation functions between the memristive layers. The size of the crossbars depends on the input size and the number of neurons in the hidden layer. The number memristors in the output layer depend on the number of output neurons. The number of layers in such network varies depending on the application of the network, input data, and the required accuracy.

10.3 Deep Convolutional Memristive Neural Networks

Most of the deep learning neural networks with memristive devices proposed in recent years investigated the implementation of convolutional neural networks (CNN). The research works [13, 14] illustrates the implementation of deep memristive CNN containing five layers. The architecture for this network is illustrated in Fig. 10.2. CNN architecture consists of feature extraction and sub-sampling layers and dense network for classification. The input data is propagated through several convolutional layers of the network and is classified in the dense layer. Feature extraction and sub-sampling layers reduce the number of features in the processed data (image); however, create several feature maps using different convolutional filters. For example, if in five layers neural network the initial size of the input image is 28×28 and the size of the convolutional filter is 5×5, the output of the first convolution layer is 24×24. The

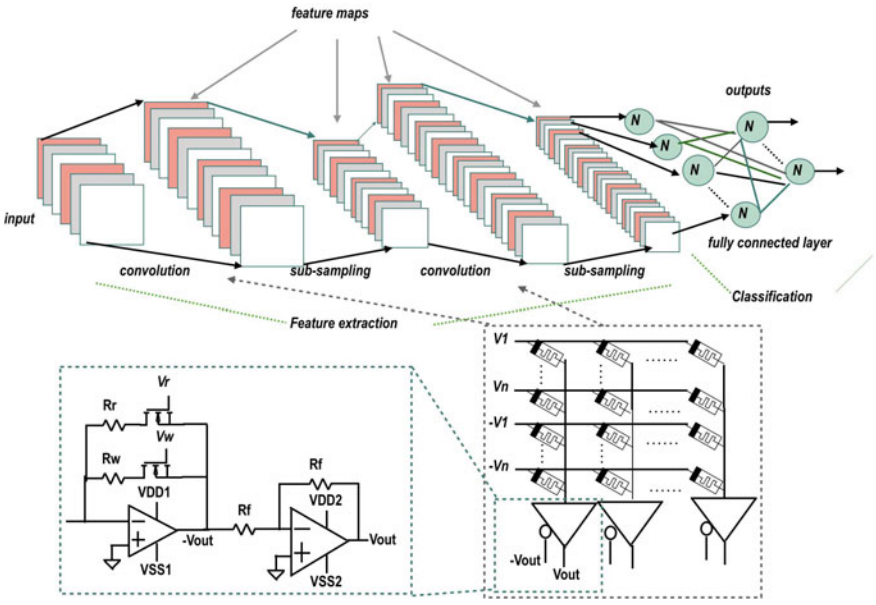

Fig. 10.2 Deep convolutional neural network implementation [13, 14]

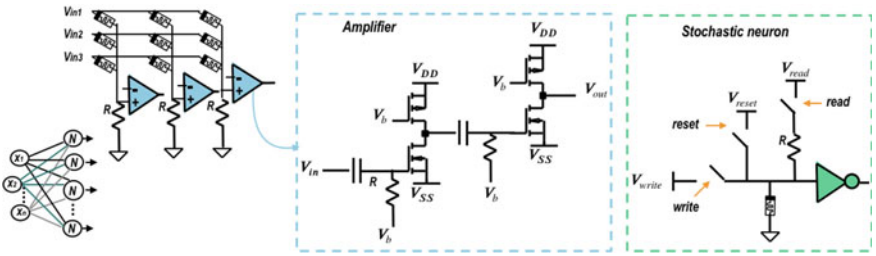

Fig. 10.3 Crossbar structure for fully connected layer and stochastic neuron used in deep stochastic spiking neural network [12]

sub-sampling layer reduces the size of the image two times and produces the images of size 12 × 12. The same is performed in the second convolutional and sub-sampling layers, and the residual output image is of the size 4 × 4. However, as the number of convolutional filters can be varied, the number of such 4 × 4 output images also vary, depending on the number of required discriminative features. All these features are fetched into the fully connected layer. These allow classifying the images more accurately in the dense layers, as more discriminative features are created. Fully connected dense layer is used for classification. Figure 10.2 illustrates the circuit level implementation of CNN layers. All the layers are based on the memristive crossbars. In the convolutional layers, memristors in a crossbar correspond to the convolutional filter weights. The dense layer is similar to two-layer neural network. The activation function consists of two amplifiers to read the output current from the memristive crossbar. The architecture has software-based learning, and the memristive weights are programmed using the values calculated on software in the learning stage while classification is purely hardware-based. The size of the memristive crossbars in the layers depends on the size of the input image [14]. The simulation results show it is possible to obtain the accuracy of 91.8% for handwritten digits classification with MNIST database using such network.

The implementation of deep stochastic spiking CNN is shown in [12]. The classification is performed based on the selection of the largest number of output spikes in the output neurons during a particular period. The crossbar structure for the fully connected layer and stochastic neuron used in the network are illustrated in Fig. 10.3. The voltage is read from measuring resistors and amplifier is used to amplify low voltage signals read across this resistors. Comparing to the CMOS based design, such a network allows reducing the power consumption eight times and on-chip area 6.4 times. This network allows achieving the accuracy of 97.84% for handwritten digits classification with MNIST database.

10.4 Analog Learning Hardware for Deep Neural Networks: Challenges and Issues

In the hardware implementation of deep learning neural networks, the main issue is scalability of the network. As the number of layers is large, the scaling of the network leads to a significant increase of power consumption and on-chip area of the network. Therefore, memristive devices are proposed to be a solution for such networks. Implementation of deep neural networks based on memristive crossbars ensures the scalability of the design and reduction of on-chip area and power consumption, comparing to the traditional CMOS-based or FPGA-based designs. However, there are specific challenges in memristive implementations.

The requirements of Internet-of-Things (IoT) applications and edge computing systems with local processing requirements force the development of local processing, where the hardware implementation of neural networks and learning systems is a solution. However, in most of the cases, the learning in memristive deep neural networks is performed on software. Even though several solutions to implement learning on hardware are proposed [6, 7, 15], this is still an open problem that should be investigated.

In the memristor-based implementation of deep neural networks, there are also specific challenges and issues, such as variability of the devices, the immaturity of the technology, the endurance of memristive devices, restricted resistive levels which can affect the accuracy of the system. The recently proposed partially fabricated memristive neural network is shown in [9]. However, the processing and update of memristive crossbars is still performed on software. And the implementation of deep learning neural network with multiple layers on hardware is still an open problem.

10.5 Conclusion

This chapter covered the deep learning systems and their implementation in memristive hardware. Memristive devices are one of the promising solutions to ensure the scalability and low power consumption of such network. Most of the hardware implementations of memristive deep learning systems focus on CNN. The open problems and challenges in deep learning memristive systems include implementation of learning algorithms on hardware and imperfections of memristive devices.

Chapter Highlights

- Deep neural network is a network with a large number of cascaded layers and different activation functions between the layers.
- Most common implementation of deep memristive neural networks is based on several crossbars with memristive devices.
- Most of the existing implementations of deep memristive neural networks with more than three layers are Convolutional Neural Networks (CNN), used for image processing applications.
- The main challenges in such networks include implementation complexity of learning circuit and imperfections of memristive devices.

References

1. CireşAn D, Meier U, Masci J, Schmidhuber J (2012) Multi-column deep neural network for traffic sign classification. Neural Netw 32:333–338
2. Courbariaux M, Bengio Y, David JP (2015) Binaryconnect: training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems, pp 3123–3131
3. Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning, vol 1. MIT press, Cambridge
4. Hwang K, Sung W (2014) Fixed-point feedforward deep neural network design using weights +1, 0, and −1. In: 2014 IEEE workshop on signal processing systems (SiPS). IEEE, pp 1–6
5. Krestinskaya O, James AP, Chua LO (2018) Neuro-memristive circuits for edge computing: a review. arXiv:1807.00962
6. Krestinskaya O, Salama KN, James AP (2018) Analog backpropagation learning circuits for memristive crossbar neural networks. In: 2018 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 1–5
7. Krestinskaya O, Salama KN, James AP (2018) Learning in memristive neural network architectures using analog backpropagation circuits. IEEE Trans Circuits Syst I: Regul Pap
8. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436
9. Li C, Hu M, Li Y, Jiang H, Ge N, Montgomery E, Zhang J, Song W, Dávila N, Graves CE et al (2018) Analogue signal and image processing with large memristor crossbars. Nat Electron 1(1):52
10. Ovtcharov K, Ruwase O, Kim JY, Fowers J, Strauss K, Chung ES (2015) Accelerating deep convolutional neural networks using specialized hardware. Microsoft Res Whitepaper 2(11)
11. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
12. Wijesinghe P, Ankit A, Sengupta A, Roy K (2018) An all-memristor deep spiking neural computing system: a step toward realizing the low-power stochastic brain. IEEE Trans Emerg Top Comput Intell 2(5):345–358. https://doi.org/10.1109/TETCI.2018.2829924
13. Yakopcic C, Alom MZ, Taha TM (2016) Memristor crossbar deep network implementation based on a convolutional neural network. In: 2016 International joint conference on neural networks (IJCNN), pp 963–970. https://doi.org/10.1109/IJCNN.2016.7727302

14. Yakopcic C, Alom MZ, Taha TM (2017) Extremely parallel memristor crossbar architecture for convolutional neural network implementation. In: 2017 International joint conference on neural networks (IJCNN). IEEE, pp 1696–1703
15. Zhang Y, Wang X, Friedman EG (2018) Memristor-based circuit design for multilayer neural networks. IEEE Trans Circuits Syst I: Regul Pap 65(2):677–686

Chapter 11
Overview of Long Short-Term Memory Neural Networks

Kamilya Smagulova and Alex Pappachen James

Abstract Long Short-term Memory was designed to avoid vanishing and exploding gradient problems in recurrent neural networks. Over the last twenty years, various modifications of an original LSTM cell were proposed. This chapter gives an overview of basic LSTM cell structures and demonstrates forward and backward propagation within the most widely used configuration called traditional LSTM cell. Besides, LSTM neural network configurations are described.

11.1 Background

Evolution of artificial neural networks would not have been possible without Rosenblatt's perceptron model, first introduced in 1958 and still valid these days. It consists of a neuron with weighted inputs and a bias (Fig. 11.1a) [1]. Traditional neural networks are formed from multiple layers of interconnected perceptrons and outperform many conventional data processing methods. However, most of them are inefficient for processing sequential data. A striking feature of such data is order-dependence that cannot be maintained in mentioned neural networks.

At the beginning of the 1980s, the problem was solved by the inclusion of feedback connections between layers and nodes of a neural network. By this, a correlation between previous and current input data was achieved. The obtained configuration is called a recurrent neural network (RNN). Theoretically, RNN is capable of processing arbitrary long sequences. However, it is impossible due to the infinite growth of constant error backflow during training. In other words, training algorithms for RNN use gradient-based approaches (e.g., Backpropagation through time (BPTT), Real-time recurrent learning (RTRL) and others) which lead to either vanishing or exploding gradient problems. As a consequence, RNN can look only several steps

K. Smagulova · A. P. James (✉)
Nazarbayev University, 53, Kabanbay Batyr ave., Astana, Kazakhstan
e-mail: apj@ieee.org

K. Smagulova
e-mail: ksmagulova@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_11

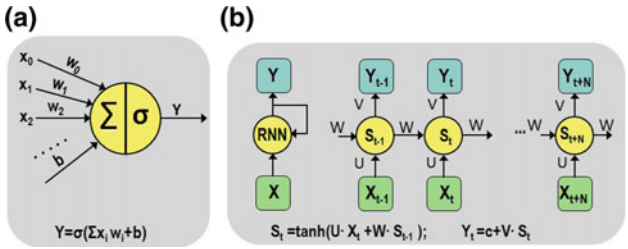

Fig. 11.1 **a** A perceptron model; and **b** Recurrent neural network

back. Therefore, Long Short-term memory (LSTM) design was suggested. It is a particular type of RNN that allows processing sequences of arbitrary length and do not suffer from RNN disadvantages [2]. In the following section different LSTM cell configurations are revised.

11.2 LSTM Cell Structure

Since the first introduction of LSTM architecture, various cell variants were implemented. This section reviews existing LSTM configurations starting from an original cell variant to most commonly used ones such as traditional LSTM, LSTM with peephole connections and convolutional LSTM. Also, phased LSTM architecture is described. To keep consistency, the following notations were adopted:

$\mathbf{x}_t$	Input vector
h_{t-1}	Output of a previous cell
C_t	Cell memory of current state
C_{t-1}	Cell memory of a previous cell
$\tilde{C}_t$	Candidate to a cell memory
i_t	Input gate
o_t	Output gate
f_t	Forget gate
g_t	Input gate
σ	sigmoid function
tanh	hyperbolic tangent function
$W^{(*)}, U^{(*)}, V^{(*)}$	weight matrices
b^*	biases

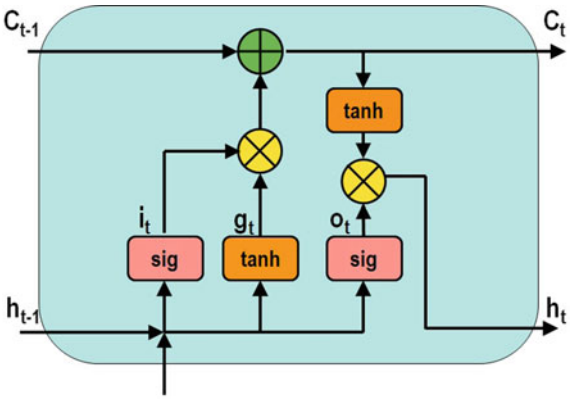

Fig. 11.2 An original LSTM cell

11.2.1 Original LSTM

Hochreiter and Schmidhuber suggested the original LSTM model in 1995 [3]. Traditional self-connected RNN units were enhanced by including memory cell and gates to control the flow of information within a cell and network [4]. Each gate is a weighted matrix with activation function layer at the output.

In a proposed configuration (Fig. 11.2), at a given time t, input data x_t concatenates with a previous cell output h_{t-1}. The resulting vector passes through input node g_t, input gate i_t and output gate o_t as in Eqs. (11.1)–(11.3):

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W^{(i)}x_t} + \boldsymbol{U^{(i)}h_{t-1}} + \boldsymbol{b^{(i)}}) \tag{11.1}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W^{(o)}x_t} + \boldsymbol{U^{(o)}h_{t-1}} + \boldsymbol{b^{(o)}}) \tag{11.2}$$

$$\boldsymbol{g_t} = \tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W^{(g)}x_t} + \boldsymbol{U^{(g)}h_{t-1}} + \boldsymbol{b^{(g)}}) \tag{11.3}$$

A hyperbolic tangent layer g_t forms a new memory cell state candidate $\tilde{C}_t$. A sigmoid layer i_t decides which values of it to be updated. A final memory state C_t is a sum of two components: $\tilde{C}_t$ and a previous cell state C_{t-1}, both pointwise filtered by forget and inpute gates respectively.

$$\boldsymbol{C_t} = \tilde{\boldsymbol{C}}_t \bigodot \boldsymbol{i_t} + \boldsymbol{C_{t-1}} \tag{11.4}$$

$$\boldsymbol{h_t} = \tanh(\boldsymbol{C_t}) \bigodot \boldsymbol{o_t} \tag{11.5}$$

Despite the success of an original LSTM cell in learning long-term dependencies, a weakness had been discovered: the absence of reset function leads to infinite growth of state value and following break down of the whole network.

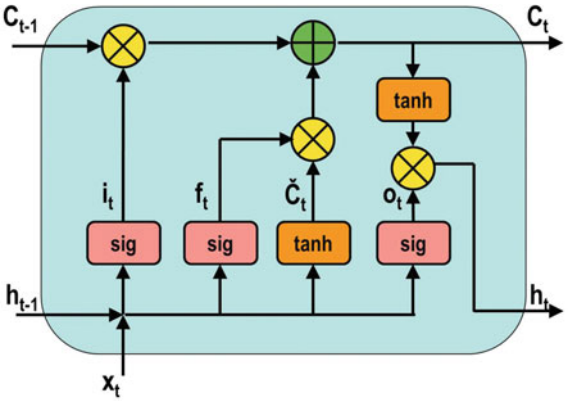

Fig. 11.3 Traditional LSTM cell

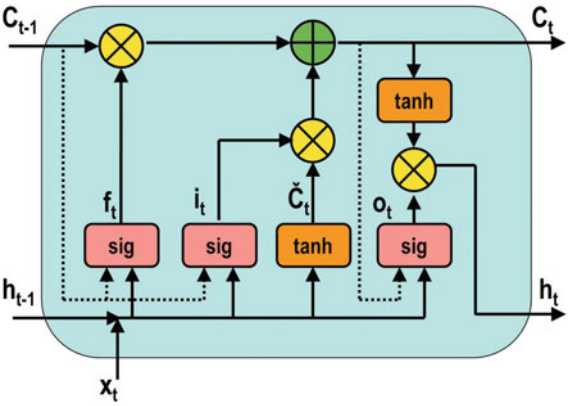

Fig. 11.4 An extended LSTM cell with peephole connections

11.2.2 Traditional LSTM

Introduction of 'forget' gate to the original LSTM cell helped to overcome the shortage [4]. Nowadays, it is the most commonly used LSTM configuration. In this LSTM (Fig. 11.3), forget gate decides whether to pass information from a previous cell memory state C_{t-1} or block it completely. Traditional LSTM forward and backward propagations are described below.

Forward Propagation

During forward propagation, $\boldsymbol{x_t}$ is multiplied with corresponding gate weight matrices W^f, W^i, W^g, W^o and $\boldsymbol{h_{t-1}}$ is multiplied with hidden unit weight matrices U^f, U^i, U^g, U^o. Biases $b^{(*)}$ are added to make adjustments to the matrix output values. Then, obtained weighted sums pass through corresponding activation function layers. The resulting LSTM gates' outputs can be calculated based on Eqs. (11.1)–(11.4).

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x_t} + \boldsymbol{U}^{(f)}\boldsymbol{h_{t-1}} + \boldsymbol{b}^{(f)}) \tag{11.6}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W^{(i)}x_t} + \boldsymbol{U^{(i)}h_{t-1}} + \boldsymbol{b^{(i)}}) \tag{11.7}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W^{(o)}x_t} + \boldsymbol{U^{(o)}h_{t-1}} + \boldsymbol{b^{(o)}}) \tag{11.8}$$

$$\boldsymbol{g_t} = \tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W^{(g)}x_t} + \boldsymbol{U^{(g)}h_{t-1}} + \boldsymbol{b^{(g)}}) \tag{11.9}$$

Eventually, a new memory state value $\boldsymbol{C_t}$ and a cell output $\boldsymbol{h_t}$ can be calculated using Eqs. (11.10) and (11.11) respectively.

$$\boldsymbol{C_t} = \boldsymbol{g_t} \bigodot \boldsymbol{i_t} + \boldsymbol{f_t} \bigodot \boldsymbol{C_{t-1}} \tag{11.10}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \bigodot \tanh(\boldsymbol{C_t}), \tag{11.11}$$

Backward Propagation

To perform backward propagation through time (BPTT) following notations are adopted:

$$\boldsymbol{gateS_t} = \begin{bmatrix} \boldsymbol{g_t} \\ \boldsymbol{i_t} \\ \boldsymbol{f_t} \\ \boldsymbol{o_t} \end{bmatrix}, \; W = \begin{bmatrix} \boldsymbol{W^{(g)}} \\ \boldsymbol{W^{(i)}} \\ \boldsymbol{W^{(f)}} \\ \boldsymbol{W^{(o)}} \end{bmatrix}, \; U = \begin{bmatrix} \boldsymbol{U^{(g)}} \\ \boldsymbol{U^{(i)}} \\ \boldsymbol{U^{(f)}} \\ \boldsymbol{U^{(o)}} \end{bmatrix}, \; b = \begin{bmatrix} \boldsymbol{b^{(g)}} \\ \boldsymbol{b^{(i)}} \\ \boldsymbol{b^{(f)}} \\ \boldsymbol{b^{(o)}} \end{bmatrix}$$

Deltas within LSTM unit are calculated using following equations [6]:

$$\delta \boldsymbol{h_t} = \Delta_t + \Delta \boldsymbol{h_t} \tag{11.12}$$

$$\delta \boldsymbol{C_t} = \delta \boldsymbol{h_t} \bigodot \boldsymbol{o_t} \bigodot (1 - \tanh^2(\boldsymbol{C_t})) + \delta \boldsymbol{C_{t+1}} \bigodot \boldsymbol{f_{t+1}} \tag{11.13}$$

$$\delta \boldsymbol{g_t} = \delta \boldsymbol{C_t} \bigodot \boldsymbol{i_t} \bigodot (1 - \boldsymbol{g_t}^2) \tag{11.14}$$

$$\delta \boldsymbol{i_t} = \delta \boldsymbol{C_t} \bigodot \boldsymbol{g_t} \bigodot (1 - \boldsymbol{i_t}) \tag{11.15}$$

$$\delta \boldsymbol{f_t} = \delta \boldsymbol{C_t} \bigodot \boldsymbol{C_{t-1}} \bigodot \boldsymbol{f_t} \bigodot (1 - \boldsymbol{f_t}) \tag{11.16}$$

$$\delta \boldsymbol{o_t} = \delta \boldsymbol{h_t} \bigodot \tanh(\boldsymbol{C_t}) \bigodot \boldsymbol{o_t} \bigodot (1 - \boldsymbol{o_t}) \tag{11.17}$$

$$\delta \boldsymbol{x_t} = \boldsymbol{W^T} \cdot \delta \boldsymbol{gateS_t} \tag{11.18}$$

$$\Delta \boldsymbol{h_{-1}} = \boldsymbol{U}^{\boldsymbol{T}} \cdot \delta \boldsymbol{gateS_1} \tag{11.19}$$

Based on above,

$$\boldsymbol{W}^{\boldsymbol{new}} = \boldsymbol{W}^{\boldsymbol{old}} - \lambda \cdot \delta \boldsymbol{W}^{\boldsymbol{old}}, \tag{11.20}$$

where λ is a Stochastic Gradient Descent (SGD) coefficient and deltas $\delta \boldsymbol{W} = \sum_{t=1}^{T} \delta \boldsymbol{gateS_t} \cdot \boldsymbol{x_t}$, $\delta \boldsymbol{U} = \sum_{t=1}^{T} \delta \boldsymbol{gateS_{t+1}} \cdot \boldsymbol{h_t}$, and $\delta \boldsymbol{b} = \sum_{t=1}^{T} \delta \boldsymbol{gateS_{t+1}}$.

11.2.3 LSTM with Peephole Connections

Precise timing in LSTM can be achieved by adding peephole connections [6]. The equations to calculate values of gates in a Peephole LSTM are provided below:

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W}^{(f)} \cdot \boldsymbol{x_t} + \boldsymbol{U}^{(f)} \cdot \boldsymbol{h_{t-1}} + \boldsymbol{V}^{(f)} \cdot \boldsymbol{C_{t-1}} + \boldsymbol{b}^{(f)}) \tag{11.21}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W}^{(i)} \cdot \boldsymbol{x_t} + \boldsymbol{U}^{(i)} \cdot \boldsymbol{h_{t-1}} + \boldsymbol{V}^{(i)} \cdot \boldsymbol{C_{t-1}} + \boldsymbol{b}^{(i)}) \tag{11.22}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W}^{(o)} \cdot \boldsymbol{x_t} + \boldsymbol{U}^{(o)} \cdot \boldsymbol{h_{t-1}} + \boldsymbol{V}^{(g)} \cdot \boldsymbol{C_t} + \boldsymbol{b}^{(o)}) \tag{11.23}$$

And as in the previous LSTM configurations,values of g_t, C_t, h_t can be obtained from Eqs. (11.9)–(11.11) (Fig. 11.4).

