Ph.D. DISSERTATION

Study on Binary Resistance Switch Array for Neuromorphic Hardware

by

Guhyun Kim

February 2020

DEPARTMENT OF MATERIALS SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING

SEOUL NATIONAL UNIVERSITY

Study on Binary Resistance Switch Array for Neuromorphic Hardware

Advisor : Prof. Cheol Seong Hwang

by

Guhyun Kim

A thesis submitted to the Graduate Faculty
of Seoul National University in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy
Department of Materials Science and Engineering

December 2019

Approved
by

Chairman of Advisory Committee : Sang Bum Kim
Vice-chairman of Advisory Committee : Cheol Seong Hwang
Advisory Committee : Dongsuk Jeon
Advisory Committee : Doo Seok Jeong
Advisory Committee : Byung Joon Choi

Abstract

Resistance switch array is a strong contender for next-generation memory. A resistance switch has low resistance state or high resistance state. Switching between states are stimulated by electric signal such as application of voltage or current. With crossbar array configuration, resistance switch array reaches to high integration density of $4F^2$ where F means minimum feature size. Analog resistance switches are also have been proposed, but most of them need very precise control of conductance. Additionally, at least one of their potentiation or depression is non-linear to pulse number (or pulse length).

Resistance switch array is also able to realize matrix-vector multiplication, or parallel operation. In other words, the current response to an applied input voltage vector naturally captures the conductance matrix-voltage vector multiplication.

Simulating resistance switch array is an efficient method to analyze its property. The most popular simulation uses Newton-Raphson methods for resistance array simulation, but this method consumes large calculation costs. As an alternative, an artificial neural network was applied for the resistance switch simulation. An artificial neural network was utilized in the behavior inference of a random crossbar array (10 × 9 or 28 × 27 in size) of nonvolatile binary resistance-switches (in a high resistance state (HRS) or low resistance state (LRS)) in response to a randomly applied voltage array. The employed artificial neural network was a multilayer perceptron (MLP) with leaky rectified linear units. This MLP was trained with 500,000 or 1,000,000 examples. For

each example, an input vector consisted of the distribution of resistance states (HRS or LRS) over a crossbar array plus an applied voltage array. That is, for a $M \times N$ array where voltages are applied to its M rows, the input vector was $M \times (N+1)$ long. The calculated (correct) current array for each random crossbar array was used as data labels for supervised learning. This attempt was successful such that the correlation coefficient between inferred and correct currents reached 0.9995 for the larger crossbar array. This result highlights MLP that leverages its versatility to capture the quantitative linkage between input and output across the highly nonlinear crossbar array. Additionally, MLP accelerates simulation 8 times faster compared to Newton-Raphson method.

With its availability of parallel operation, resistance switch array is used in various parts of neuromorphic hardware, which aims to synthesize hardware mimicking neural networks. The typical application of resistance switch array is an artificial synapse array. Because matrix-vector multiplication in resistance switch array is similar to that in neural network, neuromorphic hardware can be accelerated by implementation of a resistance switch array as an artificial synapse array.

In this paper, a learning algorithm suitable for binary resistance switch array is proposed. This algorithm is referred to as the Markov chain Hebbian learning algorithm. The algorithm pursues efficient use in memory during training in that: 1) the weight matrix has ternary elements (−1, 0, 1) and 2) each update follows a Markov chain—the upcoming update does not need past weight values. Additionally, the ternary synaptic units are easily realized by a pair of resistance switches, so that the Markov chain Hebbian learning algorithm is

appropriate for training binary resistance switch array used as synapse array. The algorithm was verified by two proof-of-concept tasks: image (MNIST and CIFAR-10 datasets) recognition and multiplication table memorization. Particularly, the latter bases multiplication arithmetic on memory, which may be analogous to humans' mental arithmetic. The memory-based multiplication arithmetic feasibly offers the basis of factorization, supporting novel insight into memory-based arithmetic.

Another application is using a resistance switch array as a content-addressable memory (CAM) as lookup table (LUT) in topology block. The LUT stores the entire connectivity among neurons. When a spike occurs from a neuron, the topology block searches the LUT and finds the destination neurons and synapses to update. Resistance switch-based CAM (RCAM) satisfies fast search ability, high integration density and low static energy consumption, and thus it is appropriate for LUT.

RCAM, however, has a low data density due to the use of a pair of resistance switches for a single bit of contents (0.5 bit/switch) in comparison with resistive random access memory (1 bit/switch). In this paper, we propose a new type of RCAM referred to as combination-encoding CAM (CECAM). In *N*-CECAM, a single unit consists of *N* high and *N* low resistance state switches whose combination collectively represents binary contents, yielding a data density of approximately 0.85 bit/switch when $N = 10$, for instance. The key to CECAM is the encoding of an *n*-bit search key as a 2*N*-digit key and its decoding. To this end, we propose a simple algorithm for encoding and decoding and its implementation in digital circuitry.

Keywords: Neuromorphic engineering, resistance switch array, multilayer perceptron, Markov chain, content-addressable memory

Student Number: 2015-20801

Guhyun Kim

Table of Contents

List of Tables

List of Figures

numbers in the range 1 – M are described by one-hot vectors. Any two of total M^2 numbers are combined to form an input vector u_1 ($u1 \in \mathbb{Z}2M;\ \ u1i \in 0,1$); for instance, when $M = 9$, u_1 for one and six is [100000000|000001000], where the first and last 9 bits indicate one and six, respectively, as shown in the figure. The correct answer serves as the label of chosen numbers; there are M^2 labels in total. Each label (bucket) has H elements so that the write vector v is a M^2H long vector that is adjusted given the correct label. Given entire pairs of numbers in the table and their multiplication results, the matrix w ($w \in \mathbb{Z}M2H \times 2M$) is adjusted. $P + 0,\ P - 0$, $b[i]$, and τ were set to 1, 0, 3, and 0.001, respectively (b) Network architecture of aliquot part retrieval given the matrix w. The transpose of w (w^{T}) finds the entire aliquot parts of a given number in a parallel manner in place. For instance, for number '6', an input vector u_1 (M^2H long vector) has a single nonzero bucket (6th bucket) that is filled with ones. The output vector z is [111001000|111001000], indicating the sum of four one-hot vectors ('1' + '2' + '3' + '6')—each of them is an aliquot part of 6. For prime numbers, the output vector includes only two 1's (1 and its own number) so that prime numbers can readily be found; for instance, 7 results in [100000100|100000100] as shown in the figure. 55

Figure 3.9.Prime factorization. (a) Memory ($wT \in \mathbb{Z}2M \times M2H$) based iterative and parallel search for prime factors. Given an input

List of Abbreviations

LRS	Low resistance state
HRS	High resistance state
RAM	Random access memory
SNN	Spiking neural network
IC	Integrated circuit
CMOS	Complementary metal oxide semiconductor
LUT	Lookup table
CAM	Content-addressable memory
MLP	Multilayer perceptron
CNN	Convolutional neural network
MAC	Multiply-accumulation
GPU	Graphics processing unit
MCMC	Markov chain Monte Carlo
MCHL	Markov chain Hebbian learning
CBA	Crossbar array
HL	Hidden layer
FPGA	Feld programmable gate array
2T2R	2 transistor-2 resistance switch
CECAM	Combination-econding content-addressable memory

1. Introduction

1.1. Resistance switch array

Resistance switch is regarded as a promising candidates for next-generation memory [1]. Resistance switch has two states called low resistance state (LRS) and high resistance state (HRS). The states of resistance switch are non-volatile so that it allows lower energy consumption compared to the conventional memories such as dynamic random access memory (DRAM), which needs refreshment. The switching between LRS and HRS is triggered by electrical stimulation such as applying voltage or current to resistance switch. In the resistance switch array, each resistance switch is placed at the crossing point between each horizontal and vertical metal electrode lines (Fig. 1.1). Note that these horizontal and vertical electrodes have roles of word and bit line. Therefore, resistance switch array is regarded as a two-terminal memory. This simple structure without transistor allows high integration density, the minimum cell size of $4F^2$, where F means the minimum feature size [2].

Recently, several analog resistance switches have been proposed [3], [4]. These analog resistance switches enable high data density because a single resistance switch express multi-bit data. Yet, analog resistance switches have bottlenecks such as high non-linear write-pulse number dependency [4], and they also need extremely dedicate control to reach desired resistance [5].

An important feature of resistance switch array is that it realizes matrix-vector multiplication [6]-[8]. From the Kirchhoff's law, the output current

response is derived as the multiplication between conductance matrix and input voltage vector (Fig. 1.1). This parallel operation enables resistance switch array to be applied to various field, such as analog computer, artificial synapse array, content-addressable memory (CAM). Additionally, this parallel operation enables exclusion of sneak current, which causes degradation of sensing margin, because all electrodes are connected to ground or V_{dd}. Therefore, sneak currents problem is merely considered in the parallel operation of resistance-switch array [6]-[8].

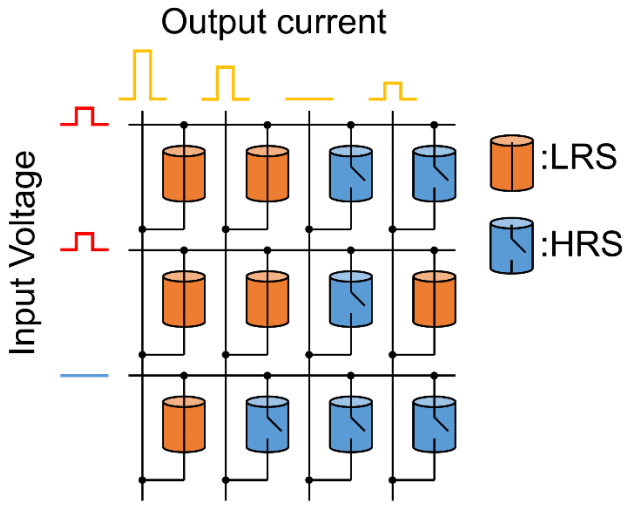


Figure 1.1. Schematic of resistance switch array. Each resistance switch is placed at each crossing points between electrodes. The output current from resistance switch array is same as multiplication between conductance matrix and input voltage vector.

1.2. Resistance switch array application in neuromorphic hardware

Neuromorphic engineering aims for implementing biologically plausible spiking neural network (SNN) into hardware [9]. With SNN, neuromorphic hardware is expected to be energy-efficient similar to human brain [10]. Also it is suitable for temporal learning, including temporal difference learning [11] and temporal sequence learning [12], and thus expected to be appropriate to solve time-dependent problem.

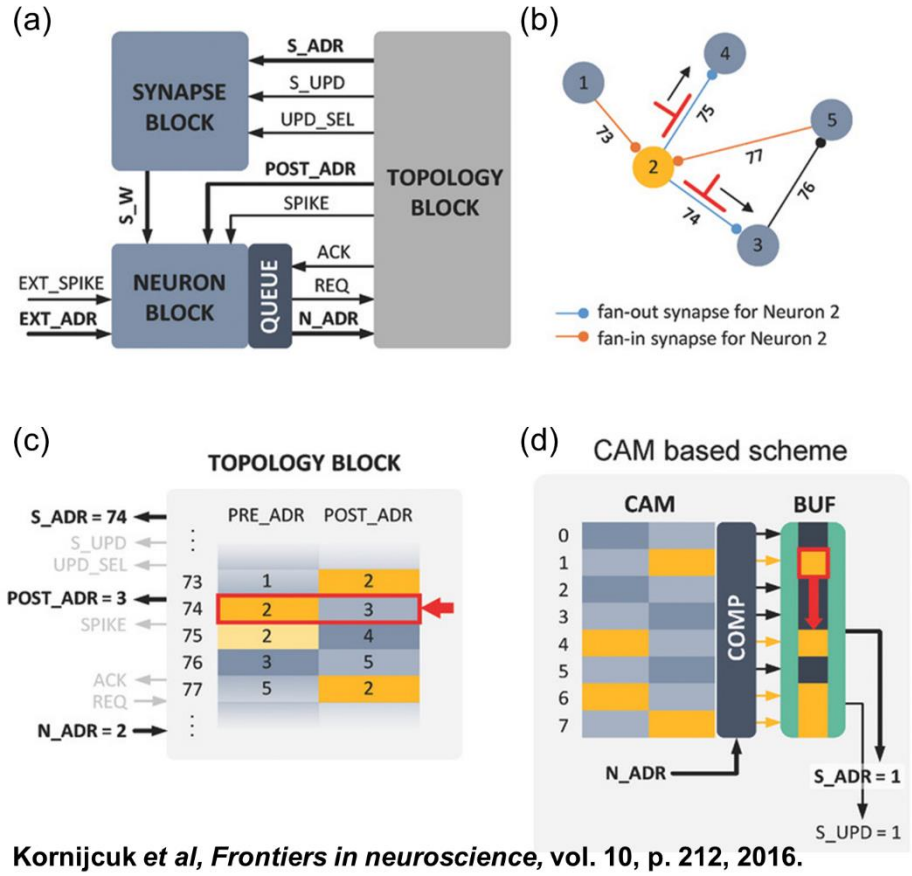
A neuromorphic hardware consists of neurons that are interconnected through synapses. Implementing neurons and synapses commonly uses analog and/or digital integrated circuits (IC) based on standard complementary metal oxide semiconductor (CMOS) technologies [13], [14]. Recently, emerging devices such as phase change memory [15], [16], magnetic tunnel junctions [17], [18], threshold switches [19], and floating-gate transistors [20] are proposed to build artificial neurons and synapses.

Resistance switch array is also a strong candidate for artificial synapses [3], [4]. In neural network, the activation of pre-synaptic neurons causes spikes and these spikes are transmitted to post-synaptic neurons. Here, post-synaptic neurons receive weighted sum of spikes from pre-synaptic neurons, not spikes. The weighted sum of spikes is expressed as $\boldsymbol{W} \times \boldsymbol{x}$, where $\boldsymbol{W}$ and $\boldsymbol{x}$ indicate a synaptic weight matrix and activation of pre-synaptic neurons, respectively. It is similar to matrix-vector multiplication in resistance switch array, mentioned section 1.1 so that resistance switch array is usable as artificial synapse array.

Another application of resistance switch array in a neuromorphic hardware is as a look-up table (LUT) in a topology block [21] (Fig. 1. 2). The entire connections between neurons through synapses are tabulated in the LUT. When a spike occurs from a neuron, the topology block searches all elements of the LUT and find the post-synaptic neurons and synapses to update. Therefore, fast-search ability is the most important factor of LUT. RAM is not a proper solution for LUT because RAM search every address sequentially and it causes significant delays. Unlike RAM, content-addressable memory (CAM) has parallel search ability and thus it is proper to be used as LUT [22]. The conventional CAMs, however, have SRAM-based structure which needs tremendous amount of transistors and have low-integration density.

Compared to SRAM-based CAM, resistance switch-based CAM (RCAM) has much higher content density because they use much less transistors [23], [24]. Also, RCAM has very low static energy consumption because of non-volatility. Consequently, RCAM is appropriate for the LUT in topology block of neuromorphic hardware.

From this features, this paper consists of three parts. At first, artificial neural network is applied to accelerate simulation of resistance-switch array. In the second part, a new learning algorithm called Markov Chain Hebbian Learning is proposed as the appropriate learning algorithm for resistance switch array. Lastly, a new type of RCAM, called combination-encoding CAM, is proposed to improve content density.



Kornijcuk *et al*, *Frontiers in neuroscience*, vol. 10, p. 212, 2016.

