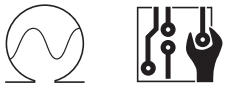
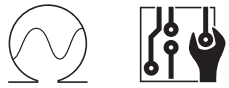


Technische Universität München
Department of Electrical Engineering and Information Technology
Chair of Electronic Design Automation

Aging-aware Lifetime Enhancement for Neuromorphic Computing

Master Thesis

Shuhang Zhang



Technische Universität München
Department of Electrical Engineering and Information Technology
Chair of Electronic Design Automation

Aging-aware Lifetime Enhancement for Neuromorphic Computing

Master Thesis

Shuhang Zhang

Supervisor :	Li Zhang
Supervising Professor :	Prof. Dr.-Ing. Ulf Schlichtmann
Topic issued :	26.02.2018
Date of submission :	21.08.2018

Shuhang Zhang
Arcisstr. 21
80333 Munich

Contents

List of Figures

List of Tables

1. Introduction

1.1. Limitations of Von Neumann Architecture

In the past decades, the von Neumann architecture [VN93] has become the foundation of today's computing systems. However, due to data explosion and high data processing requirements in recent years, this architecture has become incompetent because of its limitations in processing huge amount of data.

This limitation is caused by the separate allocation of processor and memory. Fig 1.1 shows the von Neumann architecture, where processors (Central Processing Unit) and memory are separated. In this architecture, data transferring is required between processors and memory when executing programs, leading to unavoidable latency. Although many methods have been proposed to reduce the latency of data transferring, it is still impossible to eliminate this latency and it has become the bottleneck of von Neumann architecture gradually [Bac07] because of the rapidly increasing data size.

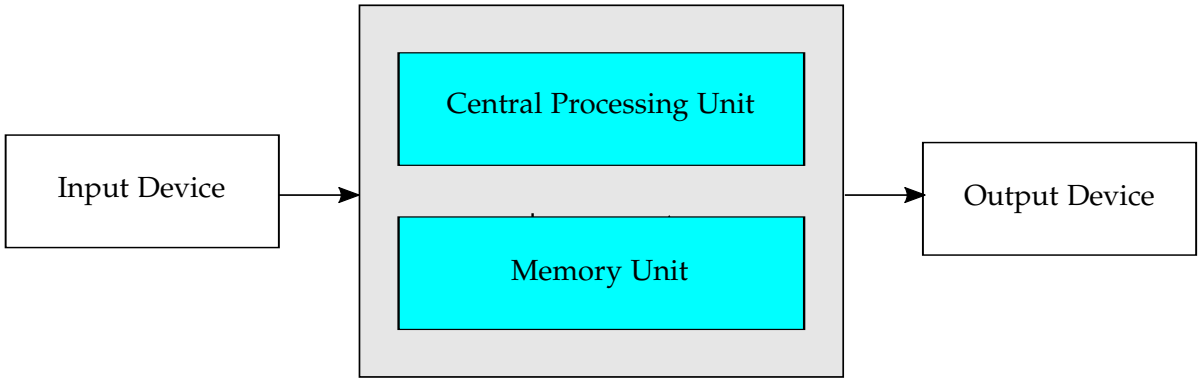


Figure 1.1.: Von Neumann architecture[VN93]

.

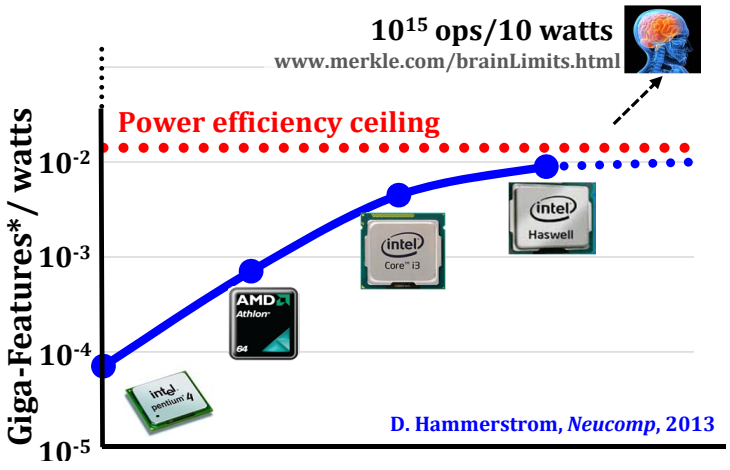


Figure 1.2.: Power efficiency ceiling.

With the development of semiconductor technology in the past decades, processor performance has been improved significantly. At the same time, the improvements of memory are mainly in density instead of data transferring speed. Therefore, the performance gap between processors and memory is becoming more pronounced. As a result, processors must spend more and more time waiting for the data transferring, which degrades the performance of von Neumann architecture significantly.

To deal with the data-heavy processing requirements, some specific hardware designs have been used, e.g., Graphic Processing Unit (GPU) and Field Programmable Gate Array (FPGA). However, these methods cannot break the bottleneck of data transferring between processors and memory. Consequently, a novel architecture is needed to remove the limitation and improve the computing capability.

Besides the data transferring limit in the von Neumann architecture, the power efficiency is also becoming one of the main concerns of modern chip design. Power consumption also increases rapidly with the improvement of processor performance. As a result, the power efficiency of processors is approaching a ceiling, as shown in Fig 1.2.

The horizontal axis represents years and vertical axis represents giga-features per watt. Although the power efficiency of main stream processors continue to grow, the growing speed is becoming slower rapidly in recent years and is hardly to be improved in the following years. However, as shown in Fig 1.2, the power efficiency of human brain is several orders

of magnitude higher than the power efficiency of state-of-the-art processors. Therefore, it might be a solution to build neuromorphic computing systems to emulate the behavior of human brain and thus to achieve a better power efficiency.

1.2. Neuromorphic Computing

To deal with the limited data transferring rate and constrained power efficiency, neuromorphic computing concept was proposed by Carver Mead in late 1980s [Mea90]. Neuromorphic computing inspired by human brain aims to mimic neural systems. In neural systems, there is no central processing unit. The basic element of neural systems, a neuron, can process and transfer information via synapses independently, so neural systems are highly parallel. These neurons do not transfer signals to neurons in the next layer all the time, but only when the accumulated excitations reach a threshold value. In this way, the neural systems have better power efficiency than current computing architecture. Therefore, by implementing neural systems with existing technologies, the computing systems can benefit from the advantages of neural systems, e.g., high-level parallelism and extremely high power efficiency.

The hardware-level implementations of neuromorphic computing realized by transistors have been published, such as TrueNorth by IBM in 2014 [MAAI$^+$14]. TrueNorth neuromorphic CMOS integrated circuit is a manycore processor network on a single chip. TrueNorth chip has 4096 cores. Each core has the capability to simulate 256 neurons. Each neuron has 256 synapses responsible for transferring information between simulated neurons.

Although TrueNorth is manufactured with traditional CMOS technology, the method of assembling transistors in TrueNorth is totally different. In TrueNorth, transistors arrangement is changed from concentric to parallel pattern. Thus, it breaks classic von Neumann architecture, leading to high data throughput and low power consumption. To implement neuromorphic computing with CMOS technology, billions of transistors have been integrated in TrueNorth chip and one of the biggest chips ever manufactured. Due to the large chip size and high manufacturing cost, it is however not practical to apply it widely.

The effort in reducing the size of neuromorphic chips did not make significant process until 2008. In this year, the memristor, which was predicted by Leon Chua and considered as the

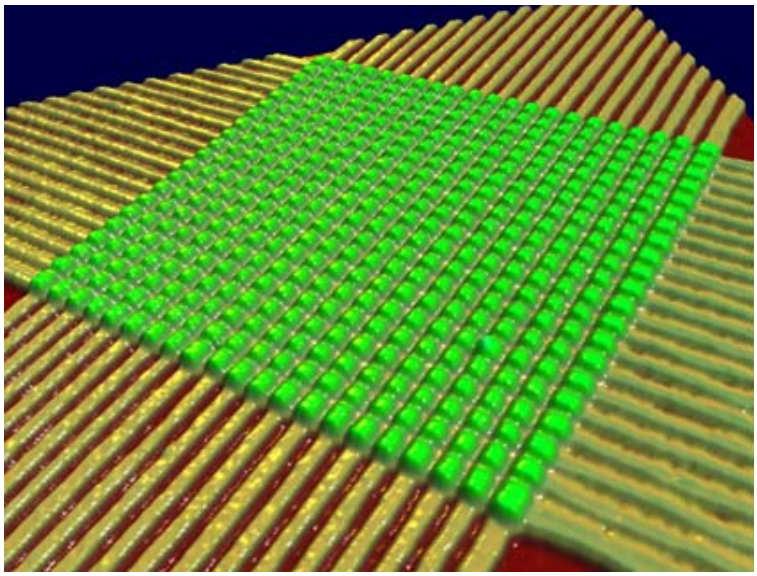


Figure 1.3.: Memristor crossbar architecture[QPMS11].

missing forth basic element [Chu71], was discovered by HP lab [SSSW08]. The resistance of a memristor can be changed, depending on the historical current flowing through it. In addition, another useful feature of memristor is its high scalability, which makes it possible to reduce area of neuromorphic chips.

Previous work (TrueNorth) uses digital circuits to build neuromorphic chips and thus results in large chip area overhead. The existence of memristor provides an alternative implementation based on analog circuits. If neurons can be modeled as voltage signals, the memristors together with current-to-voltage converter allocated between these neurons can adjust the signal transferring strength to mimic the behavior of synapse, which controls a neuron to transfer an electrical or chemical signal to the next neuron in neural systems. Based on this assumption, the neural systems can be implemented by memristors and corresponding analog devices.

Fig 1.3 shows the simplest neural systems, containing one layer pre-neurons, one layer post-neurons and the synapse connections. In this figure, pre-neurons are represented as the voltage signals applied on the rows of the matrix, the post-neurons are represented as the voltage signals on the columns, and the synapses connecting pair of pre-neuron and post-neuron are implemented by memristors which are placed between each row and column.

Based on the architecture above, more complex neural systems can be constructed by using multiple crossbar matrices and connecting them layer by layer. In this way, huge number of transistors are avoided. Due to the similarities between memristor and synapse, this analog implementation with memristor for neuromorphic computing can be high power efficiency. Furthermore, memristor has the advantage in scalability, so neuromorphic chip area can be scaled down significantly.

2. Circuits for Neuromorphic Computing

2.1. Memristor Device

Memristor has been predicted by Leon Chua in theory for more than 40 years [Chu71]. In this paper, he pointed out a conceptual symmetry between voltage, current, charge and flux and basic circuit components, resistor, capacitor and capacitor can be used to connect these four electrical elements. In Fig 2.1, the relation between voltage and current can be represented using a resistor. The inductor can represent the relation between flux and current. The capacitor connects voltage and charge. However, the representation for the relation between flux and charge is missing. In Leon Chua's theory, memristor should exist and connect flux and charge. As there was no evidence proving the existence of memristor at that time, the memristor was considered as the missing forth basic circuit element in the following decades.

Due to the lack of experimental results proving the existence of memristor, Leon Chua's theory was not widely accepted at that time. However, a team at HP claimed that they

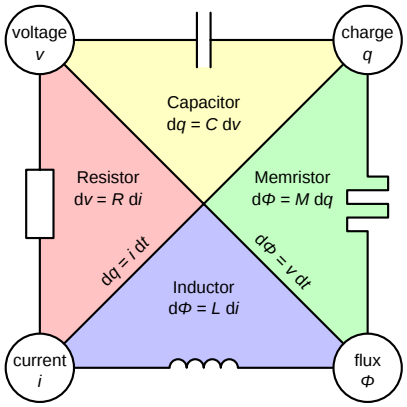


Figure 2.1.: Conceptual symmetries of resistor, capacitor, inductor and memristor.

found memristor in 2008 [SSSW08]. Based on the analysis of a thin film of titanium dioxide, this team relates the operation mechanism of Resistive Random Access Memory (ReRAM) with the memristor concept. For ReRAM, the resistance is determined by the history applied voltage across the device. For memristor, although it connects flux and charge, it should also reflect the non-linear relation between voltage and current, which is similar to the working mechanism of ReRAM. Afterwards, Leon Chua argued that all ReRAMs are memristors [Chu11]. However, there are some doubts about the existence of memristor [VM15] and some experimental results contradict Leon Chua's generalization [VLT+13].

The discussion over the existence of memristor could be continued in the following years, but without doubt, the memristor (ReRAM) changes existing circuit design methodology and revolutionize neuromorphic computing design.

2.1.1. Memristor Physical Mechanisms

With more effort put into memristor research, two main working mechanisms of memristors are proposed. The first one is dopant-drifting-based mechanism. The other one is filament-based mechanism. In this section, these two mechanisms will be discussed separately.

Dopant Drifting Mechanism

According to a paper from HP [SSSW08], the physical model of memristor is based on dopant drifting mechanism. Other research groups have also proposed their models based on the same mechanism [PPPT11, ZCJY+13]. Fig 2.2 shows the conceptual model of a memristor. Memristor is a two-terminal device, with dioxide materials between two metal electrodes. If a voltage is applied on it, current flow is generated and the ratio of doped and undoped volume is changed accordingly, leading to different resistances. Therefore, the resistance of memristor can be expressed in Equation 2.1.

$$R_{memristor} = w * R_{doped} + (1 - w) * R_{undoped}. \tag{2.1}$$

In Equation 2.1, R_{doped} and $R_{undoped}$ represent doped resistance and undoped resistance respectively and w is the state variable indicating the doping condition. If w equals 0, the

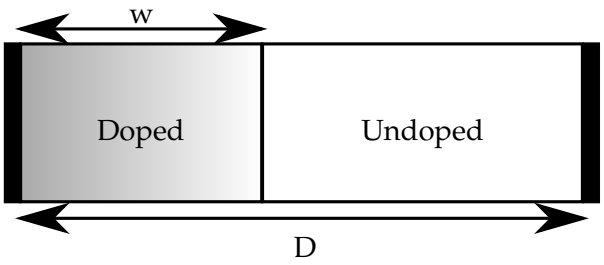


Figure 2.2.: Memristor dopant-drifting-based physical model[SSSW08].

whole memristor can be considered as fully undoped and the memristor reaches the highest resistance. If w equals 1, the whole memristor is fully *doped*, reaching lowest resistance.

Filament-based Mechanism

The second working mechanism of memristors is the filament-based mechanism of memristors. Some research groups examined the cross-sectional area and observed that not all cross-sectional area in a memristor is used. They found only a small portion of it contributes to the memristor resistance [LGY+17]. To explain this phenomenon, the filament-based memristor physical models are proposed by different groups [ZGY+15, LJH+15, Iel11]. Fig 2.3 shows the filament-based physical model. In the proposed model, the memristor resistance is a combination of conductive filament and the gap between filament and electrode. Therefore, the resistance can be expressed in Equation 2.2.

$$R_{memristor} = R_{filament} + R_{gap} \tag{2.2}$$

If a voltage is applied across the memristor, the generated electrical field in memristor constructs or ruptures the filament depending the direction of applied voltage, resulting in a change in the resistance of memristor. If the filament is fully ruptured, the memristor reaches the highest resistance and if the filament is formed fully, the memristor resistance is at the lowest value.

