Memristor Neural Network Design

Anping Huang, Xinjiang Zhang, Runmiao Li and
Yu Chi

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.69929

Abstract

Neural network, a powerful learning model, has archived amazing results. However, the current Von Neumann computing system–based implementations of neural networks are suffering from memory wall and communication bottleneck problems ascribing to the Complementary Metal Oxide Semiconductor (CMOS) technology scaling down and communication gap. Memristor, a two terminal nanosolid state nonvolatile resistive switching, can provide energy-efficient neuromorphic computing with its synaptic behavior. Crossbar architecture can be used to perform neural computations because of its high density and parallel computation. Thus, neural networks based on memristor crossbar will perform better in real world applications. In this chapter, the design of different neural network architectures based on memristor is introduced, including spiking neural networks, multilayer neural networks, convolution neural networks, and recurrent neural networks. And the brief introduction, the architecture, the computing circuits, and the training algorithm of each kind of neural networks are presented by instances. The potential applications and the prospects of memristor-based neural network system are discussed.

Keywords: memristors, neural networks, deep learning, neuromorphic computing, analog computing

1. Introduction

Neural networks, composing multiple processing layers, have achieved amazing results, such as AlphaGo, DNC and WaveNet. However conventional neural networks based on Von Neumann systems have many challenges [1]. In Von Neumann computing system, the computing process and external memory are separated by a shared bus between data and program memory as shown in **Figure 1**, which is so called Von Neumann bottleneck. In Von Neumann computing system, a single processor has to simulate many neurons and the synapses between neurons. In addition, the bottleneck leads the energy-hungry data communication when

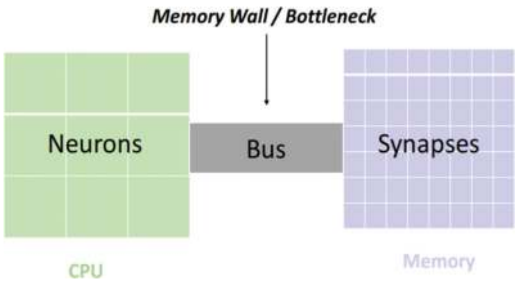

Figure 1. Von Neumann computing system bottleneck.

updating the neurons states and retrieving the synapse states, and when simulates a large-scale neural networks, the massages among processors will explode [2]. These defects make the Von Neumann computing system based neural network power hungrier, low density, and slow speed. In order to overcome these defects, a novel Nano device and computing architecture need proposing. Memristor crossbar is considered to be the most promising candidate to solve these problems [3]. Memristor crossbar is a high density, power efficiency computing-in-memory architecture. Thus, this chapter presents different design paradigm of memristor-based neural networks, including spiking neural networks (SNNs), multilayer neural networks (MNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs).

2. Memristor neural networks

2.1. Memristor

Memristor was conceived by Leon Chua according to the symmetry of circuit theory in 1971 [4] and funded by HP lab in 2008 [5]. Memristor is a nano two-terminal nonvolatile device, with a Lissajous' IV curve. In mathematical, the model of memristor can be express as (take an example of HP memristor) [6]

$$i(t) = \frac{1}{R_{ON}w(t) + R_{OFF}(1 - w(t))} v(t) \tag{1}$$

$$i(t) = G(\varphi(t))v(t) \tag{2}$$

Here, w(t) stands for the normalized position of the conduction front between the O^{2-} vacancy-rich and O^{2-} vacancy-poor regions. The range of w(t) is from 0 to 1. $G(\varphi(t))$ is the conductance. The conductance of memristor can be continuous changing when applied control pulse on the memristor. When the negative pulse is applied, the O^{2-} vacancy moves to O^{2-} vacancy-rich region, which cause the conductance decrease, and vice versa. This result is similar to the phenomenon in biological synapse, such that memristor can simulate the dynamic of synapse.

2.2. Memristor merits

Memristor as the forth device, comparing with conventional computing system such as CPU and GPU, has many advantages. First, memristor is a two-terminal nonvolatile device, resulting in the low power consumption [7]. Second, memristor is compatible with the CMOS, and it can be integrated with higher density [4]. Third, the size of memristor is in nanoscale, such that the switching speed fast [8]. These characteristics make memristor become a promising candidate for neuromorphic computing. In recent years, many researchers have performed various experiments in neural network with memristor for synapse and neurons.

2.2.1. Memristor as synapse

Human brain can perform complex tasks such as unstructured data classification and image recognition. In human brain, excitatory and inhibitory postsynaptic potentials are delivered from presynaptic neuron to postsynaptic neuron through chemical and electrical signal at synapses, driving the change of synaptic weight, as shown in **Figure 2**. The synaptic weight is precisely adjusted by the ionic flow through the neurons. In neural networks, this mechanism can be simulated by memristors. There are many samples that memristor used as synapse. In this section, we use SNN as a sample to explain how memristor used as synapse.

As shown in **Figure 3**, a memristor acts as a synapse between two CMOSs neuron, which acts as pre-/postsynaptic neurons, respectively. The input signal of presynaptic neurons reached the postsynaptic neurons through the synapse. When a presynaptic spike is triggered before a postsynaptic spike, equivalently there is a positive voltage applied on the memristor, and then the synaptic weight is increased and vice versa, which is [6] explained as

$$\Delta t = t_{pre} - t_{post} \tag{3}$$

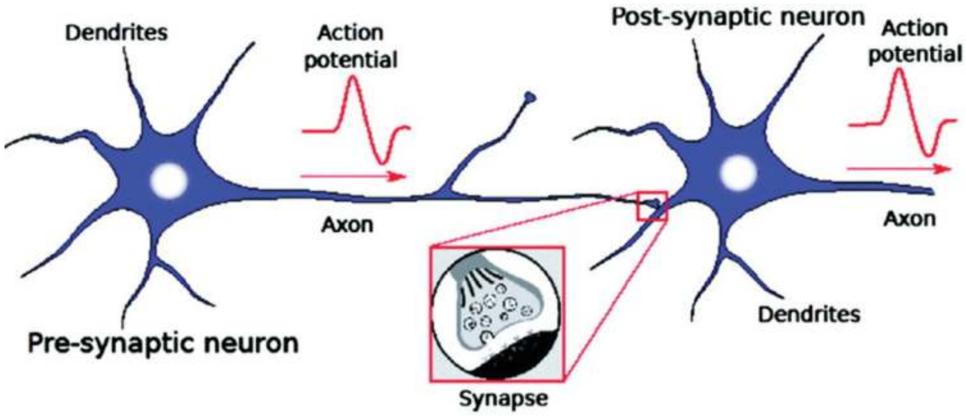

Figure 2. Biological neuron and synapse.

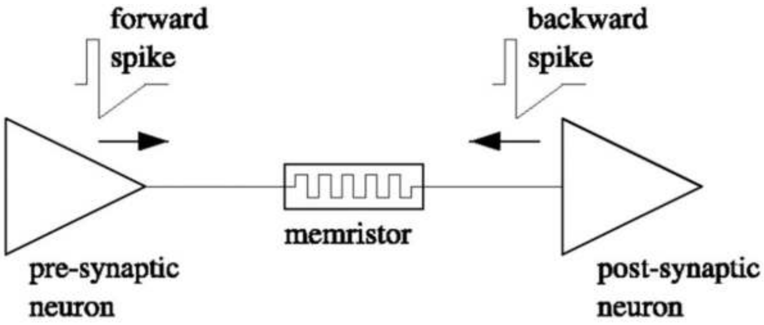

Figure 3. A paradigm of memristor based synapse.

where $t_{pre}(t_{post})$ is the pule weight the presynaptic neuron (postsynaptic neuron) spikes. Δt is the difference between neurons spike time. That means, when $\Delta t > 0$, the synapse weight is increased, and when $\Delta t < 0$, the synaptic weight is decreased.

2.2.2. Memristor as neuron

In biology, the membrane separates the inter-cell ions and enter-cell ions. Based on the electrochemical mechanism, the potential on the sides of membrane is balanced. When the excitatory and inhibitory postsynaptic potentials are arrived, the signals through the dendrites of the neurons and the balance will be destroyed. When the potential surpasses a threshold, the neuron is fired. Emulating these neuronal mechanism, including maintaining the balance of potential, the instantaneous mechanism, and the process of neurotransmission, is the key to implement biological plausible neuromorphic computing system [9].

When a memristor is used to act as a neuron in neural networks, it is not essential that the conductance of memristor implement continuous change, instead to achieve accumulative behavior. When competent pulses applied, the neuron is fired. These pulses can change the conductance state of memristor.

2.3. Memristor crossbar

Memristor crossbar consists of two perpendicular nanowire layers, which act as top electrode and bottom electrode, respectively. The memristive material is laid between two nanowire layers; as a result, memristor is formed at each crosspoint [11]. The schematic diagram of memristor crossbar is shown in **Figure 4**.

Memristor crossbar is suitable for large-scale neural networks implementations. First, it is high density, since crossbar can be vertical stack, and each crosspoint is a memristor. In addition, memristor is nonvolatile, nanoscale and multistate. Second, it is low power consumption, since the crossbar allow memory and computation integrating [10], and memristor is nonvolatile device with a low operation voltage. These advantage of memristor crossbar such that this architecture applied in a wide range of neural networks.

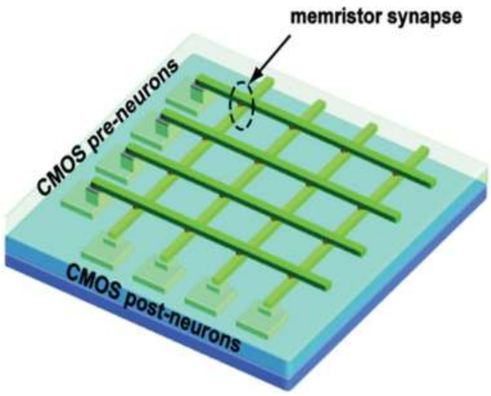

Figure 4. Memristor crossbar.

In neural networks, memristor crossbar has three operations such as read, write, and training. In this section, we use a sample to illustrate how the memristor crossbar read, write and training.

2.3.1. Read operation

In memristor crossbar, the conductance of a single memristor can be read individually. As shown in **Figure 5**, we assume that we will read the m_{ij} memristor, which is the crosspoint of i_{th} top wires and j_{th} bottom wires. The voltage V is applied on the i_{th} top wire, and other top wires and bottom wires are grounded. In this situation, only the m_{ij} memristor is applied the V bias, the current i can be collected on the j_{th} bottom wire. According to Ohm's law, the conductance of m_{ij} memristor M is caculated by $M=V/i$ [11].