Figure 1.2. (a) Scheme of neuromorphic hardware. It consists of neuron block, synapse block which realize artificial neurons and synapses, respectively, and topology block. (b), (c) Topology block stores neuronal connectivity. When a spike occurs, it searches LUT and find the destination neurons and synapses to be updated. (d) Scheme for CAM as LUT. CAM enables fast-searching for topology block.

1.3. Bibliography

[1] A. Beck, J. Bednorz, C. Gerber, C. Rossel, and D. Widmer, *Applied Physics Letters*, vol. 77, no. 1, pp. 139-141, 2000.

[2] J. Y. Seok, S. J. Song, J. H. Yoon, K. J. Yoon, T. H. Park, D. E. Kwon, H. Lim, G. H. Kim, D. S. Jeong, and C. S. Hwang, *Advanced Functional Materials*, vol. 24, no. 34, pp. 5316-5339, 2014.

[3] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, *Nature*, vol. 521, no. 7550, pp. 61-64, 2015.

[4] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, and E. U. Giacometti, *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498-3507, 2015.

[5] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, and J. J. Yang, *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[6] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, and C. S. Hwang, *Advanced Electronic Materials*, vol. 2, no. 9, p. 1600090, 2016, Art no. 1600090.

[7] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, in *Proceedings of the 53rd annual design automation conference*, 2016: ACM, p. 19.

[8] L. Gao, P. Y. Chen, and S. Yu, *IEEE Electron Device Letters*, vol. 37, no. 7, pp. 870-873, 2016.

[9] C. Mead, *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629-1636, 1990.

[10] G. Cauwenberghs, *Proceedings of the national academy of sciences*, vol. 110, no. 39, pp. 15512-15513, 2013.

[11] R. P. Rao and T. J. Sejnowski, *Neural computation*, vol. 13, no. 10, pp. 2221-2237, 2001.

[12] F. Wörgötter and B. Porr, *Neural computation*, vol. 17, no. 2, pp. 245-319, 2005.

[13] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, and S. Renaud, *Frontiers in neuroscience*, vol. 5, p. 73, 2011.

[14] M. R. Azghadi, N. Iannella, S. F. Al-Sarawi, G. Indiveri, and D. Abbott, *Proceedings of the IEEE*, vol. 102, no. 5, pp. 717-737, 2014.

[15] T. Tuma, A. Pantazi, M. Le Gallo, A. Sebastian, and E. Eleftheriou, *Nature nanotechnology*, vol. 11, no. 8, p. 693, 2016.

[16] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, and N. C. Farinha, *Nature*, vol. 558, no. 7708, p. 60, 2018.

[17] A. Sengupta, P. Panda, P. Wijesinghe, Y. Kim, and K. Roy, *Scientific reports*, vol. 6, p. 30039, 2016.

[18] A. Mizrahi, T. Hirtzlin, A. Fukushima, H. Kubota, S. Yuasa, J. Grollier, and D. Querlioz, *Nature communications*, vol. 9, no. 1, p. 1533, 2018.

[19] H. Lim, H.-W. Ahn, V. Kornijcuk, G. Kim, J. Y. Seok, I. Kim, C. S. Hwang, and D. S. Jeong, *Nanoscale*, vol. 8, no. 18, pp. 9629-9640, 2016.

[20] V. Kornijcuk, H. Lim, J. Y. Seok, G. Kim, S. K. Kim, I. Kim, B. J. Choi, and D. S. Jeong, *Frontiers in neuroscience*, vol. 10, p. 212, 2016.

[21] V. Kornijcuk, J. Park, G. Kim, D. Kim, I. Kim, J. Kim, J. Y. Kwak, and D. S. Jeong, *Advanced Materials Technologies*, vol. 4, no. 1, p. 1800345, 2019.

[22] K. Pagiamtzis and A. Sheikholeslami, *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712-727, 2006.

[23] A. Grossi, E. Vianello, C. Zambelli, P. Royer, J.-P. Noel, B. Giraud, L. Perniola, P. Olivo, and E. Nowak, *IEEE Transactions on Very Large Scale Integration Systems*, no. 99, pp. 1-9, 2018.

[24] R. Han, W. Shen, P. Huang, Z. Zhou, L. Liu, X. Liu, and J. Kang, *Japanese Journal of Applied Physics*, vol. 57, no. 4S, p. 04FE02, 2018.

2. Artificial neural network for response inference of a nonvolatile resistance-switch array

2.1. Introduction

An artificial neural network (ANN) is a layered graph of nodes (activation units) and edges (nonzero connection weights), offering an immensely versatile hypothesis for various types of data description and different training methods [1]. Among feed-forward neural networks, multilayer perceptrons (MLP) and convolutional neural networks (CNN) are the most frequently applied types of neural network [2]. MLP is a prototypical feed-forward architecture in which every unit in a layer is fully wired to all units in the adjacent layers. In contrast, CNN has interlayer connections that are sparse and localized in the network topology [3,4]. A weight matrix in the CNN filters an input matrix fed into the next layer, and this filter (also known as convolution kernel) skims over the input layer. This is mathematically identical to convolving around the input layer, thus this architecture is termed CNN. In fact, the CNN has been successfully applied to a wide range of tasks including image recognition [1], [3]-[5] and natural language processing [6].

The scope of tasks (other than conventional tasks mentioned above) within the capability of ANN has been markedly expanding, including quantum mechanical problems such as estimation of quantum mechanical ground state given a two-dimensional potential distribution [7] and modelling a mechanical

system in presence of noise [8]. These examples highlight the neural network as a versatile hypothesis and the capability of backpropagation for supervised learning as a widely applicable training method.

Meanwhile, a crossbar array of nonvolatile resistance-switches, i.e., passive resistive random access memory (RRAM), ideally meets the 4F2 design rule (F is the minimum feature size), offering a solution to high-density nonvolatile memory [9]-[11]. Additionally, its current response to an applied voltage array naturally captures the multiply-accumulate (MAC) operation so that crossbar arrays have often been used for physical implementation of the matrix–vector product [12]-[14]. The benefit of this approach is obvious in comparison to the digital MAC operation: high speed due to the fully parallel operation and energy-efficiency due to no need for data transference during the operation. Given that the MAC operation is at the heart of MLP for both training and inference, the passive RRAM can substantially improve efficiency in MLP, which is an important field of neuromorphic engineering [12], [14]-[19].

Considering the beneficial relationship between passive RRAM and MLP (particularly, the aforementioned passive RRAM for MLP), it is of interest to seek the reverse approach (MLP for passive RRAM). To this end, this work exemplifies the feasible application of MLP to the response inference of passive RRAM in which, once trained, the inference merely costs a few steps of matrix-vector product (depending on the depth of the network). Our new method may offer a new feasible means of crossbar circuit simulations as an alternative to conventional circuit simulation methods.

2.2. Description of model system

Passive RRAM as a model system is a $M \times N$ matrix R loaded with R_{HRS} and R_{LRS} that denote resistance in a high resistance state (HRS) and low resistance state (LRS), respectively, i.e., $R \in \{R_{\mathrm{HRS}}, R_{\mathrm{LRS}}\}^{\mathrm{M} \times \mathrm{N}}$. This model system outputs an N-long real-valued current vector ($\in \mathbb{R}^N$) in response to an M-long real-valued input voltage vector V ($\in \{0,1\}^M$). The model system is illustrated in Fig. 2.1(a).

The model is a nonlinear system because the HRS features a highly nonlinear current-voltage (*I-V*) relationship in contrast to the linear (or almost) *I-V* of the LRS. In this regard, the HRS was provided with a nonlinear *I-V* characteristic as follows: $\mathrm{I} = \mathrm{I}_0 \mathrm{e}^{aV}$, where I_0 and a denote a pre-exponential factor and voltage coefficient, respectively. The larger *a*, the higher nonlinearity is given to the *I-V* behavior. Such nonlinearity in the HRS has been observed in an enormous number of resistance-switches given the usual thermal activation of current transport in the HRS [10, 20, 21]. In contrast, the LRS was given a linear *I-V* characteristic, keeping fidelity to experimental systems that generally represent linear or very weakly nonlinear *I-V* characteristics.

Two types of resistance-switch were addressed in this study: Type A and B, whose detail is tabulated in Table 2.1. The *I-V* behavior for each switch is plotted in Fig. 2.1(b). They differ in the $R_{\mathrm{HRS}}/R_{\mathrm{LRS}}$ ratio (evaluated at 1 V); the ratio for Type A is 100 times larger than Type B. For each type, two different array sizes (10×9 and 28×27; M = 10 and N = 9, and M = 28 and N = 27, respectively) were considered.

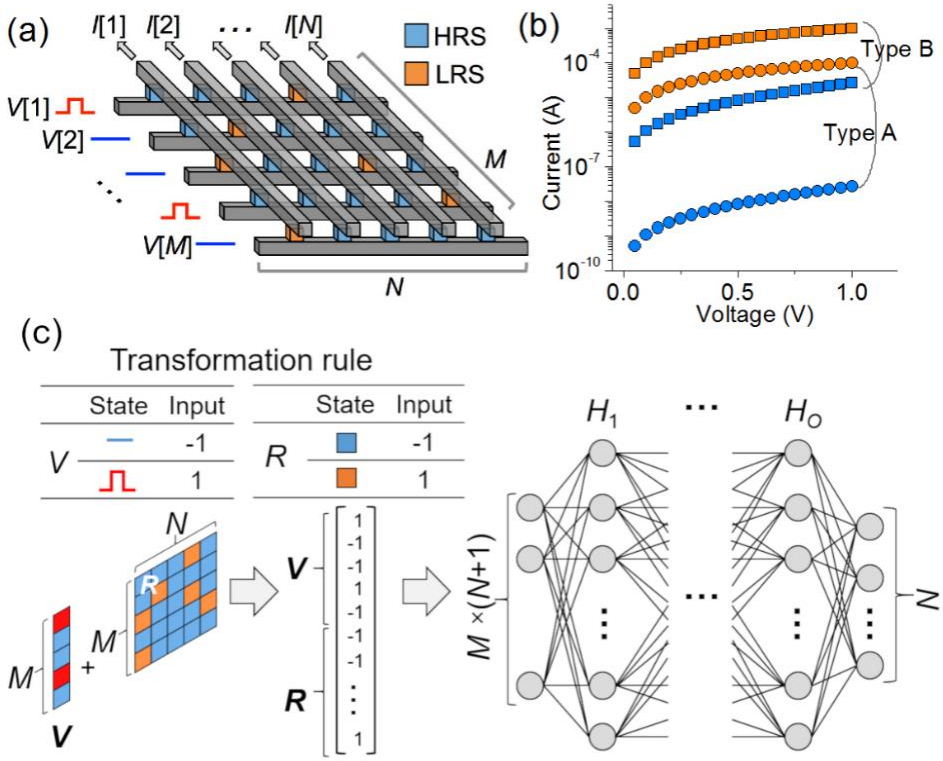


Figure 2.1. (a) Schematic of an $M \times N$ crossbar array. (b) Assumed *I*-*V* characteristics of the model resistance-switches (Types A and B). (c) Schematic of the MLP with $M \times (N + 1)$ input and N output units, and O hidden layers. The rule for mapping resistance-switches and input voltage arrays to an input vector is tabulated in the inset.

Table 2.1. Parameters of model switch.

Heading	Type A	Type B
R_{HRS} (Ω)	$10^8 \times e^{-V}$	$10^5 \times e^{-V}$
R_{LRS} (Ω)	10k	1k
R_{HRS}/R_{LRS} at 1 V	3679	36.79

2.3. Description of artificial neural network

The passive RRAM outputs a current vector I that is determined by the configuration of switches over the whole array instead of their local configuration. A fully connected feed-forward network is, therefore, suitable for the model system instead of a CNN capturing patterns over local areas. Additionally, given the aforementioned nonlinearity of the model system, a hidden layer(s) needs to be incorporated in the network, rendering an MLP most suitable. Thus, an MLP was chosen as an appropriate network for the crossbar array. Fig. 2.1(c) illustrates the employed MLP with $M \times (N + 1)$ input units, N output activation units, and O hidden layers, each of which is filled with Hi activation units where $i \in \{1, 2, \cdots, O\}$. The input into the MLP is the resistance-state (+1 and −1 for the LRS and HRS, respectively) distribution over the $M \times N$ array (R) plus an M-long vector for input voltage (+1 and−1 for $V[i] = 1$ and $V[i] = 0$, respectively) as sketched in Fig. 2.1(c). This matrix is then vectorized to feed into the MLP. The output is the estimated output current of the crossbar array at a given voltage. Note that successful training is crucial to rescale the original physical input (resistance and voltage) and output (current) in a heuristic manner such that the rescaled (scale-free) values stay in an “acceptable” range. To this end, symbolic (+1 and−1), rather than physical, values were given to the input components. Likewise, the desired (correct) output values (currents) were rescaled such that $L[i] = 10 \times I[i] \times R_{\mathrm{LRS}}$.

The leaky rectified linear unit (ReLU) was deployed as an activation unit: $f(x) = \max(x, 0.1x)$. The leakage when $x < 0$ is required for the negative input

components. Otherwise, the negative input components are merely ignored as for the simple ReLU, $f(x) = \max(x, 0)$. The ReLU is a workaround for the notorious vanishing gradient problem, which is significant when the network is deep.

2.4. Training and test datasets

The output $\boldsymbol{I}$ in response to an input $\boldsymbol{V}$ for a given $\boldsymbol{R}$ was evaluated by applying the Kirchhoff's circuit law to each switch. The obtained nonlinear equations were solved using the Newton-Raphson method, which resulted in the output $\boldsymbol{I}$. The calculation was elaborated in [22]. A training dataset was produced by randomly sampling resistance state distribution over the array and input $\boldsymbol{V}$. First, p_1 ($0 \le p_1 \le 1$) was randomly sampled from a uniform probability distribution function (PDF) and used as the probability that $V[i] = 1$. That is, if p_1 is 0.4, 40% of all input lines are pulled high (1 V), and the rest lines (60%) are pulled down (0 V). Another number p_2 ($0 \le p_2 \le 1$) was subsequently sampled for each input line from a uniform PDF to randomly distribute 1 V signals over all input lines at a probability of p_1 such that, when $p_2 \le p_1$, $V[i] = 1$, and 0 otherwise. This process was repeated with different p_2's over M rows, resulting in an input $\boldsymbol{V}$ for this training example. A third number p_3 ($0 \le p_3 \le 1$) was picked from a uniform PDF and taken as the percentage of LRS switches in the entire array. For each switch in the array, p_3 was compared with another random number, p_4 ($0 \le p_4 \le 1$) was sampled for each switch, and $R[i, j] = R_{\mathrm{LRS}}$ when $p_4 \le p_3$, and $R[i,j] = R_{\mathrm{HRS}}$ otherwise. The label of this training example was the current response for $\boldsymbol{I}$ given $\boldsymbol{R}$ and $\boldsymbol{V}$. The complete dataset was acquired by repeating this process. The test dataset was separately made for the fair evaluation of

inference accuracy. Two different crossbar array sizes (10×9 and 28×27) for each type of switch were considered so that four different training and test datasets were produced. Each training dataset included 500,000 training examples (V, R, and I) unless otherwise specified. The network was examined for every training epoch using 10,000 test examples. Backpropagation using the mean-squared error loss function was employed with Adam optimizer that leverages learning rate adaptation for each parameter to accelerate training [23]. The MLP was batch-trained with a batch size of 100 (100 examples were randomly chosen for each training epoch). Both training and inference were performed using TensorFlow [24]. Note that for successful training, the network should vary on its hyper-parameters such as the number of ReLU units in each hidden layer (H_i) and the network depth (O) depending on the input array length.