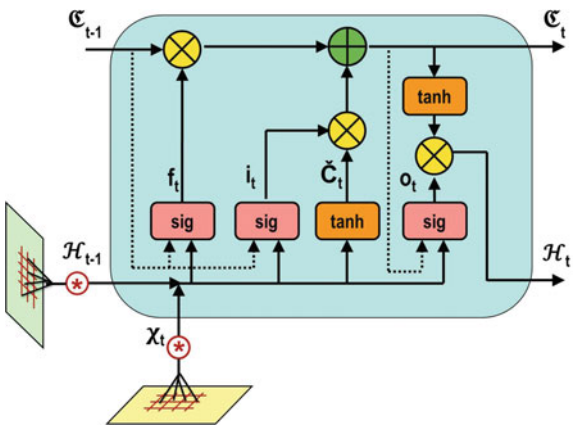

Fig. 11.5 A ConvLSTM

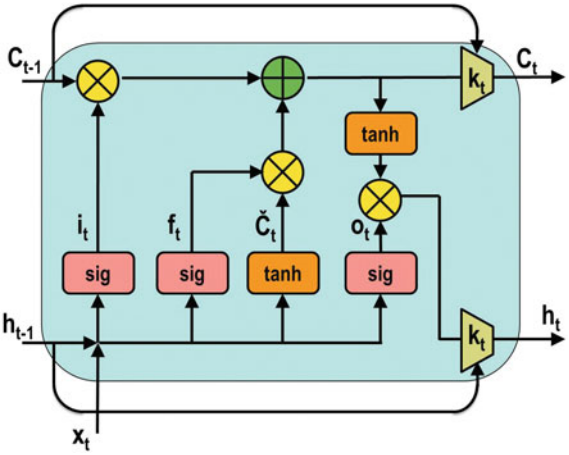

Fig. 11.6 A phased LSTM

11.2.4 Convolutional LSTM

Convolutional LSTM (ConvLSTM) in the (Fig. 11.5) allows to process spatial data without redundancy of fully-connected LSTM. In this configuration, all inputs χ_t, $\mathcal{H}_{t-1}$ and gates f_t, i_t, o_t are convolutional structures [7]. They can be presented as vectors in the form of 3D Tensors with two spatial dimensions. ConvLSTM are found to be effective in spatiotemporal sequence forecasting problems (Fig. 11.6).

$$f_t = \sigma(W^{(f)} * \chi_t + U^{(f)} * \mathcal{H}_{t-1} + V^{(f)} \cdot \mathcal{C}_{t-1} + b^{(f)}) \tag{11.24}$$

$$g_t = \tanh(W^{(g)} * \chi_t + U^{(g)} * \mathcal{H}_{t-1} + b^{(g)}) \tag{11.25}$$

$$i_t = \sigma(W^{(i)} * \chi_t + U^{(i)} * \mathcal{H}_{t-1} + V^{(i)} \cdot \mathcal{C}_{t-1} + b^{(i)}) \tag{11.26}$$

$$o_t = \sigma(W^{(o)} * \chi_t + U^{(o)} * \mathcal{H}_{t-1} + V^{(o)} \cdot \mathcal{C}_t + b^{(o)}) \tag{11.27}$$

$$\mathcal{C}_t = \tilde{\mathcal{C}}_t \bigodot i_t + f_t \bigodot \mathcal{C}_{t-1} \tag{11.28}$$

$$\mathcal{H}_t = o_t \bigodot \tanh(\mathcal{C}_t), \tag{11.29}$$

11.2.5 Phased LSTM

A phased LSTM (Fig. 11.6), is a LSTM with opening and closing *timing gate* k_t. In this configuration, a cell memory state C_t and output h_t can be updated only when

gates k_t are in an 'open phase'. The phase is controlled by independent oscillation with three basic parameters: τ the real-time period of the oscillation, r_{on} the ratio of an 'open' phase duration to the whole period, and s the phase shift of the oscillation. Phased LSTM is suitable for processing asynchronously sampled input data [8].

Example

To illustrate the forward and backpropagation through time in LSTM let's consider a cell with two hidden units. The input data at timesteps t_0 and t_1 are as follow:

$$\boldsymbol{x_0} = \begin{bmatrix} 0.25 \\ 0.30 \end{bmatrix} \text{with label} \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix},$$

$$\boldsymbol{x_1} = \begin{bmatrix} 1 \\ 0.80 \end{bmatrix} \text{with label} \begin{bmatrix} 1.25 \\ 1 \end{bmatrix}.$$

Corresponding weight matrices values are provided below:

$$\boldsymbol{W^{(f)}} = \begin{bmatrix} 0.11 & 0.32 \\ 0.42 & 0.19 \end{bmatrix}, \ \boldsymbol{W^{(i)}} = \begin{bmatrix} 0.60 & 0.17 \\ 0.16 & 0.17 \end{bmatrix}, \ \boldsymbol{W^{(g)}} = \begin{bmatrix} 0.46 & 0.74 \\ 0.75 & 0.65 \end{bmatrix}, \ \boldsymbol{W^{(o)}} = \begin{bmatrix} 0.98 & 0.08 \\ 0.15 & 0.54 \end{bmatrix}$$

and hidden unit weight matrices:

$$\boldsymbol{U^{(f)}} = \begin{bmatrix} 0.87 & 0.50 \\ 0.23 & 0.67 \end{bmatrix}, \ \boldsymbol{U^{(i)}} = \begin{bmatrix} 0.30 & 0.89 \\ 0.64 & 0.65 \end{bmatrix}, \ \boldsymbol{U^{(g)}} = \begin{bmatrix} 0.60 & 0.12 \\ 1.00 & 0.01 \end{bmatrix}, \ \boldsymbol{U^{(o)}} = \begin{bmatrix} 0.41 & 0.62 \\ 0.62 & 0.14 \end{bmatrix}$$

$$\boldsymbol{b^{(f)}} = \begin{bmatrix} 0.30 & 0.1 \end{bmatrix}, \ \boldsymbol{b^{(i)}} = \begin{bmatrix} 0.67 & 0.13 \end{bmatrix}, \ \boldsymbol{b^{(g)}} = \begin{bmatrix} 0.47 & 0.07 \end{bmatrix}, \ \boldsymbol{b^{(o)}} = \begin{bmatrix} 0.75 & 0.09 \end{bmatrix}$$

Forward Propagation

1. First of all, let's calculate gates at t_0 using Eqs. (11.6)–(11.9):

$$\boldsymbol{g_0} = \tanh\left(\begin{bmatrix} 0.46 & 0.75 \\ 0.74 & 0.65 \end{bmatrix} \cdot \begin{bmatrix} 0.25 \\ 0.3 \end{bmatrix} + \begin{bmatrix} 0.6 & 1.00 \\ 0.12 & 0.01 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.47 & 0.07 \end{bmatrix}\right) = \begin{bmatrix} 0.66959 \\ 0.42190 \end{bmatrix};$$

or

$$\boldsymbol{g_0} = \tanh\left(\begin{bmatrix} 0.46 & 0.75 & 0.6 & 1.00 & 0.47 \\ 0.74 & 0.65 & 0.12 & 0.01 & 0.07 \end{bmatrix} \cdot \begin{bmatrix} 0.25 \\ 0.3 \\ 0 \\ 0 \\ 1 \end{bmatrix}\right) = \tanh\left(\begin{bmatrix} 0.81 \\ 0.45 \end{bmatrix}\right) = \begin{bmatrix} 0.66959 \\ 0.42190 \end{bmatrix};$$

and similarly $\boldsymbol{f_0} = \begin{bmatrix}0.61147\\0.55897\end{bmatrix}$; $\boldsymbol{i_0} = \begin{bmatrix}0.70432\\0.55564\end{bmatrix}$; $\boldsymbol{o_0} = \begin{bmatrix}0.73885\\0.56758\end{bmatrix}$.

2. Since there is no C_{t-1}, a memory state at time t_0 is $\boldsymbol{C_0} = \begin{bmatrix}0.47161\\0.23442\end{bmatrix}$.
3. Eventually, a cell output at t_0 is $\boldsymbol{h_0} = \begin{bmatrix}0.32472\\0.13067\end{bmatrix}$.
4. After repeating steps 1–4 for a timestep t_1, $\boldsymbol{f_1} = \begin{bmatrix}0.74248\\0.69481\end{bmatrix}$; $\boldsymbol{i_1} = \begin{bmatrix}0.82911\\0.69219\end{bmatrix}$; $\boldsymbol{o_1} = \begin{bmatrix}0.88740\\0.69463\end{bmatrix}$; $\boldsymbol{g_1} = \begin{bmatrix}0.95231\\0.87876\end{bmatrix}$.

   Consequently, $\boldsymbol{C_1} = \begin{bmatrix}1.13973\\0.77115\end{bmatrix}$. The output of a cell is $\boldsymbol{h_1} = \begin{bmatrix}0.72263\\0.44984\end{bmatrix}$.

Back Propagation

1. Delta at backpropagating for a timestep t_1 using Eqs. (11.12)–(11.17):

$$\delta \boldsymbol{h_1} = \left(\begin{bmatrix}0.72263\\0.44984\end{bmatrix} - \begin{bmatrix}1.25\\1\end{bmatrix}\right) + \begin{bmatrix}0\\0\end{bmatrix} = \begin{bmatrix}-0.5274\\-0.5502\end{bmatrix};$$

$$\delta \boldsymbol{C_1} = \begin{bmatrix}-0.5274\\-0.5502\end{bmatrix} \odot \begin{bmatrix}0.88740\\0.69463\end{bmatrix} \odot \left(\begin{bmatrix}1-0.81432^2\\1-0.64760^2\end{bmatrix}\right) + 0 \odot 0 = \begin{bmatrix}-0.1577\\-0.2219\end{bmatrix};$$

$$\delta \boldsymbol{g_1} = \begin{bmatrix}-0.1577\\-0.2219\end{bmatrix} \odot \begin{bmatrix}0.82911\\0.69219\end{bmatrix} \odot \left(\begin{bmatrix}1-0.9068^2\\1-0.7722^2\end{bmatrix}\right) = \begin{bmatrix}-0.01218\\-0.03498\end{bmatrix};$$

$$\delta \boldsymbol{i_1} = \begin{bmatrix}-0.1577\\-0.2219\end{bmatrix} \odot \begin{bmatrix}0.95231\\0.87876\end{bmatrix} \odot \left(\begin{bmatrix}1-0.82911\\1-0.69219\end{bmatrix}\right) = \begin{bmatrix}-0.02567\\-0.06005\end{bmatrix};$$

$$\delta \boldsymbol{f_1} = \begin{bmatrix}-0.1577\\-0.2219\end{bmatrix} \odot \begin{bmatrix}0.47161\\0.23442\end{bmatrix} \odot \begin{bmatrix}0.74248\\0.69481\end{bmatrix} \odot \left(\begin{bmatrix}1-0.74248\\1-0.69481\end{bmatrix}\right) = \begin{bmatrix}-0.01422\\-0.01103\end{bmatrix};$$

$$\delta \boldsymbol{o_1} = \begin{bmatrix}-0.5274\\-0.5502\end{bmatrix} \odot \begin{bmatrix}0.81432\\0.64760\end{bmatrix} \odot \begin{bmatrix}0.88740\\0.69463\end{bmatrix} \left(\begin{bmatrix}1-0.88740\\1-0.69463\end{bmatrix}\right) = \begin{bmatrix}-0.04292\\-0.07558\end{bmatrix}.$$

$$\delta \boldsymbol{x_1} = \begin{bmatrix}-0.09810\\-0.07964\end{bmatrix};$$

$$\Delta \boldsymbol{h_0} = \begin{bmatrix}-0.16145\\-0.12819\end{bmatrix}.$$

2. The same steps were taken to identify deltas for t_t:

$$\delta \boldsymbol{h_0} = \left(\begin{bmatrix} 0.32472 \\ 0.13067 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix} \right) + \begin{bmatrix} -0.21041 \\ -0.12374 \end{bmatrix} = \begin{bmatrix} -0.33673 \\ -0.29752 \end{bmatrix};$$

$$\delta \boldsymbol{C_0} = \begin{bmatrix} -0.31781 \\ -0.31409 \end{bmatrix};\ \delta \boldsymbol{g_0} = \begin{bmatrix} -0.12348 \\ -0.14345 \end{bmatrix};\ \delta \boldsymbol{i_0} = \begin{bmatrix} -0.08948 \\ -0.12264 \end{bmatrix};\ \delta \boldsymbol{f_0} = 0;$$
$$\delta \boldsymbol{o_0} = \begin{bmatrix} -0.13132 \\ -0.04757 \end{bmatrix};\ \delta \boldsymbol{x_0} = \begin{bmatrix} -0.43607 \\ -0.16384 \end{bmatrix};\ \Delta \boldsymbol{h}_{-1} = \begin{bmatrix} -0.63738 \\ -0.62015 \end{bmatrix}.$$

3. Weight matrix deltas: $\delta \boldsymbol{W} = \begin{bmatrix} -0.0916 & -0.1051 \\ -0.1135 & -0.1222 \\ -0.0565 & -0.0576 \\ -0.0959 & -0.0911 \\ -0.0142 & -0.0113 \\ -0.0110 & -0.0088 \\ -0.0758 & -0.0737 \\ -0.0875 & -0.0747 \end{bmatrix};$

$$\delta \boldsymbol{U} = \begin{bmatrix} -0.0040 & -0.0016 \\ -0.0114 & -0.0046 \\ -0.0083 & -0.0034 \\ -0.0195 & -0.0078 \\ -0.0046 & -0.0019 \\ -0.0036 & -0.0014 \\ -0.0139 & -0.0056 \\ -0.0245 & -0.0099 \end{bmatrix};\ \delta \boldsymbol{b} = \begin{bmatrix} -0.3300 \\ -0.3491 \\ -0.1492 \\ -0.2035 \\ -0.0142 \\ -0.0110 \\ -0.1742 \\ -0.1231 \end{bmatrix}.$$

4. Eventually, after applying one iteration of Stochastic Gradient Descent (SGD) with learning rate $\lambda = 0.1$ (Eq. 11.20), new weight values are:

$$\boldsymbol{W^{(g)}} = \begin{bmatrix} 0.4614 & 0.7411 \\ 0.7011 & 0.6009 \end{bmatrix};\ \boldsymbol{W^{(i)}} = \begin{bmatrix} 0.6092 & 0.1814 \\ 0.1105 & 0.1122 \end{bmatrix};\ \boldsymbol{W^{(f)}} = \begin{bmatrix} 0.1157 & 0.3296 \\ 0.4058 & 0.1091 \end{bmatrix};\ \boldsymbol{W^{(o)}} = \begin{bmatrix} 0.9876 & 0.0887 \\ 0.1074 & 0.5075 \end{bmatrix};$$

$$\boldsymbol{U^{(g)}} = \begin{bmatrix} 0.6005 & 0.1204 \\ 1.0002 & 0.0101 \end{bmatrix};\ \boldsymbol{U^{(i)}} = \begin{bmatrix} 0.3004 & 0.8911 \\ 0.6402 & 0.6505 \end{bmatrix};\ \boldsymbol{U^{(f)}} = \begin{bmatrix} 0.8708 & 0.5019 \\ 0.2303 & 0.6708 \end{bmatrix};\ \boldsymbol{U^{(o)}} = \begin{bmatrix} 0.4114 & 0.6225 \\ 0.6206 & 0.1409 \end{bmatrix};$$

$$\boldsymbol{b^{(g)}} = \begin{bmatrix} 0.4714 & 0.0711 \end{bmatrix};\ \boldsymbol{b^{(i)}} = \begin{bmatrix} 0.7030 & 0.1649 \end{bmatrix};\ \boldsymbol{b^{(f)}} = \begin{bmatrix} 0.3149 & 0.1203 \end{bmatrix};\ \boldsymbol{b^{(o)}} = \begin{bmatrix} 0.7674 & 0.1023 \end{bmatrix}.$$

11.3 LSTM Neural Network Configurations

LSTM is a competitive neural network that is acknowledged for performing a variety of tasks being processed by modern computers. The most common applications using LSTM are a prediction, classification, recognition, machine translation, and sequence generation. Input and output sequence lengths affect LSTM connectivity and network topology. Additionally, LSTM neural networks can have different dimensionality, directionality and their combination (e.g., multidimensional and multidirectional LSTM). The forward propagation in LSTM is illustrated in Figs. 11.7 and 11.8.

11.3.1 LSTM Models

Depending on the problem to be solved, LSTM units can be mapped into various topologies. Particularly, a number of timesteps define models as 'one-to-one', 'one-to-many', 'many-to-one' and 'many-to-many'. Figure 11.9 illustrates schematics of basic models. In a *'one-to-one'* model, LSTM unit does not unroll with time. This configuration is successful in dealing with classification tasks. An output of *'one-to-*

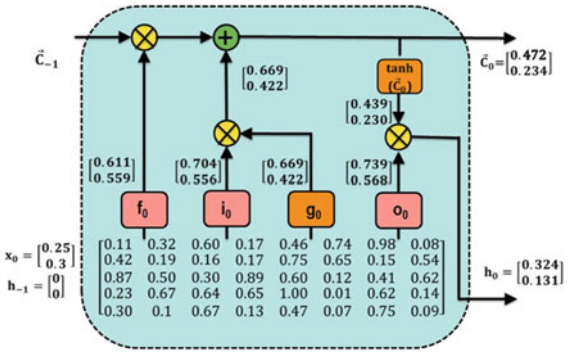

Fig. 11.7 Illustration of forward propagation in LSTM at t_1

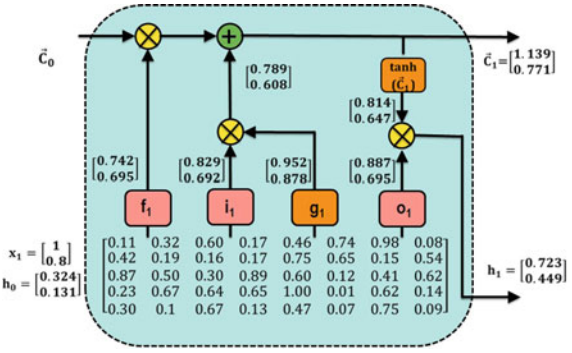

Fig. 11.8 Illustration of forward propagation in LSTM at t_0

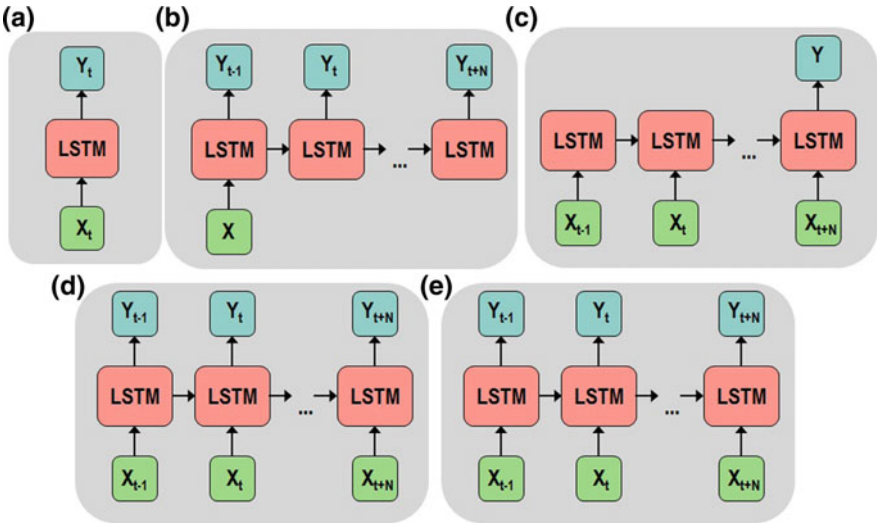

Fig. 11.9 STM models: **a** One-to-One; **b** One-to-Many; **c** Many-to-One; **d** Many-to-Many **e** Many-to-Many

many' configuration is a sequence. This allows generation of a sequential description of input data. *'Many-to-one'* can be used to classify video and audio materials or to make a prediction of following data. *'Many-to-many'* model is widely used in machine translation tasks when both input and output are sequences [9].

11.3.2 Multilayer LSTM

Multilayer LSTM architectures can build higher-level representations of input data. They can be created by stacking multiple LSTM hidden layers on the top of each other.

Stacked LSTM or *Deep LSTM* is a particular type of hierarchical LSTM. It is comprised of multiple LSTM layers with multiple hidden units. *Tree-LSTM* (Fig. 11.10b) is a hierarchic architecture where a memory cell can reflect information of multiple child cells and multiple descendant cells. Because of the structure, it outperforms existing systems in semantically related problems and sentiment classification.

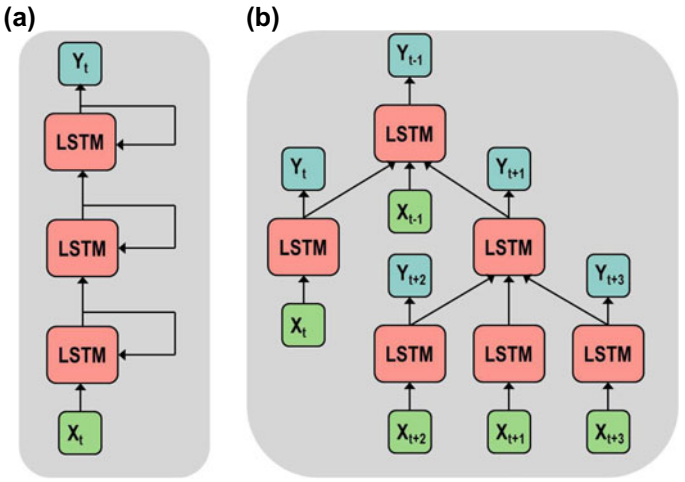

Fig. 11.10 **a** Stacked LSTM and **b** Tree-structured LSTM neural networks

11.3.3 Multidimensional and Multidirectional LSTM

In multidimensional LSTM, also called a 'spatial' LSTM, a hidden layer at a timestep t receives input $x_t$and the n previous activations f_t along all dimensions. This allows processing multidimensional data without scaling into one-dimensional form [13].

Along with multiple dimensions, LSTM cells can have different directionality. Most commonly used LSTM network architecture is a repeating linear chain. It is formed from input, output and (fully) self-connected hidden layers. The chain-structured *unidirectional* LSTM (Fig. 11.11a) has proven to be powerful in speech recognition and machine translation problems. The idea of bidirectional recurrent neural networks (BRNN) was proposed in 1997 by Schuster et al. [10]. This architecture is efficient for processing data of fixed length. It consists of two hidden layers, both connected to input and output. Unlike in unidirectional LSTM, the flow of information in layers of bidirectional LSTM are opposite to each other. Combination of BRNN with LSTM results in Bidirectional LSTM (BiLSTM) in Fig. 11.11b. BiLSTM is utilized in image and speech recognition problems and outperforms Deep Neural networks and GMM methods [11].

Often real time-series data suffers from missing measurements. There are three popular methods utilized for missing data estimation - interpolation, imputation, and matrix completion. Multidirectional LSTM allows a simultaneous combination of interpolation and imputation approaches to process dataset of N arrays of data [12].