2.3.2. Write operation

Similar to reading operation, the conductance of m_{ij} memristor can be written individually. We assume that we will write the m_{ij} memristor. Different amplitude and duration of writing

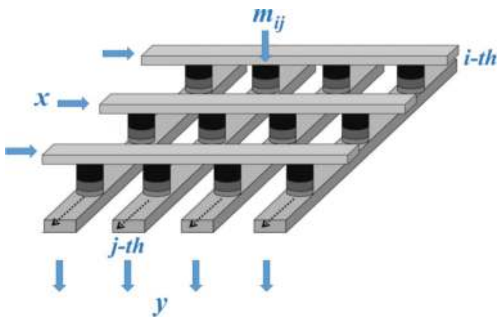

Figure 5. Memristor crossbar readout.

pulses will be directly applied on the target memristor. The i_{th} top wire is applied voltage V, and the j_{th} bottom wire is grounded. Other top wires and bottom wires are applied voltage V/2, then, only the m_{ij} memristor is applied the full voltage V, which is above the threshold and can change the conductance of target memristor. The conductance of other memristors is not changed because the voltage applied on them is 0 [12].

2.3.3. Training operation

Based on the read and write operation, the memristor neural networks are trained to implement practical neural networks. We use a single-layer neural network to explain the training process of neural network. As shown in **Figure 6**, the relationship between input vectors ***U*** and output vectors ***Y*** can be illustrated as [12]:

$$Y_n = W_{n\times m} \times U_m \tag{4}$$

Here, the weight matrix $W_{n\times m}$ represents the synaptic strengths between the two-neuron groups, which are represented by the conductance of corresponding memristors. When we train a memristor crossbar, we first assume we have a set of data. We input the training data, the synaptic weight matrix *W* is updated repeatedly until the difference between the output *y* and the target output *y** become minimum. In each repetition, *W* is adjusted across the gradient of the output error $|y\text{-}y^*|$ as [12]

$$\Delta w_{ij} = \mu\left(\frac{\partial(y-y^*)^2}{\partial w_{ij}}\right) \tag{5}$$

Here, w_{ij} is the synaptic weight in the ***W*** connecting the neuron *i* and *j*, Δw_{ij} is the change of w_{ij} during per update. μ is the training rate.

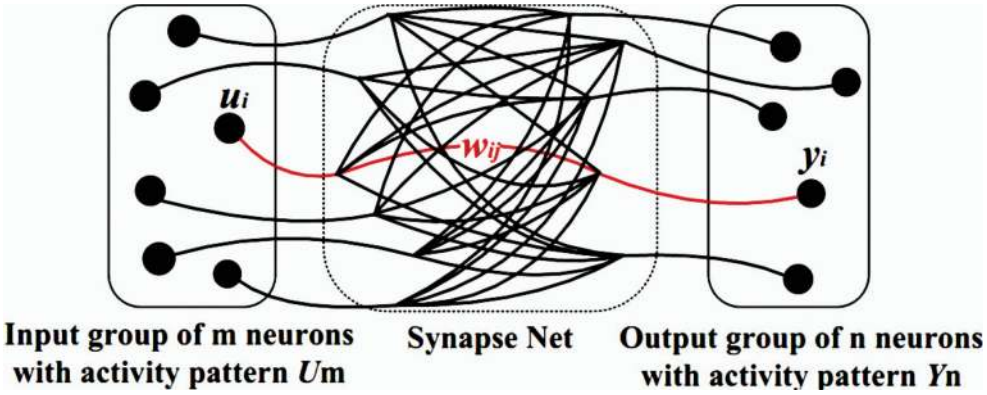
u_i
w_{ij}
y_i
Input group of m neurons with activity pattern *U*m
Synapse Net
Output group of n neurons with activity pattern *Y*n

Figure 6. A single layer neural network [12].

http://dx.doi.org/10.5772/intechopen.69929

3. Design of memristor neural networks

This section discusses different memristor-based neural network design paradigm, including spiking neural networks (SNN), multilayer neural networks (MNN), convolutional neural networks (CNN), and recurrent neural networks (RNN). Each part of these neural networks consists of five subsections, which are the concepts, the architecture, the algorithm, the circuits, and the instance.

3.1. Spiking neural networks

3.1.1. SNN concept

Spiking neural network (SNN), a neural network of neurons interchange information using spikes [13], is neural network based on individual spikes [14]. SNN is a brain-like architecture. The signal in SNNs uses pulse coding rather than rate coding, and allows multiplexing of information as frequency and amplitude. In some electronic SNNs, spikes have the similar waveform shape than biological spikes, but normally in electronic systems spikes are much simpler being represented by a square digital pulse [13]. In SNN, the presence and timing of individual spikes are considered as the means of communication and neural computation. The basic idea on biology is that the more intensive the input, the earlier the spike transmission. Hence, a network of spiking neurons can be designed with *n* input neurons Ni whose firing times are determined through some external mechanism [14].

3.1.2. SNN architecture

In this section, we use a three-layer neural network to illustrate the structure of SNN. In this structure, as shown in **Figure 7**, the multilayer SNNs are fully connected feedforward networks; all neurons between two adjacent layers are connected. All the input neurons and output neurons are multiple spikes, i.e., spikes trains.

In this structure, neurons have a model. Spike response model describes the response of both the sending and receiving neuron to a spike. In this model, the spikes of sending neuron transmitted from presynaptic neurons via synapses to postsynaptic neurons. When all spikes arrive, a postsynaptic potential is accumulated in receiving neuron. The internal state of neuron is defined as the sum of postsynaptic potential induced by all the spikes and affected by the weights for synapses that transmit these input spikes.

Suppose an input neuron has N input synapses. The i_{th} synapse transmits G_i spikes. The arrival time of each spike is denoted as $g_i = t_i^1, t_i^2 \ldots\ldots t_i^g$. The time of the most recent output spike of the neuron prior to the current time t (>0) is $t^{(fr)}$. Then the internal state of the postsynaptic neuron is expressed as

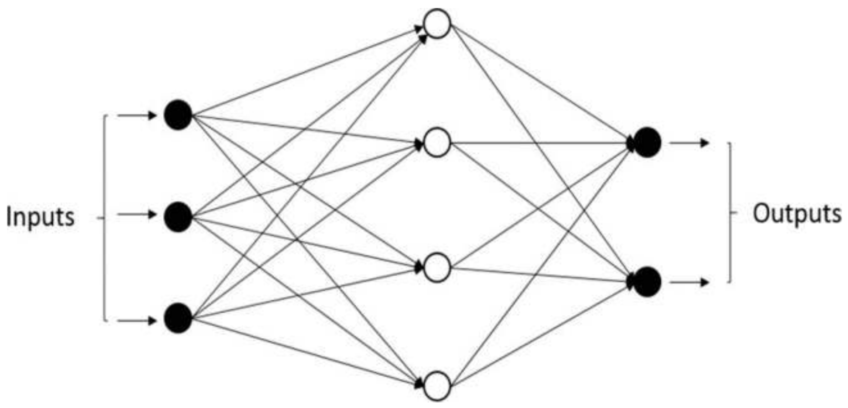

Figure 7. The architecture of SNNs.

$$u(t) = \sum_{i=1}^{N} \sum_{\substack{t_i^{(g)} \epsilon \mathcal{G}_i \\ t_i^{(g)} > t^{(fr)}}} w_i \varepsilon\left(t - t_i^{(g)}\right) + \eta(t - t^{(fr)}) \tag{6}$$

where w_i is the weight for the ith synapse. The postsynaptic potential induced by one spike is determined by the spike response function *ε(t)*, expressed as

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau} e^{1-\frac{t}{\tau}} & \text{if } t > 0 \\ 0 & \text{if } t \le 0 \end{cases} \tag{7}$$

In additional to the model of postsynaptic neuron, SNN has a model, too. For convenience, we assume that the layers are numbered backwards starting from the output layer numbered as layer 1 to the input layer. Every two neurons in adjacent layers connected by K synapses with different transmit delays and weights. The delay of the kth synapse is denoted as d^k.

We assume that there are N_{l+1} neurons in layer $l + 1$ and neuron i, belongs to the layer $l + 1$, has emitted a spike train composed of F_i spikes, the times of firing are denoted $F_i = t_i$, the time of the t_i spike which through the kth synapse arrive at postsynaptic neuron j which is in layer l is $t_i + d^k$. At time t, the internal state of the jth postsynaptic neuron in layer l can be expressed by

$$u_j(t) = \sum_{i=1}^{N_{l+1}} \sum_{k=1}^{K} \sum_{\substack{t_i^{(f)} \epsilon \mathcal{F}_i \\ t_i^{(f)} + d^k > t_j^{(f_r)} + R_a}} w_{ij}^k \varepsilon\left(t - t_i^{(j)} - d^k\right) + \eta(t - t_j^{(f_r)}) \tag{8}$$

where w_{ij}^k is the weight of the k_{th} synapse between presynaptic neuron i and postsynaptic neuron j; $t_j^{(f_r)}$ is the time of the most recent output spike for neuron j prior to the current time t [15].

3.1.3. SNN algorithm

Spike-Timing Dependent Plasticity (STDP) is the synapse strength changing mechanism according to the precise timing of individual pre- and/or postsynaptic spikes. As illustrate in Section 2, the sign of the difference between the pre-/postsynaptic neurons times determines the synaptic weight whether increased. STDP learning in biology is inherently asynchronous and online which means that synaptic incremental update occurs while neurons and synapses transmit spikes and perform computations. In experiment, the synaptic strength is a function of relative timing between the arrival time of a presynaptic spike and the time of generation of a postsynaptic spike as shown in **Figure 8**.

Although the data show stochasticity, we can infer an underlying interpolated function as shown in **Figure 9**.

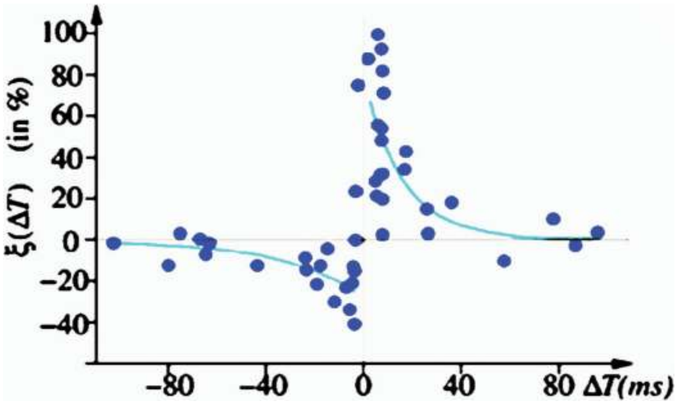

Figure 8. Experimentally measured STDP function on biological synapses [13].