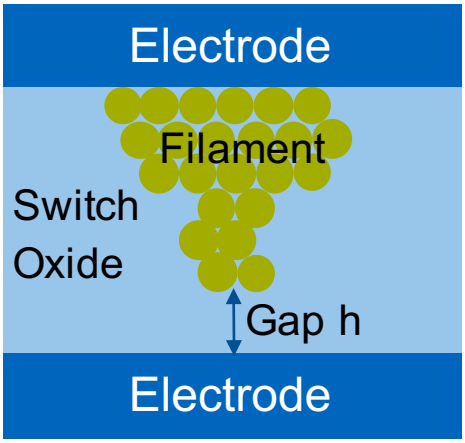


Figure 2.3.: Memristor filament-based physical model.

Comparison between Dopant-drifting-based and Filament-based Mechanisms

The two working mechanisms of memristor above are still under debate, but both mechanisms agree on the fact that applied voltage changes the memristor resistance. In Fig 2.4, the resistance switching mechanism using memristor model proposed in [ZGY+15] is presented. In this figure, the pulses are the applied voltage and the curve is memristor resistance. When a positive voltage is applied, the memristor resistance starts to decrease. On the contrary, when a negative voltage is applied, the memristor resistance is reset to high values gradually. Based on this feature, memristor resistance can be set to a specific value and will keep it if no voltage applied on it, because the resistance is determined by the history of applied voltage.

2.1.2. Memristor Applications

This non-volatile and tunable resistor can be used in various kinds of applications. Two main applications of memristor are memory cell and neuromorphic computing unit, because memory can use different resistance values to represent different information and tunable resistance can be used to mimic changeable synapse strength in neural systems.

When memristor is used as memory, memristor is also called as Resistive Random Access

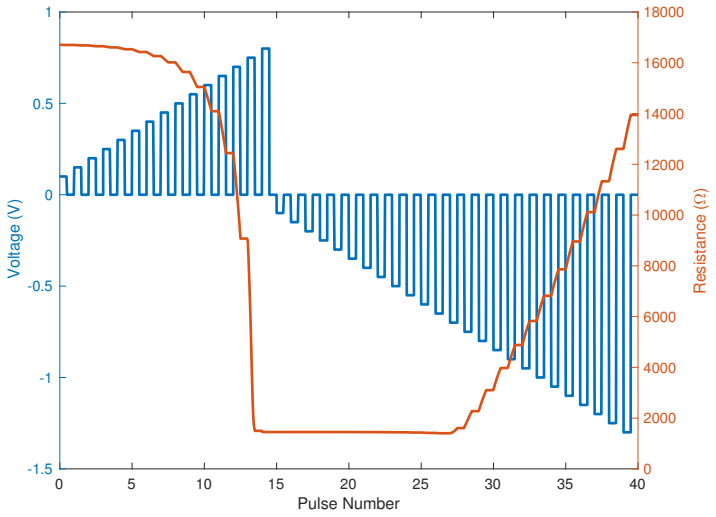


Figure 2.4.: Memristor resistance changing.

Memory (ReRAM). The highest and lowest resistance states of a memristor can represent 0 and 1 correspondingly, so that one memristor device can store 1bit information. However, with the progress in memristor programming technology, different resistance levels can be achieved to represent different logical levels, leading to multiple bits stored in only one memristor cell, which improves memory density significantly.

The other important application of memristor is used to build neuromorphic computing systems due to the similar behavior between memristor and synapse in neural systems. The state of memristor is affected by the applied voltage and synapse state is also influenced by the neurons information. Therefore, memristor is a promising candidate for hardware implementation of neuromorphic architecture.

2.2. Memristor-based Crossbar Architecture

A neural system consists of numerous neurons and synapses. The synapses connect neurons by transferring electrical or chemical signals between these neurons, so that these neurons and synapses can work together to implement specific functions. Fig 2.5 shows a simplified natural system, including only one layer pre-neurons, one layer post-neurons and one

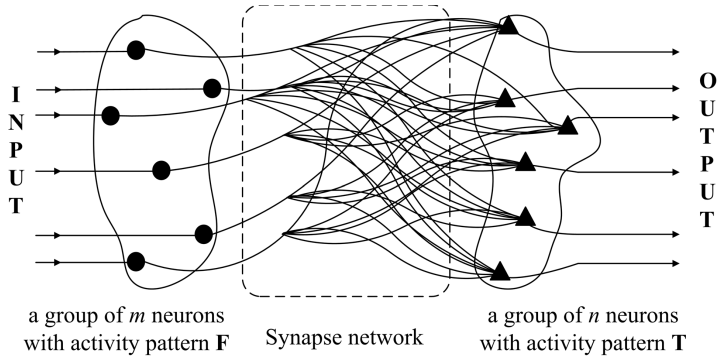


Figure 2.5.: Natural neural connection[HLC+13].

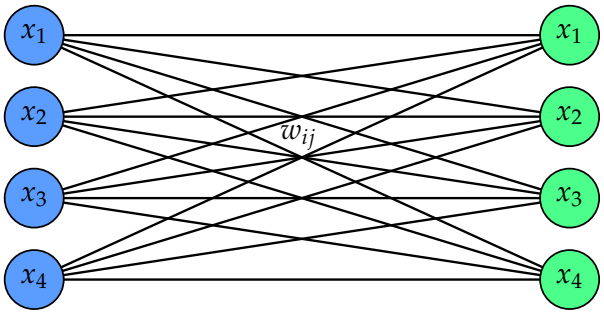


Figure 2.6.: Simplified representation of neural connections.

synapse network between these pre-neurons and post-neurons. The information from pre-neurons of the previous layer is processed by the synapse network and delivered to the next layer of neurons.

To simplify the representation, this synapse-based connection is modeled as shown in Fig 2.6. In this simplified structure, these pre-neurons are represented by an input vector x. The post-neurons are simplified as an output vector y. The synapse connections between neurons are abstracted to a weight matrix. In this way, the relation between pre-neurons and post-neurons can be expressed as follows.

$$y = x * W. \tag{2.3}$$

The information received by each post-neuron is affected by every pair of pre-neuron and the corresponding weight of synapse. Therefore, the complex neural system can be transformed into a mathematical matrix-vector multiplication, which reduces the modeling complexity of neural systems significantly.

As introduced in the previous section, neural systems can be implemented by memristor-based crossbar architecture, because of the similarities between memristor and synapse. According to the way to encode input signals, there are two types of implementations, level-based design [HSL+16] and spiking-based design [LYY+15]. In both designs, the crossbar architecture is used. The differences are encoding methods of input signals and the current conversion mechanisms.

2.2.1. The Level-based Design

Fig 2.7 shows the level-based crossbar architecture. In this design, the amplitude of voltage is used to represent original information. A higher voltage means a larger number in mathematical representation and vice versa. The synapse between input and output vectors can be represented by the memristor crossbar architecture. In this architecture, the input voltage vector V_{in} is applied on the rows of crossbar architecture, thus generating currents flowing through each memristor. According to Kirchhoff's current law, the currents of the memristors in the same column are summed. Then the summed current is converted to voltage values V_{out}. In Equation 2.4, V_{in} and V_{out} represent input and output vectors respectively, and M represents the memristor-based connections between input and output vectors.

$$V_{out} = V_{in} * M. \tag{2.4}$$

The representation between input and output voltages is the same as the mathematical representation of vector-matrix multiplication, so the memristor-based architecture can implement vector-matrix operations effectively.

In this architecture, the summed current in each column is transformed to voltage by a transimpedance amplifier. Then, this voltage is converted to digital signal by an Analog to Digital Converter (ADC) for future processing, such as shifting and scaling. Afterwards, the processed digital signals are converted to analog signals with a Digital to Analog Converter (DAC), which will be used as input signals for the next crossbar architecture.

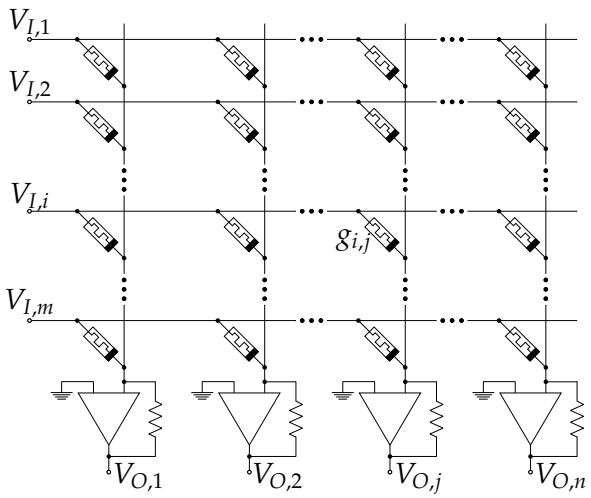


Figure 2.7.: Level-based memristor crossbar architecture.

2.2.2. The Spiking-based Design

In Fig 2.8, spiking-based design is shown. In this design, the density of voltage pulses instead of amplitude is used to represent original information. More dense pulses mean a larger number in mathematical representation and vice versa.

In this architecture, an Integrate and Fire Circuit (IFC) is used to convert currents to digital signals directly. The summed current generated in each column charges the capacitor in IFC, and generates a pulse signal when the voltage across the capacitor is higher than the threshold voltage value (V_{ref}). Afterwards, a counter is used to measure how many pulses are generated, which can reflect the current strength in an alternative way.

2.2.3. Comparison of Two Approaches

For level-based approach, it is compatible with current circuit design tool chain, because the designs of ADC and DAC are mature, so level-based design can be integrated in current design flow without any further design effort. In addition, the computing speed of level-based architecture is usually higher than spiking-based approach, because the level-based approach does not need to charge and discharge capacitor, which are time-consuming. However, the adoption of ADC and DAC usually leads to a high area overhead, which increases the chip

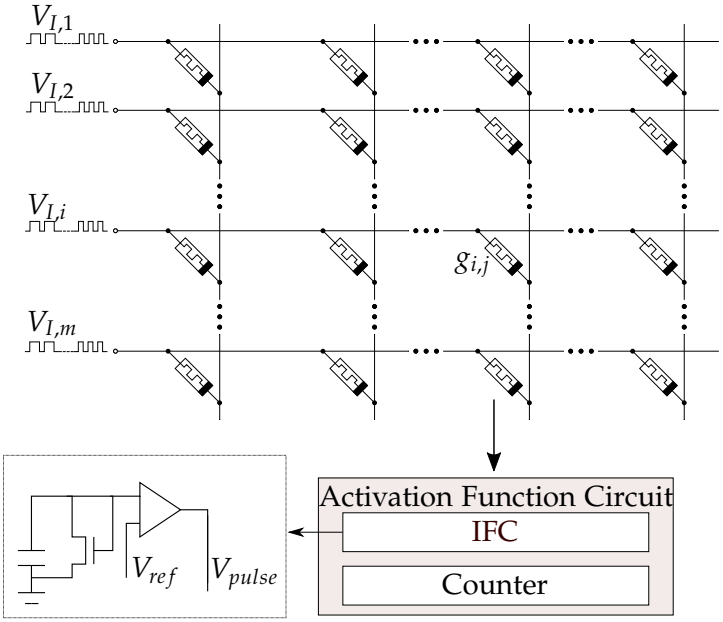


Figure 2.8.: Spiking-based memristor crossbar architecture.

manufacturing cost.

For spiking-based approach, it removes the ADC and DAC from the architecture, which usually take a lot of chip area. Another advantage of spiking-based neuromorphic architecture is that spiking-based behavior is closer to biological systems, because neurons only start to transfer information when enough energy from previous neurons is accumulated. In this scenario, the spiking-based architecture does not need to work all the time, and it only works when accumulated voltage reaches a threshold value, which can further improve the power efficiency of neuromorphic computing. However, it is essential to design the IFC carefully, because it affects the accuracy of the computing architecture significantly.

In conclusion, it is hard to evaluate which approach is better, because each approach has its own advantages and disadvantages. Both approaches can be used to implement neuromorphic computing architecture effectively, so that the choice depends on the detailed specifications.

Both designs have proved that the memristor-based crossbar architecture can be used to implement neuromorphic computing effectively. Therefore, memristor-based architecture will be the foundation of neuromorphic computing systems.

2.3. Deep Neural Networks

Memristor-based neuromorphic computing architecture has shown high efficiency in processing vector-matrix multiplications, so that the neuromorphic computing architecture can be used as a hardware accelerating framework and applied in deep learning area to relax the higher computing requirements pressure caused by large data size and complex algorithms.

With deep learning developing rapidly in recent years, deep neural networks, as one of the main deep learning architectures, have been applied in numerous fields, e.g., computer vision, speech recognition and natural language processing and has achieved remarkable performance.

A deep neural network usually consists of multiple layers which are responsible for transferring and processing information between input and output layers [B+09]. In a deep neural network, the data fed into the deep neural network are processed and transferred by the previous layer to the next layer. At the last layer, a probability is calculated as the output result.

Among deep neural networks, the convolutional neural networks (CNNs) have attracted the most attention from researchers, because these convolutional neural networks applied in pattern recognition have made a huge progress and the performance of CNN is already comparable with that of human experts in distinguishing objects in pictures. The following sections will focus on analysis of CNNs.

2.3.1. Basic Layers in Convolutional Neural Networks

In Fig 2.9, a typical convolutional neural network is presented. It consists of multiple types of layers, such as convolution layer, pooling layer and fully-connected layer. The convolution layers are used for feature extractions. The pooling layer is used to reduce extracted feature size. The fully-connected layer is usually arranged at the end of these neural networks, which merges the extracted information and performs the classifications. A deeper convolutional neural network usually stacks more convolutional layers followed by pooling layers. By

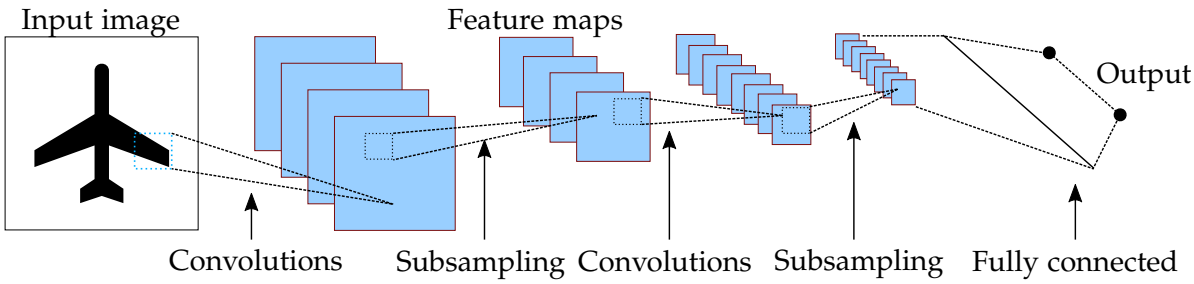


Figure 2.9.: Typical convolutional neural network architecture.

using pooling layers repeatedly, the size of data is reduced. Afterwards, the reduced data image is fed into fully-connected layers. The fully-connected layers generate final results of this neural network. Therefore, a common expression of convolutional neural network can be represented as follows.

$$INPUT- > [CONV * N- > Pool] * M- > FC * K \tag{2.5}$$

In Equation 2.5, $INPUT$ represents input, $CONV$ indicates convolutional layer, $Pool$ represents pooling layer and FC means fully-connected. The N, M and K used in Equation 2.5 represent corresponding number of layers, respectively.

Fig 2.10 shows a convolutional layer, which is used to extract the features from input images. In a convolutional layer, multiple filers are used, which are also known as kernels. These kernels represent different features. Then, the sum of element-wise multiplications between kernels and same size block of the input image is calculated. The larger the summed value, the more similarities shared by the kernel and the block from the image and vice versa. This type of calculation is carried out continually until all blocks from the image are compared with these kernels. Consequently, these kernels can extract useful features of input images for further processing in the next layer. These kernels are the most important part in convolutional layers, so that the decision of the kernels' size usually requires careful design and experiments.