2.5. Training results

Fig. 2.2 shows a reduction in the discrepancy between the output (inferred) current Iout and desired (correct) current I_{cor} in due course, revealing successful training for all four cases conditional on the network structure. For the small crossbar array (10×9), a network including a single hidden layer ($O = 1$) loaded with 100 ReLU units could successfully be trained with the 500,000 training examples (Fig. 2.2(a), (b)). However, the use of fewer units (50 and 75) falls short of the capability of learning the dataset so that a high error level is maintained for both types of switch. This is a result of underfitting referring to the use of an unsuitable network for capturing the input pattern. Here, the network is too simple (insufficient number of units) to describe the complexity

of input data. The successfully trained network infers the output current of a random 10×9 crossbar array $\boldsymbol{R}$ at a random $\boldsymbol{V}$. The inferred currents for 10,000 test examples are plotted against the desired (correct) currents in Fig. 2.2(e), (f), each of which includes 90,000 data points (10,000 test examples, each of which produces 9 current values). The error histogram for each case is plotted in the inset, indicating a root mean squared error (RMSE) of 0.313 μA and 17.8 μA, respectively. The larger error for Type 2 switch arises from the higher current in both HRS and LRS due to the lower RHRS and RLRS. The results for the larger crossbar array (28 × 27) of Types A and B switches are shown in Fig. 2.2(c), (d), respectively. Given the larger input dimension (28×28 = 784), a network needs more units in each hidden layer and/or more hidden layers for success in training. The employed network varies on the number of units (1500 and 2500) in a hidden layer and the network depth (1 and 2). The three networks among four are given the capability to estimate the response of a random 28 × 27 crossbar array $\boldsymbol{R}$ at a random $\boldsymbol{V}$. As such, the network fully trained along the green curve for Types A and B switches represents low inference-error (a RMSE of 4.85 μA and 62.7 μA, respectively) as elucidated in Fig. 2.2(g), (h), and their insets.

The correlation coefficient r for each case was also evaluated as another measure of success of training, which is given by $r = cov(I_{out} - I_{cor})/\sqrt{var(I_{out}) \cdot var(I_{cor})}$, where *cov* and *var* denote a covariance and variance, respectively. The correlation coefficient is asymptotic to 1 when the inference error tends to zero, and thereby r = 1 implies zero error (perfect match). The calculated r for each case is written in Fig. 2.2. The failure of training for the

network with 2,500 units in each of the two hidden layers is due to overfitting (see orange curves in Fig. 2.2(c), (d)). Although the network is given sufficient complexity (a large number of units and hidden layers) to learn the complex input pattern, insufficient training examples lead to faulty training as shown in the orange curves (Fig. 2.2(c), (d)). Overfitting could be avoided by training with a larger training dataset (here 1,000,000 examples for Type B switch) as shown in Fig. 2.3(a). The inference-error for the overfitting case is detailed in Fig. 2.3(b) which represents a substantial discrepancy between the inferred and desired outputs, the extent to which the RMSE reaches 438.2 μA (r = 0.99571). The error statistics are plotted in the inset. In contrast, a remarkable reduction in inference-error is identified for the non-overfitting case (Fig. 2.3(c)) whose RMSE is lowered down to 49.2 μA (r = 0.9995).

Finally, we compared the time-efficiency of the proposed method with the conventional Newton-Raphson method [22]. The run time of a 10×9 resistance array calculation was measured for both methods using the same computer. The result shown in Fig. 2.4 ensures an acceleration in calculation by approximately 8 times, identifying a feasible benefit of fast calculation from the proposed method.

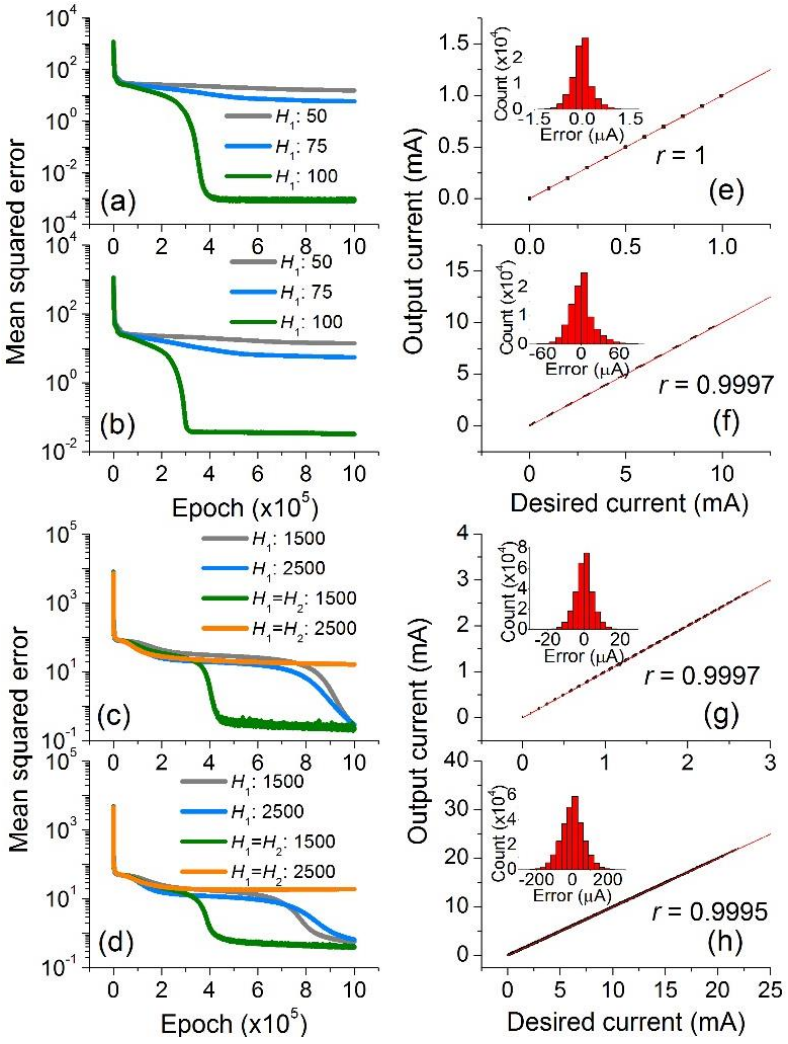


Figure 2.2. Inference-error reduction while training a network with the dataset of a 10 × 9 crossbar array of (a) Type A and (b) Type B switches. Their output results (inferred currents) for the entire 10,000 test datasets after successful training (green lines) are plotted against the desired currents in (e) and (f), respectively. The histogram of the error (the difference between inferred and desired currents) for each case is shown in the inset. The red solid lines denote the perfect match of inference with the desired (correct) results. The results are

shown for a 28 × 27 crossbar array of (c) Type A and (d) Type B switches, and their statistics in (g) and (h), respectively.

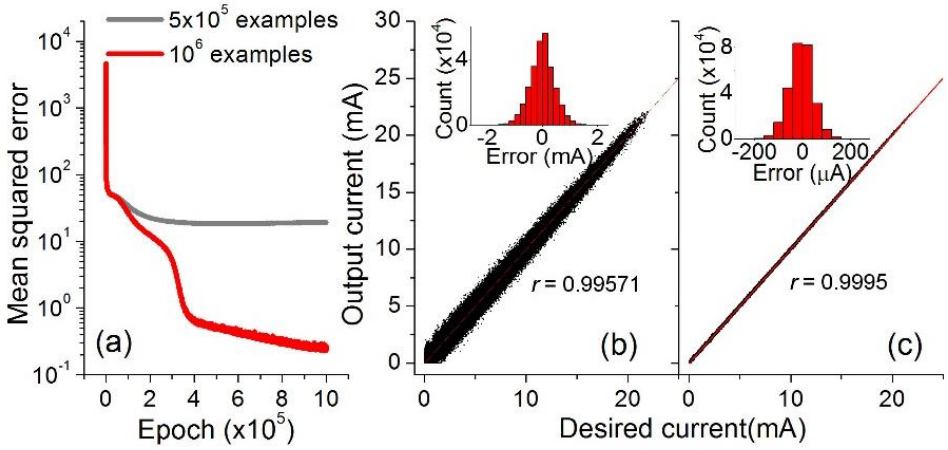


Figure 2.3. (a) Training the network (2,500 units in each of two hidden layers) with 500,000 and 1,000,000 examples for Type B switch. The capability of response inference is shown for the network trained with (b) 500,000 and (c) 1,000,000 examples. The insets address the distribution of inference-error.

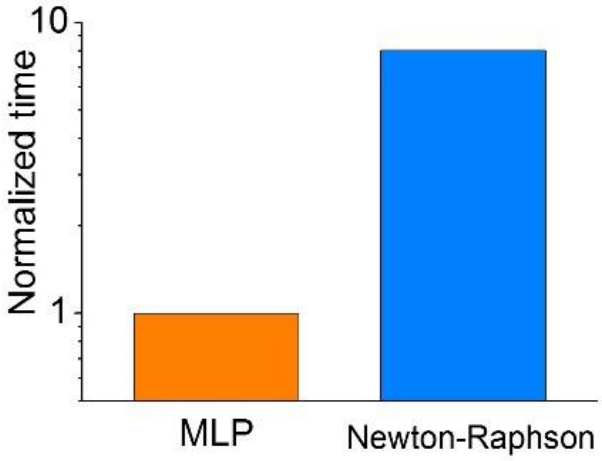


Figure 2.4. Comparison of run time for the proposed method and Newton-Raphson method.

2.6. Conclusions

A fully connected feed-forward network with different structures (depth and the number of activation units) was successfully trained to infer the current response of a random crossbar array to a randomly applied voltage array. This work first verifies the capability of ANN to capture the highly nonlinear input-output relationship of a crossbar array model system. Secondly, MLP for supervised learning provides a means of real-valued array inference beyond the classification of input patterns. Thirdly, this work offers a distinct view of crossbar array evaluation — a numerical solution of a number of simultaneous equations can be avoided at the expense of a few steps of matrix-vector product for inference. However, training the network and preparing datasets can be expensive, depending on the network hyper-parameters and model crossbar array size. Thus, we leave this efficiency issue open for the moment.

2.7. Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, *Nature*, Insight vol. 521, no. 7553, pp. 436-444, 2015.

[2] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, and A. Borchers, in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017: IEEE, pp. 1-12.

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Proceeding of IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[4] Y. LeCun, K. Kavukcuoglu, and C. Farabet, in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 253-256.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in Neural Information Processing Systems*, 2012, pp. 1097-1105.

[6] R. Collobert, J. Weston, L. o. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, *Journal of Machine Learning Research*, vol. 12, pp. 2493-2537, 2011.

[7] K. Mills, M. Spanner, and I. Tamblyn, *arXiv:1702.01361*, 2017.

[8] M. Nentwig and P. Mercorelli, in *2008 7th IEEE International Conference on Cybernetic Intelligent Systems*, 2008: IEEE, pp. 1-6.

[9] R. Waser, R. Dittmann, G. Staikov, and K. Szot, *Advanced Materials*, vol. 21, pp. 2632-2663, 2009.

[10] D. S. Jeong, R. Thomas, R. Katiyar, J. Scott, H. Kohlstedt, A. Petraru, and C. S. Hwang, *Reports on Progress in Physics*, vol. 75, no. 7, p. 076502, 2012.

[11] J. Y. Seok, S. J. Song, J. H. Yoon, K. J. Yoon, T. H. Park, D. E. Kwon, H. Lim, G. H. Kim, D. S. Jeong, and C. S. Hwang, *Advanced Functional Materials*, vol. 24, no. 34, pp. 5316-5339, 2014.

[12] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, and C. S. Hwang, *Advanced Electronic Materials*, vol. 2, no. 9, p. 1600090, 2016.

[13] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, *Proceedings of the 53nd Annual Design Automation Conference*, 2016, pp. 1-6.

[14] L. Gao, P. Y. Chen, and S. Yu, *IEEE Electron Device Letters*, vol. 37, no. 7, pp. 870-873, 2016.

[15] D. S. Jeong, I. Kim, M. Ziegler, and H. Kohlstedt, *RSC Advances*, vol. 3, no. 10, pp. 3169-3183, 2013.

[16] J. J. Yang, D. B. Strukov, and D. R. Stewart, *Nature Nanotechnology*, vol. 8, no. 1, pp. 13-24, 2013.

[17] P. Y. Chen, L. Gao, and S. Yu, *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 4, pp. 257-264, 2016.

[18] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, and W. D. Lu, *Nature Nanotechnology*, Article vol. advance online publication, 2017.

[19] S. Choi, J. H. Shin, J. Lee, P. Sheridan, and W. D. Lu, *Nano Letters*, vol. 17, no. 5, pp. 3113-3118, 2017.

[20] D. S. Jeong, H. Schroeder, and R. Waser, *Electrochemical Solid-State Letters*, vol. 10, p. G51, 2007.

[21] D. S. Jeong, H. Schroeder, and R. Waser, *Physical review B*, vol. 79, p. 195317, 2009.

[22] D. S. Jeong, H.-W. Ahn, S.-D. Kim, M. An, S. Lee, and B.-k. Cheong, *Electronic Materials Letters*, vol. 8, no. 2, pp. 169-174, 2012.

[23] D. P. Kingma and J. Ba, *arXiv:1412.6980*, 2014.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, and M. Devin, *arXiv:1603.04467*, 2016.

3. Markov chain hebbian learning algorithm with ternary synaptic units

3.1. Introduction

Recent progress in machine learning (particularly, deep learning) endows machines with high precision recognition and problem-solving capabilities beyond the human level [1]-[3]. Computers on the von Neumann architecture are the platform for the breakthroughs albeit frequently powered by hardware accelerators, e.g., graphics processing unit (GPU) [4]. The main memory stores intertwined fragmentary information, e.g., weight matrix, representation of hidden neurons, input datasets, and so forth. However, essential to efficient memory retrieval is memory organization such that the whole weight matrix can readily be recalled when necessary. In this regard, a high-density crossbar array (CBA) of two-terminal memory elements, e.g., oxide-based resistive memory and phase change memory, is perhaps a promising solution to machine learning acceleration [5]-[9].The connection weight between a pair of neurons is stored in each memory element in the CBA as conductance, and the weight is read out in place by monitoring current in response to a voltage [5]-[9].

Albeit promising, this approach should address the following challenges; each weight should be calculated beforehand using a conventional error-correcting technique, and the pre-calculated value needs to be programmed in a single memory element. The former particularly hinders online learning. In this study, an easy-to-implement algorithm based on a stochastic neural

network—termed the Markov chain Hebbian learning (MCHL) algorithm—is proposed. The most notable difference between the MCHL and restricted Boltzmann machine (RBM) [10]-[15] is that the MCHL is a discriminative learning algorithm with the aid of "external field" that realizes supervised learning. Also, each update uses only local (spatial and temporal) data rather than global data such as energy of the entire network. The MCHL algorithm also features as follows: (a) Each weight $w[i,j]$ is a ternary number: $w[i,j] \in \{-1,0,1\}$

(b) Given (a), each update of weight follows a finite-state Markov chain, and the update probability is in line with the Hebbian learning.

(c) A group of output neurons in a bucket (rather than a single neuron) simultaneously represent a data class (label), which is comparable to concept cells [16]-[18].

(d) When the network is deep, the network is trained in a greedy layer-wise manner, and each layer is trained in a greedy edge-wise manner.

