Quantization, training, parasitic resistance correction, and programming techniques of memristor-crossbar neural networks for edge intelligence

NEUROMORPHIC
Computing and Engineering

OPEN ACCESS

RECEIVED
25 November 2021

REVISED
19 May 2022

ACCEPTED FOR PUBLICATION
13 June 2022

PUBLISHED
7 July 2022

TOPICAL REVIEW

Quantization, training, parasitic resistance correction, and programming techniques of memristor-crossbar neural networks for edge intelligence

Tien Van Nguyen[1], Jiyong An[1], Seokjin Oh[1],
Son Ngoc Truong[2] and Kyeong-Sik Min[1,*]

[1] School of Electrical Engineering, Kookmin University, Seoul, Republic of Korea
[2] Ho Chi Minh City University of Technology and Education, Ho Chi Minh City, Vietnam
[*] Author to whom any correspondence should be addressed.

E-mail: mks@kookmin.ac.kr

Keywords: memristor crossbars, neural networks, edge intelligence, quantization, training, parasitic resistance correction

Abstract
In the internet-of-things era, edge intelligence is critical for overcoming the communication and computing energy crisis, which is unavoidable if cloud computing is used exclusively. Memristor crossbars with in-memory computing may be suitable for realizing edge intelligence hardware. They can perform both memory and computing functions, allowing for the development of low-power computing architectures that go beyond the von Neumann computer. For implementing edge-intelligence hardware with memristor crossbars, in this paper, we review various techniques such as quantization, training, parasitic resistance correction, and low-power crossbar programming, and so on. In particular, memristor crossbars can be considered to realize quantized neural networks with binary and ternary synapses. For preventing memristor defects from degrading edge intelligence performance, chip-in-the-loop training can be useful when training memristor crossbars. Another undesirable effect in memristor crossbars is parasitic resistances such as source, line, and neuron resistance, which worsens as crossbar size increases. Various circuit and software techniques can compensate for parasitic resistances like source, line, and neuron resistance. Finally, we discuss an energy-efficient programming method for updating synaptic weights in memristor crossbars, which is needed for learning the edge devices.

1. Introduction

Many internet-of-things (IoT) sensors and edge devices have recently been widely used to make human life more comfortable and safe. To accomplish this, a massive number of IoT sensors and edge devices must continuously collect massive amounts of unstructured data from nature and human life (Plastiras *et al* 2018, Keshavarzi and van den Hoek 2019, Krestinskaya *et al* 2019, Zhou *et al* 2019, Deng *et al* 2020, Keshavarzi *et al* 2020, Xue *et al* 2020, Qin *et al* 2020a, Amin and Hossain 2021, Ghosh and Grolinger 2021). One issue with dealing with big data from IoT devices is energy. If all of the collected data is delivered to cloud servers, energy consumption for not only data communication but also big-data computing will be increased to unacceptable level.

Figure 1 depicts a conceptual block diagram of the computing device hierarchy, which includes IoT sensors, edge devices and cloud servers (Sun and Ansari 2016, Gusev and Dustdar 2018, Premsankar *et al* 2018, James 2019, Krestinskaya *et al* 2019, Pham *et al* 2019a). IoT sensors can detect data from the physical world, such as nature, human life, and so on. On a higher level, we can think about edge devices like mobile phones, wearable watches, and so on. Cloud servers are positioned at the top of the hierarchy, where high-performance computing can be prioritized over energy efficiency. In contrast to cloud servers, edge devices and IoT sensors should perform computing with high energy efficiency in order to extend battery life.

Figure 2 depicts the amount of data created, captured, copied, and consumed globally from 2010 to 2025 (Holst 2021). In 2025, the amount of data is expected to reach 181 zeta bytes. The rate of data volume growth

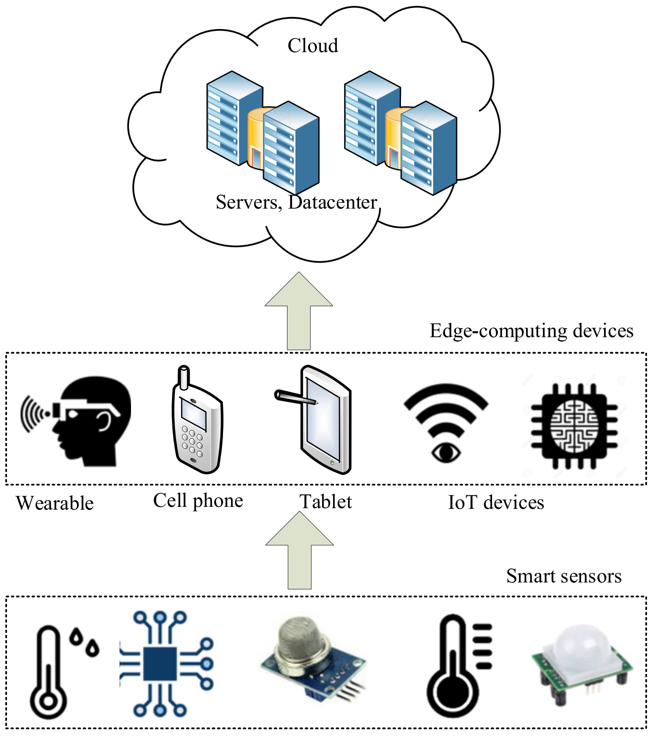

Figure 1. The conceptual block diagram of hierarchy of computing devices such as cloud servers, edge devices, and IoT sensors.

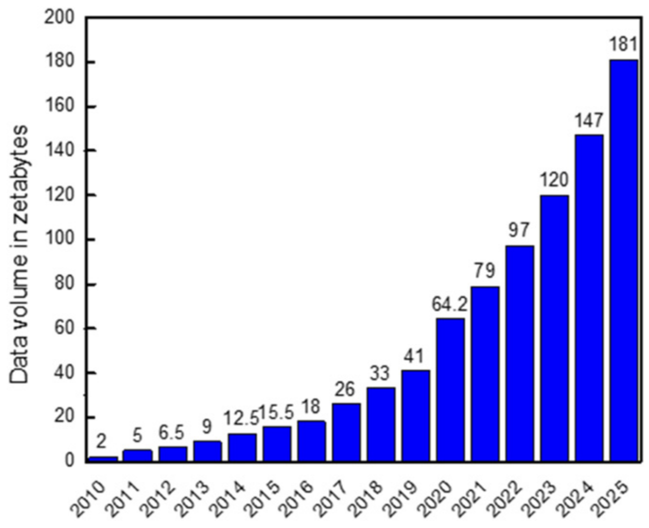

Figure 2. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025 (in zettabytes).

appears to be increasing with time. This rapid increase in figure 2 can be attributed to the recent explosion in the number of IoT sensors and edge devices.

Figure 3(a) depicts a conceptual block diagram of a traditional edge-cloud system for connecting IoT sensors to cloud servers, where the IoT sensors are assumed to lack edge intelligence functions such as neural networks (Shang *et al* 2014, Ronao and Cho 2016, Nguyen *et al* 2021a). As shown in figure 3(a), IoT sensors lacking edge intelligence must send all sensed data to cloud servers. As a result, cloud servers should consume a tremendous amount of energy in both computing and communication.

Let us instead assume that the IoT sensors have some level of edge intelligence, as shown in figure 3(b). Despite the fact that the amount of data sensed from the physical world in figure 3(b) is the same as in figure 3(a), IoT sensors with edge intelligence can send much less data to cloud servers. This is due to the edge intelligence's ability to extract important features from raw data sensed by IoT sensors. As a result, IoT

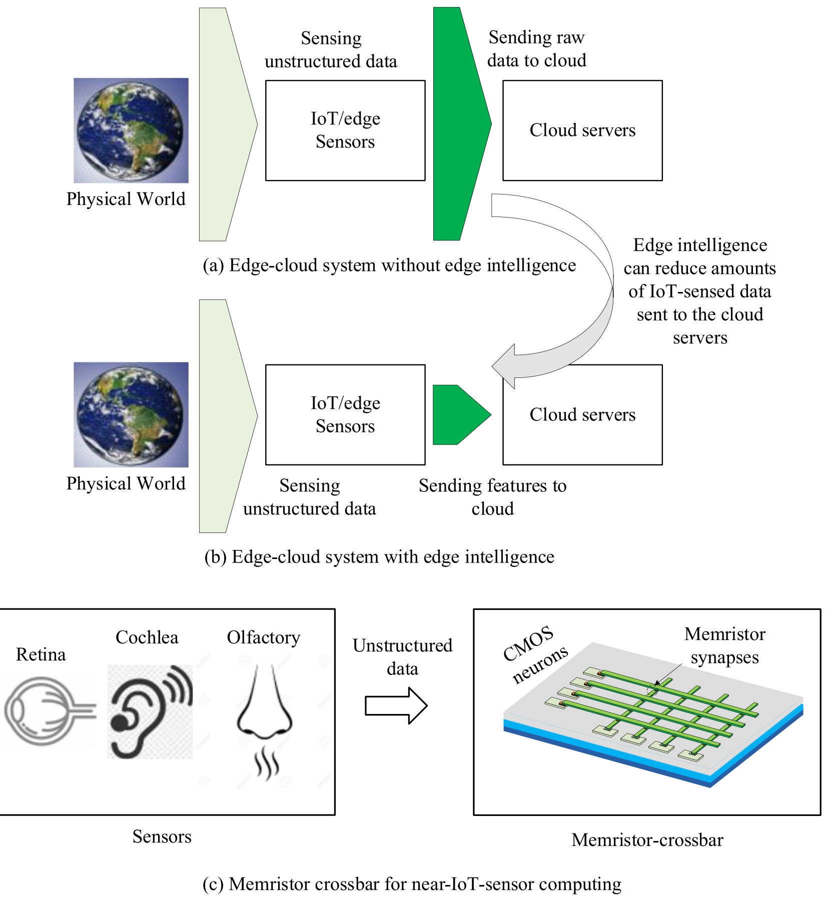
Sensing unstructured data
Sending raw data to cloud
IoT/edge Sensors
Cloud servers
Physical World
(a) Edge-cloud system without edge intelligence
Edge intelligence can reduce amounts of IoT-sensed data sent to the cloud servers
IoT/edge Sensors
Cloud servers
Physical World
Sensing unstructured data
Sending features to cloud
(b) Edge-cloud system with edge intelligence
Retina
Cochlea
Olfactory
Unstructured data
CMOS neurons
Memristor synapses
Sensors
Memristor-crossbar
(c) Memristor crossbar for near-IoT-sensor computing

Figure 3. (a) The block diagram of edge-cloud system without edge intelligence. (b) The block diagram of edge-cloud system with edge intelligence. The edge intelligence can reduce amounts of data to be sent to the cloud servers significantly, because the near-IoT-sensor computing can extract important features from the unstructured raw data collected from IoT sensors. (c) IoT sensors and memristor crossbar for near-IoT-sensor computing.

sensors can deliver only important features to cloud servers rather than sending all sensed data to the cloud. This data compression due to edge intelligence can significantly reduce the computation and communication burden for cloud servers.