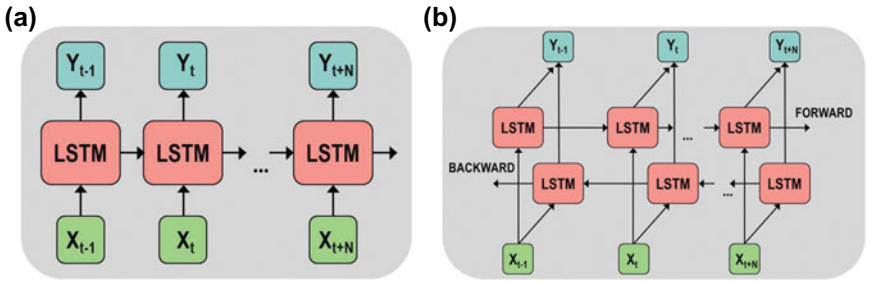

Fig. 11.11 **a** Unidirectional and **b** bidirectional LSTM neural networks

11.4 Summary

Long Short-term memory is an advanced variant of recurrent neural networks that do not suffer from vanishing or exploding gradient problems during training. Therefore it is recognized as a state-of-the-art approach for processing sequential data.

There are several LSTM cell architectures, and 'Vanilla' LSTM configuration is found to be the most popular among them. Multigated LSTM cell can be mapped into various models which allows processing sequences of arbitrary length. Also, LSTM neural networks can have different dimensionality, directionality, and a combination of them. Multidimensional and multidirectional LSTM neural networks are powerful in the implementation of different tasks including prediction, classification, recognition, machine translation, and sequence generation.

Chapter Highlights

- *Long short-term memory (LSTM)* is a special type of recurrent neural network (RNN).
- LSTM unit has a memory and multiple weighted gates. Therefore it does not suffer from vanishing or exploding gradient problems of RNN and can process sequences of arbitrary length.
- An *original LSTM* unit has no forget gate (NFG).
- Traditional LSTM configuration:

$$\begin{pmatrix} g_t \\ i_t \\ f_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} \cdot \begin{pmatrix} W^{(g)} & U^{(g)} \\ W^{(i)} & U^{(i)} \\ W^{(f)} & U^{(f)} \\ W^{(o)} & U^{(o)} \end{pmatrix} \cdot \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix};$$

$$C_t = f_t \bigodot C_{t-1} + i_t \bigodot g_t; \qquad h_t = o_t \bigodot \tanh(C_t).$$

- Traditional *LSTM with peephole connections* is distinguished for precise timing and often referred as 'Vanilla' LSTM.
- *ConvLSTM* are effective in spatiotemporal sequential problems.
- Updates in *Phased LSTM* occur at irregularly sampled time points t_j which can be controlled.
- Depending on input and output sequnces' length, following LSTM models are differentiated: 'One-to-One', 'One-to-Many', 'Many-to-One', 'Many-to-Many'.
- LSTM architecture can have different directionality, dimensionality and combination of them.

References

1. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386
2. Lipton ZC, Berkowitz J, Elkan C (2015) A critical review of recurrent neural networks for sequence learning. arXiv:1506.00019
3. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
4. Gers FA, Schmidhuber J, Cummins F (1999) Learning to forget: Continual prediction with LSTM
5. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2017) LSTM: a search space odyssey. IEEE Trans Neural Netw Learn Syst 28(10):2222–2232
6. Gomez, A. (2016). Backpropogating an LSTM: A Numerical Example. Aidan Gomez blog at Medium
7. Xingjian SHI, Chen Z, Wang H, Yeung DY, Wong WK, Woo WC (2015) Convolutional LSTM network: a machine learning approach for precipitation now casting. In: Advances in neural information processing systems, pp 802–810
8. Neil D, Pfeiffer M, Liu SC (2016) Phased lstm: accelerating recurrent network training for long or event-based sequences. In: Advances in Neural Information Processing Systems, pp 3882–3890
9. Karpathy A (2015) The unreasonable effectiveness of recurrent neural networks. Andrej Karpathy blog
10. Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Signal Process 45(11):2673–2681
11. Graves A, Jaitly N, Mohamed AR (2013) Hybrid speech recognition with deep bidirectional LSTM. In: 2013 IEEE workshop on automatic speech recognition and understanding (ASRU). IEEE, pp 273–278
12. Yoon J, Zame WR, van der Schaar M (2017) Multi-directional recurrent neural networks: a novel method for estimating missing data
13. Graves A, Schmidhuber J (2009) Offline handwriting recognition with multidimensional recurrent neural networks. In: Advances in neural information processing systems, pp 545–552

Chapter 12
Memristive LSTM Architectures

Kazybek Adam, Kamilya Smagulova and Alex Pappachen James

Abstract Mainstream standard LSTM architecture that is currently used in Tensorflow library does not use the original architecture. In fact, there are many different architectures of LSTM. One of the more widely used architectures of LSTM is Coupled Input and Forget Gate (CIFG). It is known more as Gated Recurrent Units (GRU). This chapter will introduce the existing architectures of LSTM. Further it will present memristive LSTM architecture implementation in analog hardware. The implementation realizes the standard version of LSTM architecture. Other architecture variations can be easily constructed by rearranging, adding, and deleting the existing analog circuit parts; and adding extra crossbar rows.

12.1 Vanilla LSTM

Hochreiter and Schmidhuber [1] invented the very first LSTM. Later the original LSTM evolved into the most common architecture [2] known as vanilla LSTM [3]. Gers et al. [4] and Gers and Schmidhuber [5] made the modifications that resulted in the vanilla LSTM which started to use full gradient training.

Figure 12.1 shows the detailed diagram of the vanilla LSTM cell. Vanilla LSTM differs from the standard LSTM used in Tensorflow by additional weighted connections known as peepholes. Peephole weight vectors; and half-transparent dashed lines and math elements (multiplication and addition) are extra additions that separate the standard tensorflow LSTM from the vanilla LSTM. All the lines in the diagram represent vectors. Blue line is the concatenation of following inputs along column axis: input vector (or feature vector) $\boldsymbol{x_t}$ of size $1 \times N$, output vector from previous cell $\boldsymbol{h_{t-1}}$ of size $1 \times M$, and a scalar bias value. Then the size of the blue line is $N + M + 1 = S$ (denoted as S for short). The matrices inside Vector Matrix Multiplication (VMM)

K. Adam · K. Smagulova · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: alex.james@nu.edu.kz

K. Adam
e-mail: kazybek.adam@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_12

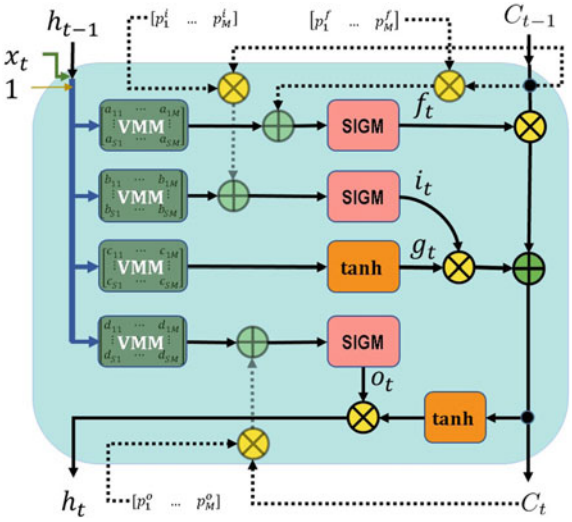

Fig. 12.1 Detailed Vanilla LSTM cell diagram. Letters subscripted with t or $t-1$ represent vectors of size M, except x_t which have size N. Bias is a scalar which is equal to unity

units represent weight matrices. They combine input, recurrent, and bias weights. The outputs of VMM units are row vectors of length M. All the black lines have size $1 \times M$, where M is the number of LSTM hidden units or blocks. The hidden units refer to the number of neurons in an LSTM layer. Therefore, ideally, we should get M parallel operations and M outputs at each stage in the diagram. However, in hardware parallel operations may be implemented sequentially to save up a chip area. Then, as expected, multiplication elements perform element-wise (Hadamard) multiplications, and addition elements perform vector additions.

As a recap, a number of neurons (hidden units) in an LSTM layer is equal to the number of cells. An LSTM cell is analogous to a layer in Feed-forward Neural Networks (FNNs) when unrolling the LSTM layer through time. That is, number of time steps is equivalent to the number of cell unrollings. It is important to understand the hidden unit size and look-back size (number of cells) correctly in order to implement LSTM in both software and hardware.

12.2 LSTM Architectures

There are total of nine existing architectures of LSTM [3]. They can be classified into six categories: (1) Vanilla; (2) Standard (No Peepholes); (3) Full Gate Recurrence (FGR); (4) Coupled Input and Forget Gate (CIFG); (5) with a Linear Activation Function; and (6) with a Constant Gate of Unity.

Vanilla is shown in Fig. 12.1 and can be described mathematically below [3], where $\boldsymbol{Q}$ having size of $S \times M$ is the concatenation of input weight matrix $\boldsymbol{W}$, recurrent weight matrix $\boldsymbol{R}$, and bias weight vector $\boldsymbol{b}$ along row axis; and $\boldsymbol{q}$ is the input vector

of size $1 \times S$:

$$\boldsymbol{f_t} = \sigma(\boldsymbol{q_t Q_f} + \boldsymbol{C_{t-1}} \bigodot \boldsymbol{p_f}) \tag{12.1}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{q_t Q_i} + \boldsymbol{C_{t-1}} \bigodot \boldsymbol{p_i}) \tag{12.2}$$

$$\boldsymbol{g_t} = tanh(\boldsymbol{q_t Q_g}) \tag{12.3}$$

$$\boldsymbol{C_t} = \boldsymbol{i_t} \bigodot \boldsymbol{g_t} + \boldsymbol{f_t} \bigodot \boldsymbol{C_{t-1}} \tag{12.4}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{q_t Q_o} + \boldsymbol{C_t} \bigodot \boldsymbol{p_o}) \tag{12.5}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \bigodot \boldsymbol{tanh}(\boldsymbol{C_t}), \tag{12.6}$$

No peepholes version is self-explanatory. It has the same architecture as that of the vanilla LSTM except for no peephole connections in its topology.

FGR is the most complex architecture of LSTM. As its name suggests, Full Gate Recurrence LSTM is featured by recurrent connections between its all gates. FGR is basically vanilla LSTM (except peephole weights of output gate element-wise multiplied to previous cell state vector) plus the new recurrent connections among the gates. It adds to the weight matrix $\boldsymbol{Q}$ of each gate additional $3M \times M$ recurrent weight matrix. Mathematically in detail, it can be described as following [3]:

$$\boldsymbol{f_t} = \sigma(\boldsymbol{q_t Q_f} + \boldsymbol{i_{t-1} R_{if}} + \boldsymbol{f_{t-1} R_{ff}} + \boldsymbol{o_{t-1} R_{of}} + \boldsymbol{C_{t-1}} \bigodot \boldsymbol{p_f}) \tag{12.7}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{q_t Q_i} + \boldsymbol{i_{t-1} R_{ii}} + \boldsymbol{f_{t-1} R_{fi}} + \boldsymbol{o_{t-1} R_{oi}} + \boldsymbol{C_{t-1}} \bigodot \boldsymbol{p_i}) \tag{12.8}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{q_t Q_o} + \boldsymbol{i_{t-1} R_{io}} + \boldsymbol{f_{t-1} R_{fo}} + \boldsymbol{o_{t-1} R_{oo}} + \boldsymbol{C_{t-1}} \bigodot \boldsymbol{p_o}) \tag{12.9}$$

CIFG architecture of LSTM is more known as GRU [6]. Again, its name (Coupled Input and Forget Gate) explains itself: $\boldsymbol{f_t = 1 - i_t}$. However, other than that, (1) there are also no peephole connections and no output activation function; (2) candidate cell state's recurrent inputs are filtered through output gate before being multiplied with recurrent weight matrix; and (3) cell state and cell output are combined together. The differences can be easier to see in mathematical equations:

$$\boldsymbol{g_t} = tanh(\boldsymbol{x_t W_g} + (\boldsymbol{h_{t-1}} \bigodot \boldsymbol{o_t}) \boldsymbol{R_g} + \boldsymbol{b_g}) \tag{12.10}$$

$$\boldsymbol{h_t} = (\boldsymbol{1 - f_t}) \bigodot \boldsymbol{g_t} + \boldsymbol{f_t} \bigodot \boldsymbol{h_{t-1}}, \tag{12.11}$$

The next category falls into LSTM architectures with a linear activation function either in (a) candidate cell state generation stage or (b) in output gate stage. From

the vanilla LSTM equations only Eqs. 12.3 and 12.6 change. In architecture (a), known as No Input Activation Function (NIAF), Eq. 12.3 becomes $\boldsymbol{g_t} = (\boldsymbol{q_t}\, \boldsymbol{Q_g})$. In architecture (b), known as No Output Activation Function (NOAF), Eq. 12.6 becomes $\boldsymbol{h_t} = \boldsymbol{o_t} \bigodot \boldsymbol{C_t}$.

The last category includes LSTM architectures which have a constant gate of unity in one of its gates, and everything else is the same as in the vanilla LSTM. Then there are three different such architectures: (a) No Input Gate (NIG): $\boldsymbol{i_t} = \mathbf{1}$; (b) No Forget Gate (NFG): $\boldsymbol{f_t} = \mathbf{1}$; and (c) No Output Gate (NOG): $\boldsymbol{o_t} = \mathbf{1}$.

12.3 Memristive Standard LSTM

In this section, we will transition from equations to the hardware implementation of LSTM using memristive crossbar circuits. Memristors will represent the weights in the equations, and the accumulated current at each column of the memristive crossbar will represent VMM operation. The rest of the circuitry includes analog hardware implementations of addition, multiplication, and activation functions. Also, analog storage elements are used to store recurrent values for the next time-step operations.

In literature works, there exists memristive LSTM architectures and can be found in [7–12]. All of them implement standard LSTM architecture - no peepholes LSTM. Architectures from [7, 11, 12] mainly use voltage-based circuits. Whereas, [8] proposes current-based crossbar architecture. TSMC 0.18 μm process technology was used in all of the above works.

Since [7, 8] does not show results for system-level simulations, the details about their circuit components were not presented here. However, their details can be found in [13]. Since the work by [9] is not done in circuit simulator; and the work by [10] is not fully analog, they do not have much circuit elements to be presented here.

12.3.1 Initial Work on Memristive LSTM

In [7], purely analog implementation of memristive LSTM in 0.18 μm CMOS technology is proposed. However, this work does not provide a full circuit simulation of the whole system solving a particular machine learning problem. They propose only the separate analog building blocks of the whole system. Particularly, analog circuits for activation function and element-wise (Hadamard) multiplication operation are proposed. In addition, an existing crossbar configuration is presented as a possible implementation of a memristive crossbar circuit for use with the rest of the circuit.

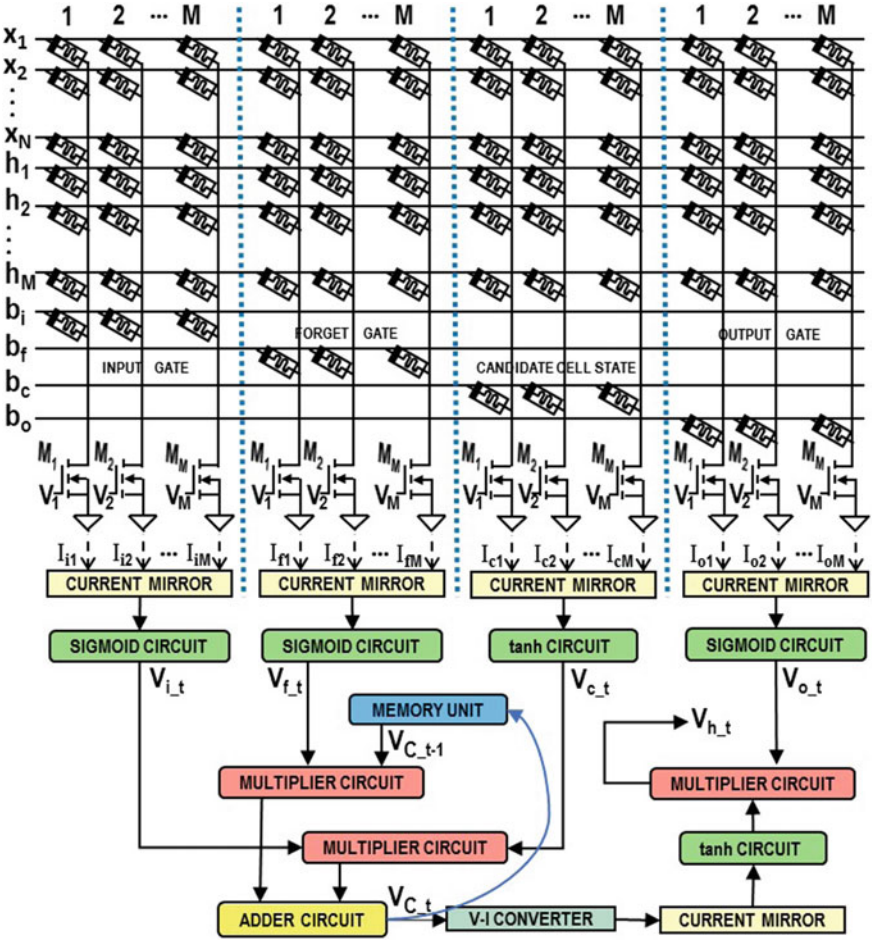

Fig. 12.2 Current-based memristive LSTM architecture. Lines represent scalar values, not vectors [8]

12.3.2 Current-Based Memristive LSTM

As an attempt to put the parts of LSTM together and solve a real-world problem, a current-based LSTM architecture is proposed by [8]. The current-based architecture can be seen in Fig. 12.2. The blue dashed line delimits the four main parts of the LSTM architecture: input gate, forget gate, candidate cell state, and the output gate. M and N matches with our previous notations for the number of hidden units and number of inputs (features), respectively. The architecture is designed to be used in a sequential manner. It means that we need to run through M cycles to obtain hidden unit vector and cell state vector. Columns with the same indices are operated

in parallel so that we can compute corresponding hidden unit's output and cell state. NMOS transistors are used to switch on and off the columns. Voltages $V_1, \ldots, V_M$ are used to control the switches. Since this design uses current-based activation function circuits, it requires current mirrors that are used as buffers for isolating the effect of interaction. Further down, voltage-based multiplier and adder circuits are used in the design. For that reason, the voltage-to-current converter is used before passing the voltage representing cell state to the output activation function.

12.3.3 Approximate Memristive LSTM

As opposed to the work in [7], authors of [9] solve a real-world problem (language modeling problem) and accomplish a system level simulation. However, their simulations are done in their own built software tool (written in C++) rather than on circuit simulator such as SPICE. They use so-called non-linear function units (NLFs), which are digital blocks, to implement all the mathematical operations required to implement LSTM except for vector-matrix multiplication (since it is implemented using memristive crossbars). They set some constraints in the software to incorporate the non-idealities associated with the analog VMM operation. Their main finding is that memristors need to have a symmetric change in conductance values when given positive and negative voltages across for good performance results. The slightest variation as low as 2% in the asymmetry can severely affect the performance results in large LSTM networks with the sizes of up to 512 hidden units compared to fully connected networks with smaller size and smaller training dataset.

12.3.4 Memristive LSTM Chip

Finally, the work by [10] presents a fabricated chip consisting of one-transistor one-memristor (1T1R) type crossbar array that implements the VMM operation part of the LSTM algorithm. However, the rest of the operations were implemented in software in their work. They were able to train their weight matrix of memristors in-situ and solve successfully time-series prediction and classification problems. The used memristors were from Ta/HfO2.

12.3.5 Voltage-Based Memristive LSTM

To fill the gaps from the previous works, fully functional and purely analog circuit-level simulation (using SPICE) of LSTM has been done in [11, 12]. In [11], time-series prediction and in [12], time-series classification problems are solved. Both of the works employ the same circuit architecture and circuit parts. The only difference

is the type of op-amps used in the designs. Two-stage op-amp from [13] was used in [11] and three-stage op-amp from [14] was used in [12]. The latter op-amp exhibited better accuracy when operating with small signals. This was the reason for the change. The whole system-level circuit can be divided into three main parts: Vector Matrix Multiplication; Activation Function; and Analog Multiplication circuit parts.

12.3.5.1 Vector Matrix Multiplication Circuit

The power of memristive crossbar circuits is the implementation of VMM operation in an efficient way. Op-amps are handy as always and provide virtual grounds for accumulating the currents in a crossbar column. In addition, they convert the accumulated current to voltage at its output. However, using single op-amp per column and single memristor per synapse restricts the range of implementable weights. That way we can only implement positive weights. The problem can be solved using two memristors per synapse and two op-amps per column in memristive crossbar [15]. The configuration of two op-amps in a crossbar is shown in Fig. 12.3. The first column op-amp acts as an inverting amplifier and the second column amplifier acts as a summing amplifier. This configuration enables to subtract the current in the second column from the first one. The subtracted current is then multiplied to Rf and results in an accurate voltage at the output of the second op-amp. The designs in [11, 12] use an only single pair of op-amps per single sub-crossbar (total four of them). This is due to the use of sequential mode operation in the design.

12.3.5.2 Activation Function Circuit

Both activation functions, sigmoid and hyperbolic tangent, used in LSTM equations can be implemented with the same circuit, but different parameters. This is because they have similar waveforms when plotted. Figure 12.4 shows the circuit design for implementing the activation functions. It is essentially a differential amplifier with shifted inputs and outputs. It has a property of giving output signals that follow sigmoidal shape in its DC transfer characteristics. Sigmoidal output shape can be achieved by sweeping the differential input of the amplifier. However, parameters need to be tuned to achieve accurate output results. Two summing op-amps and their parameters are used to shift input and output signals. Value of V_1 is also changed along with the shift tunings. As for the shape of the DC transfer characteristics, the slope and output range are modified by altering current I_1, supply voltage V_{dd}, and the sizes of NMOS transistors N_1 and N_2.