Provided with these features, the MCHL algorithm enables an ad hoc update of the weight matrix (online learning) in a memory-saving fashion, so that it is suitable for machine learning powered by CBA-based memory. No need for an auxiliary function for error correction, e.g., backpropagation, particularly alleviates computational complexity. Each synapse is given a ternary number during the entire learning period—distinguishable from binarizing real-valued weight at each update step [19] as well as the use of auxiliary real-valued variables [20]. A Markov chain, specifically, in Markov chain Monte Carlo(MCMC), is a common means of sampling from a complex distribution

of data to extract information in stochastic machine learning [21]. Especially, a Markov decision process offers a solution to an optimal policy that maps a current state of an agent to a certain action resulting in the maximum reward in reinforcement learning [22], [23] Additionally, MCMC yields a posterior probability distribution that is the key to Bayesian inference and learning [21] Examples also include recent attempts to apply Markov chains to multi-instance multi-label learning [24] that addresses objects embodying multiple instances (features). In this case, Markov chains are used as probabilistic classifiers mapping multiple instances to multiple labels [25].

Stochastic Hebbian learning algorithms are methods to probabilistically train a binary synapse conditional on the pre and postsynaptic activities in line with the MCHL algorithm [26], [27]. Interestingly, such algorithms can train networks to a comparable degree with its deterministic counterpart [26], [27]. Yet, these algorithms barely support supervised learning for classification tasks. Senn and Fusi proposed a single-layer perceptron with a stochastic learning algorithm for supervised learning [28]. The algorithm requires global inhibition that is applied to all output neurons so that the actual synaptic input in total (input from binary excitatory synapses plus global inhibition) is not all or nothing. Additionally, no explicit method to apply the algorithm to multilayer perceptrons (MLPs) is proposed.

Note that, regarding the feature (d), the network depth indicates repeated linear classifiers through the layers so that it differs from that of a multilayer feed-forward network that features a nonlinear classifier. Nevertheless, we term the additional layers between input and output layers as hidden layers(HLs)

given that they are literally hidden irrespective of their role in non-linear classification. Additionally, a network with such HLs is referred to as a deep network.

The MCHL algorithm was applied to two proof-of concept examples: image recognition using the MNIST and CIFAR-10 datasets and multiplication table memorization. The latter example relates the arithmetic to memory-based perception in an analogous way to humans' mental arithmetic. The weight matrix trained with the multiplication table was then applied to more complicated arithmetic such as aliquot part evaluation and prime factorization.

3.2. Model description

3.2.1. Network structure and energy

Analogous to the RBM, two layers of neurons without recurrent connection form the basis for the MCHL algorithm. However, it differs from the RBM such that the HL in the RBM is replaced by an output layer that does not feed input into the input layer. Fig. 3.1(a) depicts a stochastic neural network of M input features and N output neurons. $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ denote the input vector and activity vector of the output layer, defined as

$$\begin{cases} \boldsymbol{u}_1 \in \mathbb{R}^M, & 0 \le u_1[i] \le 1 \\ \boldsymbol{u}_2 \in \mathbb{Z}^N, & u_2[i] \in \{0,1\} \end{cases},$$

respectively. In the output layer, H neurons associatively represent each of total L labels so that the output layer includes LH neurons (N=LH). A group of such H neurons is referred to as a bucket. When the L labels are indexed from 1 to L, $u_2[(n-1)H+1{:}nH]$ is a block of output activities for the nth label. Note that $x[a{:}b]$ denotes a block ranging from the ath to bth elements of vector x. A matrix

$\boldsymbol{w}$ ($\in \mathbb{Z}^{LH\times M}$) defines the weight of feed-forward connection from the input to output layer such that the input $z[i]$ into the ith output neuron is given by

$$z[i]=\sum_{j=1}^{M} w[i,j]u_1[j] \quad (1)$$

Each element of $\boldsymbol{w}$ is given one of the ternary values (−1, 0, 1). According to the bucket configuration of the vector $\boldsymbol{u}_2$, the matrix w can be partitioned such that $\boldsymbol{w}[(n-1)H+1{:}nH,\cdot]$ is for the connection from the input vector to the output neurons of the nth label. '·' means all j=1, ..., M. Likewise, $\boldsymbol{z}(=\boldsymbol{w}\boldsymbol{u}_1)$can also be partitioned into L buckets.

The energy of this network is defined as

$$E(\boldsymbol{u}_1, \boldsymbol{u}_2) = -\left(2\boldsymbol{u}_2 - \vec{\mathbf{1}}\right)^{\mathrm{T}} \cdot \boldsymbol{w} \cdot \boldsymbol{u}_1 + \boldsymbol{b}^{\mathrm{T}} \cdot \boldsymbol{u}_2, \quad (2)$$

where $\boldsymbol{w}$ is a weight matrix, $\vec{\mathbf{1}}$ is a N-long vector filled with ones. $\boldsymbol{b}$ denotes a bias vector for the output layer. $\left(2\boldsymbol{u}_2 - \vec{\mathbf{1}}\right)$ in (2) transforms $\boldsymbol{u}_2$ such that a quiet neuron ($u_2[i] = 0$) is given an output of −1 rather than zero. This counts the cost of a positive connection ($w[i, j]$=1) between a nonzero input ($u_1[j]\neq 0$) and output neuron in an undesired label ($u_2[i]$=0). This undesired connection raises the energy by $u_1[j]$.

The following conditional probability that $u_2[i]$=1 given $z[i]$ holds:

$$P(u_2[i] = 1|z[i]) = \left[1 + \mathrm{e}^{-(2z[i]-b[i])/\tau}\right]^{-1}, \quad (3)$$

where τ denotes a temperature parameter. (3) is plotted in Fig. 3.1(b). The derivation of (3) is elaborated in Appendix A. We also define the deterministic activity of neuron i in the jth layer as

$$a_j[i] = \left[1 + \mathrm{e}^{-(2z[i]-b[i])/\tau}\right]^{-1}. \quad (4)$$

For instance, for the network in Fig. 3.1(a), $a_2[i]$ denotes the activity of neuron

i in the second (output) layer. This deterministic activity is used for inference as follows. The output from each label n ($O[n]$) is the sum of deterministic activity over all output neurons in the label. The maximum component of O designates the estimated label for a given input. (4) is also used when training a deep network (Sec. IVA).

Note that, unless otherwise stated, the bias is set to zero, simplifying (2), (3), and (4) to

$$E(\boldsymbol{u}_1, \boldsymbol{u}_2) = -\left(2\boldsymbol{u}_2 - \vec{\mathbf{1}}\right)^{\mathrm{T}} \cdot \boldsymbol{w} \cdot \boldsymbol{u}_1, \tag{5}$$

$$P(u_2[i] = 1 | z[i]) = \left[1 + \mathrm{e}^{-2z[i]/\tau}\right]^{-1}, \tag{6}$$

and

$$a_j[i] = \left[1 + \mathrm{e}^{-2z[i]/\tau}\right]^{-1}, \tag{7}$$

respectively. The description of each mathematical symbol is addressed in Table 3.1.

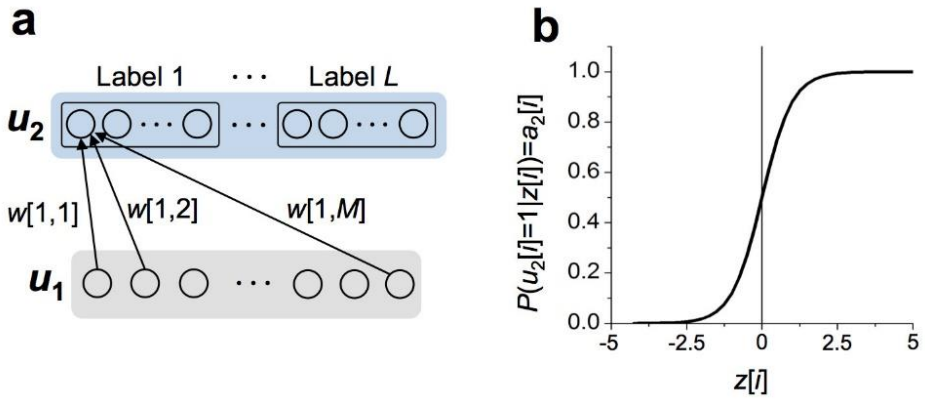


Figure 3.1. MCHL algorithm working principle. (a) Basic network of M input and N output binary stochastic neurons (u_1 and u_2: their activity vectors). (b) Behavior of $P(u_2[i] = 1)$ with z[i] when $b[i]$=0. This probability is identical to the deterministic activity $a_2[i]$ of the neuron.

Table 3.1. Symbols.

Symbol	Description
$x[i]$ ($i \geq 1$)	ith element in array $\boldsymbol{x}$
$x[i{:}j]$ ($i \geq j$)	Block ranging from the ith to jth element in array $\boldsymbol{x}$
$\boldsymbol{u}_1$	Activity vector of the M input neurons $\boldsymbol{u}_1 \in \mathbb{R}^M;$ $0 \leq u_1[i] \leq 1$
$\boldsymbol{u}_2$	Activity vector of the N output neurons $\boldsymbol{u}_2 \in \mathbb{Z}^N;$ $u_2[i] \in \{0,1\}$
$\boldsymbol{w}$	Weight matrix $\boldsymbol{w} \in \mathbb{Z}^{N \times M};$ $w[i,j] \in \{-1,0,1\}$
$\boldsymbol{b}$	Bias vector for the output neurons
$\boldsymbol{z}$	Array of inputs into the output neurons $z[i] = \sum_{j=1}^{M} w[i,j]u_1[j]$
$\boldsymbol{a}_2$	Deterministic activity of the N output neurons $\boldsymbol{a}_2 \in \mathbb{R}^N;$ $0 \leq a_2[i] \leq 1$
L	Number of total labels in a dataset
H_i	Number of neurons in a bucket in the ith layer
$\boldsymbol{v}$	Write vector $\boldsymbol{v} \in \mathbb{Z}^{LH};$ $v[i] \in \{-1,1\}$ if $H = 1$ $v[i] \in \{-1,0,1\}$ otherwise
τ	Temperature parameter
E	Energy of the model
P_+	Probability of potentiation
P_-	Probability of depression
P_+^0	Maximum probability of potentiation
P_-^0	Maximum probability depression

3.2.2. Field application and update probability

In the MCHL algorithm, write vector $\boldsymbol{v}$ designates the correct label of a given input $\boldsymbol{u}_1$. Akin to $\boldsymbol{u}_2$, $\boldsymbol{v}$ is an LH-long vector in which $v[(n-1)H+1{:}nH]$ is assigned to the nth label. The correct label (indexed N) is indicated by $\boldsymbol{v}$ such that

$$v[i]=\begin{cases} 1 & \text{if } i=(N-1)H+h \\ -1 & \text{if } i=(n-1)H+h \quad \text{for all } n\ (\neq N) \\ 0 & \text{otherwise} \end{cases}, \tag{8}$$

where $1 \le h \le H$, and h is chosen at random. That is, one of the elements for label N is endowed with 1 while one of the elements for each undesired label is given −1. Thus, only one element in $\boldsymbol{v}$ has 1, $L-1$ elements −1, and the others 0. In conjunction with the corresponding input vector $\boldsymbol{u}_1$, a field matrix $\boldsymbol{F}$ is defined as $\boldsymbol{F}=\boldsymbol{v}\cdot\boldsymbol{u}_1^{\mathrm{T}}$ and $F[i,j]=v[i]u_1[j]$ element-wise. $F[i,j]$ determines the sign and probability of weight change of $w[i,j]$ for a given input and its correct label. $F[i,j]$ (>0) causes potentiation ($\Delta w[i,j]=1$) at probability P_+ only if $u_2[i]=0$ (condition (a)) and $w[i,j]\neq 1$ (condition (b)). In contrast, $F[i,j]$ (<0) causes depression ($\Delta w[i,j]=-1$) at probability P_- only if $u_2[i]=1$ (condition (a)) and $w[i,j]\neq -1$ (condition (b)). P_+ and P_- are

$$\begin{cases} P_+ = P_+^0 F[i,j]=P_+^0 v[i]u_1[j] \\ P_- = -P_-^0 F[i,j] = -P_-^0 v[i]u_1[j] \end{cases}, \tag{9}$$

where P_+^0 and P_-^0 denote the maximum probability of potentiation and depression, respectively. Stochastic update on weight given probability is detailed in Appendix B.

This update rule is reminiscent of the Hebbian learning such that the larger the input $u_1[j]$, the more likely the update is successful since P_+ and P_- scale

with $u_1[j]$ as shown in (9). Condition (a) indicates that a quiet output neuron ($u_2[i] = 0$) supports potentiation, whereas an active one ($u_2[i] = 1$) supports depression. Condition (b) keeps $w[i, j] \in \{-1, 0, 1\}$ so that the update falls into a finite state Markov chain. $\boldsymbol{v}$ is renewed for the subsequent update with another input data and its label. h in (8) is also randomly renewed.

Specifically, the MCHL algorithm exploits inhomogeneous Markov chains that alter the transition matrices every training epoch given the update probability conditional on input and write vector according to (9). Several basic properties of the inhomogeneous Markov chains in the MCHL algorithm are addressed in Appendix C. Generally, a learning rate is of significant concern for successful learning. A learning rate in the MCHL algorithm is dictated by P_+^0 and P_-^0 in place of an explicit rate term. For extreme cases such as $P_+^0 = 1$ and $P_-^0 = 1$, the matrix barely converges, but constantly fluctuates.

When including HLs (Fig. 3.2), the network is trained in a greedy layer-wise manner as for deep belief networks [29]. That is, the matrix $\boldsymbol{w}_1$ was first fully trained with a field matrix $\boldsymbol{F}_1$ of each input vector $\boldsymbol{u}_1$ and the corresponding write vector $\boldsymbol{v}$. The matrix $\boldsymbol{w}_2$ is subsequently trained with a field matrix $\boldsymbol{F}_2$ for a given $\boldsymbol{u}_1$ and $\boldsymbol{v}$, which reads $\boldsymbol{F}_2 = \boldsymbol{v} \cdot \boldsymbol{u}_2^{\mathrm{T}}$. Such layer-wise training continues up to the topmost weight matrix $\boldsymbol{w}_{\mathrm{D}-1}$ that is trained with $\boldsymbol{F}_{\mathrm{D}-1}$ shown in Fig. 3.2.

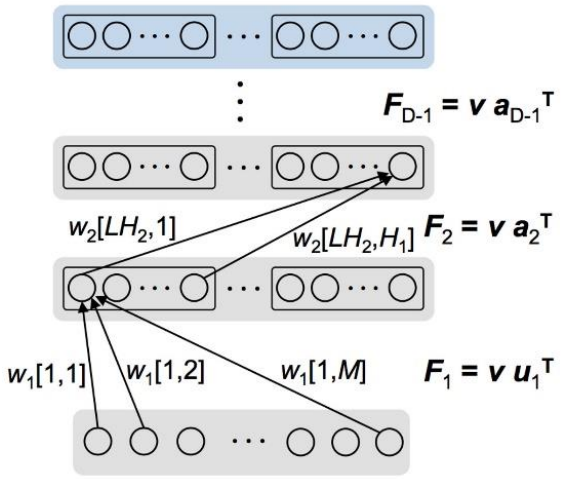


Figure 3.2. Network with hidden layers. F_2 and F_{D-1} denote a field matrix for w_2 and w_{D-1}.

3.3. Implementation of the MCHL algorithm on hardware

3.3.1. Field-programmable gate array

Implementing the MCHL algorithm on hardware boosts the advantage of the algorithm with regard to its efficient use of memory and computational simplicity in weight update. To identify the acceleration of training and inference, a field programmable gate array (FPGA) is an easy-to-implement test bed where weight matrices can be densely organized in static random access memory (SRAM) arrays that are readily accessed when necessary. We will highlight the significant acceleration of the MCHL algorithm by implementing the MCHL algorithm on an FPGA board later in Sec. IVA.