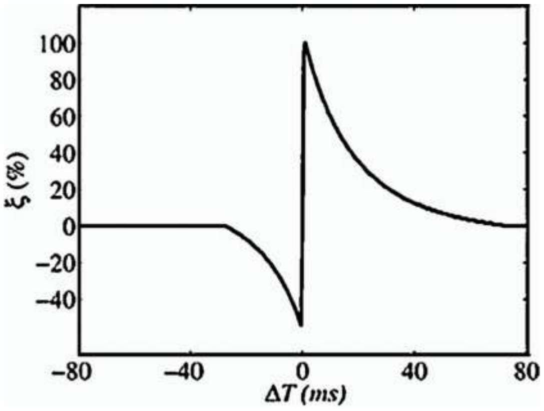

Figure 9. Ideal STDP update function used in computational models of STDP synaptic learning [13].

$$\xi \Delta \mathrm{T} = \begin{cases} a^{+} e^{-\Delta T/\tau^{+}} \ \mathit{if}\ \Delta T > 0 \\ -a^{-} e^{\Delta T/\tau^{-}} \ \mathit{if}\ \Delta T < 0 \end{cases} \tag{9}$$

3.1.4. SNN circuits

SNN with three layers of neurons and two fully connected inter-layer meshes of memristors is shown in **Figure 10**. The neuron layers are fabricated with CMOS devices, and the inter-layer meshes of memristors are made with nanowires on the top of a CMOS substrate [16]. In **Figure 10**, triangles represent the neuron soma, being the flat side its input(dendrites) and the sharp side the output (axon). Dark rectangles are memristors, representing each one synaptic junction. Each neuron controls the voltage at its input and output nodes.

In this SNN circuit, the CMOS-based spiking neurons work basically the same as conventional integrate-and-fire neuron, and use proposed spike shape and specific spike back-propagation. The total current of receiving neuron is given by Ohm's Law by conductance, g, of connected synapses and the voltage drop across the synapses. SNN training process needs building external circuit. In external circuit, the input signals are prepared, and the output signal will be measured in the external circuit.

3.1.5. SNN instances

Memristor behavior is more likely to a bidirectional exponentially grow with voltage, and many mathematical formulations can be used to simulate it. Here, we use a voltage-controlled device as a synapse, whose synaptic weight is represented by the conductance g of memristor. The function of the device is "sinh-like" in the voltage Vmem. The nano device satisfied the formulation as expressed follow

$$\frac{dg}{dt} = Asinh(V_{mem}) \tag{10}$$

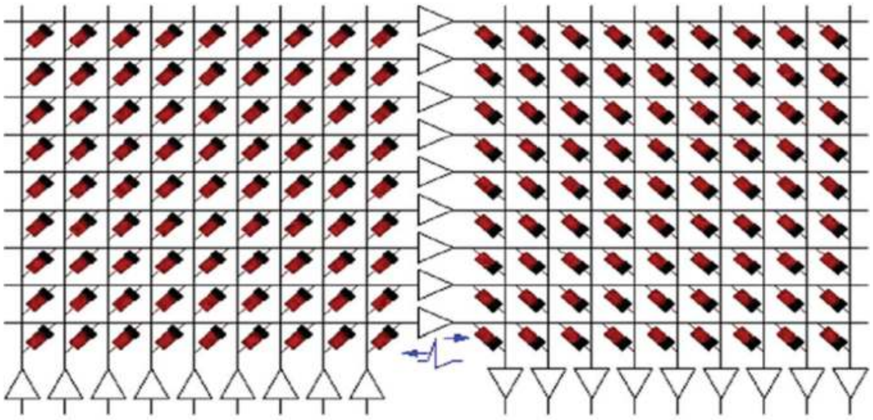

Figure 10. Memristor crossbar based SNNs paradigm [13].

A and B are the parameters which depend upon the memristor material, thickness, size, and it fabrication method.

In this section, we verify the STDP properties of the memristor-based synapses. **Figure 11** is the proposed spike shape, which is similar to the biological spikes. **Figure 12** shows the STDP curves produced by the proposed spike shape. In **Figure 12**, the vertical axis shows the average

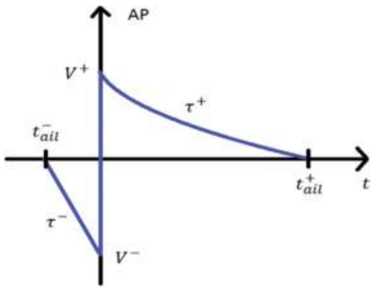

Figure 11. Proposed spike shape used for processing and learning purposed [17].

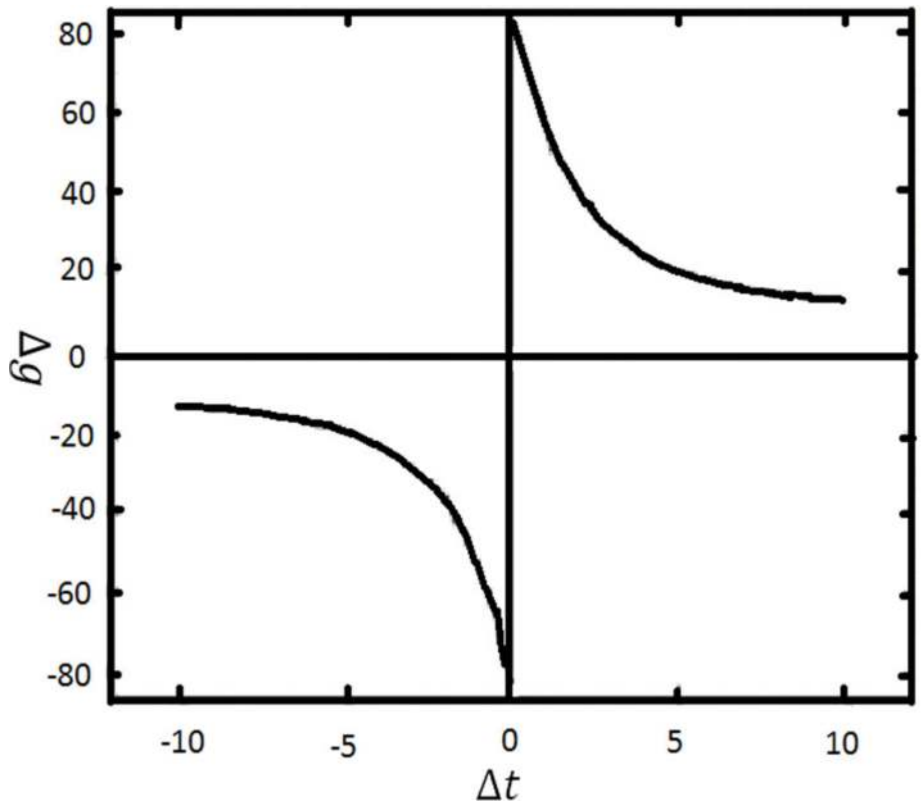

Figure 12. Simulated curve using proposed spike shape [17].

change of memristor conductance. The horizontal axis represents the difference between pre- and postsynaptic spike timings Δt. Here, the default spike parameters are e$V_{\pm} = 0.45$V volt, $t^{+}_{ail} = 11$ ms, $t^{-}_{ail} = 0.3$ ms. The result are provided for memristors with $V_{th\pm} \approx \pm 0.5$ V volt. The value of parameters A and B are 2 and 4, respectively [18, 19].

3.2. Multilayer neural networks

3.2.1. MNN concepts

Multilayer neural networks, also known as multilayer perception, are the quintessential deep networks. The advantage of MNN better than the single-layer perceptron overcomes the weaknesses that the perceptron cannot classify linearly indivisible data. To realize large scale learning tasks, MNNs can perform impressively well and produce state-of-the-art results when massive computational power is available [20, 21]. Learning in multilayer neural networks (MNNs) relies on continuous updating of large matrices of synaptic weights by local rules [22, 23]. The BP algorithm is a common algorithm in local learning, which is widely used in the training of MNNs.

3.2.2. MNN architecture

In MNN architecture, neurons of upper and lower layers are fully connected, no neuron connection exists between the same layer, and no cross layer connects to the neural network. As a quintessential deep network, multilayer neural network consists of an input layer, an output layer, and a hidden layer. MNN is the evolution of the single-layer perceptron. **Figure 13** is a double layer neural network.

The X_1, X_2 may represent the inputs single, W is the value of the weight between layers, Y is the output value. For the two-layer neural network shown above, the input signal is represented as $x_1, \ldots x_j, x_n$ (N represents the number of input neurons), b_i is represented for bias, so the result of the signal from the input layer to the hidden layer is N_{11}=f$(x_1w_{11} + x_2w_{21} + b)$, and Y_1=f $(N_{11}w_{11} + N_{12}w_{21}$+b), in which f is an activation function.

3.2.3. MNN algorithm

In this section, we give a short sketch of the back-propagation technique [25, 23]. The actual output value of the neural network is denoted by $\boldsymbol{y_j}$ and the ideal tag value is denoted by $\boldsymbol{t_j}$, and we can use the mean square error as an error function

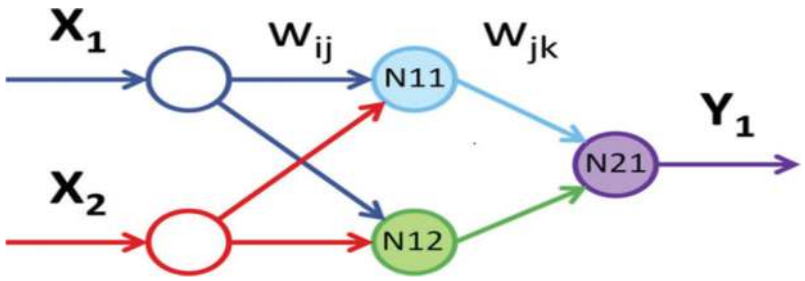

Figure 13. Logic scheme of the implemented neural network with two inputs, two hidden and one output neurons [24].

$$\boldsymbol{\varepsilon}\mathbf{MSE} = \sum_j \left(y_j - t_j\right)^2 \tag{11}$$

w_{ij} represents the weight between two layers of neurons, the neurons of the previous layer are indexed with *i*, and the next layer of neurons is indexed with *j*. The derivation of the error can be obtained by the following equation:

$$\frac{\partial \varepsilon}{\partial w_{ij}} = \frac{\partial \varepsilon}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \dot{\in}\left(y_j\right)\dot{f}(z_j)x_i = \delta_j x_i \tag{12}$$

where $z_j = \sum_i w_{ij}x_i,\ y_j = f(z_j),\ \delta_j = \dot{E}\left(y_j\right)\dot{f}z_j$

Moreover, it is assumed that the multilayer neural network uses sigmoid as a nonlinear activation function. For Eq. (3) we get

$$\frac{\partial \in}{\partial w_{ij}^{(k)}} = x_i^{(k-1)}\delta_j^{(K)} \tag{13}$$

where $\delta_j^{(L)} \equiv \delta_j = \left(y_j - t_j\right)$, x_i^0 are input signals, and $\delta_j^L \equiv \sum_i w_{ji}^{(k)}\delta_i^{(k)}\dot{f}\left(z_j^{(k-1)}\right)$.

3.2.4. MNN circuits

In this section, we enumerate an example of a memristor implementation of a two-layer neural network. As shown in **Figure 14**.