12.3.5.3 Analog Multiplication Circuit

Four-quadrant analog multiplier based on flipped voltage followers (FVFs) [16] was used in the implementation of Hadamard multiplication operation. The circuit

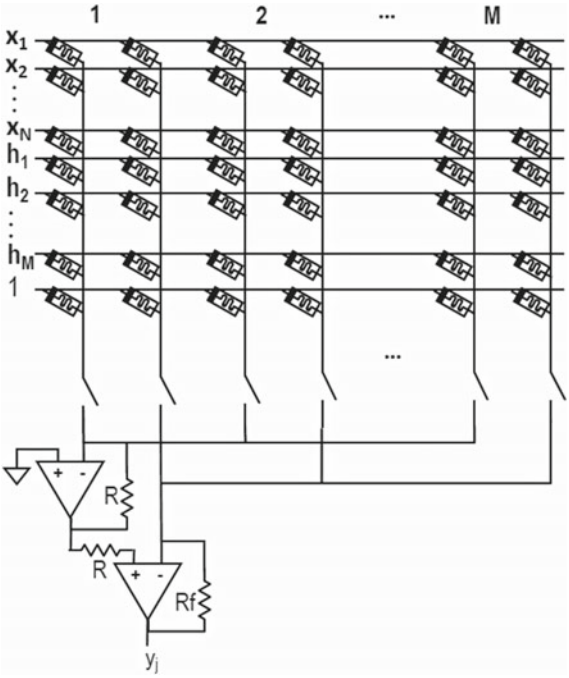

Fig. 12.3 Memristive crossbar circuit with two memristors representing single synapse. Switches represent pass logic units. They are used for implementing VMM operation in sequential manner. The whole circuit of standard LSTM would have four of these crossbars. Output voltage is read at node y_j

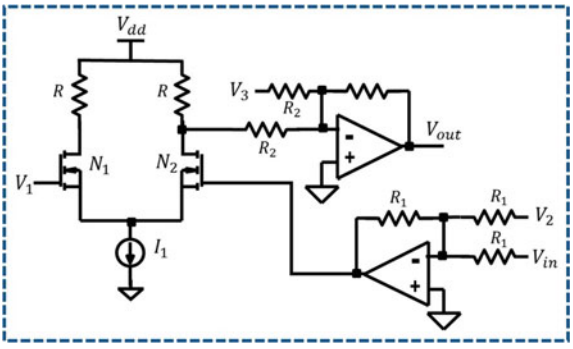

Fig. 12.4 Activation function circuit [11]. Sizes of transistors N_1 and N_2 are the same

schematic of FVF multiplier is shown in Fig. 12.5. The core of the multiplier consists of NMOS transistors $M1 - M4$. Current source I_b and transistors M_a and M_b form a FVF cell. The difference between currents I_E and I_F results in output current $I_{out} = I_E - I_F = \mu_n C_{ox} W/L$. This expression holds true only if $V_E = V_F$, all the transistors are in triode mode, and if the sources driving nodes A and B have significantly low impedance. These conditions can be met using FVFs in feedforward structure as current sensing elements and voltage buffers. The bottom FVFs create low impedance nodes E and F, and enable sensing the currents that pass through these nodes. Additionally, they are used to replicate these currents. The top FVFs

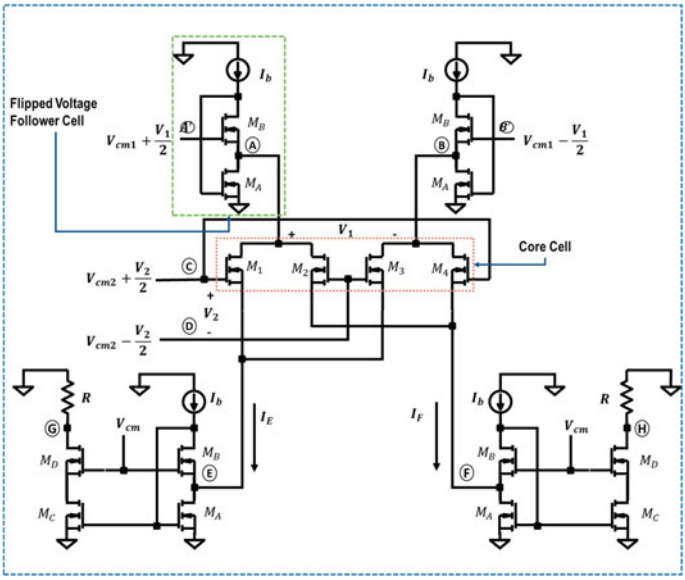

Fig. 12.5 Four-quadrant analog multiplier circuit [16]. All the transistors operate in linear region. $V_{cm} = V_{cm1} = V_{cm2} = 1.4\,\text{V}$, $I_b = 600\mu A$, $R = 2\,\text{k}\Omega$, $V_A - V_B = V_1$. Input range is between -0.4 and $0.4\,\text{V}$

implement very low impedance sources that drive nodes A and B. The resulting multiplier produces good linearity, operates at high frequencies, and has low supply (sub-volt) requirements in comparison with op-amp based multipliers.

12.4 Applications

LSTM implementation using analog CMOS-memristive circuits is slowly drawing interest. To date, a memristive LSTM circuit was validated for time-series prediction problem proposed by [17] in [11] and wafer quality classification described [18] and implemented in [12]. This section provides a brief overview of both of them.

12.4.1 Prediction

In the time-series forecasting, problem LSTM was used to predict a number of airplane passengers. The original dataset contained 144 observation points collected over 12 years. Prior to being processed by LSTM neural network, the dataset was divided into training and testing sets. Then the training set was converted into a

three-column form, where two first columns were used as input and the third column values served as a target.

For the task implementation, a two layer neural network represented in the (Fig. 12.6) was utilized. In the first layer, LSTM unit was unrolled for two timesteps and a many-to-one model was adopted. The second layer was formed of a feedforward neural network with linear activation function. Figure 12.7 illustrates a graphical representation of original dataset and predicted values. A software simulation of LSTM neural network using Python and hardware simulation was performed for the same set of weight matrix values. In Python simulation obtained MSE and RMSE values were 0.0101 and 0.1005 respectively. Close results were shown by hardware simulation where MSE = 0.0121 and RMSE = 0.1099. Area of the two-layer neural network is 108,599 μm^2 (includes sample and hold memory units). Maximum power dissipation of a single LSTM unit is 225.67 mW.

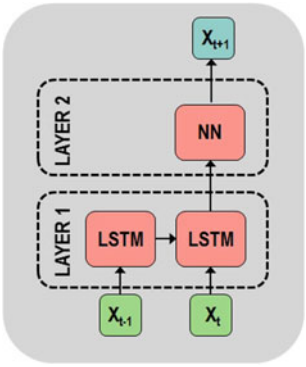

Fig. 12.6 Diagram of implementation of classification using LSTM [11]. The first layer is a many-to-one LSTM layer. The second layer consists of an artificial neural network with linear activation

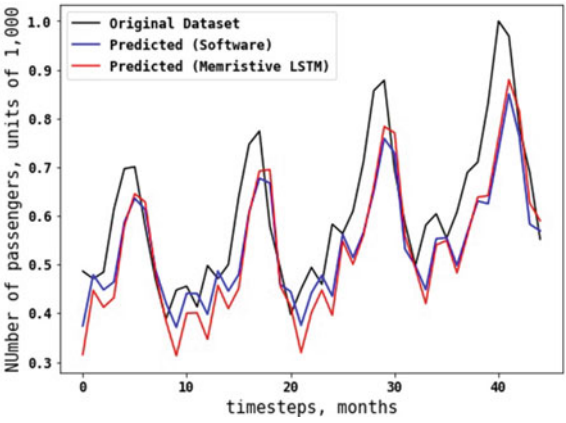

Fig. 12.7 International airline passengers: recorded dataset (black line); predictions with LSTM in Python Keras (blue line) and predictions using memristive LSTM (red line) [11]

12.4.2 Classification

Wafer quality inspection problem aimed to classify wafers into two classes: normal and abnormal. Each dataset contained 152 measurements per wafer of any class.

LSTM classification of wafers was performed using two different models represented in Fig. 12.8. In the first 'one-to-one' model (Fig. 12.8a), a single LSTM cell with one hidden unit and one timestep were utilized. The obtained accuracy reached 96.09%. The area of a cell is 115,967.4 μm^2 and power consumption is 312.4 mW. Higher classification accuracy of 98.51% was obtained with the second architecture (Fig. 12.8b). It consists of a two-layer neural network. The first layer is LSTM with four hidden units and one hundred fifty-two timesteps whereas the second layer is a feedforward neural network with linear activation. The utilized total area and

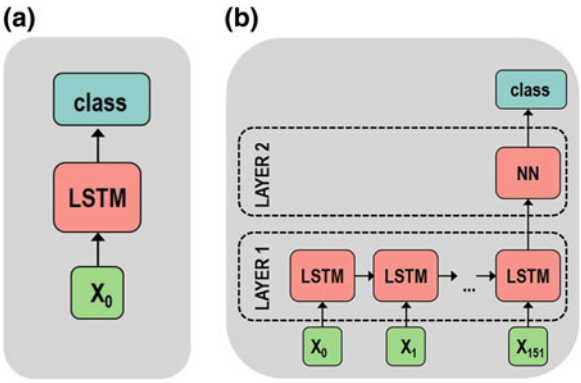

Fig. 12.8 Wafer quality classification using LSTM **a** A single LSTM layer (parallel: 1 hidden unit, 152 inputs, 1 time step;and **b** LSTM + ANN layer (sequential: 4 hidden units, 1 input, 152 time steps)

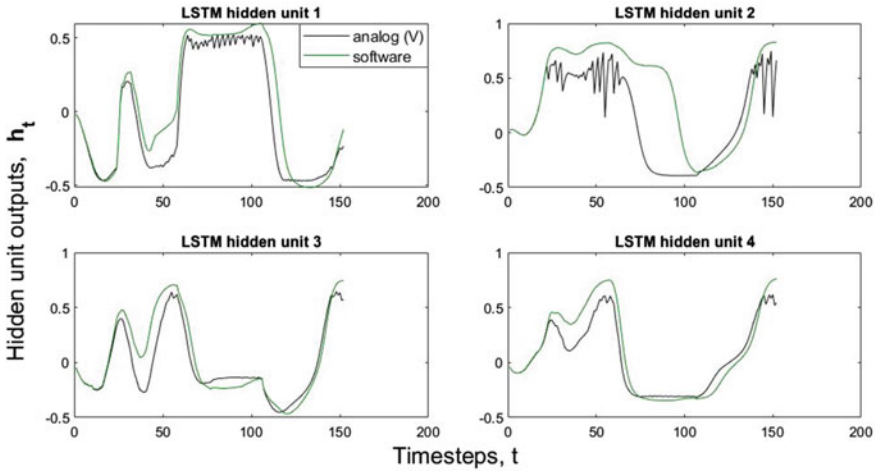

Fig. 12.9 Classification of the test wafer 23: a graphical representation of LSTM cell output values h_t during software (red line) and hardware (blue line) simulations [12]

power consumption are 257,503.20 μm^2 and 255.8 mW. Figure 12.9 demonstrates classification results of the Wafer 23.

12.5 Summary

LSTM neural networks demonstrate high accuracy during processing sequential datasets. Conventional LSTM accelerators include digital platforms such as FPGA, CPU, GPU, and ASIC. In the proposed CMOS-memristive circuits, multilevel memristor crossbars allow implementation of computationally expensive dot-product operation for weight-matrix multiplication within LSTM cell. The absence of leakage current and nanoscale size of memristors benefit in a small area and power efficiency.

Even though many LSTM architectures exist, all of them have a similar structure. Therefore, their implementation is easily realizable once the standard memristive LSTM is implemented in pure analog hardware on a chip. Implementation in analog hardware would further result in significant efficiency.

Chapter Highlights

- Long short-term memory (LSTM) unit operation can be split into several computational blocks.
- Weight-matrix multiplication in LSTM is implemented using a memristor crossbar array.
- Pointwise (Hadamard) multiplication and activation layer circuits are implemented using TSMC 180nm CMOS technology.
- Voltage-based memristive crossbar array wins over current based design due to higher accuracy.
- Memristive devices which exhibit symmetric behavior [9] would bring us closer to achieving the same performance metrics as in the software implementation of LSTM.
- The work by [10] is an example of applying real memristive crossbar for implementing LSTM. Most of the design uses digital building blocks to realize the hardware.

References

1. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
2. Graves A, Schmidhuber J (2005) Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw 18(5–6):602–610
3. Greff K, Srivastava RK, Koutník J et al (2017) LSTM: a search space Odyssey. IEEE Trans Neural Netw Learn Syst 28(10):2222–2232. https://doi.org/10.1109/TNNLS.2016.2582924
4. Gers FA, Schmidhuber J, Cummins F (1999) Learning to forget: continual prediction with LSTM. In: Proceedings of the 9th international conference on artificial neural networks (ICANN), vol 2. pp 850–855
5. Gers FA, Schmidhuber J (2000) Recurrent nets that time and count. In: Proceedings of IEEE-INNS-ENNS international joint conference on neural networks (IJCNN), vol 3. pp 189–194
6. Cho K, van Merrienboer B, Gulcehre C et al (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078
7. Smagulova K, Krestinskaya O, James AP (2018) Analog Integr Circuit Signal Process 95:467. https://doi.org/10.1007/s10470-018-1180-y
8. Smagulova K, Adam K, Krestinskaya O et al (2018) Design of CMOS-memristor circuits for LSTM architecture. In: 2018 IEEE international conference on electron devices and solid state circuits (EDSSC), 1 pp. https://doi.org/10.1109/EDSSC.2018.8487179
9. Gokmen T, Rasch MJ, Haensch W (2018) Training LSTM networks with resistive cross-point devices. Front Neurosci 12:745. https://doi.org/10.3389/fnins.2018.00745
10. Li C, Wang Z, Rao M et al (2018) Long short-term memory networks in memristor crossbars. https://doi.org/10.1038/s42256-018-0001-4
11. Adam K, Smagulova K, James AP (2018) Memristive LSTM network hardware architecture for time-series predictive modeling problem. arXiv:1809.03119
12. Adam K, Smagulova K, Krestinskaya O et al (2018) Wafer quality inspection using memristive LSTM, ANN, DNN and HTM. arXiv:1809.10438
13. Krestinskaya O, Salama K, James AP (2018) Learning in memristive neural network architectures using analog backpropagation circuits. IEEE Trans Circuits Syst I: Regul Pap. https://doi.org/10.1109/TCSI.2018.2866510
14. Saxena V, Baker RJ (2010) Indirect compensation techniques for three-stage fully-differential op-amps. In: 53rd IEEE international midwest symposium on circuits and systems, Seattle, WA, pp 588–591. https://doi.org/10.1109/MWSCAS.2010.5548896
15. Hasan R, Taha TM, Yakopcic C (2017) On-chip training of memristor based deep neural networks. In: International joint conference on neural networks (IJCNN), Anchorage, AK, pp 3527–3534. https://doi.org/10.1109/IJCNN.2017.7966300
16. Ramirez-Angulo J, Thoutam S, Lopez-Martin A et al (2004) Low-voltage CMOS analog four quadrant multiplier based on flipped voltage followers. In: 2004 IEEE international symposium on circuits and systems (IEEE Cat. No.04CH37512), Vancouver, BC, pp I–681. https://doi.org/10.1109/ISCAS.2004.1328286
17. Brownlee J (2016) Time series prediction with LSTM recurrent neural networks in python with keras. Available at: https://machinelearningmastery.com
18. Olszewski RT (2001) Generalized feature extraction for structural pattern recognition in time-series data (No. CMU-CS-01-108). Carnegie-Mellon University Pittsburgh PA School of Computer Science

Chapter 13
HTM Theory

Yeldos Dauletkhanuly, Olga Krestinskaya and Alex Pappachen James

Abstract This chapter presents the general background information about the Hierarchical Temporal Memory (HTM). HTM is a recently proposed cognitive learning algorithm that is intended to emulate the overall structural and functionality of the human neocortex responsible for the high-order functions such as cognition, learning and making predictions. The main properties of HTM is hierarchical structure, sparsity and modularity. HTM consists of two main parts: HTM Spatial Pooler (SP) and HTM Temporal Memory (TM). The HTM SP performs the encoding of the input data and produces sparse distributed representation (SDR) of the input pattern useful for visual data processing and classification tasks. The HTM TM detects the temporal changes in the input data and performs prediction making.

13.1 Introduction

Hierarchical Temporal Memory (HTM) is a machine learning algorithm proposed by Jeff Hawkins in "On Intelligence" book [4] and developed by Numenta Inc. in recent year [8]. In 2006, the first generation of the HTM algorithm called Zeta1, currently known as HTM-Zeta1, was implemented and three years later the modified version of the algorithm known as Cortical Learning Algorithm was introduced [7]. HTM algorithm is inspired from the structure and functionality of human neocortex, which has an ability to perform high level cognitive activities, such as language processing, object recognition, prediction making, etc [5]. HTM has a hierarchical structure. The main parts of HTM are HTM Spatial Pooler (SP) and HTM Temporal Memory (TM). The HTM SP produces sparse distributed representation (SDR) of

Y. Dauletkhanuly · O. Krestinskaya · A. P. James (✉)
Nazarbayev University, Kabanbay Batyr av. 53, Astana, Kazakhstan
e-mail: apj@ieee.org

Y. Dauletkhanuly
e-mail: yeldos.dauletkhanuly@nu.edu.kz

O. Krestinskaya
e-mail: ok@ieee.org

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_13

the input data, which ensures the encoding of the input pattern useful for visual data processing and classification tasks. The HTM TM detects the temporal changes in the input data in a particular time period and is able to perform the prediction task [12, 14, 15]. In recent years, several attempts to modify HTM algorithm and introduce the hardware implementation of HTM has been made [3, 10, 11, 13]; however HTM on-chip hardware implementation is still an open problem.

The main aim of this chapter is to provide in-depth information about main components, functionality and structure of HTM algorithm. The chapter focuses on HTM theory and briefly describes the applications, software and hardware implementations of HTM algorithm. Section 13.2 introduces the core principles and main concepts of HTM. Section 13.3 elaborates on the details about the main parts of the HTM SP followed by explanation of the mathematical framework of the SP algorithm. Section 13.4 focuses on the HTM TM and introduces the basic equations. Chapter 5 summarizes the applications of HTM and concludes the chapter.

13.2 HTM Overview

13.2.1 Inspiration from Biology

The HTM algorithm emulates the functionality of pyramidal neurons [19] in human neocortex [6, 7, 16]. Comparing to the traditional neuron of Artificial Neural Network (ANN), HTM has a great number of dendritic connections and the output of the HTM neuron depends not only feedforward inputs (proximal connections to the soma), but also feedback (apical connections) and lateral (distal connections) connections between the neurons. Figure 13.1 illustrates the structure of pyramidal neuron and HTM neuron model and connections.

The other main property of HTM inspired from the neocortex is a hierarchical structure of HTM. The hierarchical structure of HTM is illustrated in Fig. 13.2. HTM consists of several layers with certain regions. The smallest HTM unit is an HTM cell. Several HTM cell forms the HTM columns, in turn a set of mini-columns represent

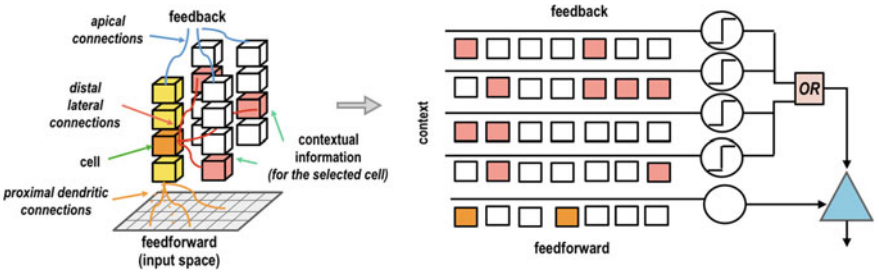

Fig. 13.1 HTM neuron and connections [7, 14]

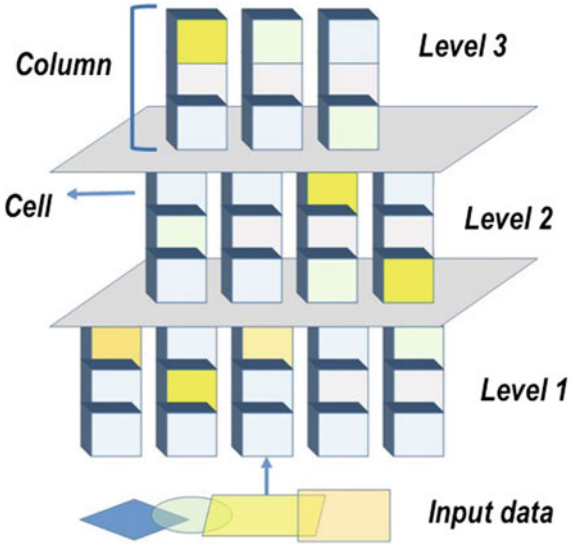

Fig. 13.2 Hierarchy in HTM with overlapping input data

Fig. 13.3 The process of the input data through the HTM algorithm [7]

the HTM region. The complexity of HTM and number of levels can vary depending on the application. Figure 13.2 illustrates 3 level HTM hierarchy, which show how the data is transformed to SDR. The example of geometrical figures illustrate how different input patterns are transformed to the activation of particular cells of the HTM regions. The cells representing the figures even with a similar shapes or similar colors is represented by similar HTM cells.

An end-to-end HTM system is shown in Fig. 13.3. This system consists of an encoder, the HTM SP, the HTM TM, and an SDR classifier [7]. The HTM SP transforms all input patterns to SDRs and the HTM TM learns temporal sequences of the SDRs from HTM SP and has an ability to make a prediction of the following input to the HTM system.

13.2.2 Sparse Distributed Representation (SDR)

Human brain represents the information in SDRs [9]. SDRs consists of a large number of 1 and 0 bits. At each point in time, only the small portion of these bits are 1s, representing active neurons in a human neucortex, while the rest are 0s, corresponding to inactive neurons. Therefore, SDRs represent the semantic attributes of the data. In

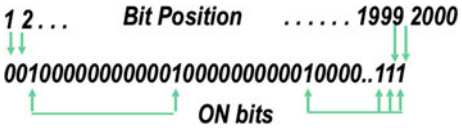

Fig. 13.4 The SDR vector with a few ON bits at a time [6, 9]

ASCII code, these are the vectors of binary numbers. The example of this concept is presented in Fig. 13.4 illustrating the case, when only 2% of bits representing the data are ON. This bit separately do not convey any meaning; however the combination of these bits transports a semantic meaning. The maim idea is to store the location of the active bit representing a particular data, rather then storing all bits, as in the dense representation of the information [9].

The other example of it is a representation and storage of a particular name. If one wants to represent a person's name, the each bit in the vector of SDR covers information related to this person's name. One bit will be responsible for the first letter of the name, another one for the second and so on. The combination of each these active bits formulate a particular name, while separately these bits do not have any meaning.

Figure 13.2 shows how the HTM SP converts the input space into SDR representation. Each mini-column contains several cells, these cells are activated based on a particular input. While the separate HTM cells do not have any particular meaning, the set of cells forms SDRs correspond to a particular geometrical figure. The overlap in SDR occurs, when two activated input vectors have the common ON bits [6]. It means that these different representations share the same attributes. The illustration of the overlap is shown in the Fig. 13.2 as an input data (geometrical figures). Each overlapping region shows that these figures have the common features. In ASCII code, this represents the case, when several representations have the ON bits at the same location.

The SDR should have a large capacity to represent the sufficient number of the combination for the given input vector. Such variation of the vector can be calculated using Eq. 13.1, where n represents the size of the vector and w is the cardinality [17].

$$\binom{n}{w} = \frac{n!}{w!(n-w)!} \tag{13.1}$$

For example, if these values are, $n = 512$ and $w = 2$, the possible combination is 130816 [17]. Increasing n and w will have the exponential increase in the output. The issue of the capacity is possible identity of random generated encoded information. To avoid this problem, the possible probability represented by Eq. 13.2 of the same output has to be very low.

$$P(a = b) = \frac{1}{\binom{n}{w}} \tag{13.2}$$

The calculation indicates that possibility of such combination is about zero for large n and w. The HTM algorithm uses non-exact matching instead exact one in

order to avoid the false repeated outputs in SDR representation. It states, if both SDRs have equal match threshold and cardinality, $\theta = w$, we define it as exact match, where $\theta < w$ indicates non-exact matching. By decreasing the θ and increasing the n will give the sufficient combinations with zero probability of false matches. Another property of the SDR is a union. The idea of the union is the ability to store several different sets in a static pattern. The union can be obtained using the simple Boolean logic OR (Fig. 13.1), which combines several sets and forms a new single pattern. The benefit of this property is its dynamic ability, representation of several patterns within a single set.