In Fig 2.11, a pooling layer is presented, which is used to reduce the computing complexity. The pooling layer is placed after a convolutional layer. The extracted features from input images by convolutional layers share spatial similarities, so that down-sampling extracted patterns does not affect the accuracy of a neural network and reduce the processing complexity significantly. There are several ways to implement pooling and the most common

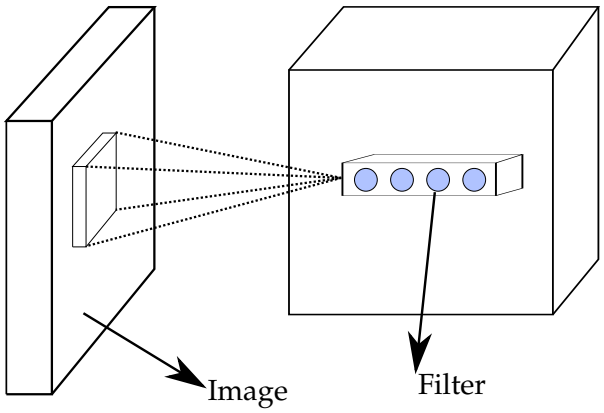


Figure 2.10.: Convolutional layer.

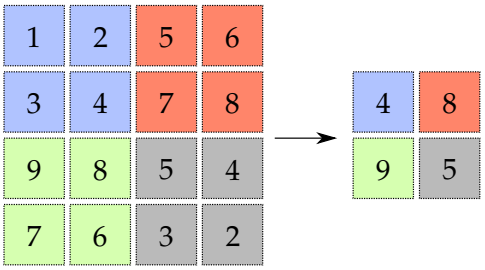


Figure 2.11.: Pooling layer.

one is max-pooling. As shown in Fig 2.11, every 2×2 block is down-sampled and this block is represented by the max value of it.

In Fig 2.12, a fully-connected layer is shown. Located at the end of the network, features have been accumulated and these features are fed into fully-connected layers for the final processing.

2.3.2. Convolutional Neural Network Training and Inference

Training of Neural Networks

In the previous sections, the architecture of convolutional neural networks has been introduced. In a neural network, the parameters in convolution layers and fully-connected layers, e.g., the kernels' weights in convolutional layers and weights in fully-connected layers, are

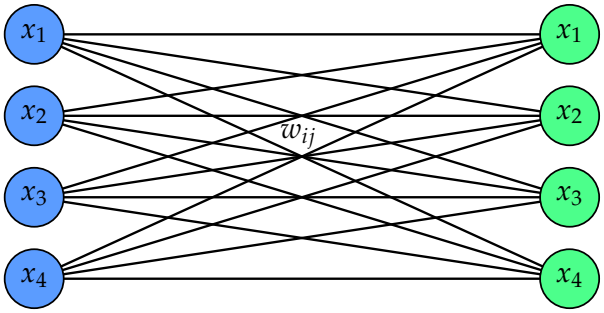


Figure 2.12.: Fully-connected layer.

critical for the accuracy of the neural network, but these optimal parameters are unknown with an untrained neural network. Therefore, the training process is necessary for neural network to find the optimal parameters.

At the beginning of training, all parameters are randomly initialized. With an initialized neural network, the accuracy of this neural network is usually low and cannot be used in any applications directly. To increase the accuracy of the neural network, training is required. Training data is fed into the neural network and corresponding correct labels are compared with the real outputs. The differences between correct labels and real outputs are considered as the cost generated by the neural network. Therefore, if the cost can be minimized, a high accuracy of neural network is obtained.

To evaluate the amplitude of neural network cost, a cost function is needed. Different kinds of cost functions can be used in training phase, e.g., square cost and cross entropy. The square cost function can be expressed in Equation 2.6, where y represents the actual output generated by neural network and y_{label} indicates the correct labels. Accordingly, reducing the cost is equivalent to push the output to the correct value as close as possible.

$$Cost = (y - y_{label})^2. \tag{2.6}$$

Another widely used cost function is cross entropy as shown in Equation 2.7.

$$Cost = -(y_{label} \log y + (1 - y_{label}) \log(1 - y)). \tag{2.7}$$

If correct label y_{label} is 0, the $Cost$ is simplified to $-\log(1 - y)$. If y generated by neural network is close to y_{label}, the cost is very small, but if the y is far away from 0, the cost

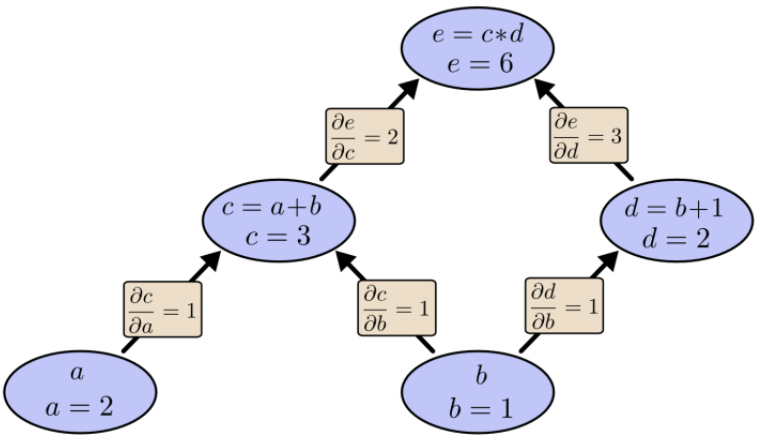


Figure 2.13.: Example for backpropagation algorithm.

becomes huge because of the log function. The other scenario is label y_{label} equals 1. Then Equation 2.7 can be simplified to $-\log y$. Similarly, if y is close to y_{label}, the cost is small, but if y is far away from y_{label}, the cost is huge.

These cost functions can not only evaluate the performance of neural networks, but also be used to update parameters of neural networks to improve its accuracy. This updating mechanism is mainly based on backpropagation algorithm, which has been used in zip code recognition [LBD+89].

In this algorithm, the parameters (weights in different layers) in the neural network need to be updated from the last layer to the first layer. In the backpropagation process, a specific updating method, gradient decent, is used.

To illustrate the effectiveness of the backpropagation method, a simple example to calculate the derivatives of $e = (a+b)*(b+1)$, is shown in Fig 2.13. In this equation, two intermediate variables c and d are introduced, representing $(a+b)$ and $(b+1)$ respectively. When $a = 2$ and $b = 1$, c, d, and e can be calculated and is equal to 6. With these values, the derivative $\frac{\partial e}{\partial c}$ can be calculated, which is 2 and the derivative $\frac{\partial e}{\partial d}$ is calculated, which equals 3. Afterwards, the derivatives, $\frac{\partial c}{\partial a}$, $\frac{\partial c}{\partial b}$ and $\frac{\partial d}{\partial b}$ can be calculated by using $\frac{\partial e}{\partial c}$ and $\frac{\partial e}{\partial d}$. Finally, the derivatives $\frac{\partial e}{\partial a}$ and $\frac{\partial e}{\partial b}$ can be expressed in Equation 2.8 and Equation 2.9

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a} \tag{2.8}$$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}. \tag{2.9}$$

It is can be noticed that the backpropagation is actually based on the chain rule and this algorithm visits each node only once, which reduces the training complexity significantly. By using this algorithm, the cost is propagated to previous layers and is used to calculate derivatives and update weight parameters in each layer. These updated weights lead to lower cost, thus, improving the neural network accuracy.

In the previous paragraphs, the backpropagation algorithm is introduced. However, by using only the cost entropy function, e.g. cross entropy function, the neural networks often get very high accuracy on the training data, but a low accuracy on the test data. This phenomenon indicates the neural network fits the training data too over, which is called over-fitting problem. In this case, the neural network has lost the generality and is limited to the training data. To deal with the over-fitting problem, a regularization function is usually appended to the cost function. Thus, the new cost function can be expressed in Equation 2.10.

$$Cost = CrossEntropy + Regularization \tag{2.10}$$

In this equation, the CrossEntropy can be considered to modify the weights for getting better accuracy and the Regularization is used to control the generality of the neural works to prevent over-fitting problem.

The regularization function used in training process is usually a L2-norm function, which can be expressed in Equation 2.11. In this way, the generality of the neural network is abstracted to the norm of weight matrix. Larger the values of the weight matrix are, the less the general of the neural network is.

$$L2\_norm = \|W\|^2 \tag{2.11}$$

Therefore, in this way, the full cost function can be expressed in Equation 2.12, where the first part is the normal cross entropy and the second part is the regularization function.

$$Cost = -(y_{label} \log y + (1 - y_{label}) \log(1 - y)) + \lambda \|W\|^2 \tag{2.12}$$

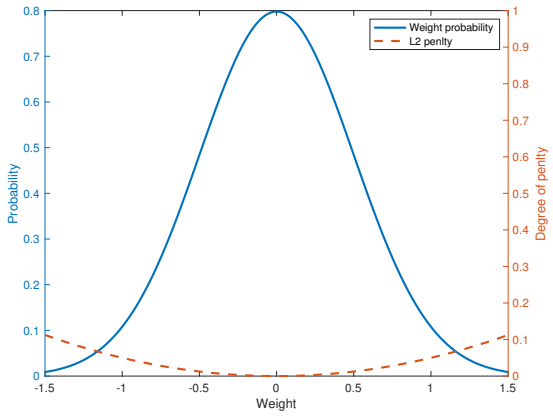


Figure 2.14.: Traditional training method and weight distribution.

Fig 2.14 shows the distribution of trained weights and corresponding L2 regularization. There two curves in this figure. The solid curve represents the conceptual distribution of trained weights and the dashed curve indicates the L2 regularization. As shown in Fig 2.14, the weights in the neural network after training usually satisfy a quasi-normal distribution, where most of the weights are close to zero. The L2 regularization pushes the weights to the center from both sides. In this way, too large weights can be avoided, leading to better generality of the neural network.

Inference of Neural Networks

The inference is much simpler than the training process. With trained parameters, input images are fed into the whole network and corresponding probabilities are calculated at the output layer. These probabilities can be used to do classifications.

2.3.3. Memristor-based Convolutional Neural Network Training

As mentioned in the previous sections, the training and inference phases require huge amount of vector-matrix multiplication operations. If the memristor-based crossbar architecture is used to implement the training and inference processes online, the power efficiency and speed can be improved significantly, especially for the training process, because

the training process requires more computing resources than the inference process. Online training can be performed fully on memristor crossbars, or after software training. The former is called hardware online training. The later is called software training together with online tuning.

Hardware Online Training

The training process implemented on memristor crossbar is similar to that of software training. Several methods implementing the hardware training process have been published [SDCG+15, HLC+13, LWW+14]. In these methods, the gradient-descent-based backpropagation is still used in online training, but the gradient calculations are simplified.

In the traditional gradient descent algorithm, calculating derivatives requires huge computing resource, which is not affordable at circuit level. Therefore, the gradient calculation is usually simplified to calculate the sign of gradients. In this way, the calculation complexity is reduced significantly, because calculating the sign requires only comparators at circuit level. Based on the sign of gradients, the positive or negative of programming voltages can be determined. Afterwards, these programming voltages are applied on memristors. Consequently, the resistances of memristors are approaching the required resistance values, with which the accuracy of the neural network reaches a given level. By programming memristors repeatedly, the required resistance values can be obtained after sufficient iterations.

Since only a positive or negative voltage with a constant value is applied on memristors, the changes of resistances might not achieve the required values. To achieve a given accuracy, a lot of iterations are required, but the total consumed time and power consumption are still less than software training. However, the accuracy after online training is lower than software training. To compensate accuracy drop, software training together online tuning method is proposed and will be explained in the next section.

Software Training with Hardware Tuning

As introduced in the previous section, although online training is faster and power efficiency, it requires more iterations and the accuracy after training is lower than that of software training. To deal with this problem, software-involved hardware training is proposed.

The training process is firstly performed at software level. The trained weights are then mapped to the resistances of memristors, as the linearity between conductance and software-trained weights. Accordingly, the max weight is mapped to the max achievable conductance and vice versa, which can be expressed as follows.

$$G = \alpha W + \beta. \tag{2.13}$$

In Equation 2.13, a linear relationship is established between weights and memristor conductance, where $\alpha = \frac{g_{max} - g_{min}}{w_{max} - w_{min}}$ and $\beta = g_{max} - \alpha \cdot w_{max}$. g_{max} and g_{min} represent the maximum and minimum conductances of a memristor, respectively. w_{max} and w_{min} represent the maximum and minimum weights, respectively. In this way, trained weights can be linearly mapped to the memristor conductance without any accuracy loss if the programming can be executed precisely.

However, to program the memristor to exact values, that correspond to the weights, a iterative programming process is required. In reality, to simplify the structure of the memristor architecture, the memristor resistances can only be programmed to some certain levels, 32 in [HSL+16] or 64 in [PKM+16]. Therefore, when ideal weights are mapped to the conductance of memristors, these weights are adjusted to approximated levels. This process is called quantization. Due to the existence of quantization, accuracy of neural networks is lower than the accuracy after software training, so that online tuning process is often necessary to improve the accuracy further.

This software training together with online tuning method is much easier than the pure online training process, because a mapped resistance is close to its required resistance value, so that only a few tuning iterations are required. After online tuning, the accuracy can be comparable with software trained accuracy.

3. Aging of Neuromorphic Circuits

3.1. Background of Aging

With CMOS technology scaling down into nanometer era, circuit aging has become a main challenge, because it causes timing failures of digital circuits. This aging effect is affected by multiple physical phenomenons, e.g., Negative-bias Temperature Instability (NBTI) [AM05] and Hot Carrier Injection (HCI) [ZFE07], Electromigration (EM), Timing Dependent Dielectric Breakdown (TDDB) and Positive BTI (PBTI).

Many factors affect aging, e.g., process variations, temperature, operation frequency and supply voltage. Due to the complexity in analyzing aging problems, it is common to maintain a safety margin to relax the aging pressure in the industry. However, with aging effect becoming more pronounced, the safety margin also increases and approaches an intolerable level, so that aging effect must be analyzed and counteraging methods should be proposed specifically.

The traditional aging effect is simulated at transistor level, which is accurate but not efficient for large circuits, because this method is too time-consuming. To reduce the simulation time, several aging models at gate level are proposed in [BM09, CWBT11, KKS06, KKS07, PKK$^+$06, KBW$^+$14, AKGH16, KME$^+$16]. In these models, the AgeGate model [LGS09, LBS10, LBS12] is based on the canonical delay model [VRK$^+$06]. Therefore, this model can be incorporated in the standard signoff flows [KS15]. With this model, the speed of aging analysis is improved in [LBS14] with no accuracy drop.

In the manufacturing phase, process variations happen to different parameters, e.g. channel length and width of transistors, for every individual chip, so that the aging effect should be considered as statistical. To deal with the aging effect, post-silicon tuning techniques

are adopted to adjust timing function of every chip. These techniques include body bias tuning [KSB06, GLL+15], voltage control [And05, KCCS+17, KLS+15], and clock tuning [NSG+06, TZ05, LN14, ZLS16c, ZLS16a, ZLS+18, ZLL+18, LCS11, LS15], as well as exploring the interdependency between clock-to-q delay and setup/hold time to alleviate the challenge in timing closure [ZLS16b]. In addition, flip-flops can also be removed to further reduce the effect of process variations [ZLHS18] and potential of aging.