3.3.2. Resistance-based random access memory

A CBA of resistance-based memories offers extremely time efficient multiply-accumulate (MAC) operation and random accessibility to each bit [30], making the MCHL come into its own. Fig. 3.3(a) illustrates a feed-forward connection between u1 and u2 for the topology in Fig. 3.1(a), where the weight matrix w is mapped onto a RAM. Each ternary unit is placed at the cross point between a word line (vertical grey line) and bit line (horizontal grey line). The input vector u1 is physically represented by a voltage array in that $u_1[j]$ is applied to the jth word line. $w[i, j]$ is implemented by the conductance of the unit at the cross point between the jth word and ith bit lines. High conductance and low conductance correspond to1 and 0, respectively. Likewise, a $w[i, j]$ of -1 corresponds to negatively high conductance. This counterintuitive concept is

realized as illustrated in Fig. 3.3(b). Each unit consists of 2 bits (two resistors), and each word line for $u_1[j]$ is paired with an additional line for negative $u_1[j]$ (Fig. 3.3(b)). Therefore, the total current through the parallel resistors I is

$$I = (G[i,j] - \overline{G}[i,j])u_1[j]$$

where $\boldsymbol{G}$ and $\overline{\boldsymbol{G}}$ are the conductance of the left and right resistors in each unit, respectively. The three combinations of $\boldsymbol{G}$ and $\overline{\boldsymbol{G}}$ in Fig. 3.3(b) realize the ternary weight. Note that $(\boldsymbol{G}, \overline{\boldsymbol{G}}) = (1,1)$ is not favorable because of high power consumption, it can represent 0 though. Therefore, in this strategy, $\boldsymbol{z}$ corresponds to an array of output currents; $z[i]$ is the current through the ith bit line, equivalent to (1). The random accessibility to each unit supports the parallel programming (training) of the units with a programming voltage applied to each bit line. An array of programming voltages corresponds to write vector $\boldsymbol{v}$ (Figs. 3.3(a) and (c)). The sign of $v[i]u_1[j]$ dictates the weight change of the unit placed between the ith bit line and jth word line. When positive, the unit is given the non-zero probability that $\Delta w[i, j] = 1$ (potentiation) while negative $v[i]u_1[j]$ gives the unit nonzero probability that $\Delta w[i, j] = -1$ (depression) as sketched in Figs. 3.3(c) and (d), respectively.

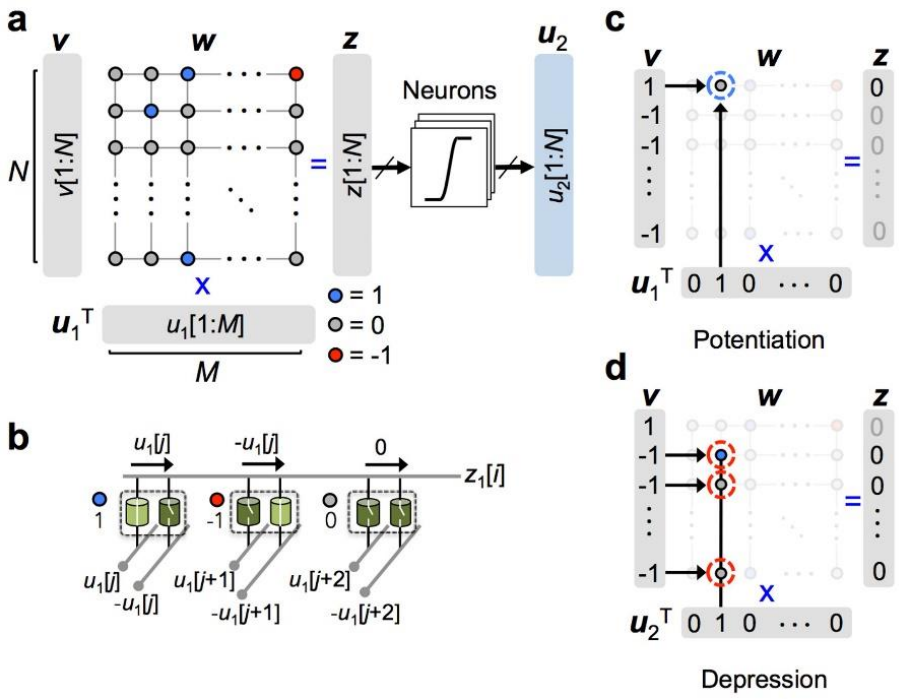


Figure 3.3. Memory-centric illustration of a neural network. (a) Graphical description of the weight matrix w that determines the correlation between the input activity u_1 and output activity u_2. The grey vertical and horizontal lines denote word and bit lines, respectively. This weight matrix w evolves in accordance to given pairs of an input u_1 and write vector v, ascertaining the statistical correlation between u_1 and v. (b) A pair of memory resistors in each synaptic unit. Three combinations of the two conductance values represent the ternary weight (1, -1, 0). (c) Potentiation: a weight component at the current step t ($w_t[i, j]$) has a nonzero probability to gain +1 (i.e. $\Delta w_t[i, j] = 1$) only if $u_1[j] \neq 0$, $v[i] = 1$, and $w_t[i, j] \neq 1$; for instance, given $u_1 = (0, 1, 0, \ldots, 0)$ and $v = (1, -1, -1, \ldots, -1)$, $w_t[1, 2]$ has a probability of positive update. (d) Depression:

all components $w_t[i, 2]$ ($i \neq 1$) are probabilistically subject to negative update (gain -1) insofar as $u_1[2] \neq 1$, $v[i] = -1$, and $w_t[i, 2] \neq -1$.

3.4. Applications

3.4.1. Image recognition

The MCHL algorithm was applied to image recognition tasks with the MNIST database ($M = 28\times28$ and $L = 10$) and CIFAR-10 database ($M = 32\times32\times3$ and $L = 10$). Fig. 3.4(a) shows a memory-centric schematic of the network for the training, which includes one HL. The implementation was two-fold. First, the MCHL algorithm was implemented on a general-purpose computer (CPU: Intel i5-4690 3.5GHz) without using a GPU. The code was written in Python. Second, the algorithm was implemented on an FPGA board (Virtex-7 XC7VX485T) to identify the acceleration of the algorithm. Hereafter, the FPGA board on which the MCHL algorithm is implemented is referred to as an MCHL accelerator. Regarding a tradeoff between recognition accuracy and training speed, parameters P_+^0 $(= P_-^0)$ and τ were set to 0.1 and 1, respectively, during training with the MNIST dataset. The effect of the parameters on training behavior is elaborated in Appendix D. Note that parameters P_+^0 $(= P_-^0)$ and τ were set to 0.01 and 1, respectively, during training with the CIFAR-10 dataset, with regard to the tradeoff.

3.4.1.1. Implementation on a general-purpose computer

When training the network with the MNIST dataset, the repeated *ad hoc* updates increase the recognition accuracy and decrease the network energy in (5) as plotted in Fig. 3.4(b). The network depth substantially alters the recognition accuracy as plotted in Fig. 3.4(c). Without HL the accuracy merely reaches approximately 88% at $H_1 = 100$ while deploying one HL improves the

accuracy up to approximately 92% at H_1 =100 and H_2 =50. Note that H_1 and H_2 denote bucket size in the HL and output layer, respectively. Improvement on accuracy continues onwards with more HLs (e.g., two HLs; blue curve in Fig. 3.4(c)), although its effect becomes smaller compared with the drastic improvement by the first HL. The training and test in detail are addressed in Appendix E. The weight matrix becomes larger with bucket size, so is the memory allocated for the matrix. Nevertheless, the benefit of deploying buckets at the expense of memory is two-fold. First, many input features (pixels) are shared among labels such that several individual features do not exclusively belong to a single particular label. The use of buckets allows such common features to be connected with elements over different labels given the sparse update on the weight matrix. For instance, without such buckets, every attempt to direct the feature at (1,1) — belonging to both labels 1 and 2 — to label 1 probabilistically weakens its connection with label 2. Second, when shared, the statistical correlation between the feature and each of the sharing labels is captured by bucket, enabling comparison among the labels. As depicted in Fig. 3.4(a), the 10 sub-matrices in the matrix $\boldsymbol{w}_2$ define 10 ensembles of H_2 output neurons; the final output from each label $O[n]$ is the sum of deterministic activity $a_2[i]$ over the neurons in the same label, i.e., the output range scales with H_2 in the range $0 - H_2$.

A single training is hardly able to capture a statistical correlation between input and write vectors. However, the larger the training numbers, the less likely the statistical error (noise) is incorporated into the data, which is similar to the error reduction in Monte Carlo simulation with an enormous number of random

numbers (RNs) [31]. The use of buckets enables the parallel acquisition of effectively multiple $\boldsymbol{w}$ matrices as opposed to repeated training trials to acquire a $\boldsymbol{w}$ matrix on average. Therefore, it is conceivable that a larger bucket size tends to improve the recognition accuracy. In fact, the bucket size and consequent memory allocation for matrix $\boldsymbol{w}$ significantly determine the recognition accuracy (see Fig. 3.5). However, in Monte Carlo simulations, the error reduction with sample number tends to be negligible when the number is sufficiently large. The same holds for the MCHL algorithm as shown in Fig. 3.5. Additionally, the memory cost perhaps outweighs the negligible improvement in the accuracy. Therefore, it is practically important to reconcile the performance with the memory cost.

Considerable reductions in memory usage and training time (for 105 epochs) for the MCHL algorithm were experimentally identified as plotted in Fig. 3.6. The networks subject to the measurements varied in the numbers of HLs and neurons in each layer. Benchmarking data were acquired from two feed-forward networks: MLP and convolutional neural network (CNN). They were trained using a backpropagation algorithm with real-valued weights. The MLP consisted of 784 input neurons, one HL including 100 neurons, and 10 output neurons. The CNN employed 3×3 kernels, 1×1 stride, and 2×2 max pooling size. Its fully-connected network was of 2,048×100×10. The MLP and CNN can infer the labels of handwritten digits with high accuracy (98% and 99.5%, respectively) at the cost of memory in use and complexity in computation (see Fig. 3.6). On the other hand, the input complexity in the CIFAR-10 dataset keeps there cognition accuracy of our network considerably low as for the MLP

trained using a backpropagation algorithm [32]. The network under training varied in the number of HLs from zero to three with a bucket size of 500. P_{+}^{0}, P_{-}^{0} and τ were set to 0.01, 0.01, and 1, respectively. The training results are plotted in Fig.3. 7, identifying a maximum accuracy of approximately 43% when incorporating three HLs. This maximum accuracy is approximately 8% lower than the benchmark accuracy from an MLP with three HLs (each of which has 500 nodes) trained using a backpropagation algorithm with real-valued weights (see the red curve in Fig. 3.7(a)).

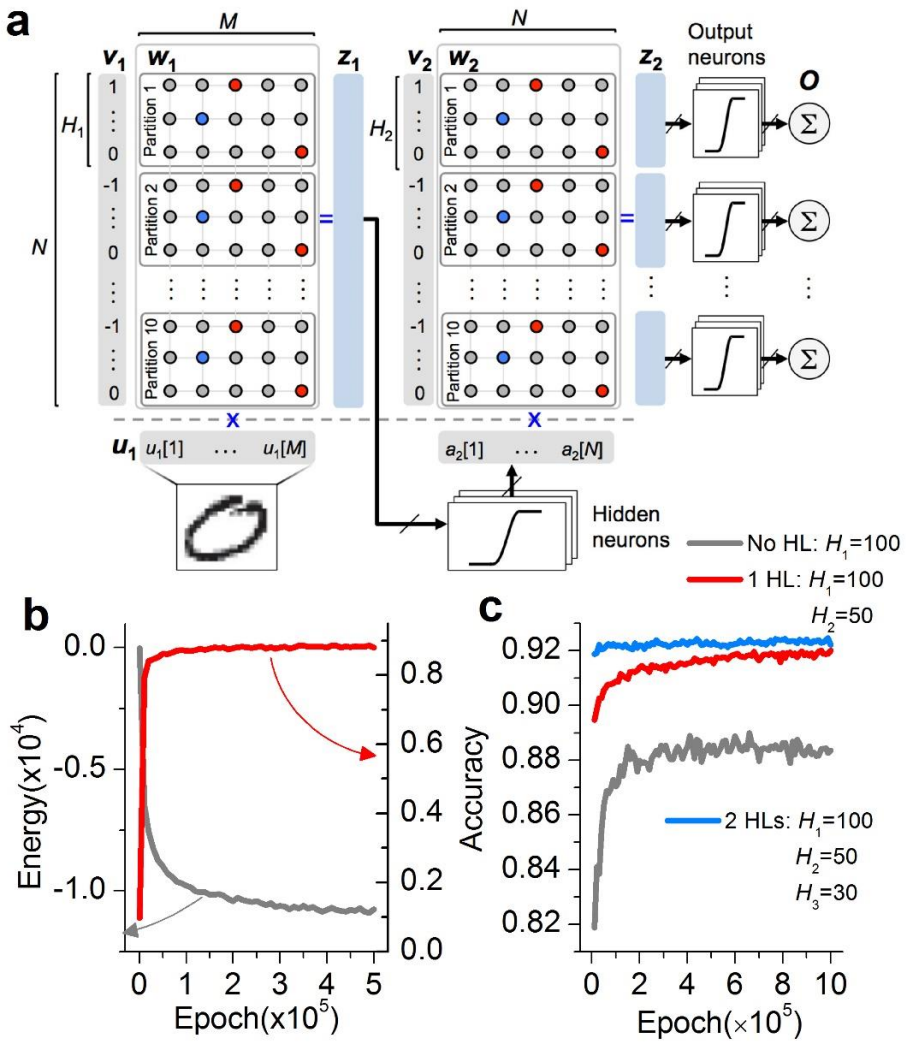


Figure 3.4. Application to handwritten digit recognition. (a) Schematic of the network architecture for handwritten digit recognition. A single HL is included. The matrix w_1 first maps the input vector u_1 to the hidden neurons. The array a_2 is taken as an input vector to w_2 that maps the input vector to the output neurons. The write vector v_1 has 10 (the number of labels) buckets, each of which has H_1 elements, i.e. $N = 10H_1$. Each thick arrow indicates an input vector to a group of neurons (each neuron takes each element in the input vector). (b) The increase of recognition accuracy (red curve) and corresponding decrease of

energy (grey curve) with training epoch. The trained network is a single-layer network (H=100). (c) Classification accuracy change in due course of training with network depth (H_1=100, H_2=50, H_3=30).

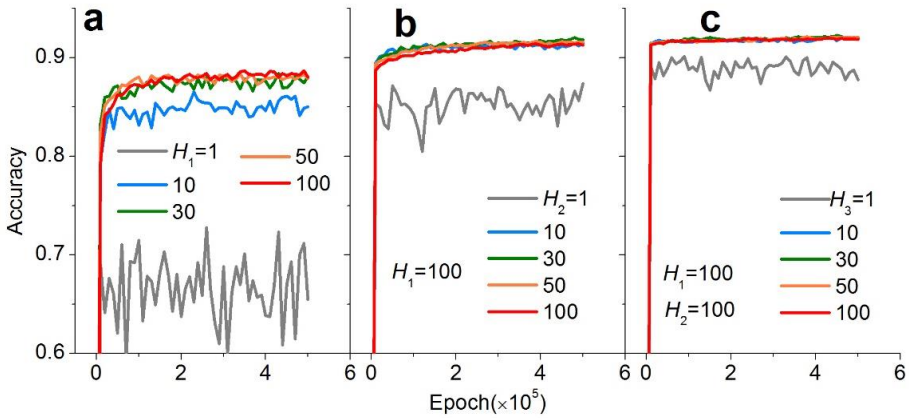


Figure 3.5. Bucket size dependence of recognition accuracy. Recognition accuracy change with (a) H_1 in a network without a hidden layer, (b) H_2 with a single hidden layer (w_1 was fully trained beforehand; H_1=100), and (c) H_3 with two hidden layers (w_1 and w_2 were fully trained beforehand; H_1=100 and H_2=100).