In hybrid-circuit based neural networks [26–28], memristors are integrated into crossbar circuits to implement density-critical analog weights (i.e., synapses). In this scheme, each artificial synapse is represented by memristors, so the weight of the synapse is equal to the conductance of the memristor. For the multilayer neural network mentioned above, each weight is represented by two memristors, so that the memristor crossbar can easily account for both "excitatory" and "inhibitory" states of the synapses. The number of memristor in the hidden layer is arranged in an 8×1 grid array as shown in **Figure 14**. The value of each weight $W = G^+ - G^-$, where G^+ and G^- is the effective conductance of each memristor. In the simplest case, neuron output x is encoded by voltages V and synaptic weight w by memristor conductance G. With virtually grounded neuron's input, the current was given by Ohm's law using the potential of postsynaptic V and the corresponding conductance G.

The memristor crossbar combined with CMOS circuitry, which implements neuron functionality and other peripheral functions. The artificial neuron body (soma) was implemented in the circuit by an op-amp based differential adder and a voltage divider with a MOSFET controlled by the output of the summator [24]. This element executed the basic neuron functions in terms of information processing—summation and threshold. The differential summator performing $y = \sum w_i x_i$ function is required to separate different classes of input combinations, where y is the output voltage of the summator, w_i, x_i – the *i*th input voltage and the corresponding weight respectively.

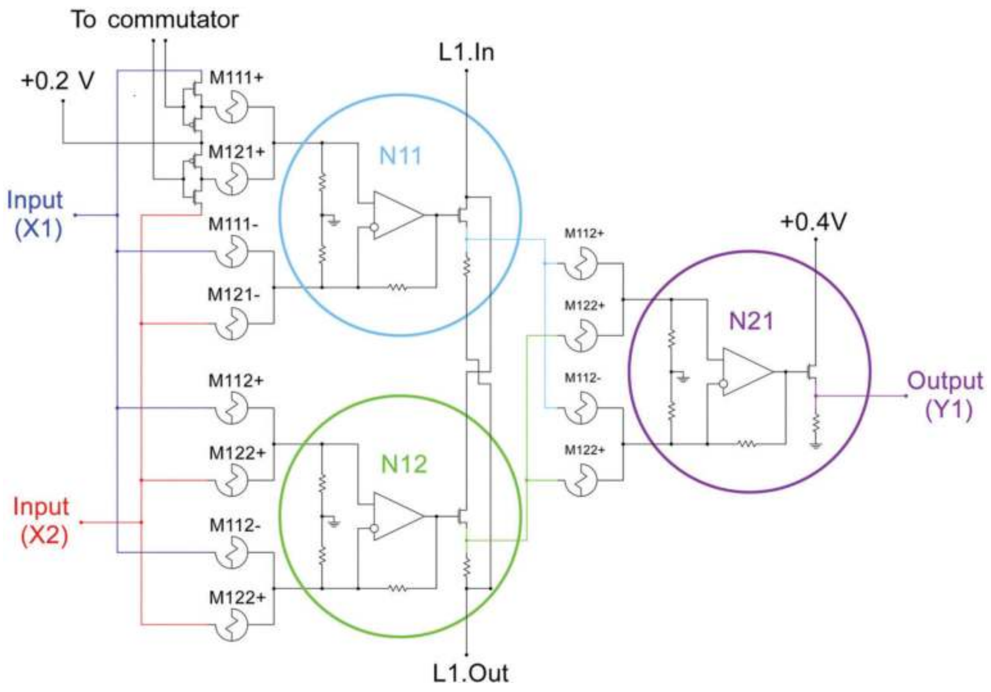

Figure 14. Circuit diagram of the ANN memristor-based hardware.

3.2.5. MNN instance

Conclusion all the experiments, we selected the image data of the MNIST data set to train and test the two-layer neural network, with the batch size 100 to speed up calculations [28]. The initial weights were selected randomly from the uniform distribution; in the experiment, the learning rate is changed depending on the training set error, and the learning rate is only constant at a level close to 0.0035.

3.3. Convolutional neural networks

3.3.1. CNN concepts

Convolutional neural network is taking inspiration from the study of biology neural science. A classical architecture of convolutional neural network was first proposed by Lecun et al. [29, 30]. As a kind of deep learning neural network, several powerful applications of CNNs were reported including pattern recognition and classification, such as human face recognition [31], traffic sign recognition [32], and object recognition [33]. Recently, in the field of image classification accuracy, convolution neural network (CNN) achieved a state-of-the art result, which can classify more than a million images into 1000 different classes [29, 34, 35].

Compared with traditional neural networks, such as fully connected NN, where each neuron is connected to all neurons of the prelayer via a large number of synapses,convolutional neural networks take advantages in weight sharing, which reduces the number of parameters need to be trained [29]. CNNs are inspired from visual cortex structure, where neurons are sensitive to small subregions of the input space, called receptive fields, exploiting the strong spatially local correlation present in images [35]. CNNs, exploiting the spatial structure of input images, has significantly fewer parameters than a fully connected network of a similar size are better suited for visual document tasks than other NN topologies such as fully connected NNs [36].

Software implementations of artificial convolutional neural networks, which require power-hungry CPU/GPU to perform convolution operations, are at the state of the art for pattern recognition applications. While achieving high performance, CNN-based methods is based on computationally expensive sums of multiplications, which is demand much more computation and memory resources than traditional classification methods. This hinders their integration in portable devices. As a result, most CNN-based algorithms and methods have to be processed on servers with plenty of resources [37].

3.3.2. CNN architectures

The overall architecture of a typical CNN consists of two main parts, the feature extractor and classifier [38, 39]. The feature extractor layers composed of two types of layers convolutional layers and pooling layers. A series of convolution and pooling are stacked, followed by fully connected layers (**Figure 15**).

In the feature extraction layers, each layer of the network receives an input from the immediate previous layer [39, 40]. Convolution neural networks are often used to handle image processing and recognition tasks. The image signal was processed by the input layer of the convolutional neural network and then enters the convolution layer for the convolution operation. Convolution operation can be expressed as [37]

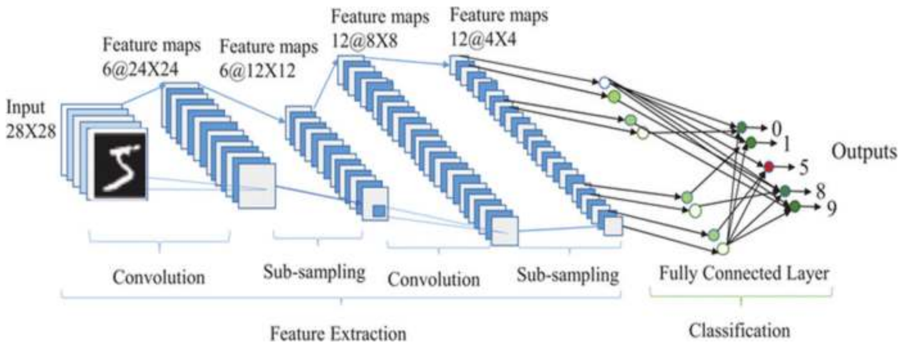

Figure 15. CNN block diagram.

$$g(x,y,z)=\sum_{i=0}^{c-1}\sum_{j=0}^{c-1}\sum_{k=1}^{l}f(x+i,y+j,k)\cdot c_z(i,j,k)=\vec{f}\cdot\vec{c_z} \tag{14}$$

where the vector $\vec{f}$ and $\vec{g}$ respectively represent the input and output feature map in the form of 3D matrix; $\vec{Cz}$ is one convolution kernel with the size of C × C; and *i* is the channel number of the convolution kernel and the input feature map.

This operation could extract different features of input images when using different convolutional kernels [29]. The input image signal will have dot product operation with kernel, and through the nonlinear transformation, the final output feature map. Then will be the subsampling process. Nonlinear neuron will be operated attached after the convolution kernel. And then, pooling computation is operated after the nonlinear neurons in order to reduce the data amount and keep the local invariance. A typical pooling unit computes the maximum of a local patch of units in one feature map (or in a few feature maps) [41]. Fully connected layers are the final layers of the CNN that all layers are fully connected by weights [37]. A feed forward neural network is usually used as a classifier in this work because it has been shown to provide the best performance compared to neural networks [42, 43].

3.3.3. CNN algorithm

In this section, the backpropagation learning algorithm for CNNs will be introduced [36]. The input of a convolution layer is the previous layer's feature maps, and the output feature map is generated by a learnable kernels and the activation function, which may combine the kernel convolutions with multiple input maps. In general, we have that

$$x_j^l=f\left(\sum_{i\in M_j}x_i^{l-1}*k_{ij}^l+b_j^l\right) \tag{15}$$

We can repeat the same computation for each map *j* in the convolutional layer, pairing it with the corresponding map in the subsampling layer:

$$\delta_j^l=\beta_j^{l+1}\left(f'\left(u_j^l\right)\circ up\left(\delta_j^{l+1}\right)\right) \tag{16}$$

where up(·) denotes an up sampling operation that simply tiles each pixel in the input horizontally and vertically n times in the output if the subsampling layer subsamples by a factor of n. One possible way to implement this function efficiently is to use the Kronecker product, $\mathrm{up}(x)\equiv X\bigotimes 1_{n\times n}$. Since the sensitivities of a given map are known, the bias gradient can be immediately computed by simply summing over all the entries in δ_j^l, $\frac{\partial E}{\partial b_j}=\sum_{u,v}(\delta_j^l)_{uv}$.

Finally, the gradients of the kernel weights are computed using backpropagation. Then, the gradient of a given weight is summed over all the connections that mention this weight

$$\frac{\partial E}{\partial k_{ij}^{l}} = \sum_{u,v} \left(\delta_j^l\right)_{uv} (P_i^{l-1})_{uv} \tag{17}$$

$$\frac{\partial E}{\partial k_{ij}^{l}} = rot180(conv2\left(x_i^{l-1}, rot180\left(\delta_j^l\right), \text{'}valid\text{'}\right)) \tag{18}$$

A subsampling layer produces down sampled versions of the input maps. If there are N input maps, then there will be exactly N output maps, although the output maps will be smaller. More formally, $x_j^l = f(\beta_j^l down\left(x_j^{l-1}\right) + b_j^l)$, where down($\cdot$) represents a subsampling function, which sum over each distinct n-by-n block in the input image so that the output image is n-times smaller along both spatial dimensions. Each output map has multiplicative bias β and an additive bias b. Since every other sample in the image $\delta_j^l = f'(u_j^l) \circ conv2(\delta_j^{l+1}, rot180\left(k_j^{l+1}\right), \text{'}full\text{'})$ can be thrown away, the gradients of b and β can be computed. The additive bias is again just the sum over the elements of the sensitivity map $\frac{\partial E}{\partial b_j} = \sum_{u,v} \left(\delta_j^l\right)_{uv}$. The multiplicative bias β will involve the original down-sampled map generated by the current layer during the forward propagation. Therefore, the maps generated during the forward propagation should be saved, to aviod recomputing them during backpropagation. Defining $d_j^l = down(x_j^{l-1})$, then the gradient of β can be represented as

$$\frac{\partial E}{\partial \beta_j} = \sum_{u,v} (\delta_j^l \circ d_j^l)_{uv} \tag{19}$$

Meanwhile, it is better to provide an output map that involves a sum over several convolutions of different input maps. Generally, the input maps combined to form a given output map are typically chosen by hand. However, such combinations can be learned during training. Let α_{ij} represents the weight given to input map i when forming output map j. Then output map j is calculated by

$$x_j^l = f\left(\sum_{i=1}^{N_{in}} \alpha_{ij}(x_i^{l-1} * K_i^l) + b_j^l\right) \tag{20}$$

where

$$\sum_i \alpha_{ij} = 1, \text{and } 0 \le \alpha_{ij} \le 1 \tag{21}$$

By setting the α_{ij} variables equal to the softmax over a set of unconstrained weights c_{ij}, these constraints can be enforced

$$\alpha_{ij} = \frac{\exp(c_{ij})}{\sum_k \exp(c_{kj})} \tag{22}$$

Since each set of weights c_{ij} for fixed j are independent of all other such sets for any other j, only the updates of a single map need considering and the subscript j can be dropped. Each map is updated in the same way, except with different j indices. The derivative of α_k with respect to the α_i variables at layer is the derivative of the softmax function is given by

$$\frac{\partial \alpha_k}{\partial c_i} = \delta_{ki}\alpha_i - \alpha_i\alpha_k \tag{23}$$

where δ is used as the Kronecker delta.