Aging of semiconductor devices is caused by voltage stressing. Same as transistors, extensive programming of memristors also result in aging effects. These aging effects can be considered as the degradation of highest resistance R_{off} and lowest resistance R_{on}. In the following sections, memristor aging effect is analyzed in detail and a compact aging model is proposed. At the end of this chapter, state-of-the-art counteraging methods are discussed.

3.2. Memristor Aging

Memristor aging attracts the attention of researchers when it is used as memory storage. To write data into a memristor, a high voltage, e.g., $2V$, is applied on the memristor, leading to the degradation of the ratio between the highest and lowest resistances (R_{off} and R_{on}). This ratio is critical for memristor as memory storage cell, as it is used for distinguishing stored information.

To evaluate the lifetime of memristor, the concept of endurance is introduced to measure the number of available writing times that a memristor can be set to the lowest resistance and reset to the highest resistance while guaranteeing the stored information can be still recognized.

According to the endurance data measured by some research groups [CGG+12, CLG+11], memristor aging types can be classified into three different categories.

- Failure type 1: Both R_{off} and R_{on} decrease to a resistance value that is smaller than the original R_{on}

- Failure type 2: Both R_{off} and R_{on} increase to a resistance value that is higher than the

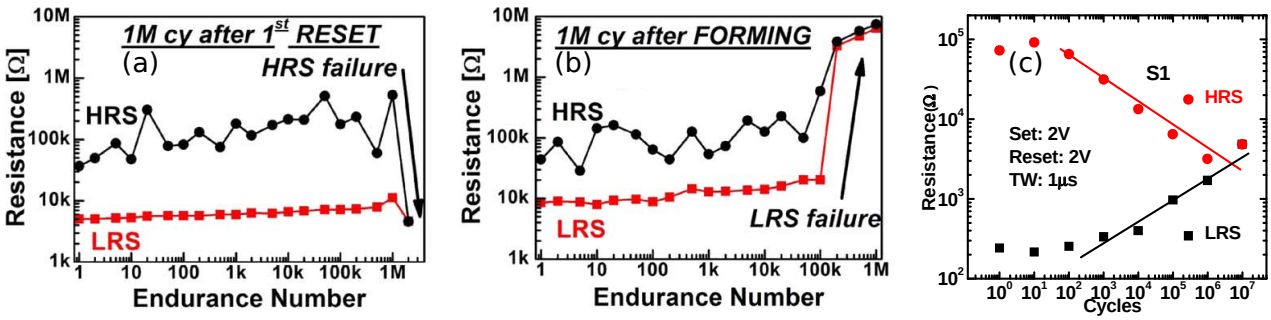


Figure 3.1.: Three failure types[CGG+12, CLG+11].

original R_{off}

- Failure type 3: R_{off} decreases and R_{on} increases and both reach intermediate resistances.

The measurement data of these three failure types are shown in Fig 3.1 respectively. Failure type 1, shown in Fig 3.1 (a), is observed in [CLG+11, CGG+12]. Failure type 2, shown in Fig 3.1 (b), is mentioned in [CGG+12]. Failure type 3, shown in Fig 3.1 (c), was discovered in [CLG+11, BAW+15].

These three different memristor failure types can cover almost all measurement data of memristor degradations. However, failure type 1 is more widely adopted, because it is observed by almost all research groups. The remaining two failure types are not always observed. In addition, for failure type 2, it is reported in [CGG+12], that this failure can be recovered, so that aging may not exist. Therefore, failure type 2 should not be considered as an aging problem, and it is just a recoverable fault. Type 1 failure causes the highest and lowest resistances of memristors to approach much lower resistances than that of type 3 failure. In this process, type 1 causes more power consumption than type 3 due to the lower resistance, leading to an acceleration of aging. Therefore, failure type 1 is used to represent the aging effect of memristor resistances failure, although the proposed framework in this work can deal with aging at all these three types.

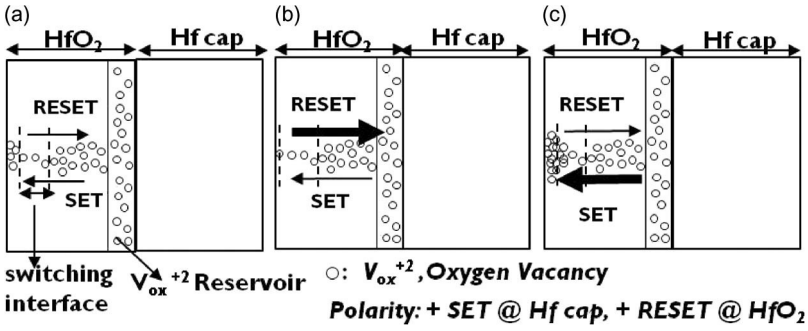


Figure 3.2.: Failure types 1 and 2 mechanism[CGG+12].

3.2.1. Failure Mechanism of Type 1 & 2

The failure types 1 and 2 can be explained as the effects of over-set and over-reset with applied voltage on memristors [CGG+12]. Fig 3.2 illustrates the failure mechanism of the two types. In this mechanism, applied voltage is considered to cause the formation and rupture of conductive filament inside the memristor, leading to the competition between formation and rupture. If the formation effect is stronger than the rupture effect, more oxygen vacancies are generated, leading to a wider conductive filament, thus causing the decreasing of resistance . On the contrary, if the rupture effect is stronger, more oxygen vacancies are consumed, leading to a much thinner conductive filament because of lack of oxygen vacancies.

The memristor resistance is proportional to filament cross area as shown in Equation 3.1. The length $L_{filament}$ is usually determined by the physical size of memristor, but the area $A_{filament}$ is affected by the formation and rupture processes.

$$R_{memristor} \propto \frac{L_{filament}}{A_{filament}} \tag{3.1}$$

When the area increases, the memristor resistance decreases and vice versa.

3.2.2. Failure Mechanism of Type 3

The failure mechanism of type 3 is explained in Fig 3.3. The degradation of R_{off} is similar to that of failure type 1. Too many oxygen vacancies caused by strong formation process

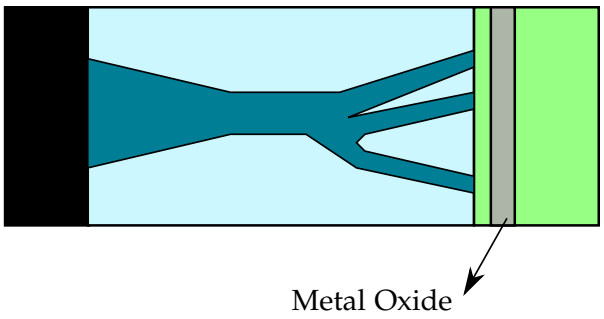


Figure 3.3.: Failure type 3 mechanism[CLG+11].

result in the decrease of R_{off}. However, for the resistance R_{on}, there is some metal oxide generated near the electrode, because of high temperature, large current and the existence of oxygen ions. Due to the existence of the metal oxide, the resistance R_{on} of memristor increases slightly.

3.3. Aging Models for Memristor

To establish the relation between the aging mechanisms and the failure types, several aging models have been proposed in [HCW+13, BAW+15, DFR+15]. These models point out that the aging effect is actually caused by temperature-activated filament changes, which is also confirmed in [KWH+17]. When voltage is applied on the memristor, the temperature inside conductive filament rises rapidly, leading to the degradation of memristor. In this section, three different aging models are thus discussed. Based on them, a compact aging model is proposed.

3.3.1. Aging Model 1

In [HCW+13], the aging effect is considered as the radius and length changes of filament. The changes of filament radius and length are accumulated in each set and reset cycle and finally cause the aging effects of memristors. The changes of filament radius and length can be modeled using Arrhenius-based equations as shown in Equation 3.2. In this equation, v_r and v_x represent the speed of filament in radius and length changes. E_a represents a physical

constant, k is the Boltzmann constant and T is the voltage-activated temperature.

$$v_r, v_x \propto exp(\frac{-E_a}{kT}). \tag{3.2}$$

Based on this model, applied voltages cause small filament changes in each set and reset cycle. These changes can be considered as the differences between formed filament size in set process and ruptured filament size in reset process. If the set effect is stronger, a wider and longer filament is formed and the reset process cannot rupture this formed filament fully, thus causing the filament to grow continuously. In this scenario, the resistances of R_{off} and R_{on} decrease gradually. If the reset effect is stronger, more filament is ruptured in the reset process and the filament becomes less conductive. In this case, the resistances of R_{off} and R_{on} increase gradually.

3.3.2. Aging Model 2

In [DFR+15], the aging effect is considered as the changes of filament size. This filament change is also modeled using a Arrhenius-based equation, because in this model, temperature is also regarded as the key reason for aging problem. This aging model can be expressed in Equation 3.3. In this equation, p_{gen} and p_{ann} represent oxygen vacancies generated and annihilated in the set and reset processes. E is a physical constant determined by memristors, k is the Boltzmann constant and T is temperature.

$$p_{gen}, p_{ann} \propto exp(\frac{-E}{kT}). \tag{3.3}$$

In this model, the filament changes can be considered as the competition between conductive oxygen vacancies generated and annihilated in each cycle, which is similar with the aging model 1.

3.3.3. Aging Model 3

In [BAW+15], the aging effect is considered as temperature-activated filament defects. However, in this model, it is not explained what is the exact defect of memristors. This model can

be expressed in Equation 3.4. In this equation, f_d represents the defect accumulated in each cycle. E_a is a physical constant determined by memristors, k is the Boltzmann constant and T represents filament temperature.

$$f_d \propto exp(\frac{-E_a}{kT}). \tag{3.4}$$

In this model, the aging effect is simplified to a virtual defect and no physical changes of memristors are considered. If a defect threshold value is defined for a memristor, the maximum number of cycles of a memristor can be expressed as in Equation 3.5. In this equation, $N_{C,max}$ is the maximum usable cycles. $f_{d,th}$ is the defect threshold value for a memristor and f_d is the defect generated in one cycle.

$$N_{C,max} = \frac{f_{d,th}}{f_d}. \tag{3.5}$$

In this way, the memristor failure is transformed to the accumulation of applied voltage on memristors, which reduces the aging modeling complexity significantly.

3.3.4. The Proposed Aging Model

In the previous section, three different aging models are introduced. In the aging models [DFR+15, HCW+13], the resistance degradations can be modeled, but these models adopt many physical parameters, obtained by fitting measurement data, and use high order aging representations, leading to unaffordable simulation time for a large crossbar architecture. Aging model [BAW+15] cannot model the resistance degradations of memristors. In this section, a compact memristor aging model is proposed to model the degradations of the highest resistance R_{off} and the lowest resistance R_{on}.

With the same defect concept proposed in [BAW+15], the proposed model also builds a relationship between the accumulated defects and the changes of R_{off} and R_{on}. As shown in Equation 3.6, where T is proportional to the multiplication of voltage and current, the calculation of defect is the same as that in [BAW+15].

$$f_d \propto exp(\frac{-E_a}{kT}) \tag{3.6}$$

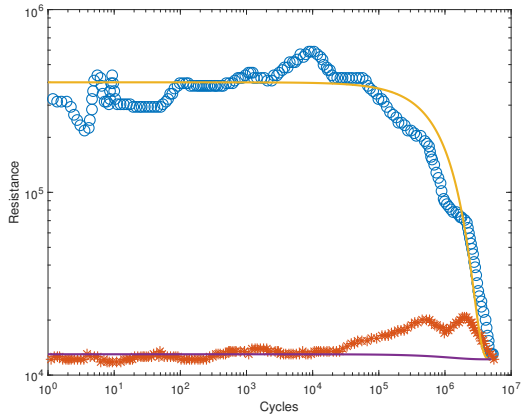


Figure 3.4.: Proposed model data fitting.

In Equations 3.7 and 3.8, the relationship between defects and degraded R_{off} and R_{on} is given. Based on the accumulated defects, the degraded resistances can be calculated. The transformation parameters α and β are obtained by data fitting.

$$R_{off}^{'} = R_{off} - \alpha(f_d) \tag{3.7}$$

$$R_{on}^{'} = R_{on} - \beta(f_d) \tag{3.8}$$

The simulated results of proposed aging models and measurement data from [CGG+12] are shown in Fig. 3.4, where the curves represent the simulation results of Equations 3.7 and 3.8 and points represent measurement data. These curves can match most of the measurement data, but cannot cover all of them, because of testing uncertainties and resistance variations.

Based on this proposed aging model, the degradations of the highest and lowest resistances of memristors can be calculated. The proposed aging model is used to evaluate the performance of counteraging methods.

3.4. State-of-the-art Counteraging Methods

The aging of memristors degrades the performance of memristor-based crossbar applications significantly. As a memory cell, aging leads to the loss of stored information, because the decreasing ratio between R_{off} and R_{on} causes unclear separation between 0 and 1. As a computing element, aging causes the decrease of linear resistance region. When software trained weights are mapped to the resistances of memristors, some weights cannot be set to the calculated resistance values because of the decrease of resistance levels.

To deal with the aging effect, several counteraging methods are proposed based on the aging models in previous sections. These methods can be classified into two categories: counteraging methods with additional hardware and counteraging methods with optimized training process.

3.4.1. Counteraging Methods at Hardware Level

The applied voltage on memristors causes the increase of filament temperature. The high temperature causes filament changes, leading to aging of memristors. To reduce the effect of aging, applied voltages can be modified. In [CLG+11], triangular and sine voltage sources are used. Consequently, the applied voltage causes less aging effect because of the average of applied voltage is lower than the constant DC voltage.

Another method to improve the lifetime of memristor is to connect a resistor and a memristor in series [KYS+16]. This method is also based on the idea of adjusting the voltage across the memristor, but using a connected resistor to implement this purpose, instead of modifying the voltages directly. The existence of connected resistor can suppress the irregular voltage drop on the memristor, leading to a longer lifetime[KYS+16].

Although the failure of memristors can be delayed by adjusting voltages, the memristor degradation is unavoidable because of the limited lifetime. To deal with accuracy loss resulted from the degradations of memristors, re-mapping method is proposed in [XLN+17, PAR15, LHSL17]. The re-mapping method uses a redundant memristor or a not heavily used memristor to replace the function of failed memristor, because in a memristor crossbar, not

all memristors are worn out. By using this method, the maximum lifetime can be improved by 65% according to [PAR15].

3.4.2. Counteraging Methods with Optimized Training Process

Besides these counteraging methods with additional hardware, counteraging methods with optimized training process have also been proposed. A main function of memristors is to be used as computing elements in neural network training and inference processes. The inference process does not contribute to the aging effect in a large degree, because the small applied voltage does not affect the structure of memristors too much. However, in training process, a memristor usually needs to be programmed with a high voltage, e.g. $2V$, for thousands even millions times. This exhaustive programming times degrade memristor rapidly. Therefore, several algorithms are proposed to optimize the online training process.

During online training, each programming operation is triggered by the weight changes, which are called delta weights in every iteration. By collecting the distribution information of delta weights, [XLN$^+$17] shows that most of delta weights are close to zero. If these small delta weights can be neglected, most of memristors are not needed to be programmed to new resistances, leading to less writing times, so that the aging effect can be alleviated. Based on the analysis, an optimized training process is proposed, which is named as Threshold-training Algorithm. In this method, small delta weights are ignored in the training process, to reduce the writing times. By adopting this method, the lifetime of neuromorphic computing architecture can be improved by approximately 15$\times$ according to [XLN$^+$17].