Another point to consider is computing architecture. The traditional von Neumann architecture that has been used thus far is becoming increasingly inadequate and inefficient, especially for edge-intelligence hardware. This is due to the memory access bottleneck in the von Neumann architecture, which separates a memory block from a computing block. To process the large amounts of unstructured data generated by IoT sensors, the computing block should frequently access the memory block in order to move large amounts of data back and forth between the two blocks. If the two blocks are separated, frequent and mass memory access can significantly increase power consumption.

To overcome the von Neumann bottleneck, a new computing architecture based on brain-inspired architecture, analog arithmetic, processing-in-memory, parallel computing, new emerging memories, and so on can be considered (Linn *et al* 2012, Wright *et al* 2013, Bohr and Young 2017, Sebastian *et al* 2019). This brings us closer to energy-efficient computing hardware, which is required for implementing hardware of edge intelligence such as IoT sensors.

A memristor crossbar in figure 3(c) can be one promising candidate for implementing energy-efficient and low-precision AI hardware including edge-intelligence (Keshavarzi and van den Hoek 2019, Krestinskaya *et al* 2019, Zhou *et al* 2019, Deng *et al* 2020, Keshavarzi *et al* 2020, Ran *et al* 2020, Xue *et al* 2020, Qin *et al* 2020a, Singh *et al* 2021). In-memory computing with the memristor crossbar in figure 3(c) can be used to overcome the von Neumann machine's memory access bottleneck mentioned earlier. Memristors are non-volatile memories that allow for fast and energy-efficient read and write operations and they can be stacked layer by layer for forming 3D structure (Strukov *et al* 2008, Jo *et al* 2010, Truong *et al* 2014, Hu *et al* 2014, 2018, Adam *et al* 2016, Bhat *et al* 2017, Chakrabarti *et al* 2017, Li *et al* 2018b, Li and Belkin 2018, Li *et al* 2018a, Jeong and Shi 2018, Sheng *et al* 2019, James 2019, Amirsoleimani *et al* 2020, Lin *et al* 2020, Wang *et al* 2020). Memristor fabrication can be combined with conventional CMOS processing technology, where memristor crossbars can be integrated with CMOS devices.

Furthermore, the current–voltage relationship of memristors make it possible physical vector-matrix multiplication performed in crossbars (Hu *et al* 2018, Amirsoleimani *et al* 2020). By doing so, memristor crossbars can be used for in-memory computing. In-memory computing with memristor crossbars, in particular, can be useful for implementing edge intelligence near IoT sensors, where computing should be very energy-efficient. IoT sensors and a memristor crossbar are shown in figure 3(c) for near-IoT sensor computing.

In this paper, we discuss many technical issues of memristor crossbars for implementing hardware of edge-intelligence. In next section 2, we try to use memristor crossbars to realize quantized neural networks having binary and ternary synapses for simple but robust operation of in-memory computing with memristors. In section 3, we discuss the chip-in-the-loop training of memristor crossbars for not allowing memristor defects to degrade performance of edge intelligence. One more non-ideal effect in memristor crossbars is parasitic resistances such as source, line, and neurons resistance, which become more severe as the crossbar size is bigger. In section 4, we discuss various techniques to compensate the parasitic resistances such as source, line, and neuron resistance. In section 5, we explain energy-efficient programming method for updating synaptic weights in memristor crossbars, which is required for low-power learning of edge devices. Finally, in section 6, we summarize this paper.

2. Binary and ternary neural networks with memristor crossbars

Quantized neural networks such as binary neural network (BNN) and ternary neural network (TNN) have been studied extensively for many years to alleviate the hardware burden of high-precision computation (Alemdar *et al* 2017, Qin *et al* 2020b). Only binary synaptic weights of -1 and $+1$ are used in BNN. TNN makes use of three synaptic weights: -1, 0 and $+1$. BNN and TNN are particularly well suited for memristor crossbars, where the binary and ternary weights can be represented by some combinations of high resistance state (HRS) and low resistance state (LRS) (Truong *et al* 2014, Pham *et al* 2018, 2019a, Kim *et al* 2019, Chen *et al* 2021).

Figure 4(a) depicts a conceptual block diagram of an artificial neural network (ANN), which can be realized with memristor crossbars with binary and ternary weights (Pham and Min 2019, Pham *et al* 2019a, Pham *et al* 2019c). The ANN is made up of synaptic weights, input, hidden, and output neurons, as shown in figure 4(a). In this case, X_1, X_2, and so on are input neurons. Y_1, Y_2, and so on represent hidden neurons. Z_1, Z_2, and so on are output neurons. The numbers m, n, and k represent the number of input, hidden, and output neurons, respectively, in figure 4(a).

Figure 4(b) depicts a neural network realized using a memristor crossbar. Like figure 4(a), X_1 and X_2 apply input voltages to the crossbar, with open and solid circles representing HRS and LRS, respectively. The plus and minus column currents are represented by I_{1+} and I_{1-}, respectively. I_{1+} and I_{1-} are transferred to Y_1 neuron, which is hidden. I_{2+} and I_{2-} are similarly transferred to Y_2, which is hidden neuron, too. The hidden and output neurons can be binary or multi-bit precision. If we assume binary neuron in figure 4(b), the neuron can be implemented simply with a comparator circuit. In figure 4(b), I_{1+} and I_{1-} are compared to each other in the binary neuron. If I_{1+} exceeds I_{1-}, Y_1 becomes $+1$. On the other hand, if I_{1+} is less than I_{1-}, Y_1 becomes -1. C_1 represents the comparator circuit for I_{1+} and I_{1-} in this case.

Figure 4(c) depicts a detailed schematic of a memristor crossbar in figure 4(b), with each memristor accessible via a metal oxide field effect transistor. The sneak leakage problem can be mitigated in this 1T–1M architecture, because the memristors can be electrically isolated column by column by turning off the access transistors.

The memristor crossbars shown in figures 4(b) and (c) were trained and tested for the CIFAR-10 data set. Convolutional neural network (CNN) is the neural network architecture used in training and testing the crossbars for CIFAR-10 data set (Duan *et al* 2015, Yakopcic *et al* 2016, Krizhevsky *et al* 2018, Yao *et al* 2020). The simulated architecture is made up of four convolution layers and two fully connected layers. The CIFAR-10 data set contains 50 000 training images and 10 000 testing images divided into 10 categories. The input image of CIFAR-10 data set has a resolution of $32 \times 32 \times 3$ (RGB). PyTorch is used in the simulation of CNN with memristor crossbars. The quantization-aware training method is used to calculate binary and ternary synaptic weights when training the memristor-crossbar CNN (Bengio *et al* 2013, Courbariaux *et al* 2016).

The CIFAR-10 recognition rate for the simulated memristor crossbar with different neuron and synapse types is shown in figure 5(a). Here FW-FN denotes the floating-point synapse and floating-point neuron in figure 5(a). BW-BN is an abbreviation for binary synapse and binary neuron. TW-BN stands for ternary synapse and binary neuron. As previously stated, the binary synapse can only have weight $= +1$ and weight $= -1$. Synaptic weight $= +1$, synaptic weight $= 0$, and synaptic weight $= -1$ are the values of the ternary synapse. Weight $= +1$ is realized in the memristor crossbar by LRS on the plus column and HRS on the minus one. Similarly, weight $= -1$ denotes HRS on the plus side and LRS on the minus side. Weight $= 0$ indicates that HRS cells are present on both the plus and minus columns.

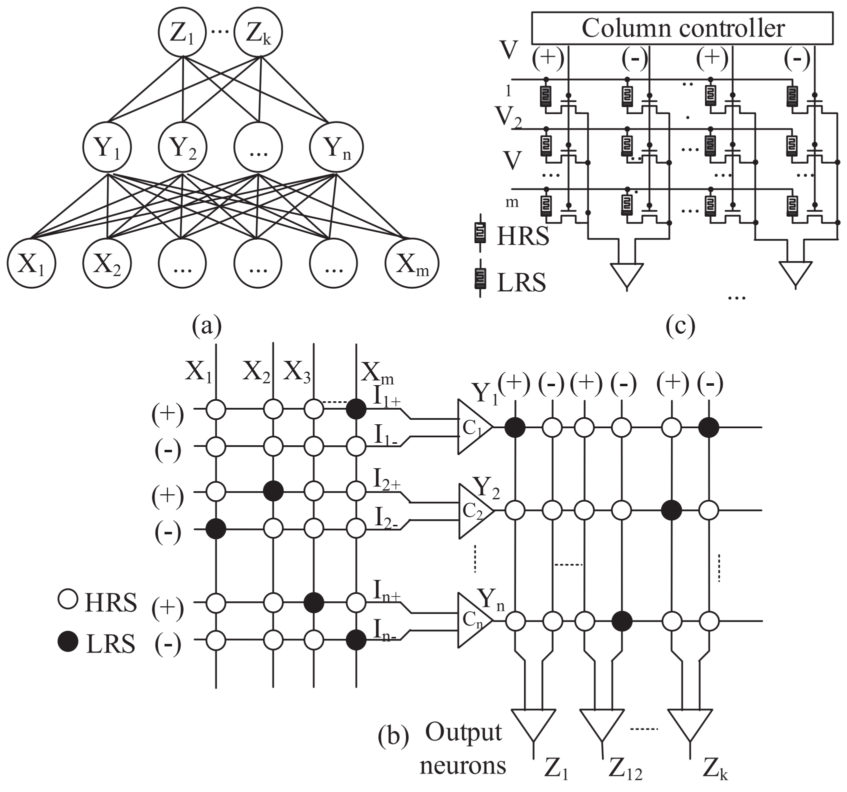

Figure 4. (a) The block diagram of two-layer artificial neural networks with input, hidden, and output neurons. (b) The memristor crossbars for implementing two-layer artificial neural networks. (c) The detailed schematic of the memristor crossbar with 1Transistor–1Memristor cell.