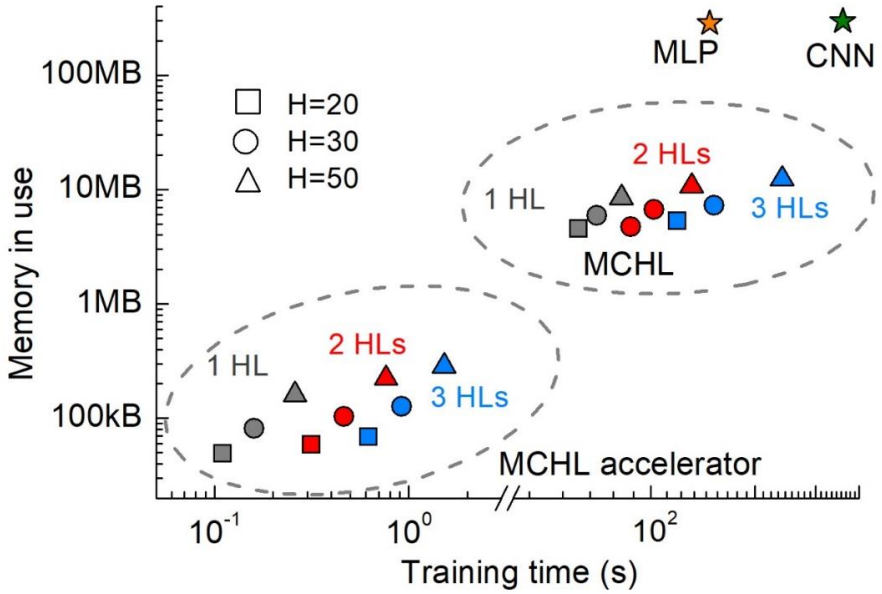


Figure 3.6. Memory usage and training time (for 105 epochs) for the MCHL algorithm. The networks subject to the measurements varied in the numbers of HLs (1, 2, and 3) and neurons (HD20, 30, and 50) in each bucket. Each HL included the same number of neurons. The data were compared with the memory usage and training time for the MCHL accelerator and two feed-forward networks (MLP and CNN) trained using a backpropagation algorithm (105 training epochs). The clock speed of the FPGA board was set to 20 MHz.

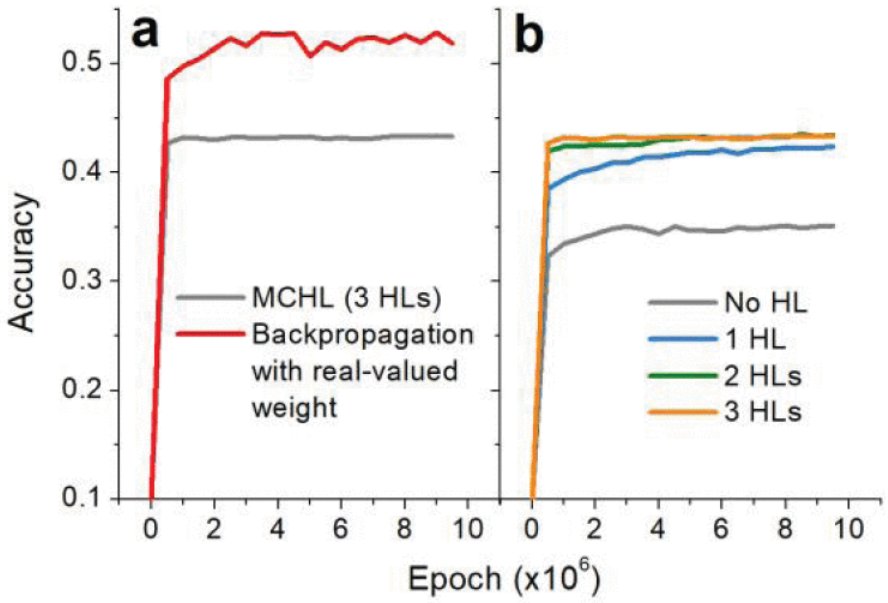


Figure 3.7. Recognition accuracy of networks trained with the CIFAR-10 dataset. (a) Accuracy evolution with training epoch for a network including three HLs, each of which embodies 500 nodes, reaching approximately 43%. An MLP trained using backpropagation with real-valued weights represents approximately 51%. (b) Recognition accuracy upon the number of HLs. Each HL includes 500 nodes

3.4.1.2. MCHL accelerator

The same type of network was built on an FPGA board and trained using the MCHL algorithm that was modified to save the resource. The modification includes representation of $a_j[i]$ in (4) and (7) using an 8-bit integer value. The original input data (8 bits/pixel) was downsized to 2 bits/pixel to accelerate the input data transfer from the computer to the FPGA board (bandwidth: 300 kb/s). The MCHL accelerator is of bucket-wise parallel structure such that the evaluation of neuronal activities in one bucket is performed in parallel with the other buckets. Accordingly, the partitions of each weight matrix a real so structured in parallel so that an update on weight in each partition can be executed in parallel. The MCHL accelerator is elaborated in Appendix F.

A network with one HL (H_1=20, H_2=10) was trained with the downsized MNIST dataset, resulting in are cognition accuracy of 88%. The reduction in recognition accuracy for the FPGA implementation arises from the downsized input data and the use of 8-bit numeric data type for $a_j[i]$.

The MCHL accelerator markedly accelerates training and minimizes a need for memory (see Fig. 3.6). Evaluating the activity of each neuron in a bucket using (1) and (7) merely needs one clock cycle T_{clk} (=1/f_{clk}, where f_{clk} denotes clock speed). Inferring a single handwritten digit needs to evaluate all neurons in the network, $(H_1+H_2)L$ in total. The evaluation for each bucket is executed in parallel. Thus, each inference takes $(H_1+H_2)T_{clk}$, i.e., $(H_1+H_2)/f_{clk}$. Setting f_{clk} to20MHz, single inference is finished in 1.5 μs. Each update on $\boldsymbol{w}_1$ needs the evaluation of $\boldsymbol{u}_2$ (performed in parallel with the update) given the current $\boldsymbol{w}_1$, $\boldsymbol{u}_1$, and $\boldsymbol{v}$ to determine the update probability detailed in Sec. IIIB. This is done

in a single clock cycle (T_{clk}) with regard to the partition-wise parallel weight update (see Appendix F). Therefore, each $\boldsymbol{w}_1$-training epoch takes $1/f_{clk}$, e.g., 50 ns at 20 MHz.

However, each update on $\boldsymbol{w}_2$ needs the evaluation of $\boldsymbol{a}_2$ given the fully trained $\boldsymbol{w}_1$ and input $\boldsymbol{u}_1$ using (1) and (7) beforehand. As such, this step takes $H_1 T_{clk}$, i.e., H_1/f_{clk}. Akin to updating w1, an update on $\boldsymbol{w}_2$ given the evaluated $\boldsymbol{a}_2$, current $\boldsymbol{w}_2$, $\boldsymbol{v}$, and $\boldsymbol{u}_3$ (also acquired in parallel with the update) merely takes one clock cycle (T_{clk}). The weight update time in total for each $\boldsymbol{w}_2$-training epoch is therefore $(H_1+1)/f_{clk}$: 1.05 μs at 20 MHz. The only memory in use was for the weight matrices $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. Given that 2-bit memory is allocated to each element, $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ need memory capacities of 313.6kb ($2\times784\times H_1 \times L$) and 40kb ($2\times H_1 \times L \times H_2 \times L$), respectively, i.e., 353.6 kb (44.2 kB) in total.

3.4.2. Multiplication table memorization and prime factorization

The MCHL algorithm can also be applied to deterministic learning. Examples include multiplication table memorization, where the MCHL algorithm spontaneously finds correct-answer-addressing matrix w. This way recalls, rather than computes, the correct answer. Matrix $\boldsymbol{w}$ $\left(\boldsymbol{w}\in\mathbb{Z}^{N\times 2M};\ w[i,j]\in\{0,1\},\ N=M^2H\right)$ was trained with the $M \times M$ multiplication table. Two integer factors in the range $(1 - M)$ were chosen and represented by two one-hot vectors, each of which had M elements. These two vectors were merged into input vector $\boldsymbol{u}_1$ $\left(\in\mathbb{Z}^{2M};\ u_1[i]\in\{0,1\}\right)$; $u_1[1{:}M]$ were allocated for the first vector, and $u_1[M{+}1{:}2M]$ for the second one. The product $(1 - M^2)$ is taken as the desired label of the input. Therefore, M^2 labels in total

are available. Given bucket size H for each label, write vector $\boldsymbol{v}$ is M^2H long.

Multiplication is deterministic so that no stochasticity intervenes in learning. Consequently, $P_+^0 = 1$ and $P_-^0 = 0$ were given to (9), and all neurons were frozen (τ = 0.01). In this regard, write vector generation does not require random sampling within the bucket in the desired label. Instead, an element in the bucket is conferred on each pair of factors in training order. For instance, 2×8 addresses the nth element in label 16, and the multiplication addressing the same label in the closest succession, e.g., 4×4, takes the (n+1)th element. Therefore, the bucket includes a set of possible multiplications yielding the same label. Notably, a prime number has only two factors, '1' and itself, and thus, the bucket includes only two multiplications. Note that bias is given to each output neuron; $b[i]$ = 3 for all i's. Therefore, (3) is expressed as $P(u_2[i] = 1|z[i]) = \left[1 + \mathrm{e}^{-(2z[i]-3)/\tau}\right]^{-1}$. The bias allows $u_2[i] = 1$ only if $z[i] > 2$ so that a single factor cannot solely activate the output neuron.

The network structure is sketched in Fig. 3.8(a); no HL is required to achieve the maximum accuracy. The training continued onwards until the entire pairs of numbers in the table were memorized. M^2 training steps were thus required to complete the memorization task. Indexing vector $\boldsymbol{A}$ $\left(\in\mathbb{Z}^{M^2}; \mathrm{A}[i] = h\right)$ was defined to count the possible multiplications (h) resulting in the same product. For instance, when $M \geq 6$, $A[6] = 4$ because 1×6, 2×3, 3×2, and 6×1 result in 6 (see Fig. 3.8(a)).

Notably, $A[i]$ is identical to the number of factors of i. The training procedure is elaborated in Appendix G. Note that the prime numbers large than M cannot

be taken as a label. Notably, the bucket size H should not be smaller than the maximum $A[i]$ ($i \leq M^2$), otherwise some buckets cannot host all multiplications. To save memory, it is necessary to calculate the integer ($\leq M^2$) that has the most factors and accordingly allocate memory to each bucket.

The trained matrix $\boldsymbol{w}$ can readily be used to find the aliquot parts of number n by transposing the matrix: $\boldsymbol{w}^{\mathrm{T}} \in \mathbb{Z}^{2M \times N}$; $N = M^2H$ (see Fig. 3.8(b)). The matrix multiplication $\boldsymbol{z} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{u}_1$ with $\boldsymbol{u}_1 \left(\in \mathbb{Z}^N;\ N = M^2H \right)$ — all H elements in the nth bucket are set to 1—yields a vector $\boldsymbol{z}$ whose upper M bits $z[1{:}M]$ are the sum of the entire aliquot parts, each of which is represented by a one-hot vector (Fig. 3.8(b)). Given the commutative property of multiplication, $z[1{:}M] = z[M+1{:}2M]$. For instance, when M = 9, input '6' yields $z[1{:}9]$ = [111001000], indicating '1' + '2' + '3' + '6'. A prime number '7' yields $z[1{:}9]$=[100000100] ('1'+'7'); two 1's in z indicates a prime number (h=2).

The matrix $\boldsymbol{w}$ trained with an $M \times M$ multiplication table also serves as the basis for prime factorization (Fig. 3.9(a)). It is a modified version of the aliquot part retrieval to avoid retrieving '1' and itself if other factors exist. A remarkable advantage consists in the parallel decomposition of many numbers; for input $\boldsymbol{u}$ (the sum of one-hot vectors under decomposition, e.g., A = a × b and B = c × d), the single matrix-vector multiplication $\boldsymbol{z} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{u}$ uncovers all a, b, c, and d. It should be noted that $u[i]$ for all i's is no longer one of the binary numbers (0 and 1); instead it can be any nonnegative integer.

An $M \times M$ multiplication table that the matrix $\boldsymbol{w}$ is trained with beforehand can be used to factorize any positive integers whose all factors are smaller than or equal to M. That is, a priori knowledge of a number subject to prime

factorization can significantly reduce the size of a multiplication table in use. Without such knowledge of integer N under prime factorization, a full $N \times N$ multiplication table is needed to safely prime factorize the number. If N is a priori known to be an even number, an $(N/2) \times (N/2)$ multiplication table is sufficient for successful prime factorization.

Fig. 3.9(b) illustrates a factor tree of '840'; the first iteration with $\boldsymbol{w}$ ($M = 50$) results in '40' + '21', the following iteration gives '2' + '3' + '7' + '20', and the third iteration 2 × '2' + '3' + '7' + '10', equivalent to $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, and $\boldsymbol{a}_3$ in Fig. 3.9(c). To demonstrate the efficiency of this method, a randomly picked integer in a multiplication table (M =300) was prime factorized, and the number of the iteration steps was counted. The results for the integers (1.62884×1010–7.75541294×1011) are plotted in Fig. 3.9(d) in comparison with benchmark results (direct search factorization). The higher efficiency of the present method over the benchmark can obviously be understood. The direct search factorization is elaborated in Appendix H. The matrix $\boldsymbol{w}$ once trained with a multiplication table can repeatedly be used to prime factorize numbers covered by the table. Therefore, the factorization iteration steps in Fig. 3.9(d) do not include the multiplication table memorization steps.

The capacity for prime factorization using the proposed algorithm is dictated by the size of a trained $M \times M$ multiplication table. As such, the larger the size M, the more the factorizable integers (Fig. 3.10). Note that the factorizable integers should be addressed as a product in the $M \times M$ multiplication table so that the number of factorizable integers is identical to that of products in the table. There exist 36 different products in the 9×9 multiplication table; all of

them are prime-factorizable. Upon enlarging the table size up to $M = 300$, the capacity reaches 24,047. Given the ternary weight in $\boldsymbol{w}$ (each element needs 2 bits), the required memory size for w (M =300) is 180 kbits (Fig. 3.10).