To further improve the lifetime of memristor, in [CLX$^+$18], the Threshold-training Algorithm, which is also called Structured Gradient Sparsification in this paper, is applied together with Aging-aware Row Swapping technology. This improved algorithm not only neglects the small delta weights in each updating cycle, but also switches the rows based on the detected aging information. Consequently, the writing times are averaged among rows of the memristor crossbar. The lifetime of crossbar architecture can be improved significantly, especially when a huge memristor crossbar architecture is used. It is reported in this paper, the lifetime extension is improved by 356$\times$, when training is performed on ResNet-50 with Imagenet dataset.

4. Aging-aware Lifetime Enhancement

As introduced in Chapter 2, software training together with programming maps trained weights to the resistance levels of memristors and online tuning process adjusts these resistances to make the neural network reach required accuracy. During the online training process, high voltage pulses, e.g. $2V$, are used to change the memristors' resistances, leading to the aging effect of memristors. To analyze this aging effect, an aging model is also proposed in the previous chapter. Although counteraging methods have been proposed, there are still some limitations of these methods. Therefore, in this chapter, an aging-aware lifetime enhancement method is proposed and the aging model is used to evaluate the effectiveness of the proposed method.

4.1. Overall Flow of the Proposed Counteraging Methods

Fig. 4.1 shows the overall flow of the proposed aging-aware lifetime enhancement method. Two main counteraging methods are proposed, skewing weights during software training and aging aware mapping. In the following sections, both methods are introduced in detail.

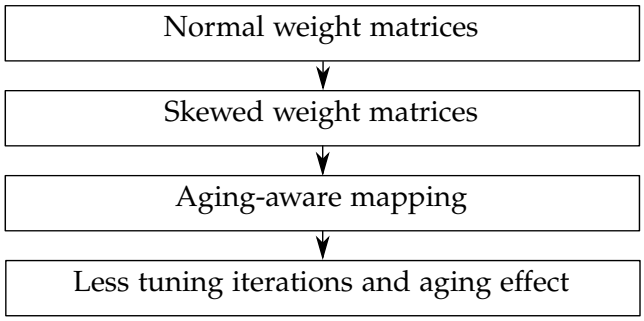


Figure 4.1.: Counteraging workflow.

4.2. Skewing Weights during Software Training

4.2.1. Limitations of Traditional Training Process

In Chapter 3, it is clear that the aging effect is caused by temperature, which results in unrecoverable changes of the filament the memristor. The temperature generated in memristor is caused by the applied voltage and corresponding current. Therefore, if the voltage or the current can be reduced, the aging effect can be reduced. However, when tuning the memristor to a required resistance value, the applied voltage is usually a constant amplitude voltage pulse. Because the implementation of variable voltage amplitudes requires additional hardware, thus the proposed counteraging method aims to reduce the memristor current by shifting the weights to be mapped to the memristor crossbar in previous methods.

The current flowing through memristors obeys Ohm's law. The flowing current is determined by the constant applied voltage, and the resistance value. If the resistance of memristors can be increased, the generated current is decreased, so that the aging effect is reduced.

During software training, weights are updated until the accuracy of the neural network achieves a given value. After software training, the weights are mapped to the conductance of memristors as shown in Equation 4.1.

$$G = \alpha \cdot W + \beta. \tag{4.1}$$

Fig. 4.2 shows the conceptual diagram of mapping method. The smallest weight is mapped to the largest resistance and vice versa. For the case of traditional trained weights, only a small portion of weights can be mapped to the large resistance range.

If most of the weights are pushed to the left side of the distribution shown in Fig. 4.3, most memristors are set to small conductances. Therefore, most weights are mapped to the large resistances of memristors, leading to the reduction of the aging effect.

Based on this observation, the proposed counteraging method aims to modify the distribution of weights by pushing most weights to the left side of the distribution. In the following section, a counteraging training method is proposed to achieve this weight shifting.

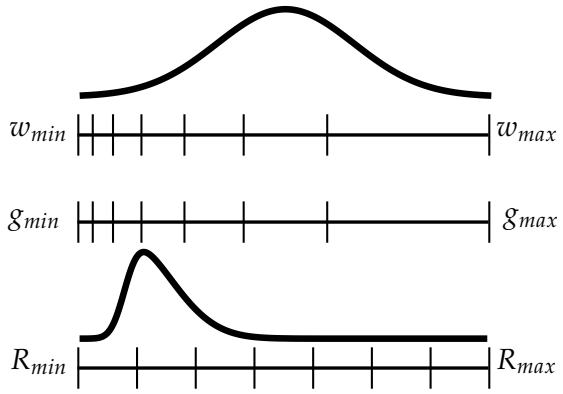


Figure 4.2.: Traditional weight distribution and corresponding mapping.

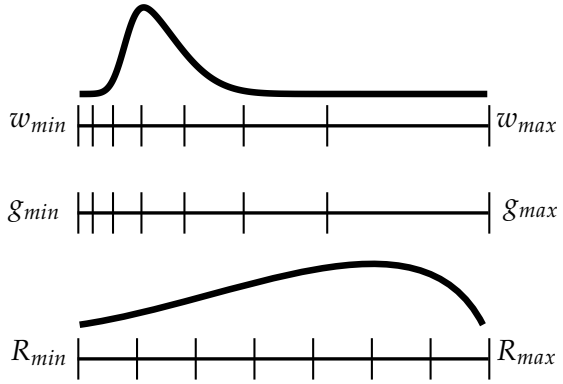


Figure 4.3.: Skewed weight distribution and corresponding mapping.

4.2.2. The Proposed Counteraging Training Method

As mentioned in the previous section, the weights obtained from the traditional training method lead to the mapping which negatively affect the lifetime of memristors, because most weights are mapped to small resistance values. In this section, a counteraging training method is proposed, which skews the distribution of trained weights to reduce the aging effect.

According to Equation 4.1, weights with small values are mapped to the small conductances, or large resistances, of memristors. To map most weights to high resistances, the expected weight distribution is shown in Fig. 4.4. With weights accumulated at the left side of the distribution, most of the weights can be mapped in a range with high resistances.

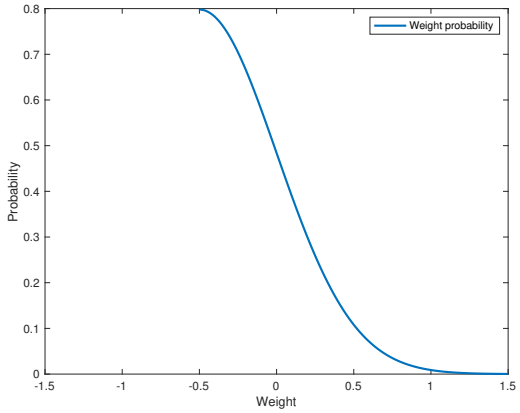


Figure 4.4.: Expected skewed weight distribution.

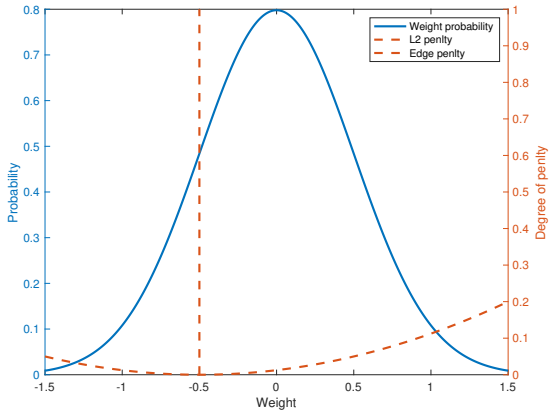


Figure 4.5.: Skewed weight training method.

To achieve the desired skewed distribution, the training method needs to be optimized. The optimized method is based on modifications of the cost function in the training process. To obtain the skewed distribution, two regularizations are used as shown in 4.5.

In Fig. 4.5, the solid curve indicates the traditional trained weights distribution, and two regularizations are represented by the dashed curves, which are appended to the cost function. In the proposed software training process, the cost function is expressed in Equation 4.2, where $\lambda \cdot \|W + \alpha \cdot \sigma\|^2$ is a general representation for both regularizations used in this training process.

$$Cost = -(y_{label} \log y + (1 - y_{label}) \log(1 - y)) + \lambda \cdot \|W + \alpha \cdot \sigma\|^2 . \tag{4.2}$$

The detailed representation of it can be expressed in Equation 4.3. Although both regularizations use modified L2-norm, the coefficients and valid ranges of both regularizations are different.

$$\lambda \cdot \|W + \alpha \cdot \sigma\|^2 = \begin{cases} \lambda_1 \cdot \|W + \alpha \cdot \sigma\|^2, & W < -\alpha \cdot \sigma \\ \lambda_2 \cdot \|W + \alpha \cdot \sigma\|^2, & W \geq -\alpha \cdot \sigma \end{cases} \tag{4.3}$$

To push the weights to accumulate at the left part of the distribution, a strong regularization (the vertical dashed curve) is used. In this case, the weights at the left side of this regularization are punished when they are updated. The closer to the center the weights, the smaller cost they cause. Therefore, to reduce the cost, in the next iteration, these weights at the left part of regularization are pushed to the right part of this regularization. The location of this regularization is critical for the skewed distribution. If the regularization is set close to zero, more weights on the left are pushed to the right, but the accuracy of training may be affected by this strong regularization.

The other regularization is similar to the L2-norm used in the traditional training process, but the central point is shifted to the same location of the first regularization. The function of the second regularization is to pull weights from the right part to the left part, which can further accumulate weights near the first regularization.

In the optimized training process, three parameters, λ_1, λ_2, and α, are introduced. It is then essential to determine the values of these three parameters and the choice of parameters are explained in the experimental chapter.

4.3. Aging-aware Mapping

4.3.1. Limitations of the Traditional Mapping Method

In the previous sections, an revised training method is proposed, which adjusts the weight distribution from a quasi-normal distribution to a skewed distribution. With the skewed distribution, most weights are mapped to high resistances, leading to less aging effect. However, during the mapping process, the degradation of the highest and lowest resistances are

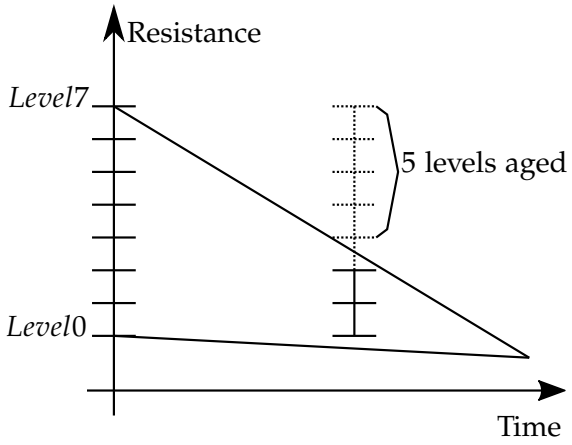


Figure 4.6.: Conceptual diagram of memristor levels decreasing.

not considered after experiencing aging. As introduced in Chapter 3, the available resistance range of memristors continues to decrease with many programming cycles, as shown in Fig 4.6. In this figure, we assume that a fresh memristor has 8 different resistance levels which can be used during mapping process. With exhaustive using of memristor, the available resistance range decreases and correspondingly available resistance levels also decrease. As shown by the second vertical line, only 3 different resistance levels are available after aging.

If the aging information of each memristor can be obtained, the mapping process can be more accurate, because the mapped resistance values are forced to maintain within the available range of these memristors. With a more accurate mapping mechanism, less tuning iterations are consumed, leading to further reduction of the aging effect. Therefore, in the following section, an aging-aware mapping method is proposed.

4.3.2. Proposed Dynamic Mapping Method

Aging Information Extraction

To extract the aging information of memristors, the aging model, proposed in the previous chapter, simulates the degraded highest and lowest resistance based on the history voltages applied on the memristors. We call this method the tracing of memristor aging. Alternatively, testing methods can be used to detect the highest and lowest resistances of memristors before

mapping. The result of testing is more accurate than using aging model to predict aging conditions, but the testing pulses also cause the aging effect.

With the aging information of memristors by tracing or testing the highest and lowest resistances of each memristors, a dynamic mapping method can be used, in which trained weights are mapped to the resistances of memristors. The dynamic mapping method uses the degraded highest and lowest resistances as the mapped boundary instead of the fresh highest and lowest resistances of memristors.

The original mapping method can be expressed in Equations 4.4, 4.5, and 4.6. g_{min} and g_{max} represent the boundary conductance of memristors. w_{min} and w_{max} represent minimum and maximum values of trained weight matrix. As shown in Equations 4.5 and 4.6, the highest and lowest resistances affect the parameters α and β directly, thus affecting the mapped resistance range.

$$G = \alpha W + \beta \tag{4.4}$$

$$\alpha = \frac{g_{max} - g_{min}}{w_{max} - w_{min}} \tag{4.5}$$

$$\beta = g_{max} - \alpha \cdot w_{max} \tag{4.6}$$

If the aging information of each memristor is known, this mapping method can be modified into Equations 4.7, 4.8, and 4.9. In these new mapping equations, α_{aging_aware} and β_{aging_aware} are determined by the smaller highest and lowest resistances of memristors after aging, which are calculated with the aging model.

$$G = \alpha_{aging\_aware} W + \beta_{aging\_aware} \tag{4.7}$$

$$\alpha_{aging\_aware} = \frac{g_{max,aged} - g_{min,aged}}{w_{max} - w_{min}} \tag{4.8}$$

$$\beta_{aging\_aware} = g_{max,aged} - \alpha_{aging\_aware} \cdot w_{max} \tag{4.9}$$

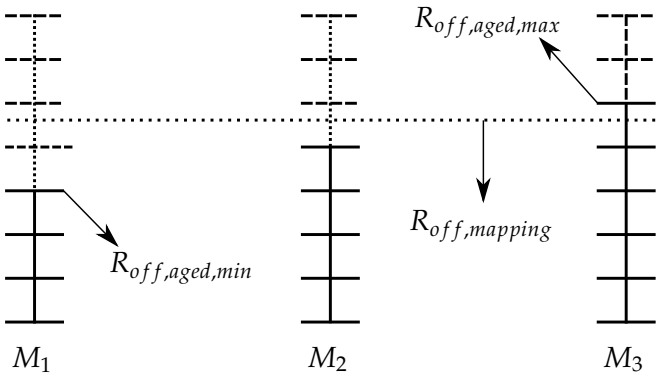


Figure 4.7.: Resistance boundaries after experiencing aging.

We use the minimum R_{off} and the maximum R_{off} after experiencing aging as references to find the mapping boundary, shown in Equations 4.10 and 4.11.

$$R_{off,mapping} = (1-\gamma) \cdot R_{min,off} + \gamma \cdot R_{max,off} \tag{4.10}$$

$$R_{on,mapping} = (1-\gamma) \cdot R_{min,on} + \gamma \cdot R_{max,on} \tag{4.11}$$

In these equations, a new parameter is introduced, γ. Before each application, this value needs to be determined by iterating γ from 0.0 to 1.0 with step 0.1. With every γ value, weight matrices can be mapped to memristor crossbars and corresponding accuracy can be obtained by simulations, so that by comparing these accuracy in different γ scenarios, the γ_{best} value is collected.