Use $\boldsymbol{\delta}^l$ represents the sensitivity map corresponding to an output map with inputs $\mathbf{u}$. Again, the convolution is the "valid" type so that the result will match the size of the sensitivity map. Now, the gradients of the error function with respect to the underlying weights c_i can be computed by the chain rule

$$\frac{\partial E}{\partial \alpha_i} = \frac{\partial E}{\partial u^l}\frac{\partial u^l}{\partial \alpha_i} = \sum_{u,v} (\delta^l \circ (x_i^{l-1} * K_i^l))_{uv} \tag{24}$$

In addition, the sparseness constraints on the distribution of weights α_i for a given map can also been imposed by adding a regularization penalty $\Omega(\boldsymbol{\alpha})$ to the final error function. Therefore, some weights will be zero. That means, only a few input maps would contribute significantly to a given output map, as opposed to all of them. The error for a single pattern can be written as

$$\frac{\partial E}{\partial c_i} = \sum_k \frac{\partial E}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial c_i} = \alpha_i(\frac{\partial E}{\partial \alpha_i} - \sum_k \frac{\partial E}{\partial \alpha_k}\alpha_k) \tag{25}$$

$$\widetilde{E}^n = E^n + \lambda \sum_{i,j} |\alpha_{ij}| \tag{26}$$

This will find the contribution of the regularization term to the gradient for the weights c_i. The user defined parameter λ controls the trade-off between minimizing the fit of the network to the training data, and ensures that the weights mentioned in the regularization term are small according to the 1-norm. Again, only the weights α_i for a given output map need considering and the subscript j can be dropped. First, there is

$$\frac{\partial \Omega}{\partial \alpha_i} = \lambda sign(\alpha_i) \tag{27}$$

Combining this result with Eq. (24), the derivation of the contribution is

$$\frac{\partial \Omega}{\partial c_i} = \sum_k \frac{\partial \Omega}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial c_i} = \lambda(|\alpha_i| - \alpha_i \sum_k |\alpha_k|) \tag{28}$$

The final gradients for the weights c_i when using the penalized error function Eq. (11) can be computed using Eqs. (13) and (9)

$$\partial \frac{\widetilde{E^n}}{\partial c_i} = \frac{\partial E^n}{\partial c_i} + \frac{\partial \Omega}{\partial c_i} \tag{29}$$

3.3.4. CNN circuits

This part introduces the construction and operation of the memristor neural networks circuit. First of all, we introduce how a single column within a memristor crossbar can be used to perform a convolution operation. Pooling operation can be seen as a simpler conversation operation [39]. The circuit diagram of each column for the convolution operation of the memristor crossbar structure is shown in **Figure 16**.

Each crosspoint of the circuit was composed of memristors, which is represented for synapses. The kernel (k) was represented by the conductivity value (G) in the crossbar circuit. Some extra manipulation include converting kernel matrix into two parallel column to express the positive and negative value of the kernel and converting kernel matrix to conductivity values ($\delta^{\pm}$) [39] that fall within the bounded range of a memristor crossbar. The op-amp circuit is used to scale the output voltage and implements the sigmoid activation function.

The convolution computation operation in memristor crossbar is the same as the matrix convolution operation. That mainly is a result of the dot-production about the matrixes of kernels and inputs. The first step is the multiplication of voltage (V) and conductance ($G = x^{-1}$) [29], which is following ohm's law ($I = V \cdot G$). Second, it will follow Kirchhoff's current law (KCL), which describes that the circuit flowing out the node will be equal to the sum of current flowing into that node. Based on KCL, novel computation architecture for implementing pot-product is implemented [29]. And then, the lower end of the op amp circuit performs activation function. As a result, each neuron of hidden and output layers implements $f\left(\sum_i (G^+ - G^-)V_i\right)$, where f is a kind of activation function. **Figure 17** shows the flow chart of the CNN image identification system.

where L is the number of layers of the CNN recognition system, the input layer ($L = 1$) holds a testing set of 500 MNIST images, whose size of data set is 28×28. $L = 2$ is the first convolution layer [39].

step 1. First convolution layer($l = 2$)

The signal size from the front input layer is 28×28. In this layer, an input image will be convolved with six different 5×5 size kernels on the memristor crossbar. According with the front description, each column is the kernel value of 5×5. And the 2D kernel was broken into two arrays in the memristor crossbar to easily account for negative values in the kernel arrays. The total number of a column in the crossbar structure is $2 \times 25 + 1$, in which "1" is the value of bias. Since we are using a memristor crossbar to perform the convolution operations, we can generate all six of these output maps in parallel. So, the crossbar circuit exist six parallel columns in a row. Therefore, this layer requires a 51×6 memristor crossbars.

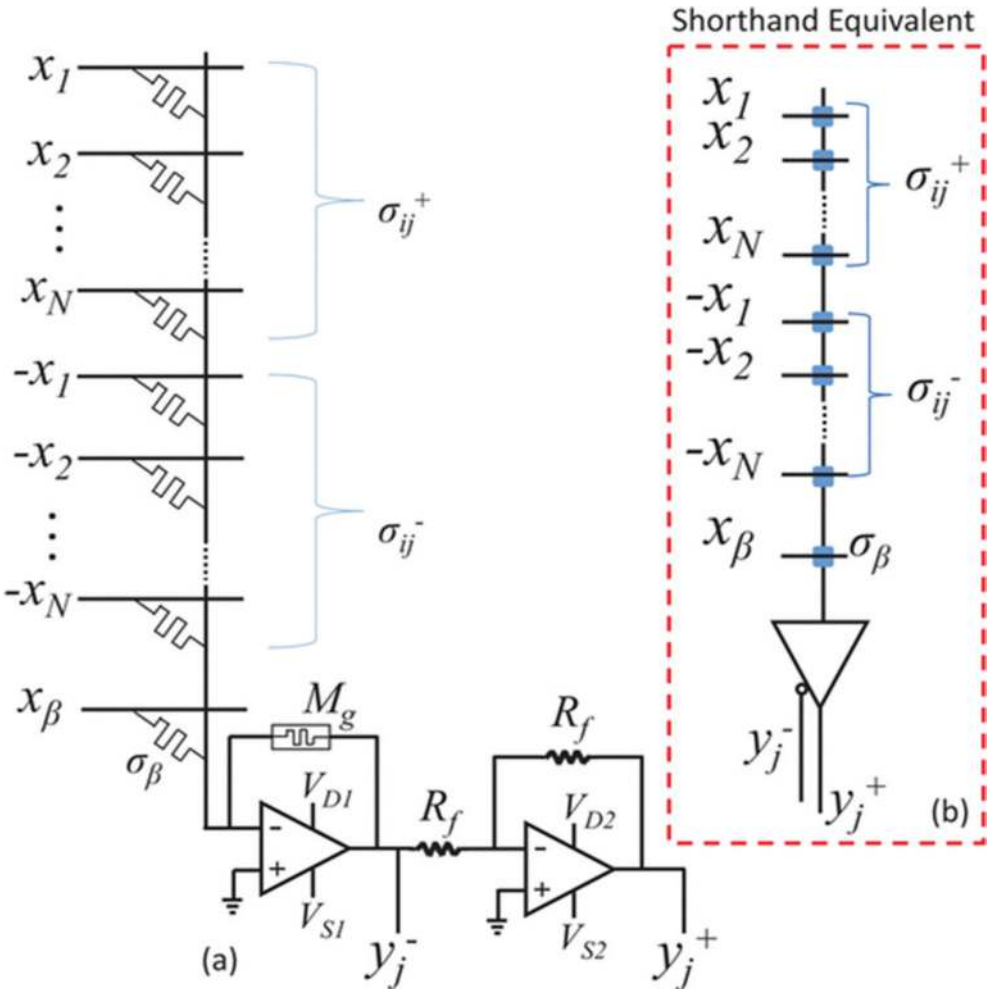

Figure 16. (a) A column circuit of MCNN circuit diagram that is capable of performing convolutional operation and (b) a shorthand depiction for this circuit.

So far, we have got the memristor crossbar structure, which simulates the synapses and stores the value of kernels in it. The circuit perform the first convolution operation is shown in **Figure 18**.