13.2.3 Encoder

The encoder is the procedure that converts input data of the HTM into the SDR format for the further processing [9]. Encoder determines which output bits should be represented as 1s, and which ones are 0s, capturing the semantic properties of the input data [18]. The process of the encoder is similar to that one in [20], where authors took cochlea as an example. Authors state that cochlea has a range of hair cells connected to the cells. And when sounds with different frequencies are applied to the cochlea, the set of the hair cells for the given frequency range will be in active state, thus sending the signal to the brain. Also, [20] states that cochlea of different animals responds to the different frequency ranges. This example shows that for different types of input data, the different encoders are required [9]. The encoding example of cochlea is shown in Fig. 13.5 [9, 18]. This encodings represents the numbers 7.0, 10.0 and 13.0. The adjacent encodings have overlap region, like Encoding 1 (representing 7.0) and Encoding 2 (representing 10.0), and no such sectors exist for Encoding 1 (representing 7.0) and Encoding 3 (representing 13.0). This means, that semantically close input data have the common ON bits, while non-matching data sets will not have an intersection area, which justifies the foreknown criteria.

Thus, when designing the encoders for distinctive data set the following criteria which are mentioned in [18], have to be considered. The first one is that input data with close meaning should have overlapping at the output. It means that two input sets with semantically identical property should have overlapping regions in SDR domain. The probability of having the same representation in SDR of input vector is about zero for sufficient n with tolerable sparsity. The cases with exactly the same SDRs are possible only for exactly matching inputs, which is the second criteria for designing encoder. The SDR always has the same dimension, number of bits, for all

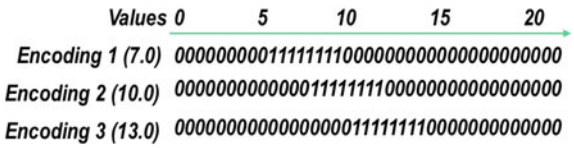

Fig. 13.5 The encoding process for different input data [9]

input data. Therefore, when designing the encoder, the output should have the equal number of bits for the entire data set. The key aspect for encoder architecture is to have enough ON bits to resist noise.

13.3 HTM Spatial Pooler

This section covers the basic idea of the HTM SP, different HTM SP phases, properties, and gives general introduction to the HTM SP algorithm. SP is a part of the HTM that continuously encodes the set of the input data into SDR representation [2]. SP has four different phases [14], some works consider them as three phases like in [21]. Before characterizing the main stages of HTM SP, it is necessary to define several terms:

- Receptive field: input space that is connected to the columns and cells.
- Permanence value: numerical value which signifies the state of the the particular synapse. If the value is below the threshold, the synapse is not connected and vice versa.
- Synapse: connection between cells within the columns.

The main idea of the HTM SP is illustrated in Fig. 13.6. Initialization is the first phase of the HTM SP, where random permanence values are assigned to the synapses. Usually, these random values are near to the threshold one due to the performance. Such method requires less iteration for synapses to be active and makes the process agile. After attaching scalar values, the potential inputs are set up. During initialization, several random columns, which receive the input data, withing the HTM region are selected (Fig. 13.6) [14]. In the initialization, the weights of synaptic connection are selected randomly. Next, the connection between the input space and the selected mini-columns is established, active connected synapses are determined.

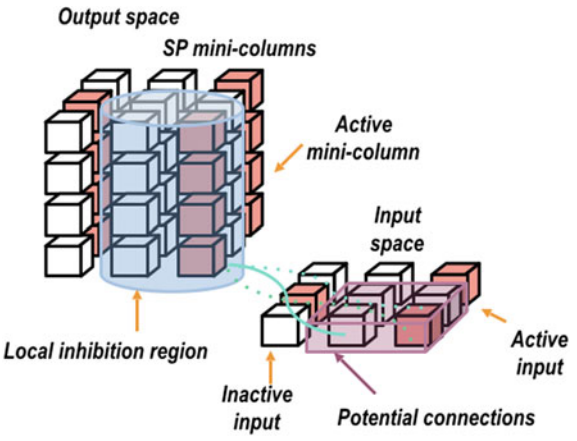

Fig. 13.6 HTM spatial Pooler [14]

The number of active connected synapses is determined by the calculation of the overlap value. The outputs are compared and the highest output is selected and activated, and the other are inhibited. This process is known as the inhibition stage. The last stage is the learning one, where the permanence values of the synapses and boosting factor are updated, and all the phases, except initialization, are repeated.

13.3.1 Properties of HTM Spatial Pooler

To tolerate different parasitic effects and to produce high reliable outputs SP has the following properties:

- SP has the same sparsity for all inputs to detect the patterns without any negative effects. Therefore, it ensures the SP to avoid the false positive errors. Already defined inhibition phase is a key property to have fixed sparsity, where winner takes all (WTA) approach is used.
- To have optimal learning outcomes, the SP system employs all resources. It means, all neurons have to respond to every set of inputs. However, [2] states that before learning process only 70% of neurons are acknowledged to the input data, while this value is about 100% after the learning process.
- Robust representation for inputs come periodically. In general, the HTM has to be tolerable to noise, subsampling and so on. Therefore, this property shows that inputs that occur frequently, even with noise, will be the same as inputs without this disturbance factor. The experiment done by [2] proves that after learning, SP has a robust representation with 40% of the input changed.
- If the input varies every time, it requires SP to be flexible for different data types. By continuous learning of the input, SP condones this variation. In the process, this is done by growing new set of synapses to adapt faster.

There are more properties of the SP in [2] with experimental results and discussion.

13.3.2 Mathematical Framework of HTM Spatial Pooler

The first step within the SP is a relationship of input and output, thus Eq. 13.1 represents the potential inputs. The term x_j shows jth input neuron, where x_i^c and γ represents the center of the cube and the length of the endge respectively.

$$PI(i) = \{j | i(x_j; x_i^c; \gamma)\} \; and \; (z_{ij} < p) \tag{13.3}$$

Randomly choosen z_{ij} belongs to the U with the range from 0 to 1 that has to be less than the fraction of the connected inputs within the space, ρ. If synapses, S_{ij}, have values greater than the threshold parameter, θ_c, it is said to be connected with their set, B_{ij}, shown in Eq. 13.2.

$$B_{ij} = \begin{cases} 1, & \text{if } S_{ij} \geq \theta_c. \\ 0, & \text{otherwise.} \end{cases} \tag{13.4}$$

Aforementioned *PI* containing *j*th input within the uniform distribution, U, represented in the matrix form, if case in Eq. 13.3 is true.

$$S_{ij} = \begin{cases} U(0,1), & \text{if } j \in PI(i). \\ 0, & \text{otherwise.} \end{cases} \tag{13.5}$$

In the previous section, we talked about four phases of the SP, which contains inhibition stage, which occurs due to the neighborhood columns, N_i, that is defined in the Eq. 13.4, where $\| y_i - y_j \|$ represents distance between two mini-columns that has to be less than the radius parameter, ϕ.

$$N_i = \{j | \parallel y_i - y_j \parallel < \phi, i \neq j\} \tag{13.6}$$

It is important to mention that, if inputs and mini-columns have the same dimensions, ϕ will be equal to the edge length defined in the Eq. 13.1. Overlap illustrated in the Fig. 13.4 can be calculated by Eq. 13.5 with the input pattern Z multiplied by the boosting factor β_i.

$$o_i = \beta_i \sum_j B_{ij} Z_{ij} \tag{13.7}$$

It was mentioned that during the learning phase, synapses and boosting factor are updated. Synapses are upgraded regarding to their activation stage.According to the Eq. 13.6, mini-column will be active if overlap is greater than stimulus threshold, θ_s, and among the top position within the neighbours, $NO(i)$.

$$\alpha = 1, \textit{if } (o_i \geq prctile(NO(i), 1 - s)) \textit{ and } (o_i \geq \theta_s) \tag{13.8}$$

Boosting factor is updated regarding to the time-average and recent activities values, which can be calculated by Eq. 13.7 and Eq. 13.8 respectively.

$$\overline{\alpha_i}(t) = \frac{(T-1) \times \overline{\alpha_i}(t-1) + \alpha_i(t)}{T} \tag{13.9}$$

$$< \overline{\alpha_i}(t) >= \frac{1}{|N(i)|} \sum_{j \in N(i)} \overline{\alpha_i}(t) \tag{13.10}$$

After obtaining the required values for the activities, with the help of the adaptation effect, η one can calculate the boosting factor proposed in Eq. 13.9.

$$\beta_i(t) = e^{-\eta(\overline{\alpha_i}(t) - <\overline{\alpha_i}(t)>)} \tag{13.11}$$

13.4 HTM Temporal Memory

13.4.1 Overview of the HTM TM

After formation of the SDR sequences from the input data, according to the Fig. 13.3, it will proceed to the Temporal Memory (TM). Figure 13.7 illustrates the HTM TM processing in time. The HTM TM learns the consistency of SDRs arranged from SP and make predictions relative to these sequences [1]. Thus, the output from the SP becomes input to the TM, where the active columns of the SP creates feed-forward inputs. There are two steps that proceeds along the TM. The first one is to activate the cells within the column which is in the predictive state. If the proximal connections are linked between the two spaces: input and output; the network among the cells called distal connection. The connection of cells can be within the same column or other columns. Thus, to determine the predictive state, the connection of the cells will be the target. The number of the synapses connected to the active cells is computed for each dendrite segment [14]. The cells that are related to the activated dendrite segment are settled to the predictive state, which is a basic procedure to perform the prediction in the HTM TM. Then, the update of the synapses according to the activity of the cell occurs, where the synaptic permanence is increased in the active cells. If the prediction is correct, this temporal change is accepted. Otherwise, this temporal change is removed, and the synaptic permanence is reduced.

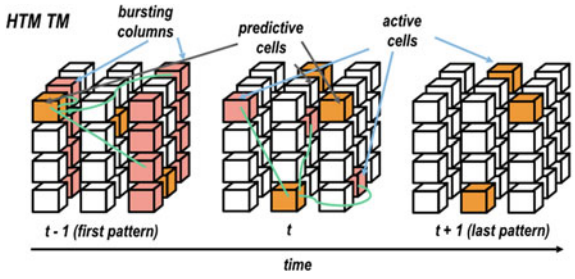

Fig. 13.7 The HTM TM illustration of bursting columns, active and predictive cells [14]

13.4.2 Mathmatical Framework of the HTM TM

If one particular cell has a connection with other cells, and those cells are in the active state, the algorithm will count the scalar value of cells in the active state through the connection.

$$\pi_{ij}^{t} = \begin{cases} 1, & \text{if} \exists_d \parallel \tilde{D}_{ij}^{d} \cdot A^{t} \parallel_1 > \theta \\ 0, & \text{otherwise.} \end{cases} \tag{13.12}$$

If this number is higher than the already defined threshold value, θ, the cell will be in the predictive state defined in the Eq. 13.10. The $\tilde{D}_{ij}^{d}$ shows dth segment of ith cell in jth column, where A^t is the activation matrix. Also, the cells for the predictive state are selected from columns that won during the inhibition stage [2]. There are cases when there is no predictive cells in such columns. In such case, all cells within the single column are active and are called bursting. After this learning procedure of the active columns of the SP, TM proceeds to the second step, which is updating the permanence values of synapses. Like in SP, the permanence value of those synapses in active state increases and decreases for inactive ones. The activation state calculation is in Eq. 13.11

$$\alpha_{ij}^{t} = \begin{cases} 1, & \text{if} j \in W^{t} \; and \; \pi_{ij}^{t-1} = 1 \\ 1, & \text{if} j \in W^{t} \; and \; \sum_i \pi_{ij}^{t-1} = 0 \\ 0, & \text{otherwise.} \end{cases} \tag{13.13}$$

where a_{ij}^{t} representing the elements within the aforementioned matrix A^t in ith cell with jth columns at time t. The variable W^t stands for the top percentage column with the most synaptic inputs.

$$\Delta D_{ij}^{d} = \rho^{+} D_{ij}^{d} \cdot A^{t-1} - \rho^{-} D_{ij}^{d} \cdot (1 - A^{t-1}) \tag{13.14}$$

Equation 13.12 shows reinforcement of the subsequently active segment with ρ^- and ρ^+ representing the negative and positive reinforcements respectively. Including a certain amount of decay to non-active segments proposes long-term depression

$$\Delta D_{ij}^{d} = \tilde{\rho}^{-} D_{ij}^{d} \tag{13.15}$$

where $\tilde{\rho}^-$ is much smaller than ρ^-.

13.5 Conclusion

HTM is a cognitive learning algorithm which mimics the behavior of the neocortex. This chapter gives a brief description about HTM properties, architecture and structure. HTM algorithm showed the promising results in various tasks, especially for visual data processing and classification, such as handwritten digits, face and speech recognition [3, 13, 15]. The hardware implementation of HTM has been shown in several recent works [3, 12, 15]; however it is still an open problem.

Chapter Highlights

- HTM is a recently proposed cognitive learning algorithm that is intended to emulate the overall structural and functionality of the human neocortex responsible for the high-order functions such as cognition, learning and making predictions.
- The main properties of HTM is hierarchical structure, sparsity and modularity.
- HTM consists of two main parts: HTM Spatial Pooler (SP) and HTM Temporal Memory (TM). The HTM SP performs the encoding of the input data and produces sparse distributed representation (SDR); while the HTM TM detects the temporal changes in the input data and performs prediction making.
- HTM algorithm showed the promising results in various tasks, especially for visual data processing and classification, such as handwritten digits, face and speech recognition.
- There are several hardware implementations of HTM; however, it is still an open problem.

References

1. Ahmad S, Lewis M (2017) Temporal memory algorithm. Technical report, Numenta, Inc
2. Cui Y, Ahmad S, Hawkins J (2017) The HTM spatial Pooler- a neocortical algorithm for online sparse distributed coding. Technical report, Numenta, Inc
3. Fan D, Sharad M, Sengupta A, Roy K (2016) Hierarchical temporal memory based on spin-neurons and resistive memory for energy-efficient brain-inspired computing. IEEE Trans Neural Netw Learn Syst 27(9):1907–1919
4. Hawkins J (2004) On Intelligence. Griffin, Macmillan
5. Hawkins J (2016) Biological and machine intelligence. Numenta. http://numenta.com/biological-and-machine-intelligence/

6. Ahmad S, Hawkins J (2016) How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites. Technical report, Numenta, Inc
7. Hawkins J, Ahmad S (2016) Why neurons have thousands of synapses, a theory of sequence memory in neocortex. Technical report, Numenta, Inc
8. Hawkins J, Ahmad S, Dubinsky D (2010) Hierarchical temporal memory including HTM cortical learning algorithms. Techical report, Numenta, Inc, Palto Alto. http://www.numenta.com/htmoverview/education/HTM_CorticalLearningAlgorithms.pdf
9. Hawkins J, Ahmad S, Purdy S, Lavin A (2016) Biological and machine intelligence (bami). Initial online release 0.4
10. Ibrayev T, Krestinskaya O, James AP (2017) Design and implication of a rule based weight sparsity module in HTM spatial pooler. In: 24th IEEE international conference on electronics, circuits and systems (ICECS). IEEE, pp 274–277
11. Ibrayev T, Myrzakhan U, Krestinskaya O, Irmanova A, James AP (2018) On-chip face recognition system design with memristive hierarchical temporal memory. J Intell Fuzzy Syst 34(3):1393–1402
12. James A, Ibrayev T, Krestinskaya O, Dolzhikova I (2018) Introduction to memristive HTM circuits. In: Memristor and memristive neural networks. InTech
13. Krestinskaya O, James AP (2018) Feature extraction without learning in an analog spatial pooler memristive-CMOS circuit design of hierarchical temporal memory. Analog Integr Circuits Signal Process 95(3):457–465
14. Krestinskaya O, Dolzhikova I, James AP (2018) Hierarchical temporal memory using memristor networks: a survey. IEEE Trans Emerg Top Comput Intell 2(5):380–395. https://doi.org/10.1109/TETCI.2018.2838124
15. Krestinskaya O, Ibrayev T, James AP (2018) Hierarchical temporal memory features with memristor logic circuits for pattern recognition. IEEE Trans Comput-Aided Des Integr Circuits Syst 37(6):1143–1156
16. Krestinskaya O, James AP, Chua LO (2018) Neuro-memristive circuits for edge computing: a review. arXiv:1807.00962
17. Lavin A, Ahmad S (2017) Sparse distributed representations. Technical report, Numenta, Inc
18. Purdy S (2017) Encoding data for HTM systems. Technical report, Numenta, Inc
19. Spruston N (2008) Pyramidal neurons: dendritic structure and synaptic integration. Nat Rev Neurosci 9(3):206
20. Webster D, Popper A (1992) The mammalian auditory pathway: neuroanatomy. Springer, New York
21. Zyarah AM (2015) Design and analysis of a Re-configurable hierarchical temporal memory architecture. Technical report, Rochester institute of technology

Chapter 14
Memristive Hierarchical Temporal Memory

Olga Krestinskaya, Irina Dolzhikova and Alex Pappachen James

Abstract This chapter covers the memristive HTM implementations on mixed-signal and analog hardware. Most of the implemented memristive systems are based on modified HTM algorithm. The HTM is often used as a feature encoding and feature extraction tool, and these features are then used with conventional nearest neighbor method for classification.

14.1 Introduction

Hierarchical Temporal Memory (HTM) is bioinspired algorithm and architecture consisting of Spatial Pooler (SP) and Temporal Memory (TM) and emulating structure and functionality of human neocortex [2, 3]. There are several HTM implementations on hardware based on FPGA [15] and analog and mixed signal circuits [9, 11, 13]. In most of the cases, FPGA implementations are more inefficient on term of power consumption and on-chip area [12]. Therefore, recent HTM implementations involve novel devices which consume less power and on-chip area, such as memristors [12] and spin-devices [1]. Therefore, this chapter will focus on the memristive HTM implementations on hardware and show how HTM can be applied for patter recognition and classification tasks.

The sections are organized as following. Section 14.2 represents the initial design of the HTM cell. Section 14.3 illustrates the HTM implementations based on the memristive crossbar. Section 14.4 shows fully analog memristive implementation of modified HTM with TM. Section 14.5 covers the systems, which use HTM for feature extraction and classification. Section 14.6 concludes the chapter.

O. Krestinskaya · I. Dolzhikova · A. P. James (✉)
Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

O. Krestinskaya
e-mail: ok@ieee.org

I. Dolzhikova
e-mail: ifedorova@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_14

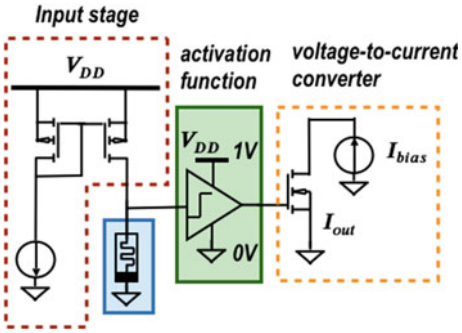

Fig. 14.1 Single synapse circuit design for SP implementation [5]

14.2 HTM Cell

One of the earliest analog implementations of HTM cell is shown in [5], where main idea is analog circuit design of the HTM SP using CMOS-memristor hybrid structure. The paper [5] presents a circuit design for single synapse as shown in the Fig. 14.1 that is used for SP column implementation with S synapses.

The input to the circuit is supplied in the form of current through the current mirror. The function of synaptic weight is implemented using a memristor. The synaptic weight is allowed to be changed only during the learning stage, while during the input supply its value is constant. This is ensured through the careful selection of the input current amplitude. The buffer in the design performs the activation function, where the high output indicates that the given weighted input will contribute to the overlap value of the column. The NMOS transistor is used to convert the voltage output from the buffer to the current value. This makes the calculation of the total overlap value for the whole column straightforward through the simple summation of the currents. This HTM design demonstrated the 80% recognition accuracy for the AR database face recognition problem [5].

14.3 Memristive Crossbar Based HTM

14.3.1 Mixed Signal HTM Design with Spin Devices

One of the first attempts to implement memristor based HTM system is presented in [1] (Figs. 14.2 and 14.3). In the design of HTM the authors combine memristive crossbar structure with the spin-torque based switches (so-called spin neurons). The proposed HTM design specifications were tested on the pattern recognition problem with MNIST handwritten digits dataset and COIL-20 objects dataset. During the training process the spatial and temporal patterns of the input image that varies with the time are extracted. The training is based on the bottom-up approach, where the

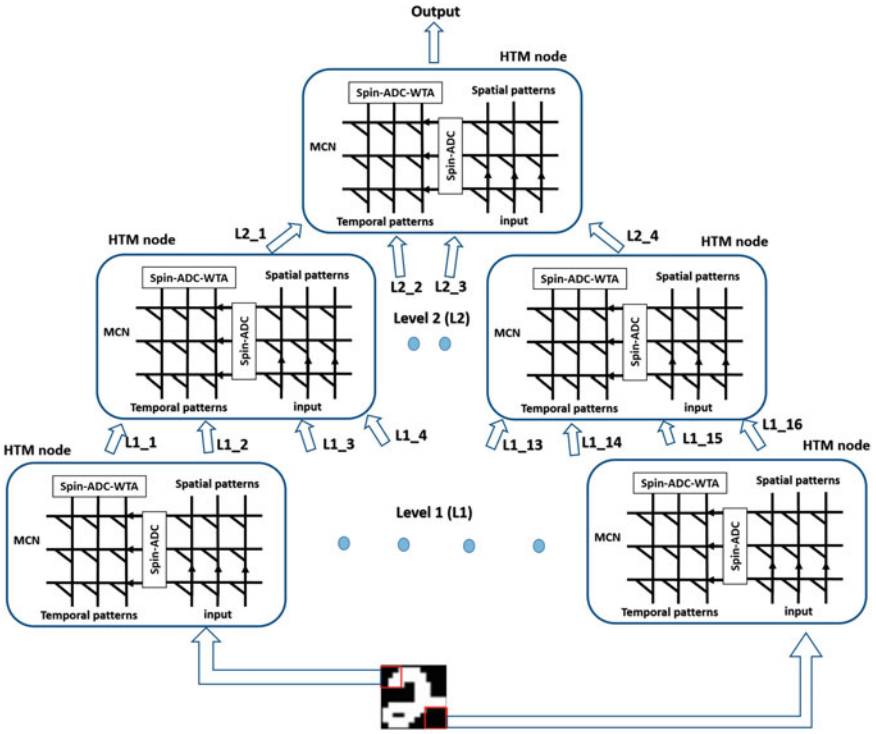

Fig. 14.2 HTM design using MCN and spin neuron proposed in [1]

lower levels with the child nodes are fully trained before the higher levels. Unsupervised training is used for all of the levels, except the output node (top level). The inference process for a node could be subdivided into 4 stages: representation of the spatial input, calculation of probability densities over the coincidences in the spatial domain, calculation of probability densities over the coincidences in the temporal domain, and computation of the output, which is represented by the index of the winning temporal class. The main function that is used for computation during the inference phase is related to the calculation of the dot product. As a result, memristive crossbar network (MCN) is highly suitable for the design of such inference process. MCN allows for direct computation of the correlation between the input (Vi) that is applied through the rows and features that are stored in columns of the crossbar (by considering the conductance values of crossbar (g_{ij})). The best matching case between the input and stored patterns could be determined by finding the largest magnitude $\sum_i V_i g_{ij}$.