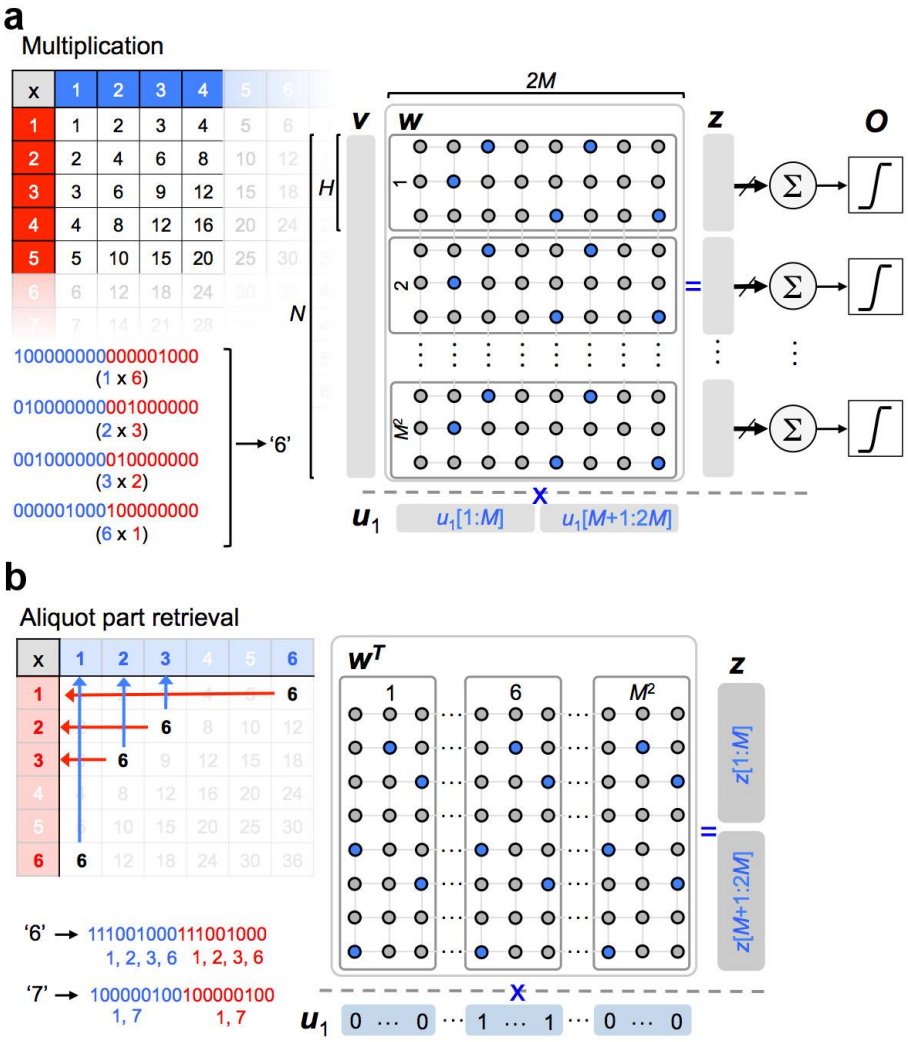


Figure 3.8. Multiplication table memorization and aliquot part retrieval. (a) Network architecture for multiplication table memorization. The numbers in the range 1 – M are described by one-hot vectors. Any two of total M^2 numbers are combined to form an input vector u_1 ($u_1 \in \mathbb{Z}^{2M}$; $u_1[i] \in \{0,1\}$); for instance, when M = 9, u_1 for one and six is [100000000|000001000], where the first and last 9 bits indicate one and six, respectively, as shown in the figure.

The correct answer serves as the label of chosen numbers; there are M^2 labels in total. Each label (bucket) has H elements so that the write vector v is a M^2H long vector that is adjusted given the correct label. Given entire pairs of numbers in the table and their multiplication results, the matrix w ($w \in \mathbb{Z}^{M^2H \times 2M}$) is adjusted. P_+^0, P_-^0, $b[i]$, and τ were set to 1, 0, 3, and 0.001, respectively (b) Network architecture of aliquot part retrieval given the matrix w. The transpose of w (w^T) finds the entire aliquot parts of a given number in a parallel manner in place. For instance, for number '6', an input vector u_1 (M^2H long vector) has a single nonzero bucket (6th bucket) that is filled with ones. The output vector z is [111001000|111001000], indicating the sum of four one-hot vectors ('1' + '2' + '3' + '6')—each of them is an aliquot part of 6. For prime numbers, the output vector includes only two 1's (1 and its own number) so that prime numbers can readily be found; for instance, 7 results in [100000100|100000100] as shown in the figure.

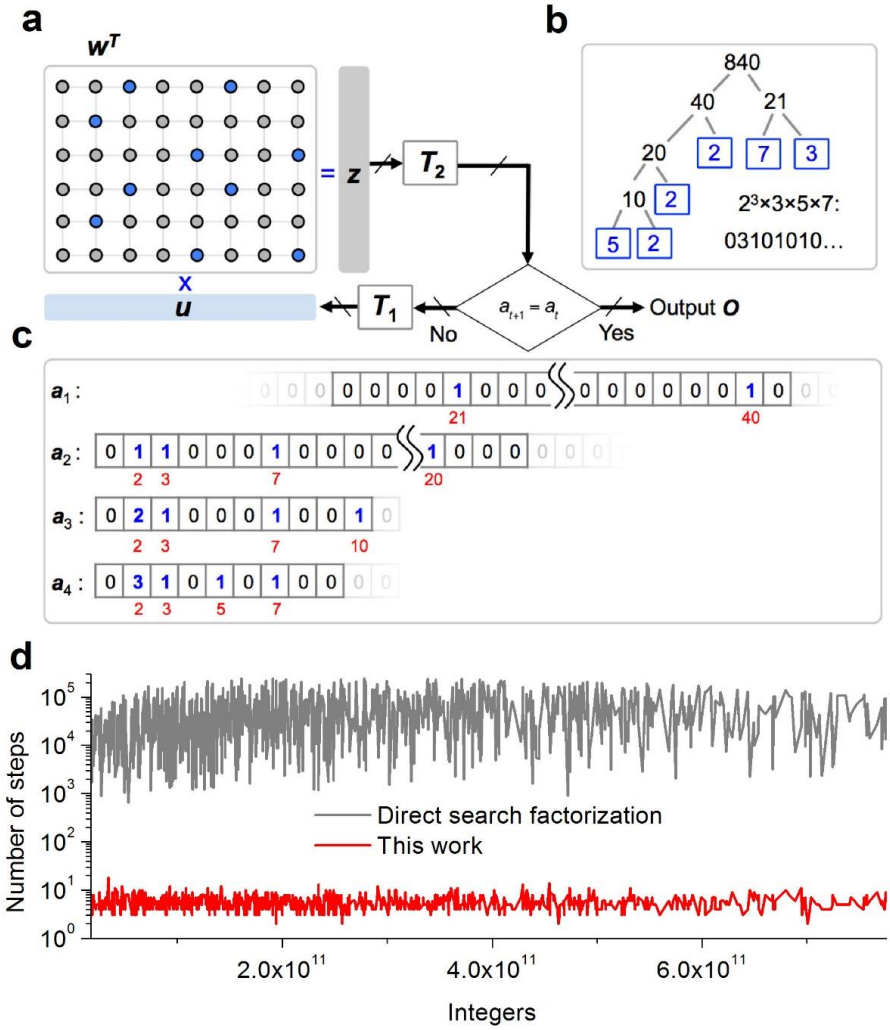


Figure 3.9. Prime factorization. (a) Memory ($w^T \in \mathbb{Z}^{2M\times M^2H}$) based iterative and parallel search for prime factors. Given an input vector u standing for a certain number n, the matrix multiplication $w^{\mathrm{T}}u$ outputs vector z ($z \in \mathbb{Z}^{2M}$; $z[i] \in \{0,1\}$) that reveals one pair of its factors—except 1 and itself—$z[1{:}M]$ and $z[M{+}1{:}2M]$ whose product yields n. Operator T_2 adds these two one-hot vectors, resulting a_t ($a_t \in \mathbb{Z}^M$). The iteration terminates upon no further change in a other than $a[1]$. Otherwise, operator T_1 transforms a_t to u, and the next cycle continues. (b) Prime factorization of 840 = 2^3×3×5×7 with a matrix

w^{T} (M = 100, H = 30). The first iterative step outputs a_1 in (c); the address of each element indicates a factor, e.g. the 21st element, a[21], means a factor of 21, and the element value its exponent. Only a_1[21] and a_1[40] in a_1 except a_1[1] are nonzero, indicating 21×40. The second iteration outputs a_2 whose nonzero elements are a_2[2], a_2[3], a_2[7], and a_2[20] (= 1, 1, 1, and 1, respectively), implying 2^2×10×21. The third iteration respectively sets a_3[2], a_3[3], a_3[7], and a_3[10] to 2, 1, 1, and 1, i.e. 2^2×3×7×10. The forth iteration sets a_3[2], a_3[3], a_3[5], and a_3[7] to 3, 1, 1, and 1, i.e. 2^3×3×5×7 and an additional iteration does not alter other elements than a[1] such that the prime factorization is completed. (d) The number of factorization steps until prime factors for the integers (1.62884×10^{10} – 7.75541294×10^{11}). The results are compared with the direct search factorization.

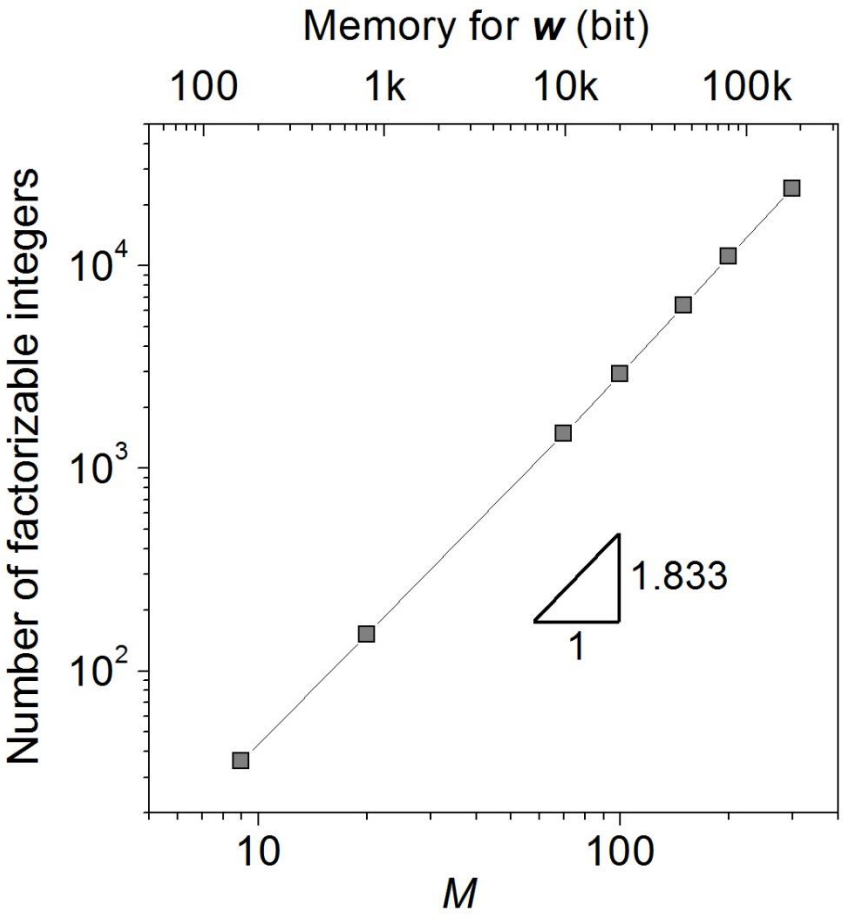


Figure 3.10. Prime factorization capacity. The number of integers factorizable using the proposed algorithm with the size *M* of a trained multiplication table and the memory for matrix *w*.

3.5. Discussion

The MCHL algorithm employs the population representation of output neurons; the population is partitioned as a consequence of bucket allocation for each label. This notion is reminiscent of 'concept cells' [16]-[18]. They fire only to specific inputs that point to the same concept even with different stimulus modalities [17]. Likewise, the 10 populations in Fig. 3.4(a) may be equivalent to concept cells, each of which represents each digit. Additionally, deploying buckets may support the integration of different stimulus modalities, each of which is directed to the same concept cell throughout different path ways. This bucket can include different neurons at the pinnacles of different pathways, e.g., in an auditory modality, so that these different stimulus modalities can complementarily activate the bucket.

Given that each bucket represents a single concept, a one-hot vector representation is most suitable for the mathematical description of concepts. The proposed multiplication table memorization algorithm therefore lays the foundation of arithmetic in association with perception via memory. All integers (factors and products) in the table are represented by one-hot vectors that are equivalent to concept cells. They may be addressed by not only arithmetic but also external stimuli in different sensory modalities. Arithmetic with the aid of memory may be akin to humans' mental arithmetic, particularly, of simple single-digit arithmetic [33]-[35]. Additionally, this memory-based multiplication may combine arithmetic with sensory modalities, e.g., visual and auditory stimuli. For instance, an agent—endowed with the handwritten digit recognition and aforementioned arithmetic capabilities—can recognize

handwritten digits (through a visual modality) and multiply them.

The MCHL algorithm offers a solution to online learning given that the algorithm enables *ad hoc* updates on a weight matrix accommodated by a random access memory (RAM) without pre-calculating the weight matrix. This approach, therefore, provides a workaround for the matrix calculation overhead that is a challenge when addressing representations with enormous features. Additionally, the ternary (−1, 0, 1) weight elements—each of which merely needs 2 bits as shown in Fig. 3.3(b)—significantly improve the areal density of the matrix mapped onto a RAM array in support of density- as well as the energy-wise efficiency of training. A CBA of resistance-based memory is perhaps most suitable for the MCHL algorithm, leveraging its capability of efficient MAC operation [5], [9], [36]. Given the stochasticity in resistance switching (particularly, on- and off-switching voltages [37], [38]) in nature, the probabilistic weight transition may be achieved by controlling driving voltage without RN generation [39]. Additionally, every update simply overwrites the current memory contents in this training scheme in that the past weight matrix no longer needs to be kept given the Markov chain nature, which also alleviates large memory needs.

A rise in handwritten digit recognition accuracy by approximately 2% was achieved by endowing each unit with 11 levels, $\boldsymbol{w}[i,j] \in \{-5, -4, \ldots, 4, 5\}$ as plotted in Fig. 3.11. The network includes no HL. This implies that the ternary weight limits the recognition accuracy below a benchmark accuracy of approximately 92% acquired from an MLP (with real-valued weight and no HL) trained using a backpropagation algorithm. Such 11 levels require five

conductance levels of each resistance-based memory. Fortunately, there are several resistance-based memory systems that exhibit multilevel operations [40]-[42].

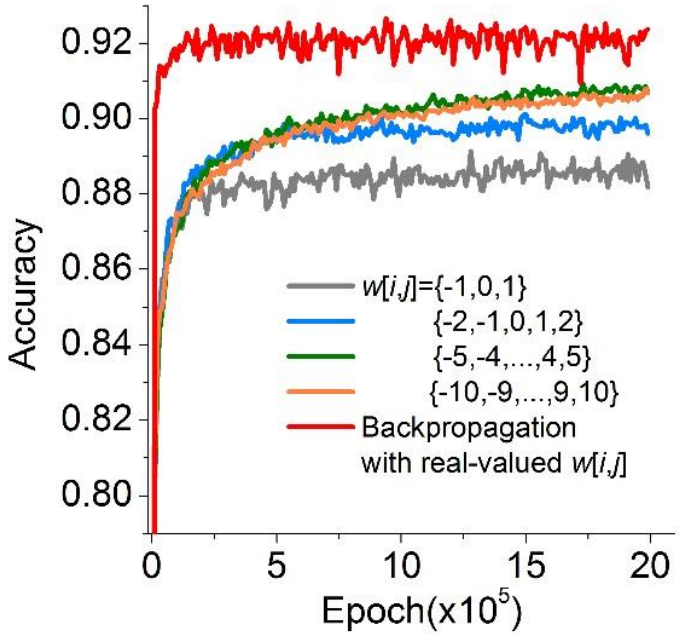


Figure 3.11. Effect of multinary synaptic weight. Improvement of handwritten digit recognition accuracy with multinary synaptic weight. The trained network is a single-layer network (H = 100). A benchmark is a single-layer perceptron with real-valued weight, which was trained with a backpropagation algorithm.