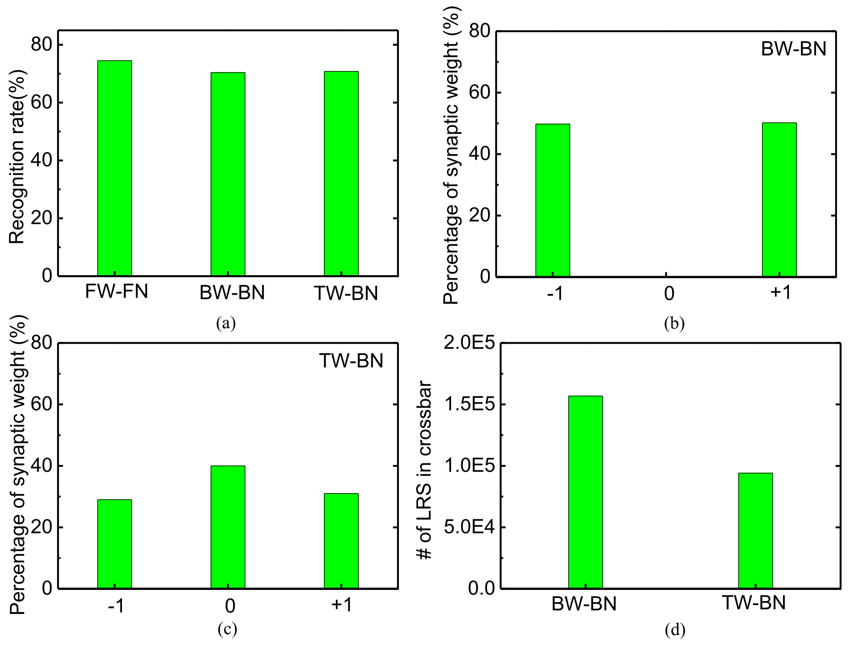

Figure 5. (a) Simulation of CIFAR-10 recognition rate of three cases of synapses and neurons (FW-FN, BW-BN, and TW-BN). (b) The percentages of weight = +1 and weight = −1 of BW-BN. (c) The percentages of weight = +1, weight = 0, and weight = −1 of TW-BN. (d) The numbers of LRS cells in the memristor crossbars for BW-BN and TW-BN.

According to figure 5(a), FW-FN has a CIFAR-10 recognition rate of up to 74.5 percent. In comparison to FW-FN, BW-BN and TW-BN are only 70.4 and 70.8 percent, respectively. In figure 5(a), the difference between BW-BN and TW-BN is insignificant (Nguyen *et al* 2021a).

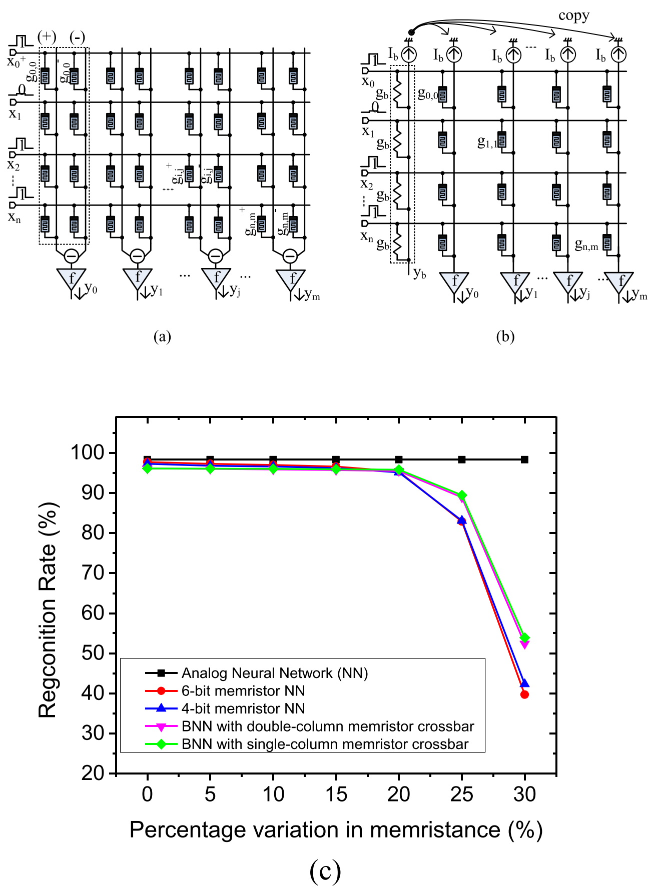

Figure 6. (a) The double-column memristor crossbar with binary synapse. (b) The single-column memristor crossbar with binary synapse. (c) MNIST recognition rate versus percentage of memristance variation (0%–30%).

Figure 5(b) depicts the percentages of synaptic weight = +1 and synaptic weight = −1 of BW-BN after the quantization-aware training. Figure 5(c) depicts the percentages of active bits of synapses of TW-BN. Figure 5(d) compares the number of LRS cells in the BW-BN and TW-BN memristor crossbars.

Because LRS cells are much more conductive than HRS cells, the power consumption of the memristor crossbar is proportional to the number of LRS cells. The HRS-LRS ratio is assumed to be 100 in this case. This means that the LRS cell is 100 times more conductive than the HRS cell. As a result, the column current increases significantly in proportion to the number of LRS cells in the corresponding column. In figure 5(d), the number of LRS cells in TW-BN is 42.9 percent lower than in BW-BN. This is due to the fact that synaptic weight = 0 in TW-BN can be implemented with two HRS on both the plus and minus columns. HRS consumes much less current than LRS.

The number of LRS cells can be reduced further by increasing the percentage of inactive bits in synapses in the crossbar. The lower the number of LRS cells, the lower the power consumption in the memristor crossbar. This means that ternary synapse can thus be more energy-efficient than binary synapse. Furthermore, if possible, a smaller number of LRS cells can significantly reduce the crossbar's power consumption more. This low power consumption of TNN memristor crossbars with as many zeros as possible can be useful in realizing low-power edge hardware.

One thing to comment here is the CIFAR-10 recognition rate shown in figure 5 can be improved better if we use a deeper neural network such as RESNET (He *et al* 2016). As the numbers of convolution layers and fully-connected layers in the tested neural networks are increased, the neural network's simulation can show better recognition rate.

As mentioned earlier, two columns are required in memristor crossbars to represent both positive and negative binary synaptic weights, as shown in figure 6(a). The current difference between the (+) and (−) columns in figure 6(a) enters the activation function circuit in the double-column crossbar. To reduce the number of memristor cells more, an alternative approach to the double-column crossbar in figure 6(a) can be considered. Figure 6(b) shows a single-column crossbar can represent both positive and negative binary weights. $G_{0,0}$ denotes a binary synaptic connection between x_0 and y_0 in figure 6(b). In Figure 6(b), g_b is used to generate the current I_b in order to calculate both positive and negative values using the single-memristor column. When compared to the double-column crossbar in figure 6(a), the single-column memristor crossbar in figure 6(b) can cut crossbar size in half. Furthermore, the energy consumption in the single-column can be reduced by half, which can be useful for implementing hardware of edge-intelligence (Pham *et al* 2018).

The I_b copy current circuit in figure 6(b) can be implemented by a simple OP amp circuit, as explained in the reference (Pham *et al* 2018). The delay time of the I_b current copy circuit is about 20 ns, when the circuit is simulated with SAMSUNG 0.13 μm CMOS technology (Pham *et al* 2018). The V_{DD} used in the circuit simulation was 1.3 V and HRS/LRS ratio of memristors was 100. Here, the circuit simulation was performed by CADENCE SPECTRE. The delay time of the I_b current copy circuit is comparable to the delay time of the activation function circuit, f, in figure 6(b), which is also implemented with the OP amp circuit. Thus, the delay time of I_b current copy circuit can be hidden in the delay time caused from the activation circuit, f, in figure 6(b).

In addition to the computing latency due to the activation function circuit, if parasitic capacitance in the memristor crossbar is combined with parasitic line resistance, this resistance-capacitance delay time can make the computing latency worse when performing the vector-matrix multiplication performed in the memristor crossbar (Sah *et al* 2015).