With this method, most mapped resistance values are mapped in the valid range. Fig. 4.7 shows an example of this mapping method to find the mapping boundary R_{off}. Memristor $M1$ degrades 4 levels, $M2$ degrades 3 levels and $M3$ degrades 2 levels because of aging effect. Based on the aging information, the mapping boundary $R_{off,mapping}$ can be calculated with γ_{best}, $R_{min,off}$ and $R_{max,off}$.

Therefore, although the mapped resistances also need to be tuned during online training, as explained in Chapter 2, the dynamic mapping method can reduce the number of iterations, because in the original mapping method, many desired resistance could be outside the valid range and it requires more iterations to reach the required accuracy.

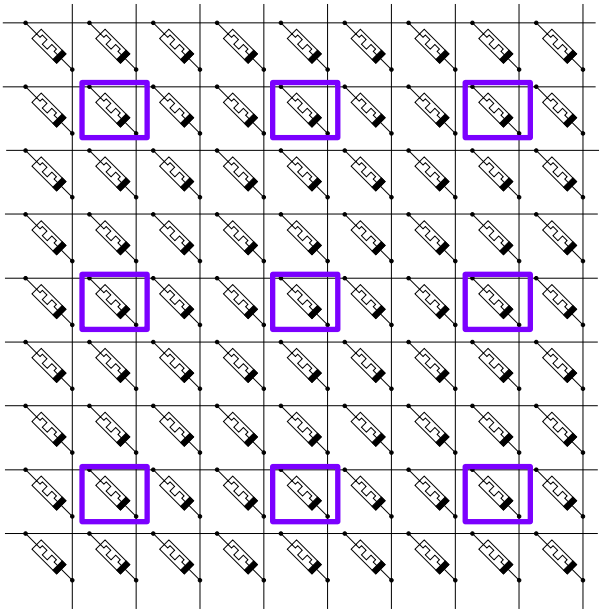


Figure 4.8.: Simplified tracing method.

Therefore, this dynamic mapping method can put most mapped resistances in the valid range by collecting the aging information of each memristor, so that the tuning iterations can be reduced, leading to less aging effect.

Simplified Tracing of Memristors

In the previous section, an aging-aware mapping method is introduced. However, in this method, the aging information of all memristors is required by tracing or testing all memristors, which requires huge computing resource if using the aging model, or unaffordable testing cost if based on testing, to predict the aged highest and lowest resistances of memristors. Therefore, the way of extracting aging information needs to be simplified.

Fig 4.8 shows a simplification of extracting the aging information. Assume there is a block with 3×3 memristors as shown in Fig 4.8 and only the memristor in the center of this block, marked with different boxes, is measured by tracing or testing. In this way, the effort of extracting aging information can be reduced by $8\times$.

In this case, not only the aging information extraction is simplified, but also the mapping

boundary calculations. As shown in Equations 4.12 and 4.13, the γ used in this case needs to be determined by the collection of γ_{best} values obtained in the previous method and it is set to the average value of previous obtained γ_{best} values.

$$R_{off,mapping} = (1 - \frac{1}{N_{application}} \cdot \sum_{i=1}^{N_{application}} \gamma_{i,best}) \cdot R_{min,off} + (\frac{1}{N_{application}} \cdot \sum_{i=1}^{N_{application}} \gamma_{i,best}) \cdot R_{max,off} \tag{4.12}$$

$$R_{on,mapping} = (1 - \frac{1}{N_{application}} \cdot \sum_{i=1}^{N_{application}} \gamma_{i,best}) \cdot R_{min,on} + (\frac{1}{N_{application}} \cdot \sum_{i=1}^{N_{application}} \gamma_{i,best}) \cdot R_{max,on} \tag{4.13}$$

5. Experimental Results

To evaluate the effectiveness of the proposed counteraging methods, three different neural networks, 2-layer fully-connected neural network, LeNet-5 and VGG16, are applied on three different images datasets, MNIST, Cifar10 and Cifar100. The neural networks and the proposed algorithm are implemented using Tensorflow [ABC+16] with an Intel 3.6GHz CPU and a NVIDA GeForce GTX 1080 Ti graphics card.

5.1. Three Neural Network Structures

2-layer Fully-connected Neural Network

The fully-connected neural network is the simplest structure, but it still achieves a good performance in simple datasets. In this work, a 2-layer fully-connected neural (FCNN) network is used on MNIST dataset. The details of the 2-layer FCNN is shown in Table 5.1. The size of the first layer is 784×256 and the size of the second layer is 256×10.

LeNet-5

LeNet [LJB+95] developed by Yann LeCun is considered as the first successful application of convolutional neural networks and this network has been used to recognize zip codes and

	Layer_1	Layer_2
2-layer FCNN	784×256	256×10

Table 5.1.: Structure of 2-layer neural network.

	Layer_1	Layer_2	Layer_3	Layer_4	Layer_5
LeNet-5	5×5(6)	5×5(16)	400×120	120×84	84×10

Table 5.2.: Structure of LeNet-5.

	Layer_1	Layer_2	Layer_3	Layer_4
	3×3(64)	3×3(64)	3×3(128)	3×3(128)
	Layer_5	Layer_6	Layer_7	Layer_8
	3×3(256)	3×3(256)	3×3(256)6	3×3(512)
VGG16	Layer_9	Layer_10	Layer_11	Layer_12
	3×3(512)	3×3(512)	3×3(512)	3×3(512)
	Layer_13	Layer_14	Layer_15	Layer_16
	3×3(512)	512×4096	4096×4096	4096×100

Table 5.3.: Structure of VGG16.

digits. LeNet-5 has five different layers, two convolutional layers with the corresponding pooling layers and three fully-connected layers. The size of each layer in LeNet-5 is presented in Table 5.2. The numbers in brackets represent the number of kernels used in that convolutional layer.

VGG16

In 2014, VGGNet [SZ14] was proposed. By stacking convolutional layers, VGGNet achieves excellent performance in ImageNet competition. The success of VGGNet shows that the depth of neural networks (number of layers) affects network performance significantly. In this work, VGG16 is used. In VGG16, there are 13 convolutional layers, 5 pooling layers and 3 fully-connected layers. Table 5.3 shows all the information of VGG16. The numbers in brackets represent the number of kernels used in that convolutional layer.

5.2. Three Image Datasets

MNIST

The MNIST dataset [LCB10] consists of handwritten digits $(0-9)$ images. There are totally 70000 black and white images. These images have been size-normalized and the digits are at the center of these images. The MNIST is a simple testing dataset. In the experiments, the simple 2-layer neural network is applied on this dataset.

Cifar10

The Cifar10 dataset [KH09] is a collection of low-resolution images. This dataset is widely used in machine learning and computer vision areas. There are 60000 images in this dataset, which are grouped into 10 different classes. These ten different classes includes airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each class contains 6000 different images. The size of each image in Cifar10 is pretty small, only 32×32 pixels in one image. To perform the classification, LeNet-5 is used.

Cifar100

The Cifar100 dataset is similar to Cifar10. The images also have a low resolution, which is 32×32 pixels for each image. The difference is that Cifar100 has 100 different classes instead of 10. Each class has 600 different images. This dataset is much more complex than MNIST and Cifar10, so that a deeper neural network, VGG16, is used on this dataset.

Dataset Classification

Before training, these datasets are usually classified into three sets, training set, validation set, and test set as shown in Fig 5.1. The training set is used to update weights during

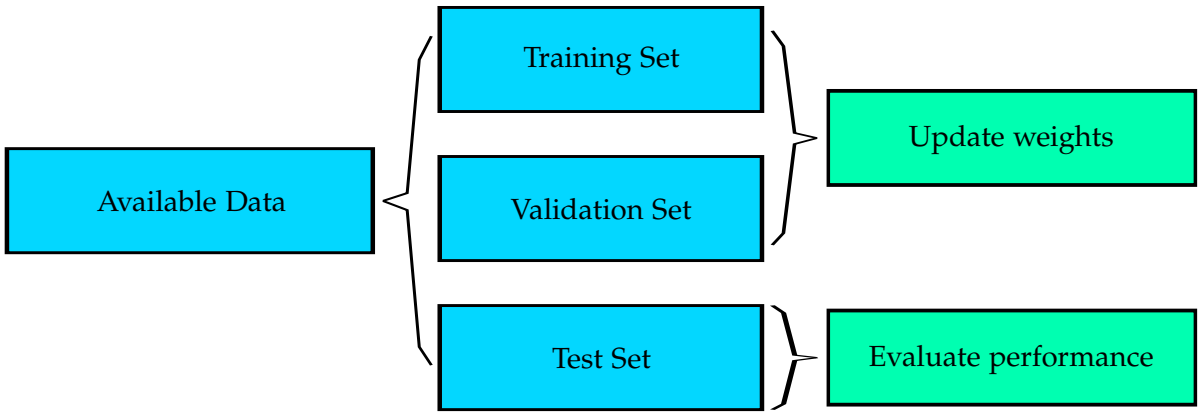


Figure 5.1.: Classification of available dataset.

	2-layer FCNN + MNIST	LeNet-5 + Cifar10	VGG16 + Cifar100
λ	0.001	0.001	0.005

Table 5.4.: λ used in traditional software training.

software training in a neural network. The validation set is used to verify the accuracy of the neural network during software training and online tuning. The test set is used to evaluate the performance of trained neural work, so that it can only be used once and cannot be involved in the process of training.

5.3. Experimental Setup

Parameters during the Modified Online Training

The traditional software training process uses cross-entropy as the cost function and L2-norm regularization is appended to the cost function, as shown in Equation 5.1.

$$Cost = -(y_{label} \log y + (1 - y_{label}) \log(1 - y)) + \lambda \|W\|^2 \tag{5.1}$$

The λ used in three networks are different, which are determined empirically. Table 5.4 shows the exact values of λ used in this work.

	2-layer FCNN + MNIST	LeNet-5 + Cifar10	VGG16 + Cifar100
$Acc_{software}$	97.90%	75.44%	71.50%

Table 5.5.: Accuracy after software training.

After software training without weight modification, the performance of three neural networks can be obtained, which is shown in Table 5.5.

In the proposed counteraging training process, the modified cost function is shown in Equation 5.2, where the second term of this cost function can be expressed in Equation 5.3. Three different parameters are introduced, strengths of two regularizations, λ_1 and λ_2, and the location parameter of both regularizations, $-\alpha$. The parameter λ_2 is the modified version of the regularization used in traditional training process, so that λ_2 can be set to the same values as displayed in Table 5.4. Therefore, only λ_1 and α are required to be determined by experiments.

$$Cost = -(y_{label} \log y + (1 - y_{label}) \log(1 - y)) + \lambda \cdot \|W + \alpha \cdot \sigma\|^2, \tag{5.2}$$

$$\lambda \cdot \|W + \alpha \cdot \sigma\|^2 = \begin{cases} \lambda_1 \cdot \|W + \alpha \cdot \sigma\|^2, & W < -\alpha \cdot \sigma \\ \lambda_2 \cdot \|W + \alpha \cdot \sigma\|^2, & W \geq -\alpha \cdot \sigma. \end{cases} \tag{5.3}$$

After traditional software training, the distributions of weights of each layer have been obtained, so that the standard deviations σ can be calculated with these distributions. To determine the parameters of regularizations used in counteraging training process, multiple combinations of location parameter α and regularization strength λ_1 are tested in training dataset. By guaranteeing that the accuracy of trained neural network meets a given value and most weights are mapped into high resistances, the values of α and λ_1 can be determined. Therefore, by adopting these parameters, skewing weights during training process is implemented afterwards. The performance of the neural network after skewed software training are shown in Table 5.6.

	2-layer FCNN + MNIST	LeNet-5 + Cifar10	VGG16 + Cifar100
Acc_{skewed}	97.17%	73.30%	71.76%

Table 5.6.: Accuracy after skewed software training.

Parameters during Online Tuning

The weights of traditional software training and counteraging software training have been obtained and are mapped to resistances of memristors. These resistances, also the corresponding weights of the neural network for one application, need to be adjusted during online-tuning process, so that it is essential to set a threshold accuracy to terminate the tuning process. The threshold accuracy should guarantee a given accuracy, even after a certain number of applications, which is set to 5×10^7. Detailed method is shown in Algorithm 1.

Algorithm 1: Determine threshold accuracy

Data: Normal training accuracy: Acc_{normal}, Skewed weights: W_{skewed}, memristor crossbar: M_{fresh}, images and labels: x and y_{label}, required number of applications: N

Result: Threshold accuracy: $Acc_{threshold}$, Threshold ratio: $ratio_{threshold}$

```
ratio_threshold = 0;
Acc_threshold = ratio_threshold * Acc_normal;
Acc_tuned = 0;
while Acc_tuned ≥ Acc_threshold do
    ratio_threshold = ratio_threshold + 0.1;
    Acc_threshold = Acc_normal * ratio_threshold;
    M = M_fresh;
    for i = 1 to N do
        M_tuned ← Online tuning(M, x, y_label);
        Acc_tuned = F(M_tuned, x, y_label);
    end
end
```

With this algorithm, $Acc_{threshold}$ and $ratio_{threshold}$ can be obtained and the $ratio_{threshold}$ values used for the three neural networks are listed in Table 5.7.

	2-layer FCNN + MNIST	LeNet-5 + Cifar10	VGG16 + Cifar100
$ratio_{threshold}$	0.985	0.95	0.98

Table 5.7.: $ratio_{threshold}$ used during online tuning.

Lifetime Definition and Comparison

With the threshold accuracy, online tuning process of application can be simulated. As the number of applications applied on the memristor crossbar increases, the aging effect becomes pronounced, because memristors are programmed during each online tuning. We define the available number of applications, which achieves a given accuracy after online tuning, as the lifetime of the memristor crossbar architecture.

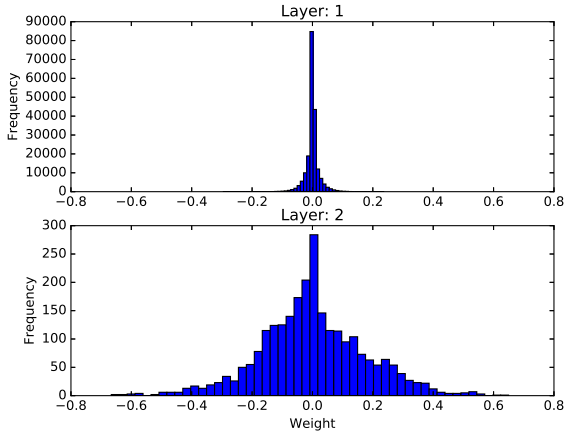
5.4. Results of Three Different Cases

To compare the lifetime improvement with the skewed weights and aging-aware mapping, we simulate the online tuning process for a certain number of applications and compare the lifetime in four scenarios, which are traditional weights and online tuning (T+N), skewed weights and online tuning (T+S), skewed weights with aging-aware mapping and online tuning (T+SA), and skewed weights with simplified aging-aware mapping and online tuning (T+SAS).

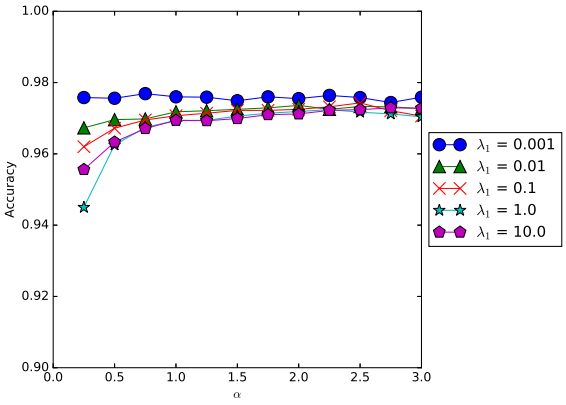
MNIST + 2-layer FCNN

Fig. 5.2a shows the traditional trained weight distributions of 2-layer FCNN. Distributions of these two layers are quasi-normal distributions. With these distributions, most weights are mapped to low resistances.