The overall architecture of the memristor and spin-neuron based HTM system [1] is demonstrated in Fig. 14.2. It composed of MCN, the successive approximation (SAR) analog-to-digital converter (ADC) and winner-take-all (WTA) circuit. The 16 *times* 16 pixel input image is subdivided into 16 patches with 4 × 4 pixels. Single

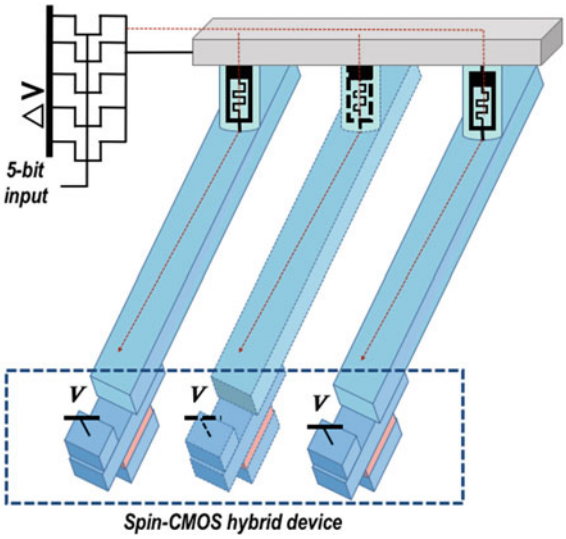

Fig. 14.3 1×3 MCN with single deep-triode current source at the input and three spin neurons in the output [1]

node of level-1 received single patch. First of all, the MCN calculates the density over the spatial features, then SAR ADC transforms the output current values into the digital inputs of the second MCN, which calculates the density over the temporal features. The outputs from the second MCN go through WTA that recognizers the winning output and forwards it to the parent node in the upper level. In case of using the proposed methodology for the MNIST handwritten digits recognition problem, the WTA output from the top-most level will represent the digit that was illustrated on the input image.

The size of the two MCNs in all of the computational nodes are $n_{child} \times n_{SP}$ and $n_{SP} \times n_{TM}$, where the n_{child} is the number of child nodes, n_{SP} is the spatial features and nTM is the number of the temporal groups. The digital input values of the MCN are converted into analog domain. This is done with help of spin neurons that are based on the domain wall (DW) magnet. DW neurons allow to construct the low energy digital-to-analog (DAC) that is based on the deep-triode current source (DTCS) PMOS transistors with binary weighting. This is illustrated in Fig. 14.3.

14.3.2 Analog Memristive Crossbar-Based HTM SP

Different method for using memristive crossbar structures for HTM implementation was noticed in [8]. The proposed hardware configuration was applied to the face and speech recognition applications. The circuit for SP of HTM allows to make separation between the unimportant and important features during the processing of the face image and speech, so that the latter ones could be used for the recognition with help of template matching technique.

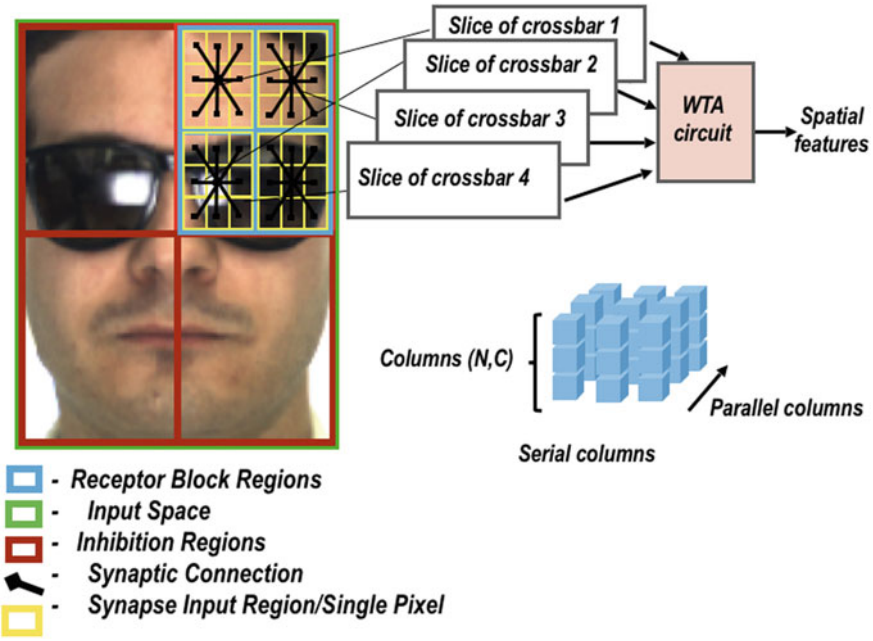

Fig. 14.4 Pre-processing of the image to group the input pixels into small blocks and inhibition region, and high-level representation of the crossbar-based architecture for SP implementation [8]

The classification of the features relies on the consideration of the threshold logic. The features has the status of the important or relevant detail in the given set of sparse data if the calculated weighted input ($u_i w_i$) is greater than the threshold value. Before the pixels of the input image are fetched into the crossbar arrays the preprocessing is performed that subdivides the whole input space (original image) into blocks that are called inhibition regions. The inhibition region in turn is divided into smaller blocks that consist of several image pixels. The example of grouping image pixels into small blocks and inhibition regions is illustrated on the Fig. 14.4. In the given example there are nine pixels in the small block and four small blocks in the inhibition region.

The proposed memristive architecture for HTM [8] requires use of several crossbar slices in parallel. The columns in a single crossbar are processed in series (one column at a time) to alleviate the problem of sneak path leakage currents. As a result columns within a single crossbar are referred to as serial columns and there are N number of such columns. The number of parallel crossbar slices, as well as the number of parallel columns C, is determined by the number of small blocks within a single inhibition block. Single small block groups S number of pixels and this produces S synapses within a column. The circuit design of the one $S \times N$ crossbar slice is shown in the Fig. 14.5.

The crossbar operates under the reading and writing modes that are selected through the set of switches in each crossbar line realized with NMOS and PMOS

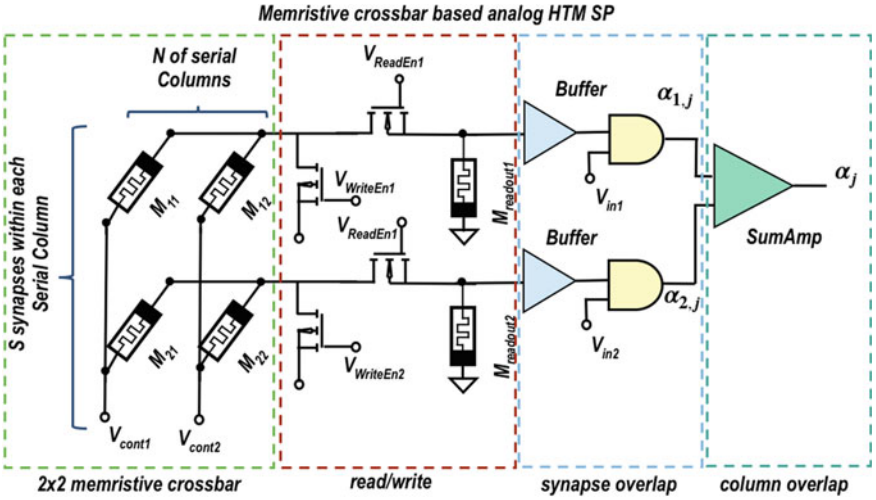

Fig. 14.5 Circuit diagram for overlap calculation using memristive crossbar of the $S \times N$ size [8]

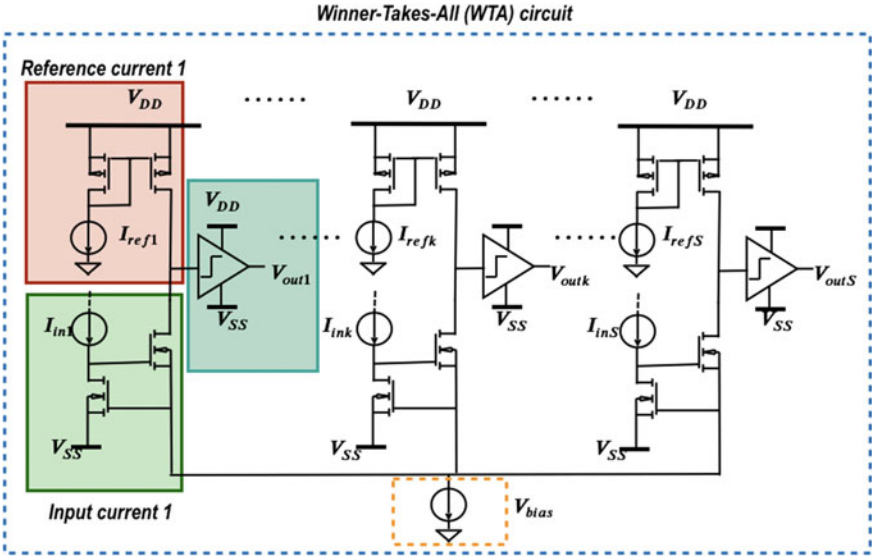

Fig. 14.6 WTA circuit proposed by Lazzaro et al. [14]

transistors. A single serial column from each of the parallel crossbar is activated during the reading mode and the voltage across the $M_{readout}$ is measured. This voltage is taken for calculation of the overlap value for each synapse in the column $(\alpha_{1,j}, \ldots, \alpha_{1,S})$. With help of summing amplifier the total overlap α_j for entire column is determined. The winning column could be found by using Winner-Take-All (WTA) circuit. WTA circuit proposed by Lazzaro et al. [14] in the Fig. 14.6 is used for inhibition phase. The structure of WTA helps to determine the column that inhibits the others and becomes the winning one by choosing from the total overlap current values $I_{in1}, \ldots, I_{inS}$. The output from the SP is the binary value that represents the classification of the image features into important and unimportant, which could be used for pattern matching.

14.4 Fully Analog Memristive HTM with Temporal Memory Implementation

The fully analog hardware implementation of memristive HTM SP and HTM TM are shown in Figs. 14.7 and 14.8, respectively [12]. This HTM design is a modification of original HTM algorithm, following the same concept. HTM SP is modified to perform mean based approach, while HTM TM follows a simplified approach based on Hebb's learning rule. The processing of the input data can be performed using parallel or sequential approaches. Depending on the on-chip area and power constraints and required accuracy and processing speed, input data from different inhibition regions of HTM SP can be processed in series as well as in parallel. In the analog implementation of HTM TM, this is also possible, as the circuit illustrated in Fig. 14.8 refers to the processing if a single input. However, for large scale data processing, the sequential processing of inputs is more feasible due to the on-chip area and power requirements of the circuit.

The main modification in HTM SP algorithm is a selection of winning columns in inhibition stage based on the mean value calculation, rather then considering kth largest overlap values. The selection of winning columns is based on the threshold value calculated as an mean of overlaps. However, the architecture of HTM SP shown in Fig. 14.8 is completely different, comparing to the crossbar-based HTM SP approach. The mean based approach simplifies the analog hardware implementation of HTM, as mean calculation is less complicated on hardware and requires smaller on-chip area and power, rather than summation and Winner-Takes-All (WTA) operation.

The analog circuit for HTM SP implementation is illustrated in Fig. 14.7. The input data is divided into inhibition blocks, and the HTM SP architecture shown in Fig. 14.7 refers to the processing of a single inhibition block of HTM SP. The inputs to each inhibition block are the values obtained from the receptor block. Receptor block performs the multiplication of the input space by random weight synapses, implemented using memristive devices with randomly assigned resistive weights. The synapses are considered to be connected, when the synapse value is R_{ON}

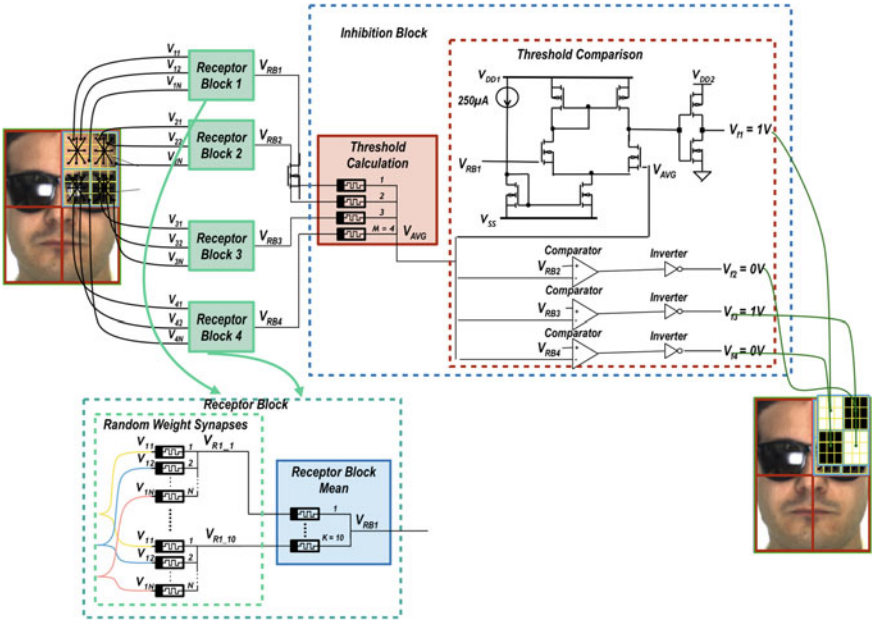

Fig. 14.7 Modified HTM SP [12]

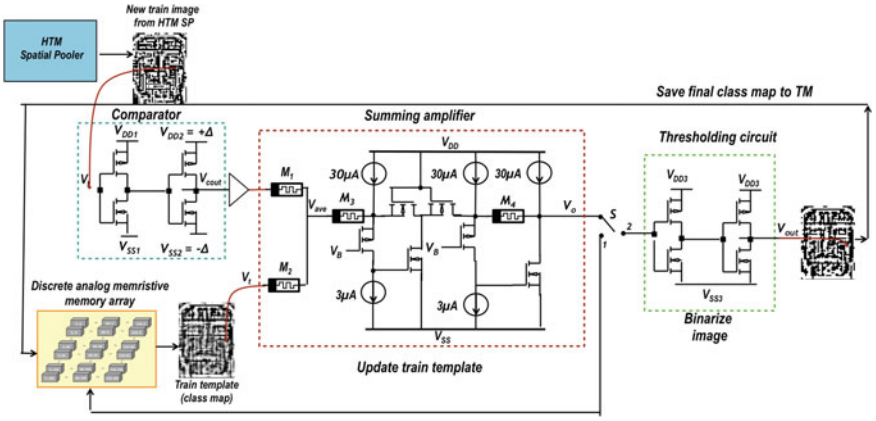

Fig. 14.8 Analog HTM TM [12]

(low resistance) and disconnected, when the resistance is R_{OFF} (high value). Each receptor block has a set of several separate weights processing the same inputs to randomize and obtain the unique HTM output, corresponding to the particular class. The output of the receptor block is the average value of multiplications of inputs by the random weights and presents the column overlap value. After the receptor block, the single threshold for the whole inhibition region is calculated by averaging of output overlap values from the receptor blocks. The circuit level implementation of averaging operation is simple and involves only the memristors with the same resistive levels. In a threshold comparison circuit each obtained overlap value is compared to the calculated threshold by the comparator circuit and the output is either accepted or inhibited by the inverter. The output of the inhibition block is represented as 1 or 0.

In this architecture, the supervised approach is used to compare input patterns with ideal data. Therefore, HTM TM in this algorithm is used to create a class map, which illustrates the ideal pattern for each class formed during HTM TM processing. Therefore, HTM TM is involved mostly in the learning of ideal patterns, rather than the whole HTM processing. Based on the input data, HTM TM performs learning of important and unimportant features in the input classes over time, and forms the templates for each data class. The features in the template of each class map are incremented or decremented based on the current input to HTM TM. The class templates are stored in memristive memory arrays, and the arrays are updated using Hebb's learning rule, by adding or subtracting Δ value from the original feature stored in the array. In this process, the new input data is processed by HTM SP to obtain binary SDRs of the input data, presented as 1 or 0. If in a new SDR, a data feature is 1, the class map feature stored in memristive memory array is incremented by HTM TM, and vice versa.

The HTM TM design consists of the comparator circuit, which is used to produce Δ values from the input data. The circuit is called a comparator, as it compares binary SDRs obtained from HTM SP output, and generates $+\Delta$ or $-\Delta$ value. Next, the Δ value is summed up with the previous value stored in a class map and read from memristive memory array. This is performed by the summing amplifier consisting of the averaging circuit and multiplication by 2. The ratio of memristors M_3 and M_4 determines the multiplication factor. Based on the obtained value, the class map stored in a memristive memory is updated. When the sufficient number of inputs of a particular class form the class map, the comparator and summing amplifier are disconnected by the switch and the class map is binarized by the thresholding circuit similar to the one from the HTM SP. HTM TM is not involved in further processing and only class map is used in the comparison stage of supervised learning, where the SDRs of the input images produced by the HTM SP are compared with the class map using memristive pattern matcher shown in Fig. 14.9. Memristive pattern matcher consists of memristive XNOR gates, followed by the averaging circuit for all the data, corresponding to the comparison of a particular class. The XNOR gate outputs 1, when the inputs from the processed data and the data stored as a class map match, and 0 otherwise. Comparing all data features, the output of all XNOR gates for a single class is averaged, which is performed for all the data classes. The

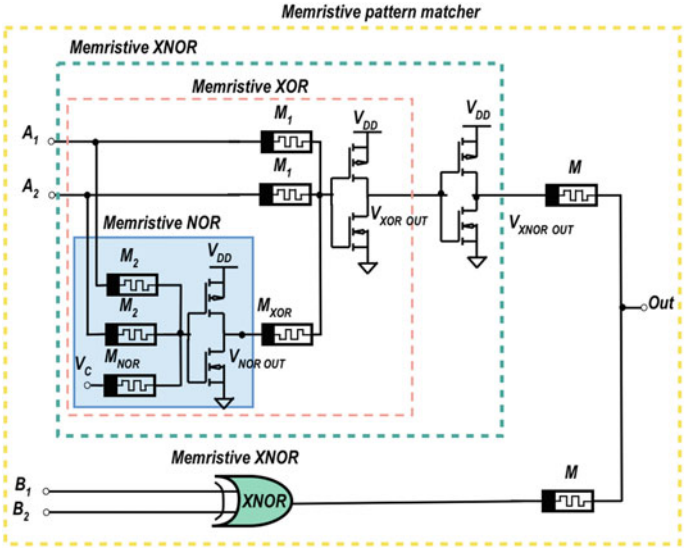

Fig. 14.9 Memristive pattern matcher [12]

winning class for the classification task is selected based on the highest output of the averaging circuit out of all the classes. This HTM system is an example of application of a concept of modified HTM for supervised learning, which is further discussed in the following section. This approach illustrated the accuracy of 87.21% for face recognition application with AR database [12].

14.5 HTM Systems for Feature Extraction and Pattern Recognition

As most of the hardware implementations of memristive HTM modify the original HTM algorithm and use the parts of HTM as the tool for feature extraction. This section covers the application of HTM as tool to extract meaningful features from the input data for pattern recognition.

14.5.1 Rule-Based HTM SP without Learning for Feature Extraction

The rule-based HTM approach explores the application of the HTM SP concept without learning [4, 10]. The research work [4] shows the algorithmic implemen-

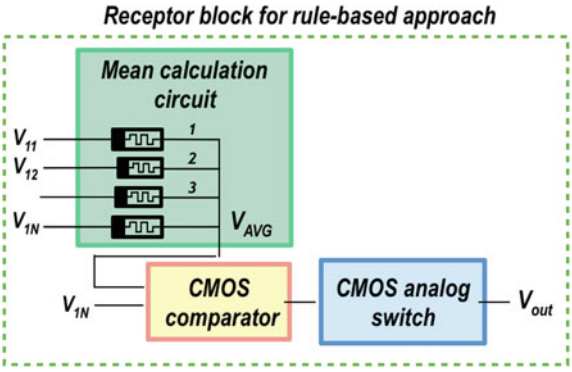

Fig. 14.10 Receptor block for rule-based HTM approach [10]

tation of this approach, where the main idea is to illustrate the sparsity principle of HTM by natural features from the input pattern. Comparing to the conventional HTM initialization, where initial weights of the HTM SP are randomized, the rule-based approach sets the initial synaptic weights based on the local mean intensity of the input pattern to preserve the natural sparsity of the input data. The local mean of the weights within a particular data window is calculated and considered as a threshold. If the synaptic weights are greater than this threshold, the weights is set to high and if less, the initial weight is set to low.

The hardware implementation of receptor block for rule-based approach in HTM SP is illustrates in Fig. 14.10, and consists of mean calculation circuit, comparator circuit for the comparison of the inputs with the mean and analog switch producing 1 or 0 output. Also, the other important part is a programming circuit to set the memristive weights. The inhibition block in the hardware implementation was the same as in the modified HTM approach in Fig. 14.7. The learning part of HTM SP is not considered in this approach, which allows to reduce the circuit level design. The method was tested as for the extraction of meaningful features from the data for classification problem without learning. The inhibition block output corresponding to the first input from the particular class was used as a ideal class template, and all further inputs processed by rule-based HTM SP were used for the classification. This approach allowed to achieve the accuracy of 80% for face recognition with AR database, while significantly reducing the size of the circuit. This approach has a limited set of application and will work only with particular input patterns, as images. Also, the natural sparsity of the images is important to achieve high classification accuracy.

14.5.2 HTM Systems for Pattern Recognition

There are several systems which use HTM as a feature extraction tool [6, 7, 12]. HTM performs a feature encoding and allows to extract meaningful features from the

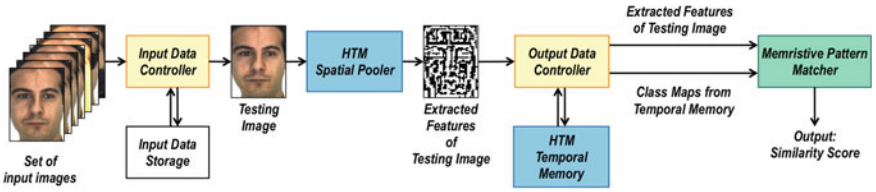

Fig. 14.11 Overall classification system with HTM circuits [6, 7, 10]

data. The implementation of overall system for pattern recognition and classification is illustrated in Fig. 14.11. The figure illustrates the example of face recognition application with the supervised learning. The set of input images, which can come directly from the sensor, is processed by input data controller. This controller is used for the sampling of input data and pre-processing, if it is required. Also, the controller stores the images required for processing in the data storage. Input data controller provides the image to the HTM SP block, which extracts the meaningful features from the input image, forwarding the image to output data controller. As here the supervised learning is illustrated, there are two different stages in this: (1) generation of the ideal image template for the supervised learning approach for all image classes, and (2) classification. As it was mentioned previously for modified HTM implementation, the HTM TM is used only for the generation of image class maps, the images that will be considered as ideal in the classification stage. Therefore, depending on the processing stage, output data controller forwards the image produced by the HTM SP to HTM TM or to the pattern matcher. At the stage of generating the ideal pattern for each class, the controller forwards data to the HTM TM and the pattern matching part is disconnected. During this process, the templates for each class are updated. In the classification stage, the pattern matcher outputs the similarity score, comparing the input image produced by the HTM SP with all the image classes stored as class templates in memristive memory arrays.

14.6 Conclusion

This chapter covered, the memristive hardware implementations of HTM systems. To implement HTM on hardware, in most of the cases HTM algorithm is modified, as an analog hardware implementation of original HTM algorithm can be computationally complex and consume high power and large on-chip area. Several systems use modified HTM algorithm for the data encoding and extraction of meaningful features that can be used for pattern recognition application. The main challenge in memristive systems is memristive devices, which have certain imperfections, which can effect the performance of the system and classification accuracy. These include the variability in memristive devices, integration to CMOS design, endurance of the memristor and limited resistive levels. The study of these effects oh the memristive

HTM systems has not been performed yet. The other open problem in hardware implementation of HTM algorithm is a complete on-chip implementation of the original algorithm without modifications.