One more thing to discuss here is memristance variation that may degrade the neural network's performance (Pham *et al* 2018). For considering memristance variation in the neural network's simulation, Monte Carlo method can be used with MATLAB. Figure 6(c) compares 5 cases of neural networks. The base line (box symbol) is analog neural network (ANN), where the synaptic weights are real numbers. For the ANN (box symbol) in figure 6(c), no memristance variation is assumed. The 6 bit and 4 bit memristor neural networks (NNs) mean the synaptic weight's precision is 6 bit and 4 bit, respectively. The 6 bit and 4 bit NNs are represented with circle and up-triangle symbols, respectively. The binary neural networks (BNNs) in figure 6(c) means the synaptic weights are binary. Here the BNNs can be implemented with the double-column crossbar in figure 6(a) and the single-column one in figure 6(b). The double-column and single-column are represented with down-triangle and diamond, respectively, in figure 6(c). The memristance variation is varied from 0% to 30%, in the simulation. The memristance variation is not considered in the ANN (box symbol). The recognition rate of ANN can be as high as 98.3%. The 6 bit NN and 4 bit NN show 97.7% and 97.3% for the memristance variation = 0%. Both the double-column and single-column BNNs show 96.1% for the variation = 0%. As the memristance variation is increased to 30%, the neural network's accuracy becomes degraded. Comparing the multi-bit NNs and the BNNs, the BNNs show better recognition rate than the 6 bit and 4 bit NNs. This is because the binary weights are more robust to the memristance variation than the weights with 6 bit and 4 bit precision. Figure 6(c) indicates the double-column and single-column BNNs are able to work well until the memristance variation becomes as large as 20%. However, when the percentage variation exceeds 20%, the recognition rate seems to fall very sharply. For the variation = 30%, the single-column BNN has the rate as high as 53.9%, whereas the 6 bit NN's accuracy is as low as 39.7%.

3. Crossbar-in-the-loop training for defect-tolerant neural networks

Unfortunately, most of fabricated memristor crossbars suffer the non-ideal effects such stuck-at-fault defects and process variations (Tunali and Altun 2017, Chakraborty *et al* 2018). Figure 7(a) depicts a conceptual schematic of memristor crossbar containing defects. In the crossbar, the solid and open circles represent LRS and HRS, respectively. Here, memristor defects are represented by red star symbols. In this section, we will look at two types of memristor defects. They are stuck-at-LRS defect and stuck-at-HRS one, respectively. Figure 7(b) depicts a random-defect map for a 400 × 256 memristor crossbar, with the percentage of memristor defects as large as 10% (Nguyen *et al* 2019), (Pham *et al* 2019b). The percentage of random defects as large as 10% in figure 7(b) was obtained from the memristor crossbars fabricated (Yeo *et al* 2019).

Aside from memristor defects, there is also the issue of memristor's conductance variation. Figures 7(c) and (d) depict the statistical distributions of LRS and HRS, respectively, from the measurement and simulation (Pham *et al* 2019a). In figures 7(c) and (d), the percentage of variation in memristor conductance is as large as 30%.

The non-ideal effects such as stuck-at-fault defects and memristance variation can affect the neural network's performance significantly, as explained in the reference (Pham *et al* 2019b). When the percentage of defects = 10% and memristance variation = 5%, the MNIST recognition rate becomes degraded as low as 78.4%. Though the non-ideal effects such as memristor defects and process variations can degrade the neural network's performance severely, the defect-tolerant re-training scheme can minimize the recognition rate loss (Pham *et al* 2019b). From the MATALB simulation, it was observed that the chip-in-the-loop learning scheme considering the memristor defects during the training could improve the recognition rate very obviously. The rate loss was reduced from 16.9% to 0.2%, when the defect-aware training was used (Pham *et al* 2019b).

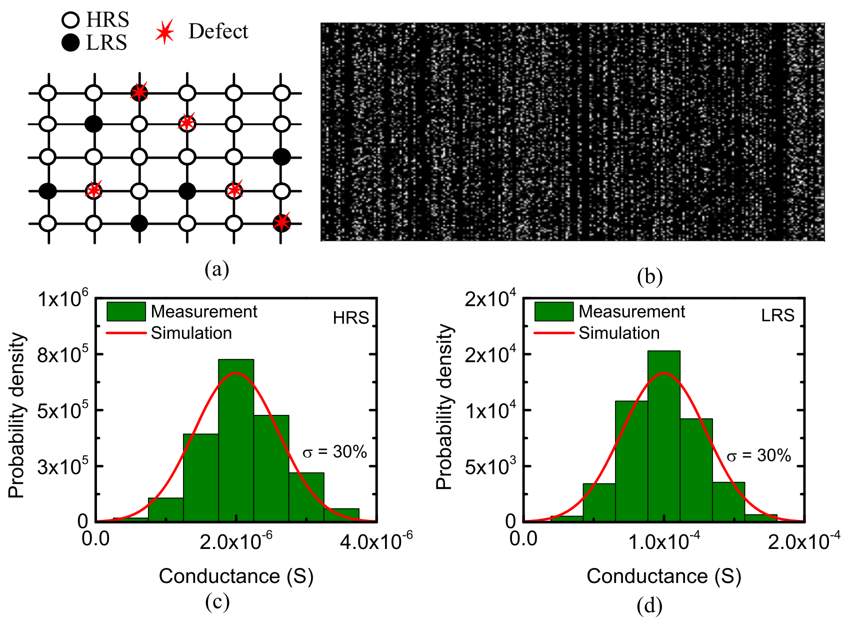

Figure 7. (a) The real memristor crossbar with defects. (b) The defect map of 400 × 256 memristor crossbar with the percentage of random defects = 10%. (c) The statistical distribution of HRS with the percentage σ variation = 30%. (d) The statistical distribution of LRS with the percentage σ variation = 30%.

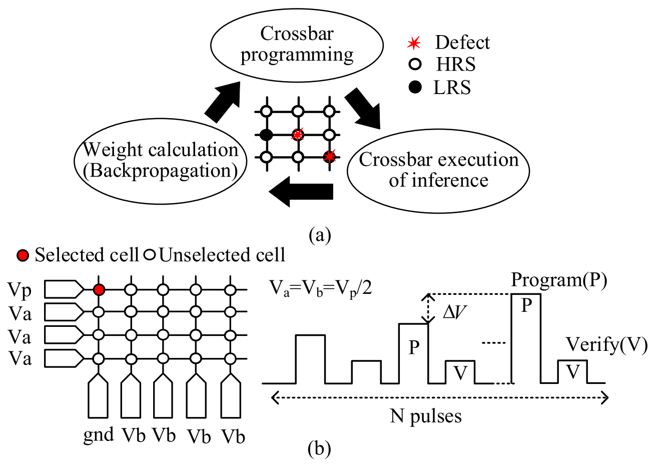

Figure 8. (a) The chip-in-the-loop training. The training loop is composed of three steps: the weight calculation, the crossbar programming, and the crossbar execution of inference. (b) The memristor crossbar with the $V_{DD}/2$ write scheme and the pulse modulation of the crossbar programming like the incremental step pulse programming ISPP.

As mentioned in the previous paragraph, the chip-in-the-loop training should be used to compensate for memristor defects such as stuck-at-faults and process variations when training memristor crossbars, as shown in figure 8(a) (Klein *et al* 1995, Pham *et al* 2019d). The training-loop incorporating memristor defects is required because the recognition rate will be greatly reduced if the memristor defects are not considered during the training of the memristor neural networks. As illustrated in figure 8(a), the training loop consists of three phases: weight calculation, crossbar programming, and crossbar's execution of inference. Among the three steps, the crossbar programming necessitates a lengthy programming time and a significant amount of crossbar programming energy. This is due to the fact that the crossbar should be programmed cell by cell using a flash memory programming method such as incremental step pulse programming (ISPP) (Suh *et al* 1995). Figure 8(b) depicts the memristor crossbar with the $V_{DD}/2$ write scheme and the pulse amplitude modulation for programming the crossbar (Pham *et al* 2019a).

As another learning method for making the crossbar tolerant against the non-ideal effects, the offline learning method can be applied to the memristor crossbar-based neural networks (Zhang *et al* 2021). Here, the pre-trained network is mapped to the memristor crossbar. In this learning method, the memristor crossbar

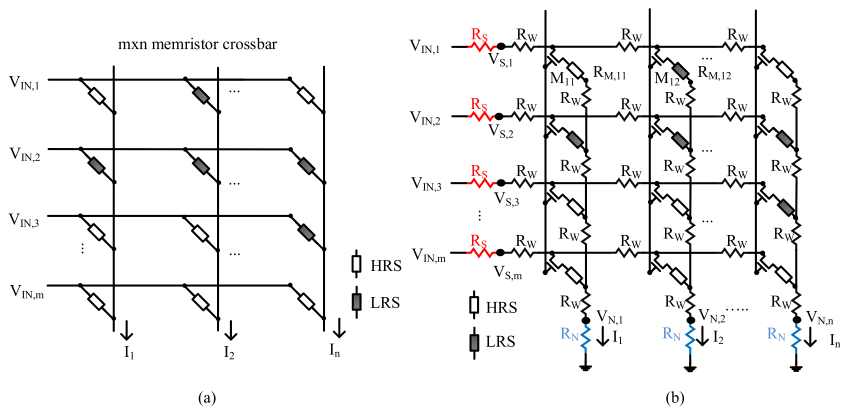

Figure 9. (a) The ideal memristor crossbar without parasitic resistances. (b) The non-ideal memristor with parasitic resistances such as source, neuron, and line resistance.

only performs the inference operation. The non-ideality of weight updates can be concealed by iterative programming with a write-verify technique, reading the conductance and rewriting it iteratively for achieving better accuracy.

The online learning or *in situ* learning can allow the weight updating process to take into account the hardware imperfections. The *in situ* learning helps the crossbar adapt itself to the hardware non-ideal effects such as process variations, cell defects, and non-linearity problems (Li and Belkin 2018). The hybrid learning scheme has demonstrated to be effective for training the memristor-based neural networks. It combines the offline learning and online learning for training the memristor neural networks to achieve better neural network's performance (Li and Belkin 2018). The pre-trained neural network is mapped to the memristor crossbar and the adaptation process is performed on the crossbar for adapting the network's parameters to the target synaptic weights.

4. Parasitic resistance correction techniques

4.1. Correction technique of inference error due to parasitic source and neuron resistance in memristor crossbar

Figure 9(a) depicts an ideal memristor crossbar with HRS and LRS memristor cells. White and grey boxes represent HRS and LRS, respectively. The input voltages $V_{\mathrm{IN},1}$, $V_{\mathrm{IN},2}$, and $V_{\mathrm{IN,m}}$ are applied to row #1, row #2, and row #*m*, respectively. I_1, I_2, and I_n are the column currents for columns #1, #2, and #*n*, respectively. The numbers '*m*' and '*n*' in this figure represent the number of rows and columns in the crossbar, respectively. In figure 9(a), parasitic crossbar resistances such as line resistance, neuron resistance, and source resistance are assumed to be zero in the ideal crossbar.