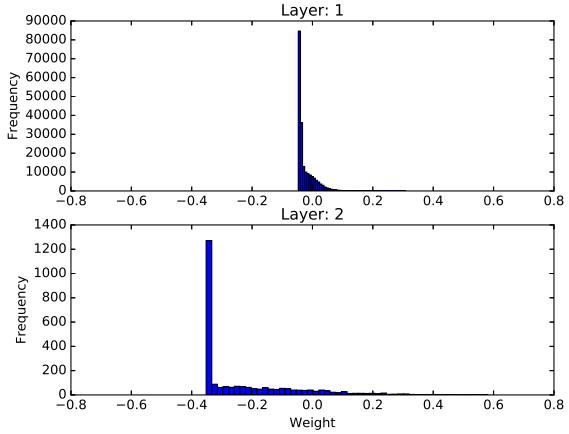
In Fig. 5.2b, the results of different combinations of the location parameter α and regularization strength λ_1 are presented.



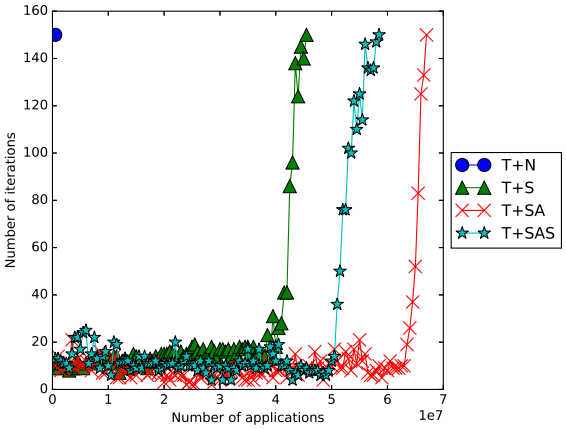
(a) Distributions of 2-layer FCNN after software training.



(b) α and λ_1 selection.



(c) Distributions of 2-layer FCNN after counteraging training.



(d) Lifetime comparison in different conditions.

Figure 5.2.: 2-layer FCNN + MNIST.

In this figure, the horizontal axis represents the location of these regularizations. The vertical axis indicates the accuracy with skewed weights after software training. In this figure, different curves mean different strengths, λ_1. According to this figure, the accuracy reaches the maximum value when the location is about $2 \times \sigma$. Therefore, the location of the regularization is set to $2 \times \sigma$. The accuracy with different regularization strengths, λ_1 and the same location is close to each other, so that λ_1 is randomly chosen and is set to 1.0.

After obtaining the proper parameters of the regularization, the software training with skewed weights can be implemented. The skewed weight distributions are presented in Fig. 5.2c. In this figure, we can observe that, most weight are pushed to the left side.

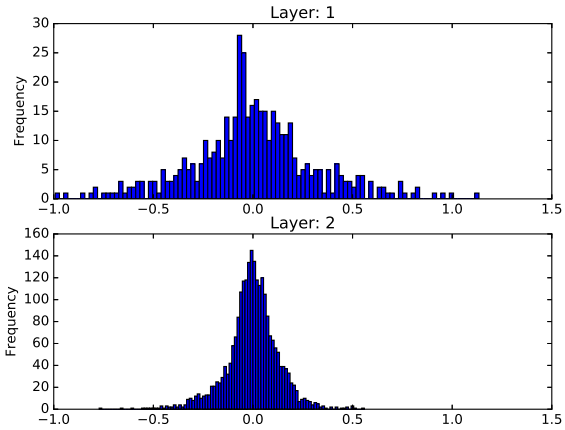
The lifetime of different scenarios of mapping and online tuning are compared in Fig. 5.2d. In this figure, there are four different curves, indicating tuning with traditional weights, tuning with skewed weights, tuning with skewed weight and aging-aware method and tuning with skewed weights and simplified aging-aware method.

In this figure, we can observe that, the proposed counteraging aging methods improve the lifetime significantly. For the traditional distribution case, the online tuning fails at the first application, because $ACC_{threshold}$ is high and requires precise tuning. The lifetime of tuning with counteraging trained weights is about 4.5×10^7 applications. Tuning with counteraging trained weights and aging-aware mapping method can improve the lifetime to about 6.7×10^7. For the simplified tracing version, the lifetime decreases, because the simplified version cannot collect all memristors' aging conditions, but the lifetime is still about 5.8×10^7.

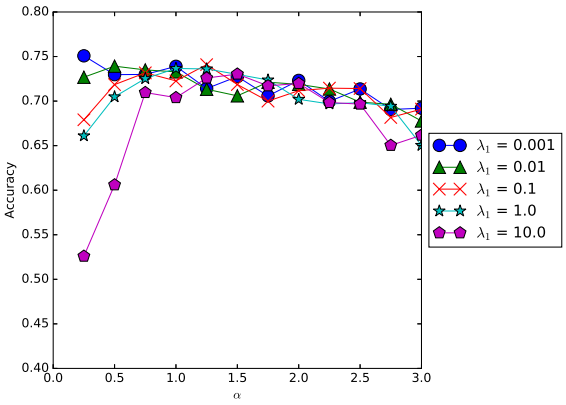
Cifar10 + LeNet-5

Fig. 5.3a shows the traditional trained weight distributions of the first and second layers in LeNet-5. These distributions of the two layers are quasi-normal distributions. With these distributions, most weights are mapped to low resistances.

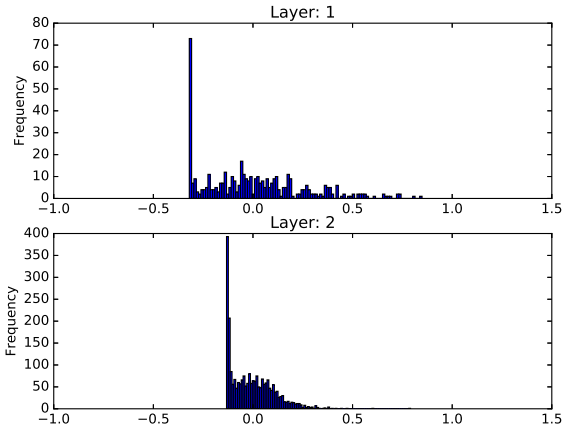
In Fig. 5.3b, the results of different combinations of the location parameter α and regularization strength λ_1 are presented.



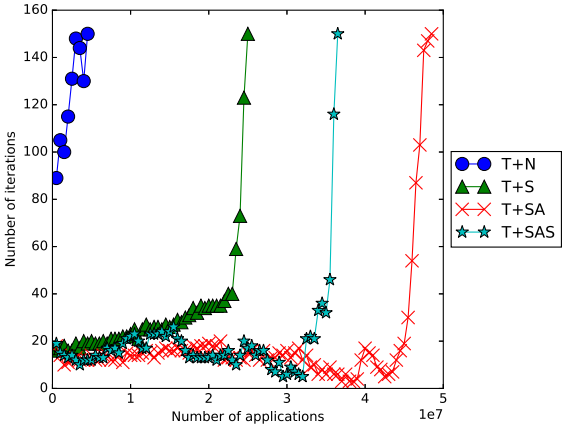
(a) Distributions of first 2 layers in LeNet-5 after software training.



(b) α and λ_1 selection.



(c) Distributions of first 2 layers in LeNet-5 after counteraging training.



(d) Lifetime comparison in different conditions.

Figure 5.3.: LeNet-5 + Cifar10.

In Fig. 5.3b, the horizontal axis represents the location of these regularizations. The vertical axis indicates the accuracy with skewed weights after software training. In this figure, different curves mean different strengths, λ_1. According to this figure, the accuracy reaches the maximum value when the location is σ. Therefore, the location of the regularization is set to σ. The accuracy with different regularization strengths, λ_1 and the same location is close to each other, so that λ_1 is randomly chosen and is set to 1.0.

After obtaining the proper parameters of the regularization, the software training with skewed weights can be implemented. The skew weight distributions of the first and second layer in LeNet-5 are presented in Fig. 5.3c. In this figure, we can observe that, most weight are pushed to the left side.

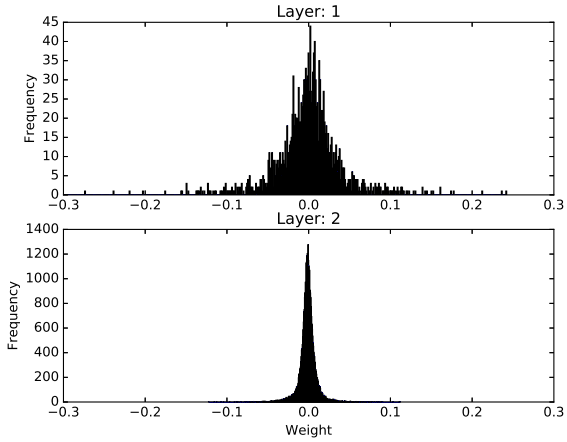
The lifetime of different scenarios of mapping and tuning are compared in Fig. 5.3d. In this figure, there are four different curves, indicating tuning with traditional weights, tuning with skewed weights, tuning with skewed weights and aging-aware method and tuning with skewed weights and simplified aging-aware method.

In this figure, we can observe that, counteraging aging methods improve the lifetime significantly. The lifetime of tuning with skewed weights is almost $5\times$ of the tuning with traditional trained weights. The lifetime of tuning with counteraging trained weights and aging-aware mapping method is almost $10\times$ of the original lifetime. For the simplified tracing version, the lifetime decreases, because the simplified version cannot collect all memristors' aging conditions, but the lifetime is still about $8\times$ of the traditional case.

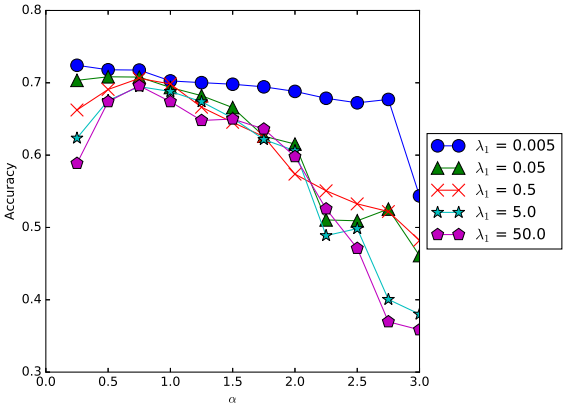
Cifar100 + VGG16

Fig. 5.4a shows the traditional trained weight distributions of the first and second layers in VGG16 as example. Distributions of these layers are quasi-normal distributions. With these distributions, most weights are mapped to low resistances.

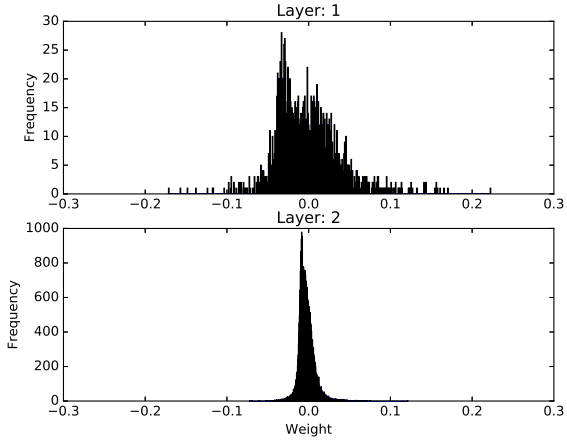
In Fig. 5.4b, the results of different combinations of the location parameter α and regularization strength λ_1 are presented.



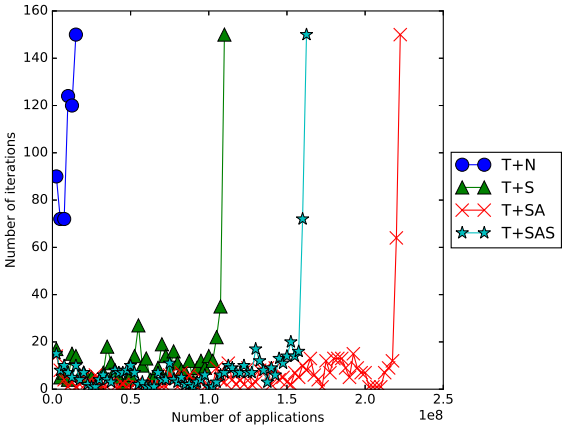
(a) Distributions of first 2 layers in VGG16 after software training.



(b) α and λ_1 selection.



(c) Distributions of first 2 layers in LeNet-5 after counteraging training.



(d) Lifetime comparison in different conditions.

Figure 5.4.: VGG16 + Cifar100.

In this figure, the horizontal axis represents the location of these regularizations. The vertical axis indicates the accuracy with skewed weights after software training. In this figure, different curves mean different strengths, λ_1. According to this figure, the accuracy reaches the maximum value when the location is $0.75 \times \sigma$. Therefore, the location of the regularization is set to $0.75 \times \sigma$. λ_1 used in this case is set to 0.005.

After obtaining the proper parameters of the regularization, the software training with skewed weights can be implemented. The skew weight distributions of the first and second layers in VGG16 are presented in Fig. 5.4c. In this figure, we can observe that, most weight are pushed to the left side.

The lifetime of different scenarios of mapping and tuning are compared in Fig. 5.4d. In this figure, there are four different curves, indicating tuning with traditional weights, tuning with skewed weights, tuning with skewed weights and aging-aware method and tuning with skewed weights and simplified aging-aware method.

In this figure, we can observe that, counteraging aging methods improve the lifetime significantly. The lifetime of tuning with skewed weights is almost $7\times$ of the tuning with traditional trained weights. The lifetime of tuning with counteraging trained weights and aging-aware mapping method is almost $15\times$ of the original lifetime. For the simplified tracing version, the lifetime decreases, because the simplified version cannot collect all memristors' aging conditions, but the lifetime is still about $10\times$ of the traditional case.

6. Conclusion

With the increasing demand for high-performance computing architecture, the memristor-based neuromorphic computing framework is introduced to relax the computing pressure caused by limitations of traditional von Neumann architecture. The neuromorphic computing architecture can break the bottleneck effectively and improve the computing capability and power efficiency significantly. Therefore, this architecture is considered as a promising candidate for the future computing system.

However, as a basic circuit component, memristor faces the aging problem. Exhaustive using of memristors will cause the degradations of the highest and lowest resistances, leading to failures of applications. To analyze the aging effect, an aging model is proposed in this work. This aging model can be used together with a memristor model to simulates the degradation of resistances according to applied voltages.

Besides the aging model, two counteraging methods are also proposed in this thesis. The first one is skewing weight during software training. It modifies the software training process, by pushing weights to the left part of the distribution, leading to smaller weights and less aging effect. The other one is dynamic mapping method, which determines the mapping boundary resistances before each application. This method extracts the aging information to map weights to valid resistance range, which can reduce the tuning iterations and lead to less aging effect. The proposed aging model is also used to evaluate the effectiveness of these counteraging methods.

In the experiments, different neural networks and image datasets are used to verify the effectiveness of the proposed methods. The simulation results show that with counteraging methods, the lifetime of neuromorphic computing architecture can be improved significantly.