Each image contains 784 pixel, but the image is applied 25 pixels at a time where each 25 pixel section generates a single output value. After these convolution kernels applied, a data array that has a size of 24 × 24 × 6 will be generated in the memristor crossbar and then will be operated in the next layer. For each column in the memristor crossbar structure,memristor is used to simulate synapses of neural networks. And, the circuit simulates neurons to produce

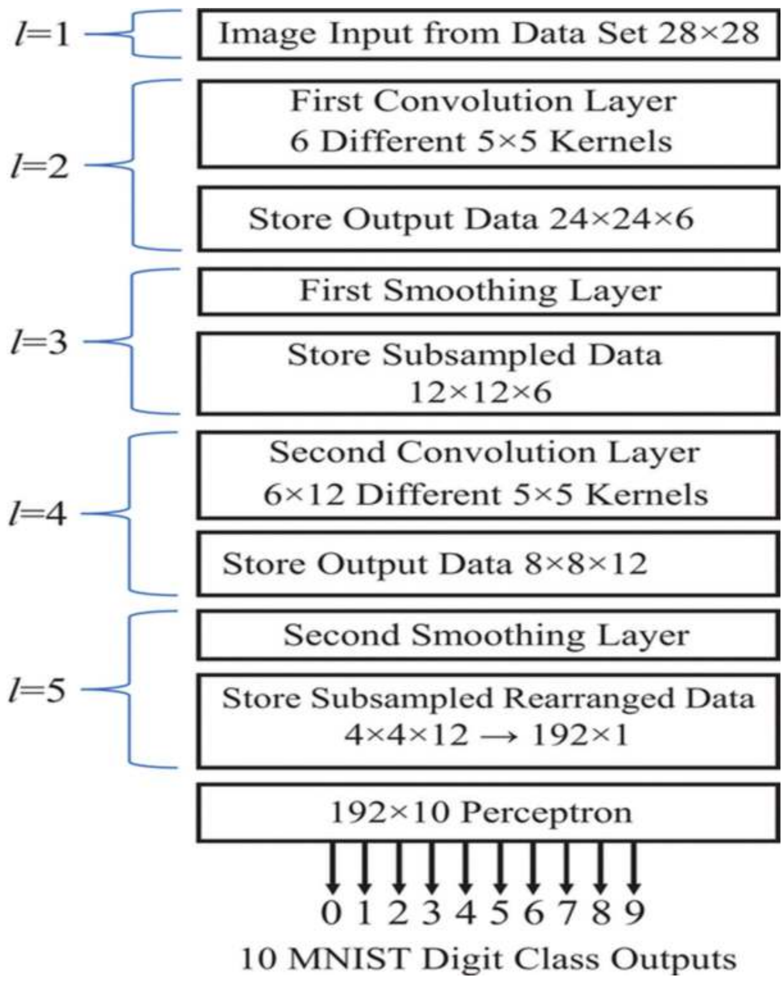

Figure 17. Flowchart describing the CNN recognition system [39].

the summation of all the product of inputs and kernels and operate activation function. According to Ohm's law and Kirchhoff's law, every single output value in this time is the input value and the kernel value of the inner product results. After the signal is input, the memristor and op-amp circuits are output later. When all 6 24×24 sizes of feature map are obtained, the first convolution operation was finished,the output is the input value of the next neuron that will be applied in pooling operation process.

Step 2. First smoothing layer ($l = 3$)

Following the first convolution layer, a smoothing operation is performed on the six generated feature maps. Pooling operation can be seen as a simpler conversation operation, with all kernel applied to each feature map is

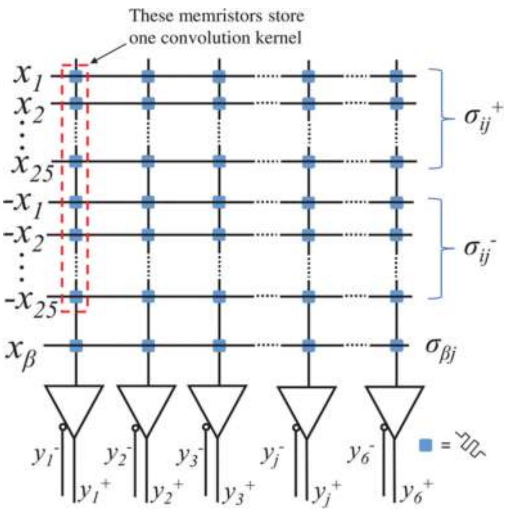

Figure 18. Circuit used to perform the convolution operations for the second ($l = 2$) in the CNN recognition system.

$$K = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}$$

Be similar with the convolution process, each column of the crossbar represents a kernel. So the memristor numbers of a single column of the crossbar is $4 \times 2 + 1$, and six column for with all six feature maps be operated in parallel, Therefore this layer requires a $6 \times (2 \times 4 + 1)$ memristor crossbars. But different with the convolution layers, the conductivities corresponding to negative elements in the kernel matrix in this layer are meaningless because all components of the pooling kernel are positive. The following circuit is shown that has pooling operation on the $6 \times 24 \times 24$ size of feature map which the convolution layer is derived (**Figure 19**).

In the pooling operation, six different feature maps obtained from the convolution layers applied to every corresponding column respectively and obtain another sets of feature maps.

A subsampling operation is performed following each of the smoothing crossbars that reduce the size of each feature map by a square factor of 2. This could be design in to place a single-bit counter on the memory array where the data output from the smoothing operation is stored. The memory address would only update for every other sample so all unwanted data would be overwritten during the smoothing step.

Step 3. Second convolutional layer (l=4)

Following the polling and subsample, operation is the second convolution operation. Different with the first one, inputs of the second convolution layers are six feature maps with 12×12 size, and it exists 12 outputs instead of six in the front one. Because the different number and size of input and output single, the structure of the second layer is distinctly different from the

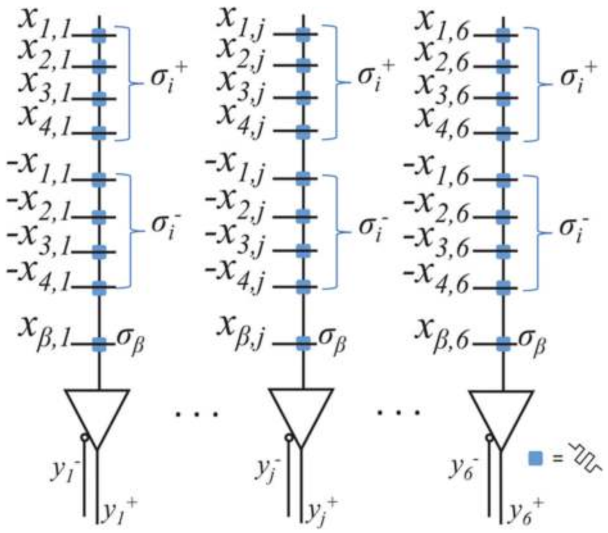

Figure 19. The group of convolution circuits that is used to implement the smoothing operation.

one. The circuit design of the second convolution layer is shown in **Figure 20**. Each column represents six different feature map convolution processes, and will be operated with 12 different kernels in parallel methods.

Step 4. Second smoothing layer($l = 5$)

Following the second convolution layer, another smoothing layer is following the second convolution layer to further reduce the size of the data array. The circuit in this layer will be identical to that displayed in **Figure 7**. With 12 feature maps will be operated, so required 12 parallel single column crossbars. After second layers of pool will produce 4×4 of the size of 12 feature map, the input to the next layer, classification layer ($l = 6$).

Step 5. Classification layer

Following the front feature extraction operations, a fully connected layer is used to classify the feature maps. The classification layer is generally a single layer perceptron or multilayer neural network.

The circuit used to complete this operation can be seen in **Figure 21**. The memristor crossbar used in classification layer is to store a weight matrix, which is different with storing a set of convolution kernels arrays in convolution circuits. The crossbar consists of 192*2+1 rows which represent 192 inputs (one input for each of the 16 value in each of the 12 outputs maps), and 10 columns which represent 10 outputs (one for each MNIST digit). So the total numbers of memristors in this layers is $(192 + 1) \times 10$.

Step 6. Digital storage layers

Following every convolution layer, a digital layer was placed at the output of each convolution. The digital storage layer reduces the amount of analog circuit error that is transmitted

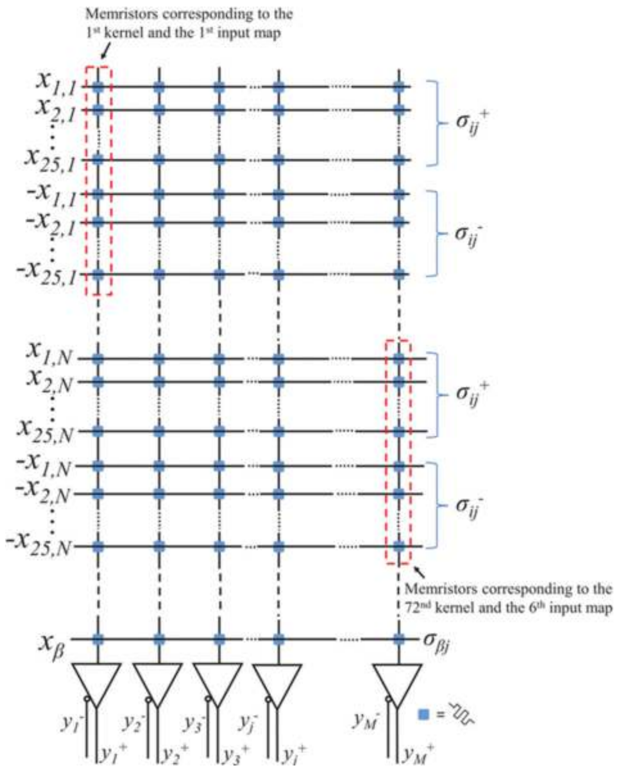
Memristors corresponding to the 1st kernel and the 1st input map
$x_{1,1}$
$x_{2,1}$
$x_{25,1}$
$-x_{1,1}$
$-x_{2,1}$
$-x_{25,1}$
σ_{ij}^{+}
σ_{ij}^{-}
$x_{1,N}$
$x_{2,N}$
$x_{25,N}$
$-x_{1,N}$
$-x_{2,N}$
$-x_{25,N}$
Memristors corresponding to the 72nd kernel and the 6th input map
x_{β}
$\sigma_{\beta j}$
y_1^- y_1^+ y_2^- y_2^+ y_3^- y_3^+ y_j^- y_i^+ y_M^- y_M^+
=

Figure 20. The circuit that is used to implement the second convolution layer.

between layers. We chose to store an entire image between layers because any benefit gained by a systematically reduced memory size would likely be outweighed by the complexity of a data controller of this nature [44].

3.3.4.1. CNN instance

The CNN algorithm purely in simulation under these training conditions results in 92% classification accuracy as shown in **Figure 22**. And, the simulation process is to test the accuracy of the memristor based CNN recognition system described in the previous section. When testing the simulated memristor crossbars, an accuracy of 91.8% was achieved.

3.4. Recurrent neural networks

3.4.1. RNN concept

Recurrent neural networks, or RNNs, are the main tool for handling sequential data, which involve variable length inputs or outputs [40]. Compared with multilayer network, the weights in an RNN are shared across different instances of the artificial neurons, each associated with

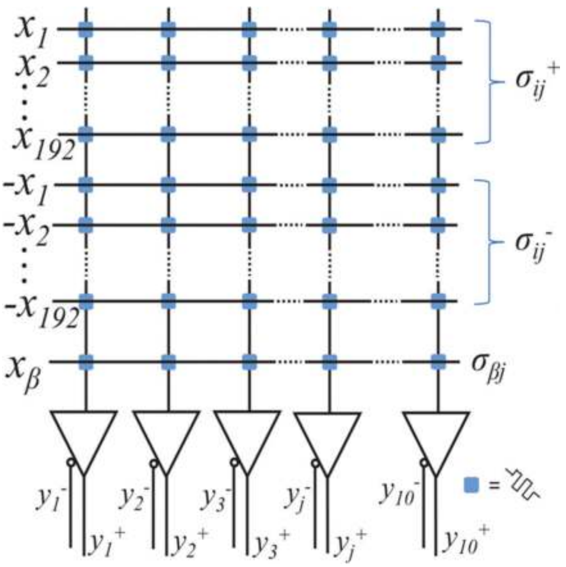

Figure 21. Circuit that is used to implement the classification layer of the CNN recognition system.