Figure 9(b) depicts a non-ideal memristor crossbar schematic with parasitic crossbar resistances such as R_{S}, R_{W}, and R_{N} taken into account (Chakraborty *et al* 2018). Source resistance, line resistance, and neuron resistance are represented by R_{S}, R_{W}, and R_{N}, respectively. HRS and LRS are represented in this diagram by white and grey boxes, respectively. $V_{\mathrm{IN},1}$, $V_{\mathrm{IN},2}$, and $V_{\mathrm{IN},m}$ are the input row voltages for row #1, row #2, and row #*m*, respectively, in figure 9(b). The numbers '*m*' and '*n*' represent the number of rows and columns in the crossbar, respectively. The source voltages on the rows are denoted by $V_{\mathrm{S},1}$, $V_{\mathrm{S},2}$, and $V_{\mathrm{S},m}$. Because of R_{S} and R_{W}, the source voltages are degraded. The crossbar currents cause voltage drops on R_{S} and R_{W}, which can degrade the source voltages in proportion to the amounts of current flowing through the resistances. I_1, I_2, and I_n are the column currents of columns #1, #2, and #*n*, respectively. Similarly, for column #1, column #2, and column #*n*, $V_{\mathrm{N},1}$, $V_{\mathrm{N},2}$, and $V_{\mathrm{N},n}$ are neuron voltages on the columns, respectively. The neuron voltages are affected by R_{N} and R_{W}, similarly with the source voltages. M_{11} and M_{12} in the crossbar are the transistors for controlling memristors $R_{\mathrm{M},11}$ and $R_{\mathrm{M},12}$.

$R_{\mathrm{M},11}$ is a memristor cell that is linked to row #1 and column #1. Similarly, $R_{\mathrm{M},12}$ is a memristor cell that is linked to row #1 and column #2. When compared to the ideal crossbar in figure 9(a), the source, line, and neuron resistance in the non-ideal crossbar can change the source voltages and column currents in figure 9(b), resulting in degrading neural network's performance.

Figure 10 depicts a schematic of a memristor crossbar with parasitic R_{S} and R_{N} correction. The correction circuit on the left compensates for the loss of source voltage. Figure 10 shows $V_{\mathrm{IN},1}$ entering row #1 and making $V_{\mathrm{R}1}$. Then, $V_{\mathrm{R}1}$ is an amplified version of the input voltage with $R_{\mathrm{R},1}$ and R_1. The amplifier with $R_{\mathrm{R},1}$ and R_1

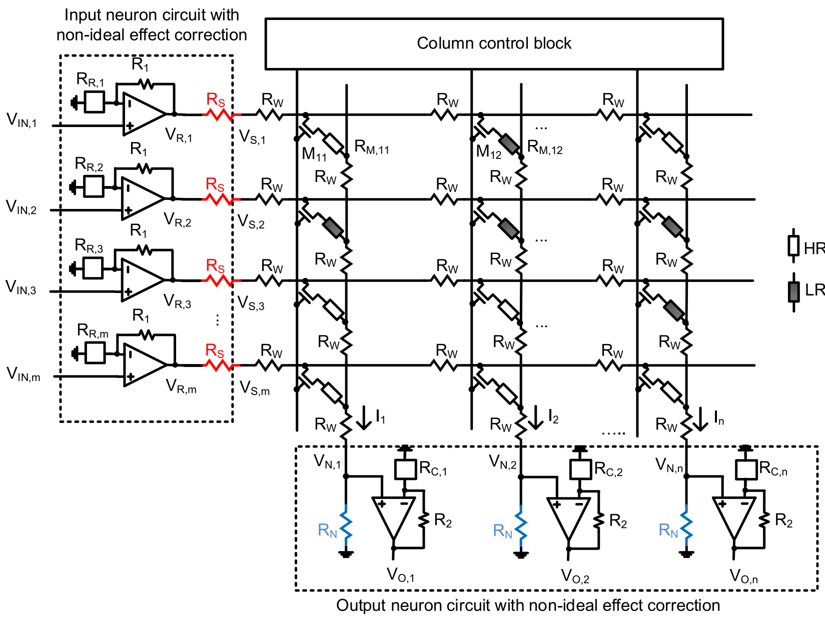

Figure 10. The schematic of memristor crossbar with correction of source and neuron resistance. Here the correction circuit in the left is for compensating the loss of source voltage. The correction circuit added to the bottom is for reducing the loss of neuron voltage.

can compensate for the source voltage degradation, which is caused primarily by the voltage drop across the source resistance. The correction circuit added at the bottom is for reducing the neuron voltage loss. The neuron voltages in figure 10 can be affected by R_N and R_W along the crossbar columns in the same way that the source voltages are degraded. $V_{N,1}$ is a neuron voltage from column #1 that is amplified with $R_{C,1}$ and R_2 to compensate for the neuron voltage degradation (Nguyen *et al* 2021b).

Figure 11(a) depicts a memristor crossbar HRS-LRS map with 64 rows and 64 columns. Figure 11(b) shows the percentage error in source voltage from row #1 to row #64. In this case, $R_S = 2\ \text{k}\Omega$, $R_N = 2\ \text{k}\Omega$, and $R_W = 1\ \Omega$ per cell. In the red line, the percentage error between the crossbars without and with parasitic resistances. The correction circuit is not considered in the red line. The correction circuit significantly reduces the percentage error, as shown by the black line in figure 11(b).

Without the parasitic resistance correction, the average percentage error can reach as high as 36.7 percent. The percentage error of the source voltage can be reduced from 36.7 percent to 7.5 percent when the correction circuit is used in the black line in figure 11(b). Figure 11(c) depicts the neuron voltage percentage error from column #1 to column #64. Without the correction, the average percentage error is as high as 65.5 percent. The correction circuit reduces the percentage error from 65.5 to 8.6 percent. If the correction circuit is used, the percentage error seems decreased to about 1/7.

In figures 11(b) and (c), the line resistance value as small as 1 Ω was obtained from the experimental measurement, where the 40 nm 128 × 128 RRAM crossbar was fabricated in TSMC foundry with the line resistance of ∼1.1 Ω per cell (Murali *et al* 2020). The source and neuron resistance used in this paper were obtained from the circuit simulation. The maximum limits of parasitic neuron and source resistance can be decided from the number of LRS cells per column and that per row in the crossbar, respectively. In addition, the LRS conductance value can affect the maximum limits of neuron and source resistance, too. In figure 11(b), if the parasitic neuron and source resistance become larger than 2 kΩ, the voltage drops on the parasitic resistance can be comparable to the voltage drops on the memristor cells. If so, the column current can be affected more by the parasitic resistance than the memristor cells. As the parasitic resistance becomes larger, the read voltage becomes smaller, resulting in the read failure. Thus, the parasitic neuron and source resistance should be limited within the maximum values. In this paper, the maximum neuron and source resistance allowable was calculated as large as 2 kΩ, considering the read voltage margin.

4.2. Correction technique of parasitic line resistance in memristor crossbar programming

In memristor crossbars, parasitic line resistance can exist between two crossing points. An amount of voltage drop on line resistance can make difference between the calculation and measurement. By doing so, the neural network's performance realized with memristor crossbars having parasitic line resistance can be degraded from the ideal calculation. Moreover, the line resistance problem becomes more significant, as crossbar's array size is increased. To mitigate the impact of parasitic line resistance, the device selector circuit or external circuit

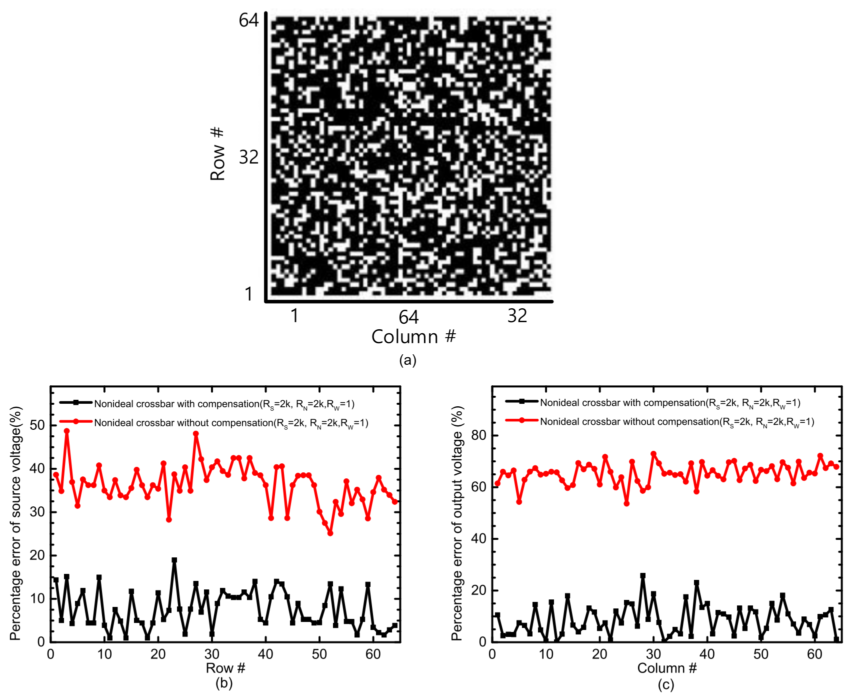

Figure 11. (a) The HRS-LRS map of memristor crossbar with 64 rows and 64 columns. Here the white and black pixels represent LRS and HRS, respectively. (b) The percentage error of source voltage from row #1 to row #64. Here $R_S = 2\ \mathrm{k}\Omega$, $R_N = 2\ \mathrm{k}\Omega$, and $R_W = 1\ \Omega$. The percentage error due to parasitic resistances is shown in the red line. The percentage error is shown in the black line for the correction circuit. (c) The percentage error of neuron voltage from column #1 to column #64.

could be added into the crossbar array (Shin *et al* 2015, Levisse *et al* 2017). However, the memristor crossbar without selection device is more potential for neuromorphic computing systems, and for neural network at least.