In conclusion, this work points out that the potential aging problem for the promising neu-

romorphic computing architecture. In addition, an aging model is proposed to analyze the problem and corresponding counteraging methods are proposed.

Bibliography

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[AKGH16] Hussam Amrouch, Behnam Khaleghi, Andreas Gerstlauer, and Jörg Henkel. Reliability-aware design to suppress aging. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[AM05] Muhammad Ashraful Alam and Souvik Mahapatra. A comprehensive model of PMOS NBTI degradation. *Microelectronics Reliability*, 45(1):71–81, 2005.

[And05] Yoshiyuki Ando. Integrated circuits having post-silicon adjustment control, October 18 2005. US Patent 6,957,163.

[B+09] Yoshua Bengio et al. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[Bac07] John Backus. *Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs*. ACM, 2007.

[BAW+15] Simone Balatti, Stefano Ambrogio, Zhongqiang Wang, Scott Sills, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. Voltage-controlled cycling endurance of HfO_x-based resistive-switching memory. *IEEE Transactions on Electron Devices*, 62(10):3365–3372, 2015.

[BM09] Altug Hakan Baba and Subhasish Mitra. Testing for transistor aging. In *2009 27th IEEE VLSI Test Symposium*, pages 215–220. IEEE, 2009.

[CGG+12] Yang Yin Chen, Bogdan Govoreanu, Ludovic Goux, Robin Degraeve, Andrea Fantini, Gouri Sankar Kar, Dirk J Wouters, Guido Groeseneken, Jorge A Kittl, Malgorzata Jurczak, et al. Balancing SET/RESET Pulse for $> 10^{10}$ Endurance in HfO_2/Hf 1T1R Bipolar RRAM. *IEEE Transactions on Electron devices*, 59(12):3243–3249, 2012.

[Chu71] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[Chu11] Leon Chua. Resistance switching memories are memristors. *Applied Physics A*, 102(4):765–783, 2011.

[CLG+11] B Chen, Y Lu, B Gao, YH Fu, FF Zhang, P Huang, YS Chen, LF Liu, XY Liu, JF Kang, et al. Physical mechanisms of endurance degradation in TMO-RRAM. In *Electron Devices Meeting (IEDM), 2011 IEEE International*, pages 12–3. IEEE, 2011.

[CLX+18] Yi Cai, Yujun Lin, Lixue Xia, Xiaoming Chen, Song Han, Yu Wang, and Huazhong Yang. Long live TIME: improving lifetime for training-in-memory engines by structured gradient sparsification. In *Proceedings of the 55th Annual Design Automation Conference*, page 107. ACM, 2018.

[CWBT11] Jifeng Chen, Shuo Wang, Nemat Bidokhti, and Mohammad Tehranipoor. A framework for fast and accurate critical-reliability paths identification. In *IEEE North Atlantic test workshop (NATW)*, 2011.

[DFR+15] Robin Degraeve, Andrea Fantini, Ph Roussel, Ludovic Goux, A Costantino, CY Chen, Sergiu Clima, Bogdan Govoreanu, Dimitri Linten, Aaron Thean, et al. Quantitative endurance failure model for filamentary RRAM. In *VLSI Technology (VLSI Technology), 2015 Symposium on*, pages T188–T189. IEEE, 2015.

[GLL+15] Hui Geng, Jianming Liu, Pei-Wen Luo, Liang-Chia Cheng, Steven L Grant, and

Yiyu Shi. Selective Body Biasing for Post-Silicon Tuning of Sub-Threshold Designs: An Adaptive Filtering Approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):713–725, 2015.

[HCW+13] P Huang, B Chen, YJ Wang, FF Zhang, L Shen, R Liu, L Zeng, G Du, X Zhang, B Gao, et al. Analytic model of endurance degradation and its practical applications for operation scheme optimization in metal oxide based RRAM. In *Electron Devices Meeting (IEDM), 2013 IEEE International*, pages 22–5. IEEE, 2013.

[HLC+13] Miao Hu, Hai Li, Yiran Chen, Qing Wu, and Garrett S Rose. BSB training scheme implementation on memristor-based circuit. In *Computational Intelligence for Security and Defense Applications (CISDA), 2013 IEEE Symposium on*, pages 80–87. IEEE, 2013.

[HSL+16] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Proceedings of the 53rd annual design automation conference*, page 19. ACM, 2016.

[Iel11] Daniele Ielmini. Modeling the universal set/reset characteristics of bipolar RRAM by field-and temperature-driven filament growth. *IEEE Transactions on Electron Devices*, 58(12):4309–4317, 2011.

[KBW+14] Veit B Kleeberger, Martin Barke, Christoph Werner, Doris Schmitt-Landsiedel, and Ulf Schlichtmann. A compact model for NBTI degradation and recovery under use-profile variations and its application to aging analysis of digital integrated circuits. *Microelectronics Reliability*, 54(6-7):1083–1089, 2014.

[KCCS+17] Jerry Chang-Jui Kao, Chien-Ju Chao, LIN Chin-Shen, Nitesh Katta, Kuo-Nan Yang, and Chung-Hsing Wang. Post-silicon tuning in voltage control of semiconductor integrated circuits, February 7 2017. US Patent 9,564,896.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[KKS06] Sanjay V Kumar, Chris H Kim, and Sachin S Sapatnekar. An analytical model for negative bias temperature instability. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 493–496. ACM, 2006.

[KKS07] Sanjay V Kumar, Chris H Kim, and Sachin S Sapatnekar. NBTI-aware synthesis of digital circuits. In *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*, pages 370–375. IEEE, 2007.

[KLS+15] Rohit Kumar, Bing Li, Yiren Shen, Ulf Schlichtmann, and Jiang Hu. Timing verification for adaptive integrated circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 1587–1590. IEEE, 2015.

[KME+16] Nils Koppaetzky, Malte Metzdorf, Reef Eilers, Domenik Helms, and Wolfgang Nebel. RT level timing modeling for aging prediction. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 297–300. EDA Consortium, 2016.

[KS15] Shushanik Karapetyan and Ulf Schlichtmann. Integrating aging aware timing analysis into a commercial STA tool. In *VLSI Design, Automation and Test (VLSI-DAT), 2015 International Symposium on*, pages 1–4. IEEE, 2015.

[KSB06] Sarvesh H Kulkarni, Dennis Sylvester, and David Blaauw. A statistical framework for post-silicon tuning through body bias clustering. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 39–46. ACM, 2006.

[KWH+17] Suhas Kumar, Ziwen Wang, Xiaopeng Huang, Niru Kumari, Noraica Davila, John Paul Strachan, David Vine, AL David Kilcoyne, Yoshio Nishi, and R Stanley Williams. Oxygen migration during resistance switching and failure of hafnium oxide memristors. *Applied Physics Letters*, 110(10):103503, 2017.

[KYS+16] Kyung Min Kim, J Joshua Yang, John Paul Strachan, Emmanuelle Merced Grafals, Ning Ge, Noraica Davila Melendez, Zhiyong Li, and R Stanley Williams. Voltage divider effect for the improvement of variability and endurance of TaO_x memristor. *Scientific reports*, 6:20085, 2016.

[LBD+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[LBS10] Dominik Lorenz, Martin Barke, and Ulf Schlichtmann. Aging analysis at gate and macro cell level. In *Proceedings of the International Conference on Computer-Aided Design*, pages 77–84. IEEE Press, 2010.

[LBS12] Dominik Lorenz, Martin Barke, and Ulf Schlichtmann. Efficiently analyzing the impact of aging effects on large integrated circuits. *Microelectronics Reliability*, 52(8):1546–1552, 2012.

[LBS14] Dominik Lorenz, Martin Barke, and Ulf Schlichtmann. Monitoring of aging in integrated circuits by identifying possible critical paths. *Microelectronics Reliability*, 54(6-7):1075–1082, 2014.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.

[LCS11] Bing Li, Ning Chen, and Ulf Schlichtmann. Fast statistical timing analysis for circuits with post-silicon tunable clock buffers. In *Proceedings of the International Conference on Computer-Aided Design*, pages 111–117. IEEE Press, 2011.

[LGS09] Dominik Lorenz, Georg Georgakos, and Ulf Schlichtmann. Aging analysis of circuit timing considering NBTI and HCI. In *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*, pages 3–8. IEEE, 2009.

[LGY+17] Chao Li, Bin Gao, Yuan Yao, Xiangxiang Guan, Xi Shen, Yanguo Wang, Peng Huang, Lifeng Liu, Xiaoyan Liu, Junjie Li, et al. Direct Observations of Nanofilament Evolution in Switching Processes in HfO_2-Based Resistive Random Access Memory by In Situ TEM Studies. *Advanced Materials*, 29(10):1602976, 2017.

[LHSL17] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai Li. Rescuing memristor-based neuromorphic design with high defects. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pages 1–6. IEEE, 2017.

[LJB+95] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.

[LJH+15] Haitong Li, Zizhen Jiang, Peng Huang, Yi Wu, H-Y Chen, Bin Gao, XY Liu, JF Kang, and H-SP Wong. Variation-aware, reliability-emphasized design and optimization of RRAM using SPICE model. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 1425–1430. IEEE, 2015.

[LN14] Zahra Lak and Nicola Nicolici. A novel algorithmic approach to aid post-silicon delay measurement and clock tuning. *IEEE Transactions on Computers*, 63(5):1074–1084, 2014.

[LS15] Bing Li and Ulf Schlichtmann. Statistical timing analysis and criticality computation for circuits with post-silicon clock tuning elements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1784–1797, 2015.

[LWW+14] Boxun Li, Yuzhi Wang, Yu Wang, Yiran Chen, and Huazhong Yang. Training itself: Mixed-signal training acceleration for memristor-based neural network. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 361–366. IEEE, 2014.

[LYY+15] Chenchen Liu, Bonan Yan, Chaofei Yang, Linghao Song, Zheng Li, Beiye Liu, Yiran Chen, Hai Li, Qing Wu, and Hao Jiang. A spiking neuromorphic design with resistive crossbar. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[MAAI+14] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[Mea90] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[NSG+06] Samuel Naffziger, Blaine Stackhouse, Tom Grutkowski, Doug Josephson, Jayen Desai, Elad Alon, and Mark Horowitz. The implementation of a 2-core, multi-threaded Itanium family processor. *IEEE Journal of Solid-state circuits*, 41(1):197–209, 2006.

[PAR15] Peyman Pouyan, Esteve Amat, and Antonio Rubio. Statistical lifetime analysis of memristive crossbar matrix. In *Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2015 10th International Conference on*, pages 1–6. IEEE, 2015.

[PKK+06] Bipul C Paul, Kunhyuk Kang, Haldun Kufluoglu, Muhammad Ashraful Alam, and Kaushik Roy. Temporal performance degradation under NBTI: Estimation and design for improved reliability of nanoscale circuits. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pages 780–785. European Design and Automation Association, 2006.

[PKM+16] Jaesung Park, Myunghoon Kwak, Kibong Moon, Jiyong Woo, Dongwook Lee, and Hyunsang Hwang. TiO_x-based RRAM synapse with 64-levels of conductance and symmetric conductance change by adopting a hybrid pulse scheme for neuromorphic computing. *IEEE Electron Device Letters*, 37(12):1559–1562, 2016.

[PPPT11] Themistoklis Prodromakis, Boon Pin Peh, Christos Papavassiliou, and Christofer Toumazou. A versatile memristor model with nonlinear dopant kinetics. *IEEE transactions on electron devices*, 58(9):3099–3105, 2011.

[QPMS11] Muhammad Shakeel Qureshi, Matthew Pickett, Feng Miao, and John Paul Strachan. CMOS interface circuits for reading and writing memristor crossbar array. In *Circuits and systems (ISCAS), 2011 IEEE international symposium on*, pages 2954–2957. IEEE, 2011.

[SDCG+15] Daniel Soudry, Dotan Di Castro, Asaf Gal, Avinoam Kolodny, and Shahar Kvatinsky. Memristor-based multilayer neural networks with online gradient descent training. *IEEE transactions on neural networks and learning systems*, 26(10):2408–2421, 2015.

[SSSW08] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80, 2008.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TZ05] Jeng-Liang Tsai and Lizheng Zhang. Statistical timing analysis driven post-silicon-tunable clock-tree synthesis. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 575–581. IEEE Computer Society, 2005.

[VLT+13] Ilia Valov, Eike Linn, Stefan Tappertzhofen, Sebastian Schmelzer, Jan van den Hurk, Florian Lentz, and Rainer Waser. Nanobatteries in redox-based resistive switches require extension of memristor theory. *Nature communications*, 4:1771, 2013.

[VM15] Sascha Vongehr and Xiangkang Meng. The missing memristor has not been found. *Scientific reports*, 5:11657, 2015.

[VN93] John Von Neumann. First Draft of a Report on the EDVAC. *IEEE Annals of the History of Computing*, (4):27–75, 1993.

[VRK+06] Chandramouli Visweswariah, Kaushik Ravindran, Kerim Kalafala, Steven G Walker, Sambasivan Narayan, Daniel K Beece, Jeff Piaget, Natesan Venkateswaran, and Jeffrey G Hemmett. First-order incremental block-based statistical timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2170–2180, 2006.

[XLN+17] Lixue Xia, Mengyun Liu, Xuefei Ning, Krishnendu Chakrabarty, and Yu Wang. Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 33. ACM, 2017.

[ZCJY+13] Lu Zhang, Zhijie Chen, J Joshua Yang, Bryant Wysocki, Nathan McDonald, and

Yiran Chen. A compact modeling of TiO_2-TiO_{2-x} memristor. *Applied Physics Letters*, 102(15):153503, 2013.

[ZFE07] John Zimmerman, Jodi Forlizzi, and Shelley Evenson. Research through design as a method for interaction design research in HCI. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 493–502. ACM, 2007.

[ZGY+15] Lu Zhang, Ning Ge, J Joshua Yang, Zhiyong Li, R Stanley Williams, and Yiran Chen. Low voltage two-state-variable memristor model of vacancy-drift resistive switches. *Applied Physics A*, 119(1):1–9, 2015.

[ZLHS18] Grace Li Zhang, Bing Li, Masanori Hashimoto, and Ulf Schlichtmann. Virtualsync: timing optimization by synchronizing logic waves with sequential and combinational components as delay units. In *Proceedings of the 55th Annual Design Automation Conference*, page 26. ACM, 2018.

[ZLL+18] Grace Li Zhang, Bing Li, Jinglan Liu, Yiyu Shi, and Ulf Schlichtmann. Design-phase buffer allocation for post-silicon clock binning by iterative learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):392–405, 2018.

[ZLS16a] Grace Li Zhang, Bing Li, and Ulf Schlichtmann. EffiTest: Efficient delay test and statistical prediction for configuring post-silicon tunable buffers. In *Proceedings of the 53rd Annual Design Automation Conference*, page 60. ACM, 2016.

[ZLS16b] Grace Li Zhang, Bing Li, and Ulf Schlichtmann. PieceTimer: A holistic timing analysis framework considering setup/hold time interdependency using a piecewise model. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 100. ACM, 2016.

[ZLS16c] Grace Li Zhang, Bing Li, and Ulf Schlichtmann. Sampling-based buffer insertion for post-silicon yield improvement under process variability. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 1457–1460. IEEE, 2016.

[ZLS+18] Grace Li Zhang, Bing Li, Yiyu Shi, Jiang Hu, and Ulf Schlichtmann. EffiTest2: Efficient delay test and prediction for post-silicon clock skew configuration under process variations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.