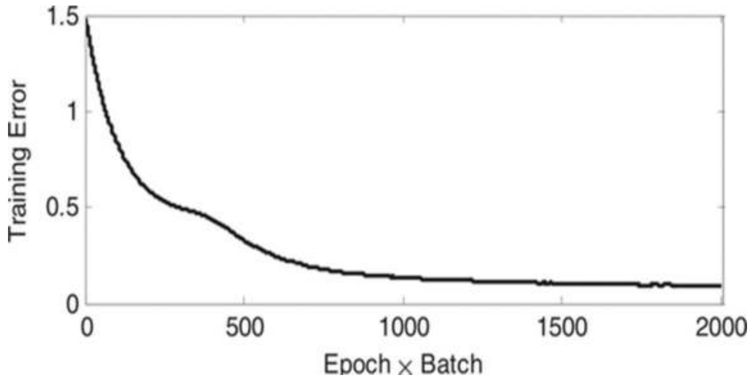

Figure 22. Error present when training the CNN algorithm is software [39].

different time steps [40, 42]. And, others, in recurrent neural networks, lengths history represented by neurons with recurrent connections, and history length is unlimited. Also recurrent networks can learn to compress whole history in low dimensional space, while feedforward networks compress (project) just single word recurrent networks have possibility to form short term memory, so they can better deal with position invariance [45] RNN architecture.

The simplest architecture of RNNs is illustrated in **Figure 23** [40]. The left of **Figure 24** shows the ordinary recurrent network circuit with weight matrices U, V, W denoting three different kind of connection (input-to-hidden, hidden-to-output, and hidden-to-hidden,

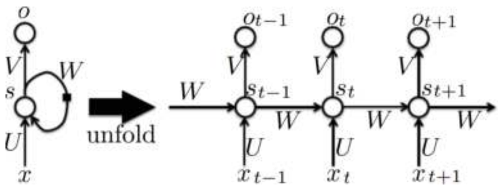

Figure 23. The architecture of recurrent neural networks.

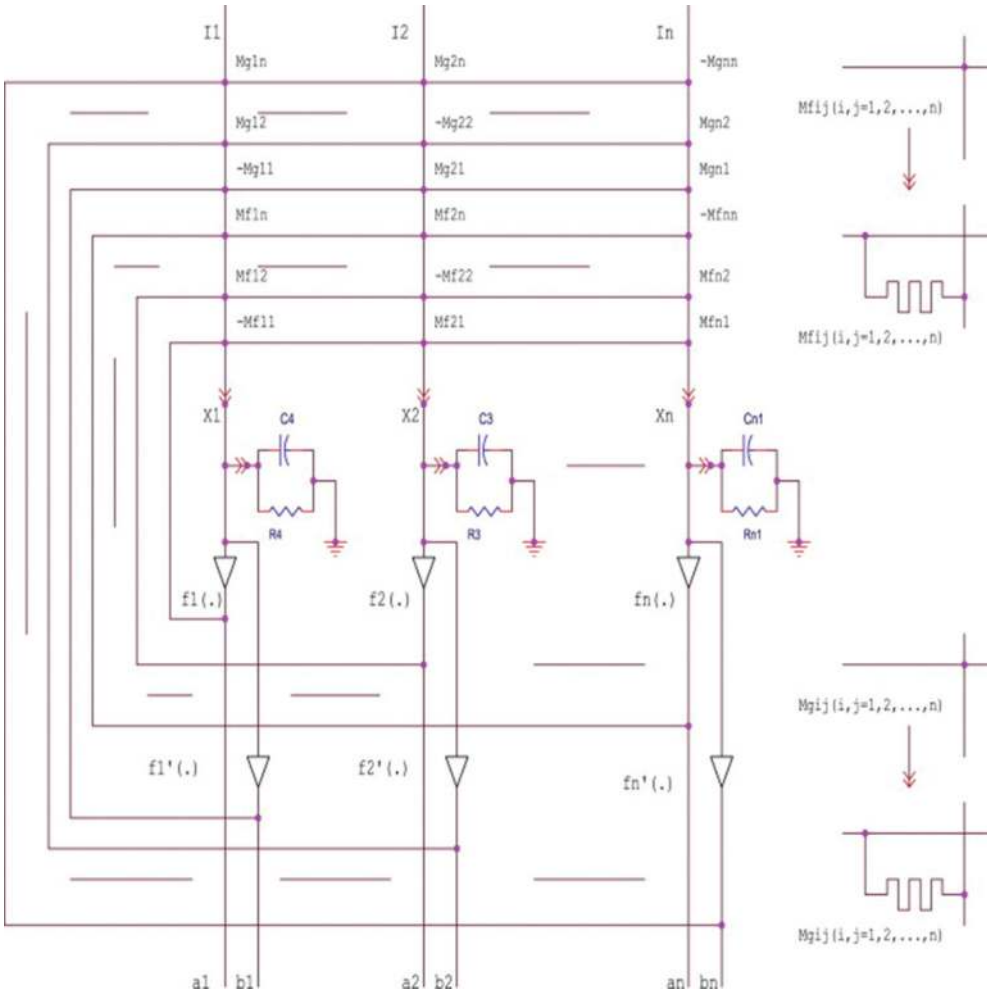

Figure 24. General class of recurrent neural network circuit.

respectively). Each circle indicates a whole vector of activations. The right of **Figure 24** is a time-unfolded flow graph, where each node is now associated with one particular time instance.

3.4.2. A Hopfield neural network design

Memristor-based Hopfield networks (MHN), which is an ideal model for the case where the memristor-based circuit network exhibits complex switching phenomena, and is frequently encountered in the applications [46]. A Hopfield network consists of a set of interconnected artificial neurons and synapses. In this case, a nine synapses Hopfield network is realized with six memristors and three neurons. As shown in **Figure 25**, the artificial neuron has three inputs and each input, Ni = ($i = 1$, 2, and 3), is connected to a synapse with synaptic weight of w_i. The output of the three-input binary artificial neuron is expressed as

$$y = \text{sign}\left(\sum_{i=1}^{3} w_i N_i - \theta\right) \tag{30}$$

where y is the neuron's threshold; and the sign function is defined as

$$\text{sign(N)} = \begin{cases} 1 & ifN \geq 0 \\ 0 & ifN < 0 \end{cases}$$

An artificial neuron was constructed, as shown in **Figure 26**. An operational amplifier is used to sum the inputs. The switches, S_1, S_2, and S_3, are controlled by external signals to obtain positive or negative synaptic weights. The synaptic weights corresponding to input N_1, N_2, and N_3 are $w_1 = \pm\frac{M_1}{M_1+R}$, $W_2 = \pm\frac{M_2}{M_2+R}$ and $W_3 = \pm\frac{M_3}{M_3+R}$, respectively ($M_1$, M_2, and M_3 are the resistance of the memristors, respectively, and the resistance of R is fixed at 3 MΩ). In the circuit shown in **Figure 26**, transmission gates B_1, B_2, and B_3 reform signals without modifying its polarity, inverters I_1, I_2 and I_3 generate negative synaptic weights.

The architecture of a 3-bit MHN implemented with nine synapses is shown in **Figure 27**. The synaptic weight from neuron i to neuron j is denoted as $w_{i,j}$, which is mapped to resistance of the corresponding memristor $M_{i,j}$,. $M_{i,j}$, and $w_{i,j}$ are represented by the resistance matrix, respectively

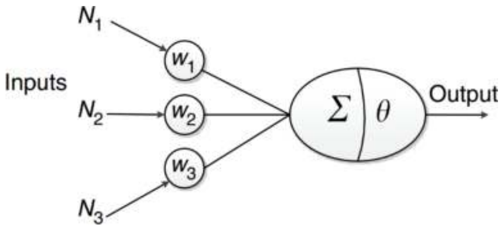

Figure 25. Mathematical abstraction of the neuron model.

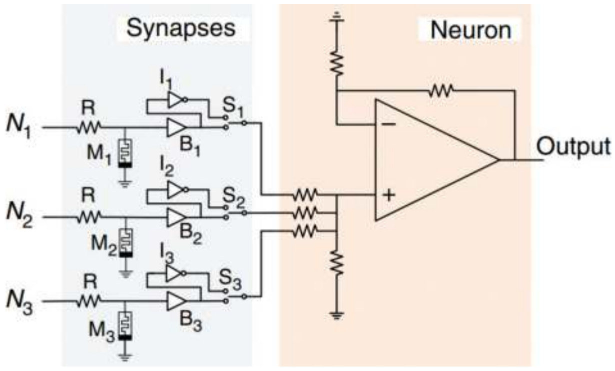

Figure 26. Circuit schematic of the designed 3-bit neuron.

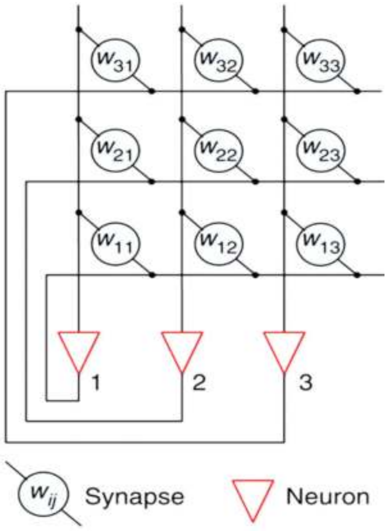

Figure 27. Architecture of the 3-bit MHN consisting of nine memristors.

$$\mathrm{M} = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}, \ \mathrm{W} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix},$$

Due to the symmetry of Hopfield network, $\mathrm{M}_{12} = \mathrm{M}_{21}$, $\mathrm{M}_{23} = \mathrm{M}_{32}$, and $\mathrm{M}_{13} = \mathrm{M}_{31}$, the implementation of the network only needs six memristors. The schematic of this circuit is shown in **Figure 28**, and all the demonstration below is based on this circuit. The threshold vector $\mathbf{T} = (\theta_1, \theta_2, \theta_3)$ represents the threshold of the artificial neurons (neurons 1, 2, and 3), and the state vector $\mathbf{X} = (\mathrm{X}_1, \mathrm{X}_2, \mathrm{X}_3)$ represents the states of the three neurons, respectively. In each updating cycle, new states of the neurons are updated by the following function

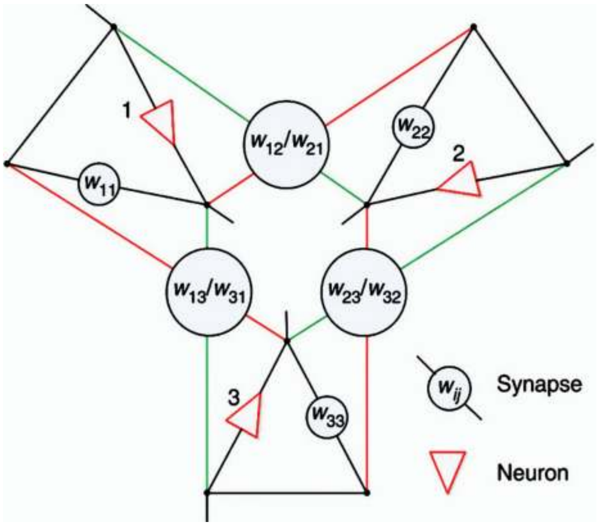

Figure 28. Architecture of the MHN with symmetrical configuration consisting of six memristors.

$$\mathbf{X}(\mathrm{t}+1) = \mathrm{sign}(X_{(t)} \cdot \mathbf{W} - \mathbf{T}) \tag{31}$$

where t represents the number of updating cycles and when $t = 0$, $\mathbf{X}(0)$ represents initial states vector. In one updating cycle, new states of the neurons are asynchronously updated from X_1, X_2 to X_3 in three stages, which are defined as stages a, b, and c, respectively [46].