An adaptive programming technique considering parasitic resistance correction was developed, to avoid to use the additional correction circuit or selecting devices. Figure 12(a) show a conventional programming method, where each memristive crossing-point is programmed to corresponding memristance value (Truong 2019). In figure 12(a), for the *i*th column, the *j*th input produces the column current, $I_{j,i}$, represented by the dashed line. In the ideal case, where parasitic line resistance is omitted, the column line current, $I_{j,i}$, depends on the memristive crossing-point, $M_{j,i}$. The equivalent resistance of the memristive crossing-point $M_{j,i}$ is as high as $M_{j,i} + iR_W + (m - j + 1)R_W$, as the parasitic line resistance is taken into account, as implied from figure 12(a). To compensate the parasitic line resistance, the memristive crossing-point, $M_{j,i}$, should be programmed to a target value that is smaller than the original value presented in figure 12(a). In particular, the memristive crossing-point of $M_{j,i}$ should be programmed to the target value of $M_{j,i} - (iR_W + (m - j + 1)R_W)$, instead of $M_{j,i}$.

Figure 12(b) shows a conceptual diagram of parasitic line resistance-adapted programing approach, where parasitic line resistance is compensated during the programming phase. The equivalent parasitic line resistance matrix is carried out and the connection matrix is updated by subtracting the equivalent parasitic line resistance matrix from the original connection matrix.

Figure 13 shows the comparison of recognition rate between the conventional programming method and the adaptive programming approach when the parasitic line resistance varies from 0.5 Ω to 3 Ω (Truong 2019). Here, a single-layer network with 26 nodes for character image recognition was implemented on a memristor crossbar array circuit made up of a single crossbar array circuit and a constant term circuit. The crossbar circuit composed 26 columns and a constant term circuit was simulated using CADENCE SPECTRE circuit simulator. The network was trained using MATLAB software and the obtained synaptic weights were mapped to the memristance values of memristor crossbar array. The memristor crossbar array was then programmed to the target values using the $V_{DD}/3$ write scheme.

The improvement of neural network's performance of the adaptive programming scheme seems obvious as shown in figure 13. Here the conventional crossbar programming is compared with the adaptive programming. In the adaptive technique, the trained synaptic weights is mapped to the connection matrix, where the weights are updated using the parasitic line resistance matrix. The updated connection matrix is used for programming the crossbar, as explained earlier. The recognition rate of the conventional programming method

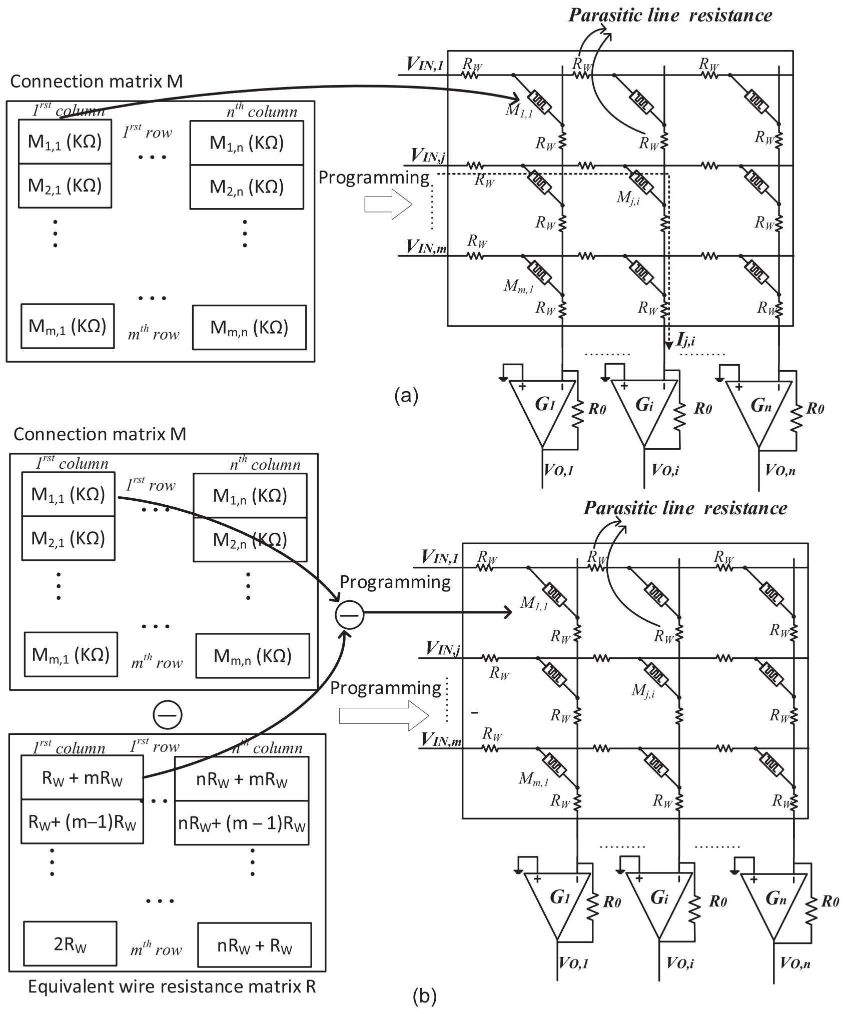

Figure 12. (a) The conventional programming method, (b) the parasitic line resistance correction programming method where the value of connection matrix is updated by subtracting the equivalent line resistance matrix from the original connection matrix.

seems declined dramatically when the line resistance is increased from 0.5 Ω to 3 Ω. In contrast, the memristor crossbar with the adaptive programming scheme could maintain the recognition rate without loss until the line resistance = 3 Ω.

5. Energy-efficient programming of memristor crossbars

LRS cells in memristor crossbars can act as active bits in binary-memristor neural networks. On the contrary, HRS cells can be considered inactive bits. The active bits affect the crossbar's column current more significantly than the inactive ones when calculating vector-matrix multiplication (Pham *et al* 2019a). As a result, calculating exactly the crossbar's column current decided by active bits is critical for achieving the neural network's performance better. To do so, a precise programming of LRS cells is needed in transferring the calculated synaptic weights into the memristor crossbar. The fine programming scheme of memristors in figure 14(a) can be used to program LRS precisely, where the number of programming pulses should be large because of the fine modulation of pulse amplitude (Pham *et al* 2019a). Figure 14(a) indicates that the fine programming scheme of memristors has as many as 30 programming pulses. By doing so, the modulation of programming pulse's amplitude can be controlled little by little for precise tuning of memristor's conductance.

On the contrary, the coarse scheme can be used for programming the inactive bits. As mentioned earlier, the inactive bits are HRS cells. The coarse programming scheme, as shown in figure 14(b), uses only three pulses to program memristors to HRS coarsely. The small number of the programming pulses in figure 14(b) is extremely beneficial in saving a significant amount of programming energy of memristor crossbars.

In figures 14(a) and (b), the open-circle symbols and the red line represent the measured data and the behavioural model, respectively. The behavioural model equations of memristors used in figure 1(b) were obtained from the reference (Truong *et al* 2016). The equations are implemented in VERILOG-A in the

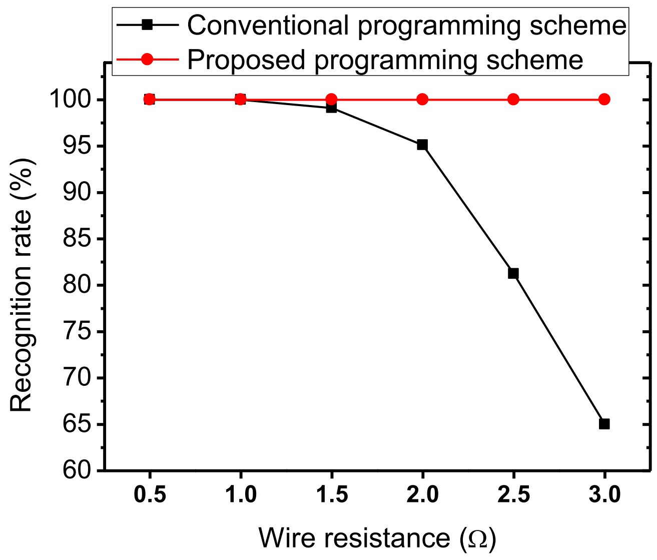

Figure 13. The comparison of recognition rate between the conventional programming method and the adaptive programming approach when the parasitic line resistance is varied from 0.5 Ω to 3 Ω.

CADENCE SPECTRE. The measured memristor devices in figures 14(a) and (b) are also explained in the reference (Truong *et al* 2016).

For analysing various combinations of the fine and coarse amplitude modulation schemes for active LRS and inactive HRS bits, four cases are compared in figures 15(a) and (b). The four cases are Fine-HRS and Fine-LRS (F-F), Coarse-HRS and Fine-LRS (C-F), Fine-HRS and Coarse-LRS (F-C), and Coarse-HRS and Coarse-LRS (F-C) (C-C), respectively. HRS and LRS programmed with the fine scheme are referred to as Fine-HRS and Fine-LRS, respectively. Coarse-HRS and Coarse-LRS are HRS and LRS that have been programmed using the coarse scheme, respectively. C-F has the lowest error-energy product among the four cases. As shown in figure 15(b), C-F reduces the product by nearly 50% more than F-F. Here, the neural network for testing MNIST data set is implemented by memristor crossbars. The neural network architecture used in the simulation is composed of 784 input neurons, 100 hidden neurons, and 10 output neurons. In the simulation, HRS/LRS ratio is assumed 10, as shown in figures 15(a) and (b).