Chapter Highlights

- Most of the hardware implementations of HTM with memristive devices are based on the modification of original HTM algorithm.
- HTM SP can be used for both: original HTM and for the extraction of meaningful features from input data in the traditional nearest neighbor classification approach.
- Even thought several HTM implementations have been proposed in recent years [1, 8, 12], the full on-chip hardware implementation of original HTM algorithm is an open problem.

References

1. Fan D, Sharad M, Sengupta A, Roy K (2016) Hierarchical temporal memory based on spin-neurons and resistive memory for energy-efficient brain-inspired computing. IEEE Trans Neural Netw Learn Syst 27(9):1907–1919
2. Hawkins J, George D (2006) Hierarchical temporal memory: Concepts, theory and terminology. Technical report, Numenta
3. Hawkins J, Ahmad S, Dubinsky D (2010) Hierarchical temporal memory including htm cortical learning algorithms. Technical report, Numenta, Inc, Palto Alto. http://www.numenta.com/htmoverview/education/HTM_CorticalLearningAlgorithms.pdf
4. Ibrayev T, Krestinskaya O, James AP (2017) Design and implication of a rule based weight sparsity module in htm spatial pooler. In: 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS). IEEE, pp 274–277
5. Ibrayev T, James AP, Merkel C, Kudithipudi D (2016) A design of htm spatial pooler for face recognition using memristor-cmos hybrid circuits. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, pp 1254–1257
6. Ibrayev T, Myrzakhan U, Krestinskaya O, Irmanova A, James AP (2018) On-chip face recognition system design with memristive hierarchical temporal memory. J Intell Fuzzy Syst 34(3):1393–1402
7. Irmanova A, Ibrayev T, James AP (2017) Discrete-level memristive circuits for htm-based spatiotemporal data classification system. IET Cyber Phys Syst Theory Appl 3(1):34–43
8. James AP, Fedorova I, Ibrayev T, Kudithipudi D (2017) Htm spatial pooler with memristor crossbar circuits for sparse biometric recognition. IEEE Trans Biomed Circuits Syst 11(3):640–651
9. James A, Ibrayev T, Krestinskaya O, Dolzhikova I (2018) Introduction to memristive htm circuits. In: Memristor and Memristive Neural Networks. InTech

10. Krestinskaya O, James AP (2018) Feature extraction without learning in an analog spatial pooler memristive-cmos circuit design of hierarchical temporal memory. Analog Integr Circuits Signal Process 95(3):457–465
11. Krestinskaya O, Dolzhikova I, James AP (2018) Hierarchical temporal memory using memristor networks: a survey. IEEE Trans Emerg Top Comput Intell 2(5):380–395. https://doi.org/10.1109/TETCI.2018.2838124
12. Krestinskaya O, Ibrayev T, James AP (2018) Hierarchical temporal memory features with memristor logic circuits for pattern recognition. IEEE Trans Comput Aided Des Integr Circuits Syst 37(6):1143–1156
13. Krestinskaya O, James AP, Chua LO (2018) Neuro-memristive circuits for edge computing: a review. arXiv preprint arXiv:1807.00962
14. Lazzaro J, Ryckebusch S, Mahowald MA, Mead CA (1989) Winner-take-all networks of o (n) complexity. In: Advances in Neural Information Processing Systems, pp 703–711
15. Zyarah AM, Kudithipudi D (2015) Reconfigurable hardware architecture of the spatial pooler for hierarchical temporal memory. In: 2015 28th IEEE International System-on-Chip Conference (SOCC). IEEE, pp 143–153

Chapter 15
Deep Neuro-Fuzzy Architectures

Anuar Dorzhigulov and Alex Pappachen James

Abstract Fuzzy logic inspires from the non-deterministic behaviour of human brain computations. The fusion of neural networks and fuzzy logic such as neuro-fuzzy architectures is natural, as both represent elementary inspiration from brain computations involving learning, adaptation and ability to tolerate noise. This chapter focuses on neuro-fuzzy and alike solutions for machine learning from perspective of functionality, architectures and applications.

15.1 Introduction

The demand for intelligent data processing resulted in the increased need to have machine intelligence and learning algorithms. Majority of the contemporary machine learning techniques inspired from the human brain, and neuro-fuzzy (NF) systems are a set of methods focused on the processing of the imprecise and vague data. It is a hybrid of the fuzzy sets and logic with the artificial neural network style of architectures and self optimization. Neuro-fuzzy architectures is often used as the robust data processing models, especially popular in the field of the system control [1].

Majority of the existing NF data architectures and control units exist as the software solution. Nevertheless, the interest in the dedicated NF hardware is growing among the researchers. This trend can be attributed to the enormous energy demand for the processing of the colossal data that requires collection, storage and processing today. Dedicated hardware tends to lack the flexibility of the software, but shows superiority in terms of the parallelism, energy efficiency, occupied area and processing speed. This feature is maximized in the specialized analog hardware. Development of the dedicated analog NF hardware will be a potential solution for energy

A. Dorzhigulov · A. P. James (✉)
School of Engineering, Nazarbayev University, Astana, Kazakhstan
e-mail: apj@ieee.org

A. Dorzhigulov
e-mail: adorzhigulov@nu.edu.kz

A. P. James (ed.), *Deep Learning Classifiers with Memristive Networks*,
Modeling and Optimization in Science and Technologies 14,
https://doi.org/10.1007/978-3-030-14524-8_15

efficient big data processing. In addition, the compact size of the analog devices is an opportunity to decentralize data processing, reducing processing on data centers, and outsourcing the preprocessing to the near sensor analog processing unit, reinforcing the edge computing paradigm.

In the last decade, there were many research projects related to the implementation of the NF hardware. It includes implementation of the wholesome NF systems, such as digital hardware [2], FPGA based [3], mixed circuits [4] and fully analog [5]. In addition, many researchers propose implementations of the separate elements of the NF systems, such as membership function generation circuits [6–8] or learning algorithms [9].

15.2 Fuzzy Sets and Logic: Overview of the Basics

Not all uncertainties could be accurately modeled as deterministic stochastic processes. To simplify the data processing, the raw data could be clustered into multiple groups. For example, the temperature sensor readings divided into certain ranges of temperature, below 0 °C, or above, which could be named as 'cold' and 'warm' temperature ranges, or sets. However, what if the difference between −1 and −30 °C is significant for our data processing? What if the degree of 'coldness' could be used for 'cold' temperature range, instead of defining multiple subranges? Such uncertainties of strict set boundaries are explored in fuzzy sets and logic. For example, the strict set A defined by inequality in Eq. 15.1 shows a strict boundary condition of 0.

$$A = \{x \mid x > 0\} \tag{15.1}$$

There will be no difference between value of x of 0.001 or 1000, it will all belong to the set A. But what if we are interested in the negative values that are very close to 0 boundary condition as well? Then, set A could be fuzzy, Fig. 15.1. As it can be observed, the negative values close to 0 are also included into the fuzzy set A, but with lesser degree of association or membership, defined by the continuously decaying function. In this example, the boundary is defined by the Gaussian function with mean c, standard deviation σ and parameter x, Eq. 15.2.

$$f_{Gauss}(x) = exp\left(\frac{-(x-c)^2}{\sigma^2}\right) \tag{15.2}$$

In general, the functions that used to represent the degree of membership or association with a fuzzy sets called *Membership Functions* (MFs). Typically, the value of MFs are normalised between 0 and 1 (where 0 represents no associations with fuzzy set and 1 represents full associations with it of some value of parameter x). The range of values, which is relevant for MF, is called *Universe of Discourse* or just *Universe X* in fuzzy logic terms.

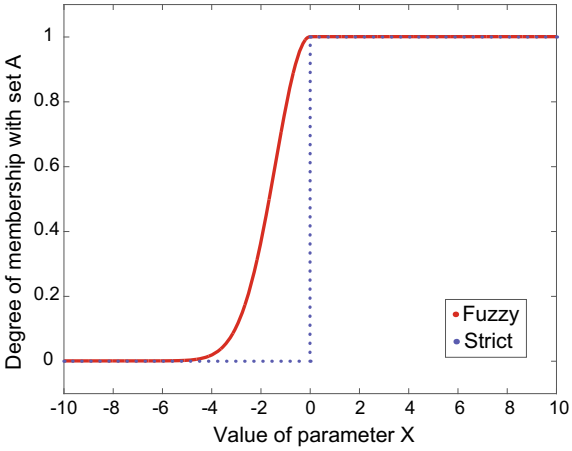

Fig. 15.1 Difference of the boundary condition representation for strict and fuzzy set A and some random parameter x, which is partly belongs to A

The fuzzy MFs are used to define inputs in fuzzy system. In order to connect the inputs and outputs, *If-Then Rules* (ITR) used, Eq. 15.3.

$$\text{if } X \in \{A\}, \text{ then } Y \in \{B\} \tag{15.3}$$

This single rule represent one simple action, or inference, in fuzzy system: if the value of parameter X is within the universe of fuzzy set A, then assign value of output Y to belong to the fuzzy set B. In fuzzy system input parameters are called *Antecedent* or *Premise*, while output parameters are *Consequent* or *Conclusion*.

Simple example of ITR: if the color of the nickel ball is red, then its temperature is high, Eq. 15.4.

$$\text{if } Nickel\ ball\ color \in \{'Red'\}, \text{ then } 'Temperature' \in \{'High'\} \tag{15.4}$$

This simple example shows how non-numerical and linguistic parameter, such as color, could be directly used for fuzzy system. Sure, the color could be deterministic parameter, if it is mapped on RGB scale for example, but fuzzy systems allows this simplistic and subjective representation of the inputs and outputs, while the numerical representations is still viable in fuzzy logic.

Equation 15.3 shows one of the simplest forms of ITR. Single ITR could be structured to infer or map multiple antecedent and consequent parameters at once. And the set of such ITRs (called *Rule Base*) defines the inference algorithm of the whole fuzzy system.

The fuzzy parameters, MFs, architecture and structure in general are defined by the *Experts*, whose knowledge and expertise is used to model the required system behaviour. As a result, the performance of fuzzy system depends on the initial setup and/or consequent fine tuning of the system. The lack of automated self optimization and strong dependence on external experts are the main disadvantages of fuzzy systems.

15.3 From Fuzzy to Neural: How to Combine Neural Networks with Fuzzy Logic

The Neuro-Fuzzy systems is the system that combines the fuzzy sets and logic with the neural network architecture and ability to self-optimize or learn. Typically, different types of the NF architectures build from the similar modules, Fig. 15.2. Those modules are: input fuzzification, inference, defuzzification, rule base and learning [10].

15.3.1 Fuzzification

The main purpose of the fuzzification module is to convert crisp (numeric or deterministic) input feature values to the fuzzy sets. Alternatively, the fuzzification could be performed on a qualitative or linguistic inputs. It could be vague verbal inputs, such as 'higher', 'lower', 'colder', etc. Irrespective to the input type, the module produces the output based on the set fuzzification function, called *Membership Function* (MF). The membership function represents the 'degree of truth' or how close the fuzzified input to the certain desired one. There are several functions that are commonly used for fuzzification: triangular, trapezoidal, Gaussian, generalized bell, S- and Z-function, shown on Fig. 15.3. It should be noted, that not differentiable shapes (triangular and trapezoidal) are rarely used as the hardware. The optimal value of these parameters in order to increase the accuracy of the NF system.

15.3.2 Inference

The inference module is used to produce fuzzy output that is the function of one or multiple fuzzified inputs. This function could be simple operators or algebraic functions, such as average, weighted average, summation, product, minimum or

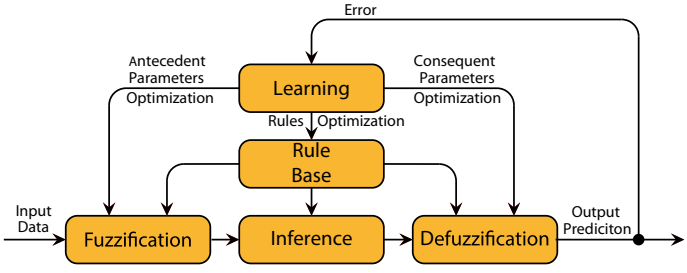

Fig. 15.2 System diagram of the typical Neuro-Fuzzy system

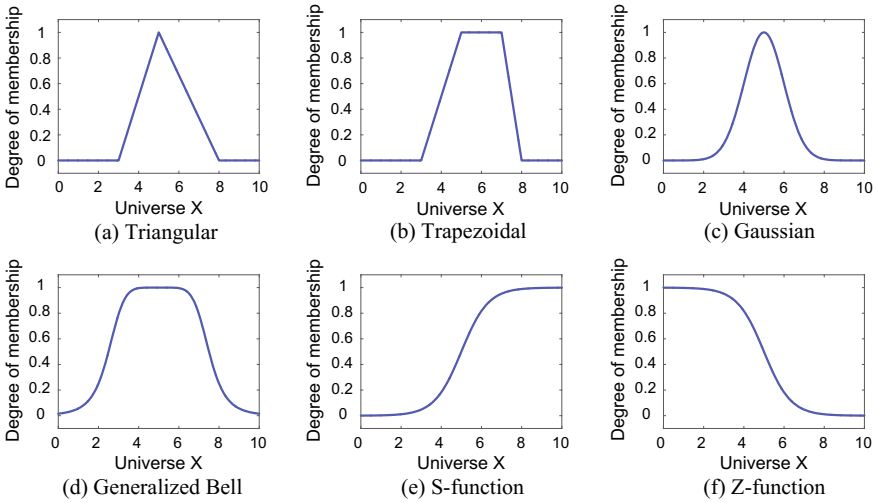

Fig. 15.3 Typical membership function shapes

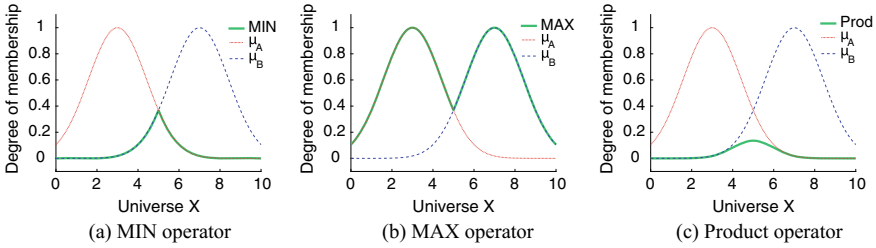

Fig. 15.4 Examples of the commonly used fuzzy operators

maximum (example on a Fig. 15.4), which are applied to all related inputs once (eq. minimum between X1, X2 and X3), or could be build more complex relationships (eq. example the minimum between X3 and the average of X1 and X2), which is common case for fuzzy decision trees. For example assume some control system rule: if the pressure is 'high' and flow is 'low', increase the temperature 'slightly'.

15.3.3 Defuzzification

Defuzzification is the process used to convert the obtained fuzzy output to the numeric crisp value. Typically it shows the relationship between estimated vague response and exact numeric response. For example, if the NF system produces response to 'increase the temperature slightly' it will equate it to 'increase temperature by 1 degree'. In addition, if the NF system is governed by the multiple rules, which is typically the

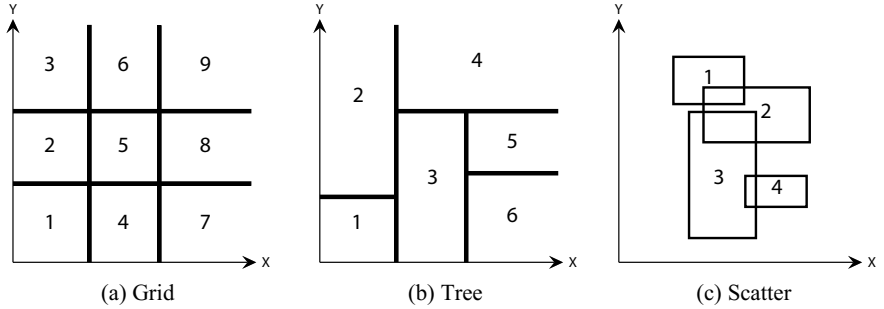

Fig. 15.5 Segmentations of the 2D input space, with input features X and Y

case, the defuzzification process is required to estimate the overall output, taking into account the current value and weight of each rule (e.q. weighted average between output of the Rule 1, Rule 2,…, Rule N). The following defuzzification algorithms are common: centroid of area, bisector of area, mean of max, smallest of max, largest of max.

15.3.4 Rule Base

Rule base based on fuzzy If-Then rules, Eq. 15.3. The primary function of the rule base is the NF system routing, or building the connections between inputs and outputs directly, or trough inference and defuzzification operators if needed. In other words, the rule base is used for segmentation of the input space, example shown on a Fig. 15.5. It should be noted that the rule base is the major limitation for a large scale implementations of the classic NF system, because the number of rules can grow exponentially with the number of inputs, known as the curse of dimensionality. This issue is common for the systems that use Grid Partitioning of the input space, Fig. 15.5a. In order to limit the number of the rules, tree and scatter partitioning can be used, Figs. 15.5b and 15.5c. However, such approaches require more complex MFs relations.

15.3.5 Learning

The learning is the process of the iterative self-optimization of the NF system. The optimization is based on the function of the desired and produced outputs (error) called cost function. There are two major types of the optimization: parameter and structural. The parameter optimization is used to gradually adjust the parameters of the NF modules, fuzzification being the most common. Structural optimization

targets the structure, adjusting the number of the modules in each layer, maximizing the performance with the minimally possible number of the rules.

Learning algorithms can be also differentiated by the frequency of the error correction. If the system performs optimization after every iteration it is called online learning. If the error value is accumulated for multiple iteration before the parametric/structural adjustment it is called batch learning. If the optimization is done only after the whole training iterations - offline learning.

And lastly, the learning can be performed locally, on-chip, or externally, off-chip, where the optimal set of the system parameters is obtained based on the external training hardware or software.

15.4 How "Deep" Neuro-Fuzzy Can Be: Combining Deep Networks and Fuzzy Inference

Neuro-fuzzy systems combines neural networks and fuzzy logic. "Deep" neuro-fuzzy systems are results of combining typical fuzzy and "deep" neural network elements, such as multi-layered structure. This section provides overview of some of the most popular NF architectures.

15.4.1 Adaptive Neuro-Fuzzy Inference System (ANFIS)

The ANFIS and its variations are one of the most popular NF system model. Originally proposed be the Jang [10], this type of architecture combines the structure and learning of the artificial neural networks with fuzzification of the inputs and rule base. Classic ANFIS uses 5 layers. Example of ANFIS network with two inputs and two MFs per input Fig. 15.6, and produced grid segmentation, Fig. 15.7. On a Fig. 15.6 the square neurons (layers 1 and 4) have adjustable parameters, while round neurons are static.

Layer 1

Each neuron in this layer is used to fuzzify corresponding input feature, Eq. 15.5

$$\begin{aligned} Out_{1,i} &= \mu_{A_i}(X),\ where\ i\ is\ node\ with\ X\ input \\ Out_{1,i} &= \mu_{B_i}(Y),\ where\ i\ is\ node\ with\ Y\ input \end{aligned} \tag{15.5}$$

Where μ is the corresponding parametrized MF.

Layer 2

Each neuron in this layer is associated with one certain fuzzy rule, and the produced neuron output is considered as the firing strength of particular fuzzy rule. The output

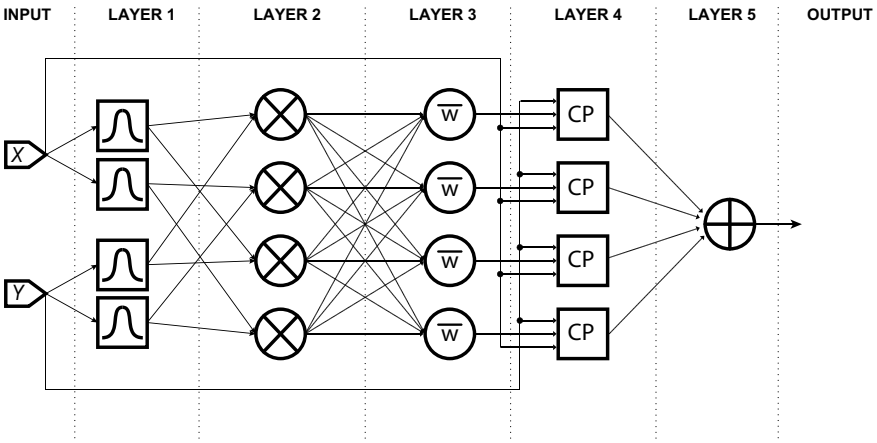

Fig. 15.6 ANFIS network for 2 inputs, X and Y, and two MFs per each input

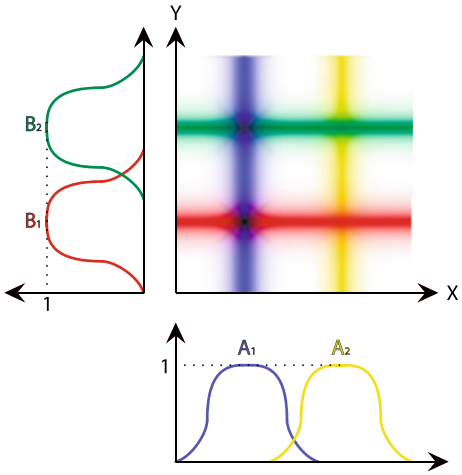

Fig. 15.7 The ANFIS network (Fig. 15.6) produces 4 If-then fuzzy rules that divide the 2D input space into Grid segments

function of this neuron is the product (T-norm or AND) of the corresponding fuzzified input features, Eq. 15.6

$$Out_{2,i} = \mu_{A_i}(X) \times \mu_{B_i}(Y) \tag{15.6}$$

Layer 3

Neurons in this layers are responsible to produce normalized strength of each associated rule, Eq. 15.7

$$Out_{3,i} = \overline{w}_i = \frac{w_i}{\sum_i w_i} \tag{15.7}$$

Layer 4

Each neuron in layer 4 keeps a set of the consequent parameters f_i which are associated with certain rule i. The unfuzzified inputs, normalized rule strength $\overline{w}_i$ and consequent parameters f_i are used to produce the output based on the Tagaki-Sugeno-Kang (TSK) fuzzy inference model, Eq. 15.8.

$$\begin{aligned} &Out_{4,i} = \overline{w}_i f_i, where \\ &\quad f_i = (p_i X + q_i Y + r_i)\ for\ first\ order\ model,\ or \\ &\quad f_i = r_i,\ \ for\ zero\ order\ model \end{aligned} \tag{15.8}$$

Layer 5

Single neuron output layer, sums outputs of the layer 4 neurons, Eq. 15.9.

$$Out_5 = \sum_i \overline{w}_i f_i \tag{15.9}$$

15.4.1.1 ANFIS Advantages

ANFIS architecture allows process both linguistic and numerical features. In addition, ANFIS tends to have fewer parameters to optimize, compared to ANN. Moreover, some consequent parameters can be adjusted by the linear learning algorithms, such as least-squares estimator (LSE), accelerating the learning process. Finally, fuzzy rules based architecture can be more comprehensive to a human operator, so the final approximated system model is more understandable, given the amount of the rules are not overwhelming.

15.4.1.2 ANFIS Disadvantages

The main disadvantage of the ANFIS is the poor scalability with the number of the features and the number of the MFs per feature. The number of the discrete rules required for N input features and M MFs per feature is M^N. This problem is called the Curse of Dimensionality. Majority of the ANFIS variants are designed to address this problem in one way or another, such as SOANFIS or VANFIS.