4. Potential applications and prospects

Hardware implementation of deep neural networks is accomplished by using neuron-synapse circuits and future devices can make deep neural networks (NNs) design and fabrication more efficient. The full power of NNs has not yet been realized, but the release of commercial chips implementing arbitrary neural networks, more efficient algorithms will no doubt be realized in these domains where neural networks can improve the performance dramatically. Memristor-based NNs promote and solve many A.I. problems such as machine translation, intelligent question-and-answer, and game play, and in the future, memristor-based NNs can be used in neuromorphic computation, brain-computer interface or computer-brain interface, cell phone A.I. application, autopilot and environment monitor.

5. Conclusions

Different memristor-based neural network design paradigms are described. With regard to neural network systems, the current neural network implementations are not sufficient but fortunately, memristor-based systems provide the potential solution. The basic concepts of memristor-based implementation, such as memristor-based synapse, memristor-based neuron, and memristor crossbar based neuromorphic computing engine, are discussed. The memristor-based neural networks, including SNNs, MNNs, CNNs, and RNNs, are possible and efficient and are expected to spur future development of A.I. It is expected that memristor-based neural networks will take the lead.

Acknowledgements

The work was jointly financially supported by the National Natural Science Foundation of China under grant nos. 11574017 and 51372008, and the Special Foundation of Beijing Municipal Science & Technology Commission (Grant No. Z161100000216149).

Author details

Anping Huang*, Xinjiang Zhang, Runmiao Li and Yu Chi

*Address all correspondence to: aphuang@buaa.edu.cn

School of Physics and Nuclear Energy Engineering, Beihang University, Beijing, China

References

[1] Rothenbuhler A. A memristor-based neuromorphic computing application [Dissertation]. Boise State University; 2013

[2] Merolla PA, Arthur JV, Alvarezicaza R, et al. A million spiking-neuron integrated circuitwith a scalable communication network and interface. Science. 2014;**345**(6197): 668-673

[3] Shahsavari M, Falez P, Boulet P. Combining a volatile and nonvolatile memristor in artificial synapse to improve learning in Spiking Neural Networks. In: IEEE/ACM International Symposium on Nanoscale Architecture; 2016. pp. 67-72

[4] Chua LO. Memristor–The missing circuit element. IEEE Transactions on Circuit Theory. 1971;**18**(5):507-519

[5] Strukov DB, Snider GS, Stewart DR, et al. The missing memristor found. Nature. 2008;**453** (7191):80-83

[6] Jo SH, Chang T, Ebong I, et al. Nanoscale memristor device as synapse in neuromorphicsystems. Nano Letters. 2010;**10**(4):1297-1301

[7] Ali S, Bae J, Lee CH, et al. Ultra-low power non-volatile resistive crossbar memory based on pull up resistors. Organic Electronics. 2017;**41**:73-78

[8] Li Y, Zhong Y, Xu L, et al. Ultrafast synaptic events in a chalcogenide memristor. Scientific Reports. 2013;**3**:1619

[9] Wu C, Kim TW, Guo T, et al. Mimicking classical conditioning based on a single flexible-memristor. Advanced Materials. 2017;**29**(10):1602890

[10] Burr GW, Shelby RM, Sebastian A, et al. Neuromorphic computing using non-volatile memory. Advances in Physics: X. 2017;**2**(1):89-124

[11] Indiveri G, Linares-Barranco B, Legenstein R, et al. Integration of nanoscale memristor synapses inneuromorphic computing architectures. Nanotechnology. 2013;**24**(38):384010

[12] Liu B. Neuromorphic System Design and Application. University of Pittsburgh; ProQuest Dissertations Publishing, 2016

[13] Liu B. Memristor-based analog neuromorphic computing engine design and robust training scheme [Dissertation]. University of Pittsburgh; 2014

[14] Serrano-Gotarredona T, Masquelier T, Prodromakis T, et al. STDP and STDP variations with memristors for spiking neuromorphic learning systems. Frontiers in Neuroscience. 2013;**7**:2

[15] Paugam-Moisy H, Bohte S. Computing with spiking neuron networks. In: Handbook of Natural Computing. Berlin Heidelberg: Springer; 2012. pp. 335-376

[16] Prezioso M, Bayat FM, Hoskins B, et al. Self-adaptive spike-time-dependent plasticity of metal-oxide memristors. Scientific Reports. 2016;**6**:21331

[17] Xu Y, Zeng X, Han L, et al. A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. Neural Networks. 2013;**43**:99-113

[18] Linares-Barranco B, Serrano-Gotarredona T. Memristance can explain spike-time-dependent-plasticity inneural synapses. Nature Precedings. 2009;**1**:2009

[19] Afifi A, Ayatollahi A, Raissi F. Implementation of biologically plausible spiking neural network models on the memristor cross bar-based CMOS/nano circuits. In: 2009 European Conference on Circuit Theory and Design (ECCTD); 2009. pp. 563-566

[20] Soudry D, Di CD, Gal A, et al. Memristor-based multilayer neural networks with online gradient descent training. IEEE Transactions on Neural Networks & Learning Systems. 2015;**26**(10):2408

[21] Rosenthal E, Greshnikov S, Soudry D, et al. A fully analog memristor-based neural network with online gradient training. In: IEEE International Symposium on Circuits and Systems. IEEE; 2016. pp. 1394-1397

[22] Chung J, Park J, Ghosh S. Domain wall memory based convolutional neural networks forbit-width extend ability and energy-efficiency. In: International Symposium on LowPower Electronics and Design. ACM; 2016. pp. 332-337

[23] Kataeva I, Merrikh-Bayat F, Zamanidoost E, et al. Efficient training algorithms for neural networks based on memristive crossbar circuits. In: IEEE International Joint Conferenceon Neural Networks; 2015. pp. 1-8

[24] Emelyanov AV, Lapkin DA, Demin VA, et al. First steps towards the realization of a double layer perceptron based on organic memristive devices. AIP Advances. 2016;**6**(11): 111301

[25] Yang C, Kim H, Adhikari SP, et al. A circuit-based neural network with hybrid learning of back propagation and random weight change algorithms. Sensors. 2016;**17**(1):16

[26] ZhangY,LiY, Wang X, et al. Synaptic characteristics of Ag/AgInSbTe/Ta-basedmemristor for pattern recognition applications. IEEE Transactions on Electron Devices. 2017;**64**(4): 1806-1811

[27] Yakopcic C, Hasan R, Taha TM. Flexible memristor based neuromorphic system for implementing multi-layer neural network algorithms. International Journal of Parallel, Emergent andDistributed Systems. 2017 DOI: http://dx.doi.org/10.1080/17445760.2017.1321761

[28] Negrov DV, Karandashev IM, Shakirov VV, et al. A plausible memristor implementation of deep learning neural networks. Computer Science. 2015

[29] Zeng X, Wen S, Zeng Z, et al. Design of memristor-based image convolution calculationin convolutional neural network. Neural Computing & Applications. 2016:1-6

[30] Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;**86**(11):2278-2324

[31] Lawrence S, Giles CL, Tsoi AC, et al. Face recognition: A convolutional neural-network approach. IEEE Transactions on Neural Networks. 1997;**8**(1):98-113

[32] Lau MM, Lim KH, Gopalai AA. Malaysia traffic sign recognition with convolutional neural network. In: 2015 IEEE International Conference on Digital Signal Processing (DSP). IEEE; 2015. pp. 1006-1010

[33] Wang J, Lu J, Chen W, et al. Convolutional neural network for 3D object recognition based on RGB-D dataset. In: 2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA). 2015. pp. 34-39

[34] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: International Conference on Neural Information Processing Systems. 2012. pp. 1097-1105

[35] Wang Y, Wen W, Song L, et al. Classification accuracy improvement for neuromorphic computing systems with one-level precision synapses. arXiv. 2017;arXiv:1701.01791

[36] Bouvrie J. Notes on Convolutional Neural Networks. Neural Networks. 2006. Unpublished. http://web.mit.edu/jvb/www/papers/cnn tutorial.pdf

[37] Wang Y, Xia L, Tang T, et al. Low power convolutional neural networks on a chip. In: IEEE International Symposium on Circuits and Systems; 2016. pp. 129-132

[38] Garbin D, Bichler O, Vianello E, et al. Variability-tolerant convolutional neural network for pattern recognition applications based on OxRAM synapses. In: 2014 IEEE International Conference on Electron Devices Meeting (IEDM); 2014. pp. 28.4.1-28.4.4

[39] Yakopcic C, Alom MZ, Taha TM. Memristor crossbar deep network implementation based on a convolutional neural network. In. International Joint Conference on Neural Networks; 2016. pp. 963-970

[40] Simard PY, Steinkraus D, Platt JC. Best practices for convolutional neural networks applied to visual document analysis. In: International Conference on Document Analysis & Recognition (ICDAR), vol. 3; 2003. pp. 958-963

[41] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;**521**(7553):436-444

[42] Mohamed AR, Dahl GE, Hinton G. Acoustic modeling using deep belief networks. IEEE Transactions on Audio, Speech, and Language Processing. 2012;**20**(1):14-22

[43] Nair V, Hinton G. Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conferenceon Machine Learning (ICML). 2010. pp. 807-814

[44] Mathiyalagan K, Anbuvithya R, Sakthivel R, et al. Non-fragile H8 synchronization of memristor-based neural networks using passivity theory. Neural Networks. 2016;**74**: 85-100

[45] MikolovT, Karafiát M, Burget L, et al. Recurrent neural network based language model. In: Interspeech Interspeech, Conference of the International Speech Communication Association, Makuhari, Chiba, Japan; 2010. pp. 1045-1048

[46] Hu S G, Liu Y, Liu Z, et al. Associative memory realized by a reconfigurable memristive Hopfield neural network. Nature Communications. 2015;**6**:7522