The fine and coarse programming schemes of memristors shown in figures 14(a) and (b) can control the memristor's conductance variation within 5% and 30%, respectively. For the fine programming scheme, the variation of programmed memristor's conductance as small as 5% can be improved better in the recent memristor devices based on WOx, where the variation measured was only as small as ∼2% (Choi *et al* 2021). The memristor's conductance variation seems to be controlled by not only the programming method but also the material property.

The fine and coarse programming schemes can be combined for reducing the number of programming pulses for multi-state memristor crossbars (Le *et al* 2021). The range-dependent adaptive resistance tuning scheme used the coarse programming pulse to reach a target conductance roughly in the starting programming phase (Le *et al* 2021). Then, as the target conductance is approached, the fine programming scheme start to be applied to the memristor to tune the programmed conductance precisely. Combining the coarse and fine programming schemes can save the programming time and energy.

Finally, it should be noted that the other non-ideal effects such as retention, endurance, drift, etc may affect the neural networks implemented with memristor crossbars (Chih *et al* 2021). The various circuit techniques were developed to mitigate these reliability problems (Chou *et al* 2018). The non-ideal effects related to the reliability issues are very important and should be considered in implementing hardware of edge intelligence, too, as memristor defects and process variations were considered in realizing the memristor-crossbar neural networks.

6. Summary

In the IoT era, edge intelligence is critical for overcoming the communication and computing energy crisis, which is unavoidable if cloud computing is used exclusively. Memristor crossbars with in-memory computing

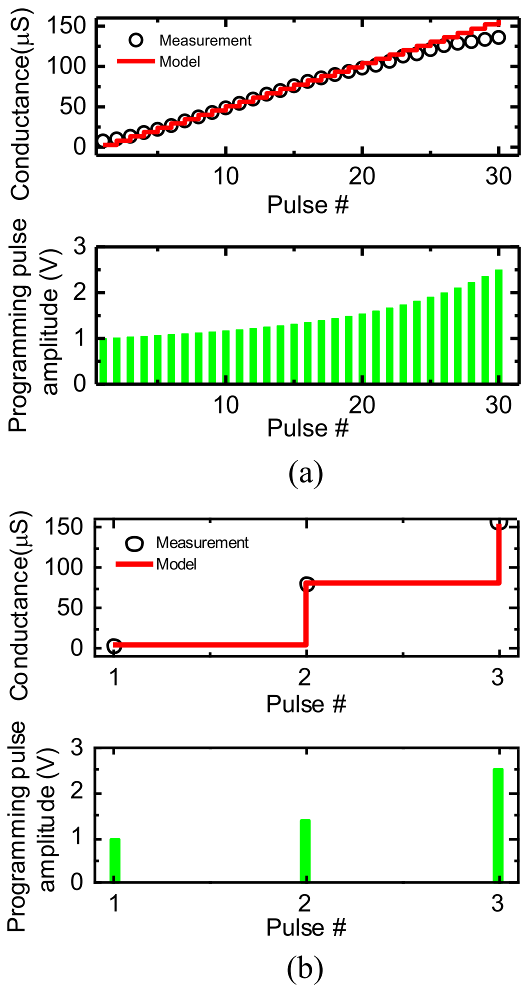

Figure 14. (a) Memristor's conductance programmed with the fine amplitude-modulation scheme of 30 programming pulses. (b) Memristor's conductance programmed with the coarse amplitude modulation scheme of 3 programming pulses.

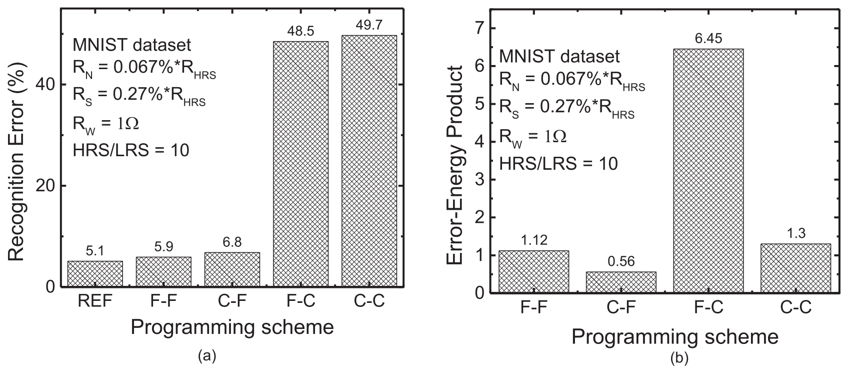

Figure 15. (a) Recognition errors of (1) Fine-HRS and Fine-LRS (F-F), (2) Coarse-HRS and Fine-LRS (C-F), (3) Fine-HRS and Coarse-LRS (F-C), and (4) Coarse-HRS and Course-LRS (C-C). (b) Error-programming energy products of the four schemes.

may be suitable for realizing edge intelligence hardware. They can perform both memory and computing functions, allowing for the development of low-power computing architectures that go beyond the von Neumann computer.

This paper discussed various techniques for implementing edge-intelligence hardware with memristor crossbars, such as quantization, training, parasitic resistance correction, and low-power crossbar programming. In particular, the memristor crossbars were used in section 2 to realize quantized neural networks with binary and ternary synapses for implementing simple but robust operation of in-memory computing with memristors. In section 3, the chip-in-the-loop training of memristor crossbars was discussed in order to prevent memristor defects from degrading edge intelligence performance. Another undesirable effect in memristor crossbars is parasitic resistances such as source, line, and neuron resistance, which worsens as crossbar size increases. In section 4, we talked about how to compensate for parasitic resistances like source, line, and neuron resistance. We discussed the energy-efficient programming method for updating synaptic weights in memristor crossbars in section 5.

Acknowledgments

The work was financially supported by NRF-2019K1A3A1A25000279, NRF-2021R1A2C1011631, and NRF-2021M3F3A2A01037972, NRF-2022R1A5A7000765, N62909-20-1-2021-P00001-Memristor-based Neural Net, and SRFC-TA1903-01. The CAD tools were supported from IC Design Education Center (IDEC), Korea.

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

ORCID iDs

Kyeong-Sik Min https://orcid.org/0000-0002-1518-7037

References

Adam G C *et al* 2016 3D memristor crossbars for analog and neuromorphic computing applications *IEEE Trans. Electron Devices* **64** 312–8

Alemdar H *et al* 2017 Ternary neural networks for resource-efficient AI applications *2017 Int. Joint Conf. Neural Networks (IJCNN)* (Piscataway, NJ: IEEE) pp 2547–54

Amin S U and Hossain M S 2021 Edge intelligence and internet of things in healthcare: a survey *IEEE Access* **9** 45–59

Amirsoleimani A *et al* 2020 In-memory vector-matrix multiplication in monolithic complementary metal–oxide–semiconductor-memristor integrated circuits: design choices, challenges, and perspectives *Adv. Intell. Syst.* **2** 2000115

Bengio Y, Léonard N and Courville A 2013 Estimating or propagating gradients through stochastic neurons for conditional computation (arXiv:1308.3432)

Bhat S *et al* 2017 SkyNet: memristor-based 3D IC for artificial neural networks *2017 IEEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH)*

Bohr M T and Young I A 2017 CMOS scaling trends and beyond *IEEE Micro* **37** 20–9

Chakrabarti B *et al* 2017 A multiply-add engine with monolithically integrated 3D memristor crossbar/CMOS hybrid circuit *Sci. Rep.* **7** 42429

Chakraborty I, Roy D and Roy K 2018 Technology aware training in memristive neuromorphic systems for nonideal synaptic crossbars *IEEE Trans. Emerg. Top. Comput. Intell.* **2** 335–44

Chen J, Wen S, Shi K and Yang Y 2021 Highly parallelized memristive binary neural network *Neural Netw.* **144** 565–72

Chih Y-D *et al* 2021 Design challenges and solutions of emerging nonvolatile memory for embedded applications *2021 IEEE Int. Electron Devices Meeting (IEDM)* (Piscataway, NJ: IEEE) pp 2–4

Choi W *et al* 2021 Hardware neural network using hybrid synapses via transfer learning: WO× nano-resistors and TiO× RRAM synapse for energy-efficient edge-AI sensor *2021 IEEE Int. Electron Devices Meeting (IEDM)* (Piscataway, NJ: IEEE) pp 21–3

Chou C-C *et al* 2018 An N40 256 K $\times$ 44 embedded RRAM macro with SL-precharge SA and low-voltage current limiter to improve read and write performance *2018 IEEE Int. Solid-State Circuits Conf. (ISSCC)* (Piscataway, NJ: IEEE) pp 478–80

Courbariaux M *et al* 2016 Binarized neural networks: training deep neural networks with weights and activations constrained to $+1$ or -1 (arXiv:1602.02830)

Deng S, Zhao H, Fang W, Yin J, Dustdar S and Zomaya A Y 2020 Edge intelligence: the confluence of edge computing and artificial intelligence *IEEE Internet Things J.* **7** 7457–69

Duan S, Hu X, Dong Z, Wang L and Mazumder P 2015 Memristor-based cellular nonlinear/neural network: design, analysis, and applications *IEEE Trans. Neural Netw. Learn. Syst.* **26** 1202–13

Ghosh A M and Grolinger K 2021 Edge-cloud computing for internet of things data analytics: embedding intelligence in the edge with deep learning *IEEE Trans. Ind. Inf.* **17** 2191–200

Gusev M and Dustdar S 2018 Going back to the roots ×2014; the evolution of edge computing, an IoT perspective *IEEE Internet Comput.* **22** 5–15

He K *et al* 2016 Deep residual learning for image recognition *Proc. IEEE Conf. Computer Vision and Pattern Recognition* pp 770–8

Holst A 2021 Amount of data created, consumed and stored 2010–2025, Statista available at: https://statista.com/statistics/871513/worldwide-data-created/

Hu M, Li H, Chen Y, Wu Q, Rose G S and Linderman R W 2014 Memristor crossbar-based neuromorphic computing system: a case study *IEEE Trans. Neural Netw. Learn. Syst.* **25** 1864–78