15.4.2 Fuzzy Neural Networks

Fuzzy neural networks (FNN) are broad category of the hybrid system architectures which combine in one way or another properties of the neural networks and fuzzy systems. One FNN could be a merge between DNNs and Fuzzy inference engine,

such as ANFIS, or it could utilize those elements separately in sequential or parallel fashion, resulting in heterogeneous NFS. With the emergence of the 'deep learning' concept and Deep Neural Networks (DNNs) some researches are introducing FL elements and modules into such systems in order to address the possible uncertainties in the raw data. The combination of the FL and NN can take many forms. It could be sequential architecture, where the fuzzy and neural data processing take place one after another, Fig. 15.8a. It could be parallel - when the data is processed separately be the fuzzy and neural modules separately and then fused, Fig. 15.8b. It could be used as a supportive or control modules of one another, Fig. 15.8c, e.g. when the NN is used to determine the optimal structure, rules, parameters of the fuzzy system, creating a cooperative neuro-fuzzy network. In addition, such systems could classified into two categories homogeneous or heterogeneous. In the first type the neural and fuzzy parts are structurally fused, so it is not possible to separately operate each other. ANFIS is the example of the homogeneous fuzzy neural network. In heterogeneous systems neural and fuzzy parts are separable.

Example of the sequential FNN is the combination of CNN and fuzzy inference. CNNs demonstrate exceptional performance in image classification due to the efficient feature extraction. Since fuzzy systems tend to struggle with large raw data processing, this property of CNNs to preprocess raw data and extract limited number of features from data could be used as antecedent part of deep NFS [11].

Example of parallel FNN could be *Fused Fuzzy Deep Neural Network* (FFDNN) [12]. In this architecture authors split the data processing between fuzzy system and DNN, resulting in adequate performance in data classification.

Example of cooperative FNN is the selection of the mutated neural architecture in genetic algorithms via fuzzy inference for multi criteria structure optimization.

15.4.3 Fuzzy Trees, Patterns and Decisions

Another emerging approach in the field of the neuro-fuzzy machine learning is the implementation of the hierarchical fuzzy trees. Instead of the conventional form of the fuzzy If-Then rule base, which is structured in single-layered parallel form, Fig. 15.9a, the fuzzy trees are typically multi-layered structures of rules with more complex internal dependencies, Fig. 15.9b. In other words, the classical rule base defined by the basic If-Then rules, while fuzzy tree is defined by the complex, multi-layered functional dependencies between fuzzified inputs and output. Compared to the classical rule base, the fuzzy tree can be use for a more compact and possibly more accurate system approximation, where multiple rules can be aggregated and clustered to create one complex rule, or 'branch' of fuzzy tree. As a result, the performance of the fuzzy tree architecture is very sensitive to the structure. There are two major approaches for fuzzy tree structuring. One approach is to use expert knowledge. Like it is done in classical fuzzy systems. Another one is to use data driven algorithms to iteratively build and evolve fuzzy tree structure (structural self adaptation/optimization) until desired performance is reached, while maintaining

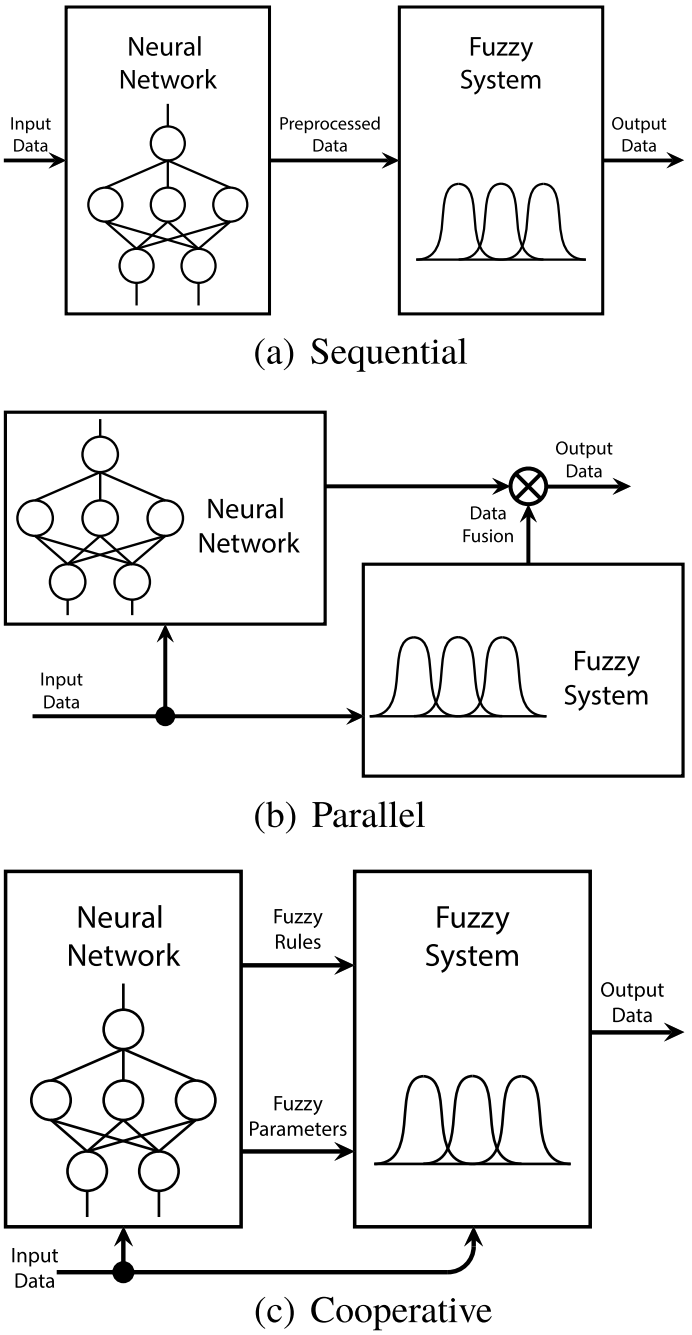

Fig. 15.8 Examples of the Heterogeneous FNNs

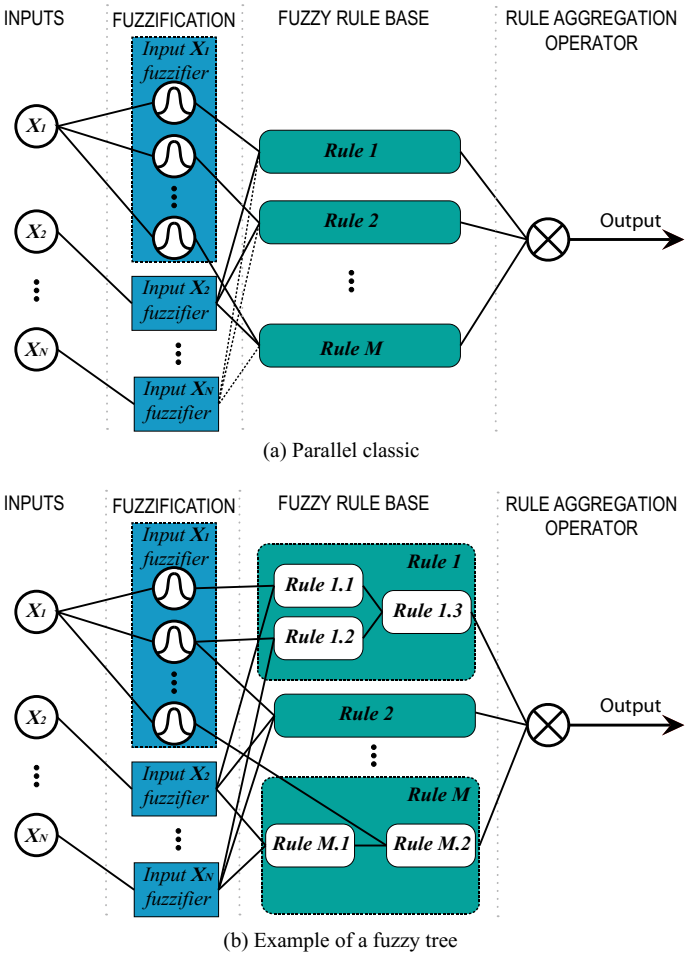

Fig. 15.9 Rule base representation in neuro-fuzzy systems

relatively low structural complexity. In addition, both methodologies could be combined, where expert knowledge can be used to built the core structure of the fuzzy tree and evolutionary algorithms to fine tune smaller branches. The parametric optimization is still important aspect of the fuzzy tree learning, because the fuzzification of the inputs still can be adjusted to increase the performance.

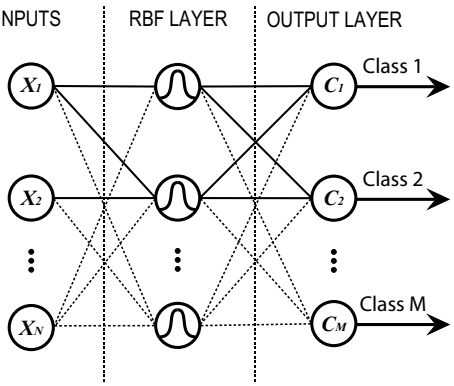

Fig. 15.10 The architecture of the RBF network

15.4.4 Fuzzy Elements in Non Neuro-Fuzzy Architectures

There are some neural architectures that typically not considered as part of neuro-fuzzy group, while using some functional elements of fuzzy system. As a result, such architectures could share some similar core elements with neuro fuzzy systems, especially if hardware implementations is considered. This section will focus on *Radial basis function* (RBF) networks and *Fuzzy Adaptive Resonance Theory Mapping* (Fuzzy ARTMAP).

15.4.4.1 Radial Basis Function Networks

RBF network, Fig. 15.10, is the particular type of the artificial neural networks used for nonlinear data classification. Its structure is similar to the 3 layered Multi-Layer Perceptrons (MLP). The key difference in activation functions in hidden layer: classical MLPs typically use sigmoidal (similar to the S-function on Fig. 15.3), while RBF networks use Bell-shaped functions, such as Gaussian, Eq. 15.2. In classical MLPs hidden layer neurons get activated if weighted inputs and bias sum above activation threshold. In RBFNs, the hidden neurons activates in case of high similarity of input pattern and pattern stored in each RBF neuron, high similarity scores result in stronger activation, Fig. 15.11. Alternatively, weighted input sum and bias could be used in RBFN as well, but it is not considered as classical RBFN.

RBFN uses Gaussian function, which is common fuzzy MF. It could be argued, that other MFs could by used as well.

The training process of the RBFN differs from typical neural networks, because it requires training of the output weights and activation functions parameters. Output weights could be trained with back-propagation algorithms, but such algorithms are not efficient for training of activation functions parameters. Typically, center points and width of each functions are obtained via unsupervised clustering algorithms and output weights trained with stochastic gradient descent (SGD) or linear training

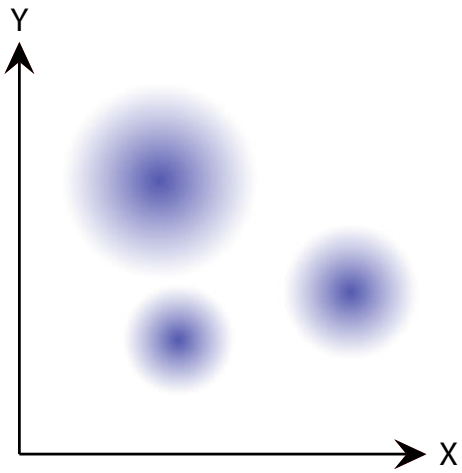

Fig. 15.11 2-D input space segmented by three RBF neurons

algorithms, such as LMS. Such training paradigm is typically faster, compared to use of back-propagation based only algorithms.

15.4.4.2 Fuzzy ARTMAP

Incremental learning is the emerging paradigm of the machine learning, that addresses the problem of the plasticity of learning, finding the balance between learning new data without forgetting the old patterns. Original Adaptive Resonance Theory (ART) architecture was proposed as the unsupervised incremental learning platform of the binary features. Later, originals authors modified the architecture to be able to process the analog features (valued between 0 and 1), introducing the fuzzy ART algorithm. In addition, the ART modules could be adapted for supervised learning, as it has been done in ARTMAP algorithm. If the fuzzy ART modules are used for ARTMAP, it becomes fuzzy as well. The supervised learning is based on the training data pairs, input features and expected results. In fuzzy ARTMAP, Fig. 15.12, one ART_a module is used for input features processing and ART_b module is used for a output mapping. Both modules are connected via map field F_{ab} for matching operation. The first layer F_0 for both modules are used for normalization by complementary coding. In this layer the M-dimensional input vector is concatenated by its complement vector. As the result, the size of the F_0 layer is 2M-dimensional vector. The amount of the neurons in the next layer, F_1 is equals to the size of the concatenated vector from F_1. Final layer, F_2, has one neuron, additional neurons are activated during the training, implementing incremental learning.

Module ART_a is responsible for incremental unsupervised clustering into categories based on the input features, as a result mapping the hyperplane in the M-dimensional input space. Each obtained category T_j is defined by the weight

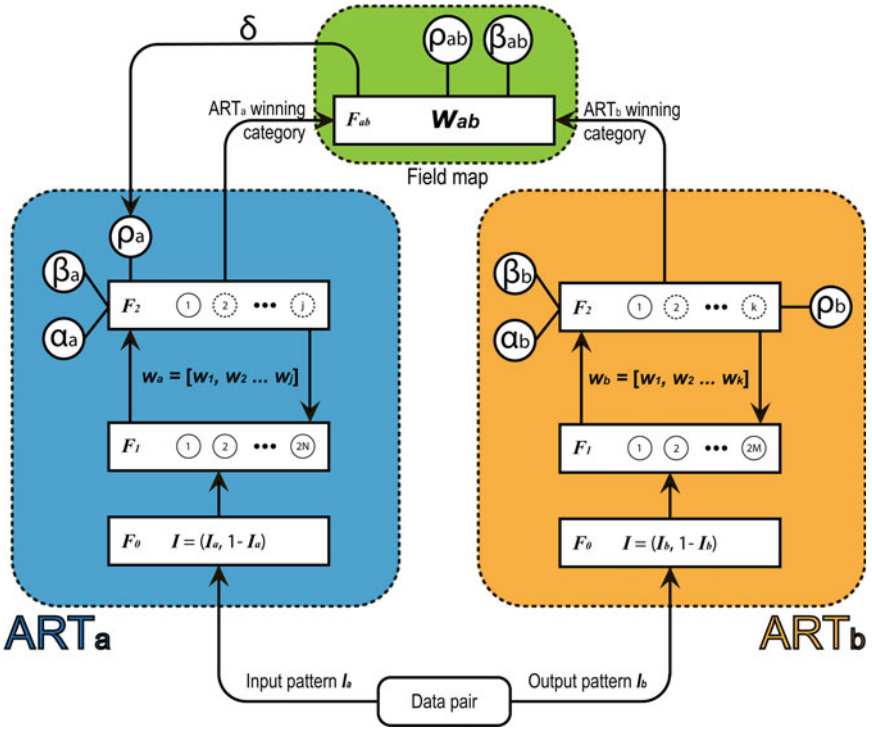

Fig. 15.12 The architecture of the fuzzy ARTMAP algorithm

vector w_j. Initially, each element in weight vector w_j is equal to 1, and the weights are gradually decreasing during the training. The process of the training within ART module is following:

1. *Category selection*:
   For the current input vector I and weight vector w_j the category function is calculated based on the equation:

$$T_j = \frac{||\mathbf{I} \wedge \mathbf{w_j}||}{\alpha + ||\mathbf{w_j}||} \tag{15.10}$$

   The category with the highest value of its respective function is selected.

2. *Vigilance test*:
   For a selected in a previous step category j, the vigilance test is performed to estimate the matching value:

$$\frac{||\mathbf{I} \wedge \mathbf{w_j}||}{||\mathbf{I}||} \geq \rho \tag{15.11}$$

If the given category satisfies this condition, it is selected for learning. In other case, the another category is chosen to undergo the vigilance test. If not existing categories satisfy this test, new category is created for given input pattern and it undergoes further learning.

3. *Learning*:
The selected for learning category gets it weight updated:

$$\mathbf{w_j}^{new} = \beta(\mathbf{I} \wedge \mathbf{w_j}^{old}) + (1 - \beta)\mathbf{w_j}^{old} \tag{15.12}$$

For the given training pair, the input and output are processed by the separate fuzzy ART modules. ART_a module learn the input pattern and makes unsupervised prediction, while map field is used to accept or decline the prediction based on the output of the ART_b module. ρ_b is the vigilance parameter for the output block, so it generates unique ART_b category for each unique output value. As a result, the map field generates the weights matrix w^{ab}, associating the categories between ART_a and ART_b. If the jth category of ART_a and kth category of ART_b are winning in respective modules, map field generates the output x_{ab}:

$$\mathbf{x}^{ab} = \mathbf{y}_k^b \wedge \mathbf{w}_j^{ab} \tag{15.13}$$

Similarly to the ART modules, map field performs it own vigilance test:

$$\frac{||\mathbf{x}^{ab}||}{||\mathbf{y}^b||} \geq \rho_{ab} \tag{15.14}$$

If conditions of Eq. 15.14 are satisfied, the jth category of ART_a is associated with kth category of ART_b and this j-k connection is considered for learning. Otherwise, the vigilance parameter of ART_a is adjusted to find better matching category among existing or creating a new one:

$$\rho_a = \frac{||\mathbf{I}_a \wedge \mathbf{w_j}^a||}{||\mathbf{I}_a||} + \delta, \ where\ 0 < \delta \ll 1 \tag{15.15}$$

Nevertheless, the jth category of ART_a that satisfying the vigilance test of map field undergoes the new weigh training for w^{ab}:

$$\mathbf{w_j}^{ab,\ new} = \beta_{ab}(\mathbf{w_j}^{ab,\ old} \wedge \mathbf{y}^b) + (1 - \beta_{ab})\mathbf{w_j}^{ab,\ old} \tag{15.16}$$

After the training, only ART_a module is used for testing.

15.4.4.3 Fuzzy ARTMAP Advantages

Compared to the traditional machine learning architectures, such as MLP, the fuzzy ARTMAP tends to demonstrate the superiority in terms of the training convergence, simplicity of the required learning, while maintaining the competitive prediction and classification accuracy. In addition, the ARTMAP has an ability to learn with minimal amount of the training data, even in dynamic environment. One of the significant features of the ART based algorithms is the ability to learn new patterns without the requirements to complete retraining. In other words, the ARTMAP is capable to memorize additional patterns, without affecting the already existing ones.

15.4.4.4 Fuzzy ARTMAP Disadvantages

Fuzzy ARTMAP architecture is vulnerable to the category proliferation, the case when the model complexity is directly affected by the input feature complexity. It could be extremely problematic for the cases that requires processing of the noisy data or multitude of the overlapping data. Many alternative versions of this architecture are designed to address exactly this problem. It has been done via optimizing theory, Bayesian decision theory, intelligent clustering or evolutionary algorithms. Additional problem related to the hardware implementation of the fuzzy ARTMAP could be the requirements of the structural flexibility if the on-chip learning considered, because accurate training of this architecture requires dynamically increasing number of neurons in F_2 layer of each ART module. In addition, the training data order affects the learning performance.

15.5 Future Perspective: Dedicated Deep Neuro-Fuzzy Hardware?

The neuro-fuzzy algorithms tends to draw interest of modern researchers, especially in the field system control and data classifications. However, the majority of the neuro-fuzzy solutions exist as software solutions. As concept of Internet-of-Things become more and more popular, the hardware solutions of machine intelligence systems become more favorable, due to the higher power efficiency and processing speed. There are neuro-fuzzy systems architecture implemented as dedicated digital hardware, such as popular nowadays FPGA modules, but mixed/analog hardware tends to be more efficient and faster, with some trade-off in flexibility. The implementation of separate elements of NFS is very popular in the field of mixed/analog implementations. Fuzzification modules/circuits are especially popular. However, there some examples of full NFS implementations. VANFIS architecture has been implemented as mixed signal processor for real-time object detection [4]. Another example utilized analog memristive crossbar arrays for data classification [13]. With

abundance of fuzzy elements analog circuits design and emerging of efficient analog hardware devices, such as memristors, the efficient hardware implementations should be expected in the nearest future.

Chapter Highlights

- The fuzziness of the NFS allows for a more relaxed input data processing.
- There are multiple functions that can be used as MFs.
- Deep learning architectures could be integrated with fuzzy sets and logic in order to introduce automated optimization of neural architectures.
- ANFIS is one of the most popular NF architectures
- FNNs separately use fuzzy and neural elements within one architecture
- Fuzzy trees allow more complex, but compact representation of neuro-fuzzy rule base.
- There are neural architectures that use some of the fuzzy elements, such as RBFNs and fuzzy ARTAMAP.
- Dedicated analog hardware allows efficient implementation of NF algorithms.

References

1. Pandiyan M, Mani G (2015) Embedded low power analog CMOS fuzzy logic controller chip for industrial applications. In: 2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp 43–48
2. Juang C, Chen C (2014) An interval type-2 neural fuzzy chip with on-chip incremental learning ability for time-varying data sequence prediction and system control. IEEE Trans Neural Netw Learn Syst 25(1):216–228
3. Juang C, Juang K (2017) Circuit implementation of data-driven TSK-type interval type-2 neural fuzzy system with online parameter tuning ability. IEEE Trans Ind Electron 64(5):4266–4275
4. Oh J, Lee S, Yoo H (2013) 1.2-mW online learning mixed-mode intelligent inference engine for low-power real-time object recognition processor. IEEE Trans Very Large Scale Integr (VLSI) Syst 21(5):921–933
5. Merrikh-Bayat F, Merrikh-Bayat F, Shouraki SB (2014) The neurofuzzy computing system with the capacity of implementation on a memristor crossbar and optimization-free hardware training. IEEE Trans Fuzzy Syst 22(5):1272–1287
6. Yaghmourali YV, Fathi A, Hassanzadazar M, Khoei A, Hadidi K (2018) A low-power, fully programmable membership function generator using both transconductance and current modes. Fuzzy Sets Syst 337:128–142
7. Abolhasani A, Tohidi M, Mousazadeh M, Khoei A, Hadidi K (2013) A high speed and fully tunable MFG with new programmable CMOS OTA and new MIN circuit. In: Proceedings of the 20th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2013, pp 169–173

8. Ghasemizadeh H, Fathi A, Ahmadi A (2012) Programmable fuzzifier circuits with high precision for analog neuro-fuzzy system. In: 20th Iranian Conference on Electrical Engineering (ICEE2012), pp 12–16
9. Afrakoti IEP, Shouraki SB, Bayat FM, Gholami M (2017) Using a memristor crossbar structure to implement a novel adaptive real-time fuzzy modeling algorithm. Fuzzy Sets Syst 307:115–128
10. Jang JR (1993) Anfis: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 23(3):665–685
11. Bonanno D, Nock K, Smith L, Elmore P, Petry F (2017) An approach to explainable deep learning using fuzzy inference. In: Next-Generation Analyst V, vol 10207, p 102070D. International Society for Optics and Photonics
12. Deng Y, Ren Z, Kong Y, Bao F, Dai Q (2017) A Hierarchical fused fuzzy deep neural network for data classification. IEEE Trans Fuzzy Syst 25(4):1006–1012
13. Afrakoti IEP, Shouraki SB, Bayat FM, Gholami M (2016) Using a memristor crossbar structure to implement a novel adaptive real-time fuzzy modeling algorithm. Fuzzy Sets Syst 307:115–128