Hu M *et al* 2018 Memristor-based analog computation and neural network classification with a dot product engine *Adv. Mater.* **30** 1705914

James A P 2019 A hybrid memristor-CMOS chip for AI *Nat. Electron.* **2** 268–9

Jeong H and Shi L 2018 Memristor devices for neural networks *J. Phys. D: Appl. Phys.* **52** 023003

Jo S H, Chang T, Ebong I, Bhadviya B B, Mazumder P and Lu W 2010 Nanoscale memristor device as synapse in neuromorphic systems *Nano Lett.* **10** 1297–301

Keshavarzi A and van den Hoek W 2019 Edge intelligence-on the challenging road to a Trillion smart connected IoT devices *IEEE Des. Test* **36** 41–64

Keshavarzi A, Ni K, van Den Hoek W, Datta S and Raychowdhury A 2020 FerroElectronics for edge intelligence *IEEE Micro* **40** 33–48

Kim Y *et al* 2019 Memristor crossbar array for binarized neural networks *AIP Adv.* **9** 045131

Klein J-O, Garda P and Pujol H 1995 Chip-in-the-loop learning algorithm for Boltzmann machine *Electron. Lett.* **31** 986–8

Krestinskaya O, James A P and Chua L O 2019 Neuromemristive circuits for edge computing: a review *IEEE Trans. Neural Netw. Learn. Syst.* **31** 4–23

Krizhevsky A, Nair V and Hinton G 2018 CIFAR-10 and CIFAR-100 datasets available online: https://cs.toronto.edu/kriz/cifar.html (accessed on 20 October 2018)

Le B Q *et al* 2021 RADAR: a fast and energy-efficient programming technique for multiple bits-per-cell RRAM arrays *IEEE Trans. Electron Devices* **68** 4397–403

Levisse A *et al* 2017 Architecture, design and technology guidelines for crosspoint memories *Proc. IEEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH)* pp 55–60

Li C and Belkin D 2018 Efficient and self-adaptive *in situ* learning in multilayer memristor neural networks *Nat. Commun.* **9** 2385

Li C *et al* 2018a Analogue signal and image processing with large memristor crossbars *Nat. Electron.* **1** 52–9

Li Y, Wang Z, Midya R, Xia Q and Yang J J 2018b Review of memristor devices in neuromorphic computing: materials sciences and device challenges *J. Phys. D: Appl. Phys.* **51** 503002

Lin P *et al* 2020 Three-dimensional memristor circuits as complex neural networks *Nat. Electron.* **3** 225–32

Linn E, Rosezin R, Tappertzhofen S, Böttger U and Waser R 2012 Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations *Nanotechnology* **23** 305205

Murali G *et al* 2020 Heterogeneous mixed-signal monolithic 3D in-memory computing using resistive RAM *IEEE Trans. Very Large Scale Integr. Syst.* **29** 386–96

Nguyen T-V, Pham K-V and Min K-S 2019 Hybrid circuit of memristor and complementary metal–oxide–semiconductor for defect-tolerant spatial pooling with Boost-factor Adjustment *Materials* **12** 2122

Nguyen T V, An J and Min K S 2021a Comparative study on quantization-aware training of memristor crossbars for reducing inference power of neural networks at the edge *2021 Int. Joint Conf. Neural Networks (IJCNN)* (Piscataway, NJ: IEEE) pp 1–6

Nguyen T V, An J and Min K-S 2021b Memristor-CMOS hybrid neuron circuit with nonideal-effect correction related to parasitic resistance for binary-memristor-crossbar neural networks *Micromachines* **12** 791

Pham K V and Min K-S 2019 Non-ideal effects of memristor-CMOS hybrid circuits for realizing multiple-layer neural networks *2019 IEEE Int. Symp. Circuits and Systems (ISCAS)* (Piscataway, NJ: IEEE) pp 1–5

Pham K V, Nguyen T V, Tran S B, Nam H, Lee M J, Choi B J, Truong S N and Min K-S 2018 Memristor binarized neural networks *J. Semicond. Technol. Sci* **18** 568–77

Pham K, Tran S, Nguyen T and Min K-S 2019a Asymmetrical training scheme of binary-memristor-crossbar-based neural networks for energy-efficient edge-computing nanoscale systems *Micromachines* **10** 141

Pham K V, Nguyen T V and Min K-S 2019b Defect-tolerant crossbar training of memristor ternary neural networks *2019 26th IEEE Int. Conf. Electronics, Circuits and Systems (ICECS)* (Piscataway, NJ: IEEE) pp 486–9

Pham K V, Nguyen T V and Min K-S 2019c Partial-gated memristor crossbar for fast and power-efficient defect-tolerant training *Micromachines* **10** 245

Pham K V, Nguyen T V and Min K-S 2019d Defect-tolerant and energy-efficient training of multi-valued and binary memristor crossbars for near-sensor cognitive computing *Proc. Int. Conf. ASIC* (Piscataway, NJ: IEEE) pp 1–4

Plastiras G *et al* 2018 Edge intelligence: challenges and opportunities of near-sensor machine learning applications *2018 IEEE 29th Int. Conf. Application Specific Systems, Architectures and Processors (ASAP)* (Piscataway, NJ: IEEE) pp 1–7

Premsankar G, Di Francesco M and Taleb T 2018 Edge computing for the internet of things: a case study *IEEE Internet Things J.* **5** 1275–84

Qin H, Gong R, Liu X, Bai X, Song J and Sebe N 2020b Binary neural networks: a survey *Pattern Recogn.* **105** 107281

Qin Y-F, Bao H, Wang F, Chen J, Li Y and Miao X-S 2020a Recent progress on memristive convolutional neural networks for edge intelligence *Adv. Intell. Syst.* **2** 2000114

Ran H *et al* 2020 Memristor-based edge computing of ShuffleNetV2 for image classification *IEEE Trans. Comput. -Aided Des. Integr. Circuits Syst.* **2** 324–34

Ronao C A and Cho S-B 2016 Human activity recognition with smartphone sensors using deep learning neural networks *Expert Syst. Appl.* **59** 235–44

Sah M P, Yang C, Kim H, Muthuswamy B, Jevtic J and Chua L 2015 A generic model of memristors with parasitic components *IEEE Trans. Circuits Syst.* I **62** 891–8

Sebastian A, Le Gallo M and Eleftheriou E 2019 Computational phase-change memory: beyond von Neumann computing *J. Phys. D: Appl. Phys.* **52** 443002

Shang C, Yang F, Huang D and Lyu W 2014 Data-driven soft sensor development based on deep learning technique *J. Process Control* **24** 223–33

Sheng X, Graves C E, Kumar S, Li X, Buchanan B, Zheng L, Lam S, Li C and Strachan J P 2019 Low-conductance and multilevel CMOS-integrated nanoscale oxide memristors *Adv. Electron. Mater.* **5** 1800876

Shin S, Byeon S-D, Song J, Truong S N, Mo H-S, Kim D and Min K-S 2015 Dynamic reference scheme with improved read voltage margin for compensating cell-position and background-pattern dependencies in pure memristor array *J. Semicond. Technol. Sci.* **15** 685–94

Singh A *et al* 2021 Low-power memristor-based computing for edge-AI applications *2021 IEEE Int. Symp. Circuits and Systems (ISCAS)* (Piscataway, NJ: IEEE) pp 1–5

Strukov D B, Snider G S, Stewart D R and Williams R S 2008 The missing memristor found *Nature* **453** 80–3

Suh K-D *et al* 1995 A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme *IEEE J. Solid-State Circuits* **30** 1149–56

Sun X and Ansari N 2016 EdgeIoT: mobile edge computing for the internet of things *IEEE Commun. Mag.* **54** 22–9

Truong S N 2019 A parasitic resistance-adapted programming scheme for memristor crossbar-based neuromorphic computing systems *Materials* **12** 1–12

Truong S N, Ham S-J and Min K-S 2014 Neuromorphic crossbar circuit with nanoscale filamentary-switching binary memristors for speech recognition *Nanoscale Res. Lett.* **9** 1–9

Truong S N, Pham K V, Yang W, Shin S, Pedrotti K and Min K-S 2016 New pulse amplitude modulation for fine tuning of memristor synapses *Microelectron. J.* **55** 162–8

Tunali O and Altun M 2017 A survey of fault-tolerance algorithms for reconfigurable nano-crossbar arrays *ACM Comput. Surv.* **50** 1–35

Wang T-Y *et al* 2020 Three-dimensional nanoscale flexible memristor networks with ultralow power for information transmission and processing application *Nano Lett.* **20** 4111–20

Wright C D, Hosseini P and Diosdado J A V 2013 Beyond von Neumann computing with nanoscale phase-change memory devices *Adv. Funct. Mater.* **23** 2248–54

Xue C *et al* 2020 15.4 A 22 nm 2 Mb ReRAM compute-in-memory macro with 121-28 TOPS/W for multibit MAC computing for Tiny AI edge devices *2020 IEEE Int. Solid-State Circuits Conf. (ISSCC)* pp 244–6

Yakopcic C, Alom M-Z and Taha T-M 2016 Memristor crossbar deep network implementation based on a convolutional neural network *Int. Joint Conf. Neural Networks (IJCNN)* pp 963–70

Yao P, Wu H, Gao B, Tang J, Zhang Q, Zhang W, Yang J J and Qian H 2020 Fully hardware-implemented memristor convolutional neural network *Nature* **577** 641–6

Yeo I, Chu M, Gi S-G, Hwang H and Lee B-G 2019 Stuck-at-fault tolerant schemes for memristor crossbar array-based neural networks *IEEE Trans. Electron Devices* **66** 2937–45

Zhang W *et al* 2021 ROA: a rapid learning scheme for in-situ memristor networks *Front. Artif. Intell.* **4** 692065

Zhou Z, Chen X, Li E, Zeng L, Luo K and Zhang J 2019 Edge intelligence: paving the last mile of artificial intelligence with edge computing *Proc. IEEE* **107** 1738–62