3.6. Appendix

3.6.1. Derivation of stochastic activity of a neuron

Given the network energy in (2), the joint probability distribution of the state $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ is described distribution of the state $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ is described as $P(\boldsymbol{u}_1, \boldsymbol{u}_2) = \mathrm{e}^{-E(\boldsymbol{u}_1, \boldsymbol{u}_2)/\tau}/Z$, where Z is the partition function of the network, $Z = \sum_{j=1}^{M}\sum_{i=1}^{N}\mathrm{e}^{-E(u_1[j], u_2[i])/\tau}$. Consequently, the conditional probability distribution of $\boldsymbol{u}_2$ given $\boldsymbol{u}_1$ is

$$P(\boldsymbol{u}_2|\boldsymbol{u}_1) = \frac{\mathrm{e}^{\sum_{i=1}^{N}\left(-a[i]u_2[i]-\sum_{j=1}^{M}w[i,j]u_1[j]+2\sum_{j=1}^{M}u_2[i]w[i,j]u_1[j]\right)/\tau}}{\prod_{i=1}^{N}\sum_{u_2[i]\in\{0,\,1\}}\mathrm{e}^{\left(-a[i]u_2[i]-\sum_{j=1}^{M}w[i,j]u_1[j]+2\sum_{j=1}^{M}u_2[i]w[i,j]u_1[j]\right)/\tau}}$$

$$= \prod_{i=1}^{N}\frac{\mathrm{e}^{\left(-a[i]u_2[i]+2\sum_{j=1}^{M}u_2[i]w[i,j]u_1[j]\right)/\tau}}{1+\mathrm{e}^{\left(-a[i]+2\sum_{j=1}^{M}w[i,j]u_1[j]\right)/\tau}}. \tag{10}$$

$P(\boldsymbol{u}_2|\boldsymbol{u}_1) = \prod_{i=1}^{N}P(u_2[i]|\boldsymbol{u}_1)$ such that $u_2[i]$'s are independent of each other owing to the lack of recurrent connection. Therefore, the following equation holds:

$$P(u_2[i] = 1|\boldsymbol{u}_1) = \frac{\mathrm{e}^{\left(-a[i]+2\sum_{j=1}^{M}w[i,j]u_1[j]\right)/\tau}}{1+\mathrm{e}^{\left(-a[i]+2\sum_{j=1}^{M}w[i,j]u_1[j]\right)/\tau}}. \tag{11}$$

Introducing $z[i]$ $\left(=\sum_{j=1}^{M}w[i,j]u_1[j]\right)$ simplifies (11) to

$$P(u_2[i] = 1|z[i]) = \frac{\mathrm{e}^{(-a[i]+2z[i])/\tau}}{1+\mathrm{e}^{(-a[i]+2z[i])/\tau}} = \frac{1}{1+\mathrm{e}^{(a[i]-2z[i])/\tau}}, \tag{12}$$

which is equal to the directed graphical model in Fig. 3.1(a).

3.6.2. Calculation of update probability

The update conditions and corresponding probability P can readily be incorporated into the following equation (when $v[i] \neq 0$):

$$P(\Delta w[i,j] = v[i] | w_t[i,j], u_1[j], v[i], u_2[i])$$

$$= \frac{u_1[j]v[i]\left[P_+^0(1-u_2[i])(v[i]+1)+P_-^0 u_2[i](v[i]-1)\right]}{2\left[1+e^{k\left(w_t[i,j]v[i]-w_0\right)}\right]}, \quad (13)$$

where P_+^0 and P_-^0 are expressed as $P(\Delta w[i,j] = 1 | w_t[i,j] \neq 1, u_1[j] = 1, v[i] = 1, u_2[i] = 0)$ and $P(\Delta w[i,j] = -1 | w_t[i,j] \neq -1, u_1[j] = 1, v[i] = -1, u_2[i] = 1)$, respectively. k and w_0 dictate the exponential function in the denominator, which are set to 100 and 0.5 through the entire simulation. Note that when $v[i] = 0$ no update on $w[i,j]$ is allowed, i.e. $P(\Delta w[i,j] = 0 |\ v[i] = 0) = 1$.

In practical computation, the stochastic variable $u_2[i]$ with the probability in (3) is acquired with the aid of a single RN before applying (13) to the $w[i,j]$ update that needs another RN. Fortunately, $u_2[i]$ can be ruled out among the conditions in (13) as follows:

$$P(\Delta w[i,j] = v[i] | w_t[i,j], u_1[j], v[i])$$

$$= P(\Delta w[i,j] = v[i] | w_t[i,j], u_1[j], v[i], u_2[i] = 1) \times P(u_2[i] = 1)$$

$$+P(\Delta w[i,j] = v[i] | w_t[i,j], u_1[j], v[i], u_2[i] = 0) \times P(u_2[i] = 0)$$

$$= \frac{u_1[j]v[i]\left[P_+^0(v[i]+1)+P_-^0(v[i]-1)\right]}{2\left[1+e^{k\left(w_t[i,j]v[i]-w_0\right)}\right]\left[1+e^{-(2z[i]-a[i])/\tau}\right]}. \quad (14)$$

Each update of $w[i,j]$, therefore, needs a single RN, rendering the computation more efficient.

3.6.3. Properties of Markov chain in MCHL

As shown in (9), the transition probability varies over the elements of $\boldsymbol{w}$ every training epoch so that the MCHL algorithm is of non-homogeneous Markov chains. The transition matrix for $w[i,j]$ at the nth epoch is given by

$$\boldsymbol{T}_n^{i,j} = \begin{bmatrix} p_n^{-1,-1} & p_n^{-1,0} & p_n^{-1,1} \\ p_n^{0,-1} & p_n^{0,0} & p_n^{0,1} \\ p_n^{1,-1} & p_n^{1,0} & p_n^{1,1} \end{bmatrix}$$

where the superscript of $p_n^{x,\,y}$ denotes the transition of $w[i,j]$ from x to y. As such, the transition matrix $\boldsymbol{T}_n^{i,j}$ differs for epochs with different $v[i]$ as follows:

$$\begin{cases} \begin{bmatrix} 1 & 0 & 0 \\ P^- & 1-P^- & 0 \\ 0 & P^- & 1-P^- \end{bmatrix} \text{ when } v[i] = -1 \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ when } v[i] = -0 \\ \begin{bmatrix} 1-P^+ & P^+ & 0 \\ 0 & 1-P^+ & P^+ \\ 0 & 0 & 1 \end{bmatrix} \text{ when } v[i] = 1 \end{cases} \tag{16}$$

where $P^- = P_-^0 u_1[j] u_2[i]$, and $P^+ = P_+^0 u_1[j] u_2[i]$. State transition diagrams of these three cases are depicted in Fig. 3.12(a). Although all individual chains notably lack ergodicity, the inhomogeneous Markov chain alternating a transition matrix among these three matrices for each epoch may meet ergodicity. Therefore, ergodicity as an important property of the Markov chain is worth checking.

To this end, matrix $\boldsymbol{H}_{n,m}$ is defined as $\boldsymbol{H}_{n,m} = \prod_{k=n+1}^{n+m} \boldsymbol{T}_k^{i,j}$. Thus, Hn,m is a single transition matrix equivalent to m successive transitions from the (n+1)th to the (n+m)th epoch. Inhomogeneous Markov chains are known to be ergodic if $\left|\boldsymbol{H}_{n,m}[x,y] - \boldsymbol{H}_{n,m}[x',y]\right| \to 0$ as $m \to 0$ for any n, x, x', and y [43]. That is,

an ergodic inhomogeneous Markov chain has identical elements in each column of $\boldsymbol{H}_{n,m}$. For the MCHL algorithm, $\boldsymbol{H}_{n,m}$ is a 3×3 matrix. During the whole training phase, a training image for each epoch appears at random so that one of the three transition matrices is chosen at random. An ergodic Markov chain thus meets the aforementioned condition irrespective of n. Here n is set to zero—ergodicity is evaluated from the first epoch. We define non-ergodicity factor *NE* as

$$NE = \sum_{x,\ x',\ y} \left| \boldsymbol{H}_{n,m}[x, y] - \boldsymbol{H}_{n,m}[x', y] \right|, \tag{17}$$

which decreases to zero with an increase in m if ergodic. The maximum *NE* is 6. We identified *NE* for randomly sampled 100 elements of $\boldsymbol{w}$ in due course during training with the MNIST dataset (see Fig. 3.12(b)). The figure explains a wide range of non-ergodicity in that several trajectories ensure ergodicity, several ones decay at low rates, and the rest remain in the initial state. Such non-ergodicity is of the elements that were barely updated because $u_1[j] = 0$ throughout the entire training phase—background pixels. This is identified by Fig. 3.12(c) displaying the 100 final *NE* values (after 2×106 epochs) with the frequency of non-zero $u_1[j]$ during the training phase. Notably, the elements of low frequencies are given high *NE* values. This is because such elements mostly receive zero input, i.e., $u_1[j] = 0$, and thus their transition matrices in (16) are mostly identity matrices irrespective of $v[i]$. The identity matrix as a transition matrix results in a non-ergodic Markov chain as illustrated in the middle panel of Fig. 3.12(a). Stationary distribution is also of concern of the inhomogeneous Markov chain. To this end, we monitored the numberof elements $w[i, j]$ filled with each of −1, 0, and 1 every MNIST dataset training epoch as plotted in Fig.

3.12(d). The data show asymptotic convergence toward the stationary probability distribution over $w[i, j] = -1$, 0, and 1.

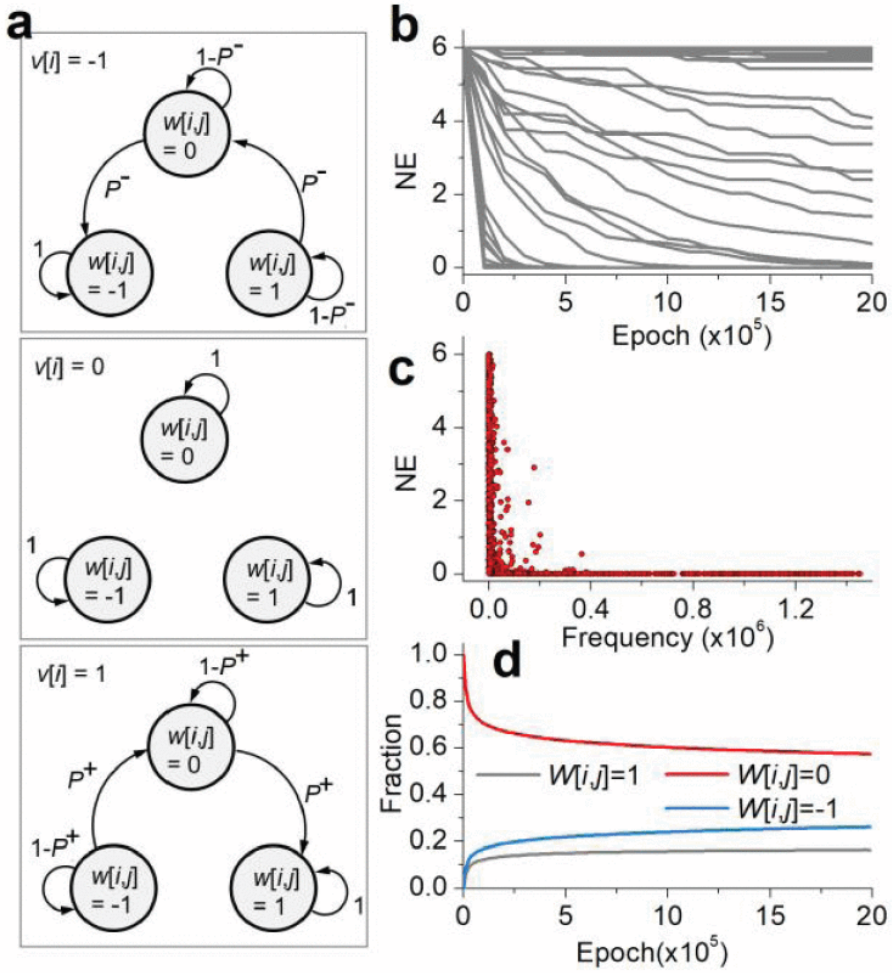


Figure 3.12. (a) State transition diagram for a weight element given three different *v*[i] values. (b) *NE* change (for 100 weight elements randomly sampled) monitored when training a network with the MNIST dataset. (c) The 100 final *NE* values plotted with respect to the frequency of non-zero input during the training phase. (d) Probability distribution over $w[i, j] = 1, 0, 1$ with training epoch.

3.6.4. Effect of update probability and temperature parameter on training

Parameters P_+^0, P_-^0, and τ considerably affect training speed and recognition accuracy. To identify the effect, a network without HL was trained with three different P_+^0 ($=P_-^0$) values (0.01, 0.1, and 1) and τ fixed to 1. The MNIST dataset was used in the training. The results are plotted in Fig. 3.13(a), ensuring their considerable effect on training speed in that the larger P_+^0 ($=P_-^0$) the sooner the recognition accuracy is saturated. Additionally, a P_+^0 of 1 keeps the accuracy fairly lower than the other values. The effect of temperature parameter τ on training was also identified by varying τ (0.1, 1, and 10) with P_+^0 ($=P_-^0$) fixed to 0.1. Fig. 3.13(b) notably indicates the lower accuracy achieved with a τ of 10 than the others. We chose the parameter values with regard to a tradeoff between learning speed and accuracy. When training with the MNIST dataset, P_+^0 ($=P_-^0$) and τ were set to 0.1 and 1, respectively, regarding the tradeoff. The same tendency holds for the CIFAR-10 dataset. Yet, the tradeoff in detail slightly differs so that we set P_+^0 ($=P_-^0$) to 0.01 while setting τ to the same value (1).

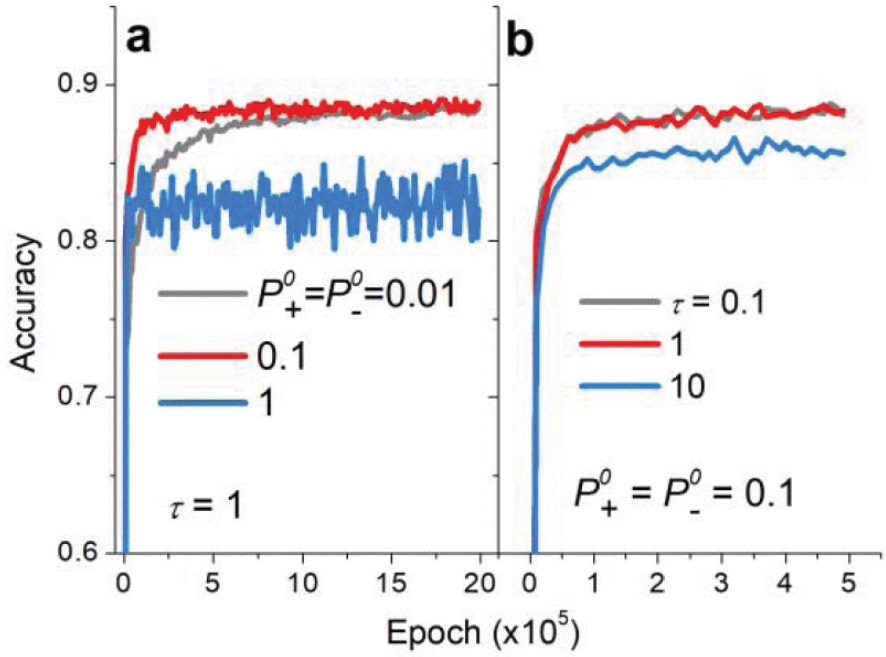


Figure 3.13. Effect of (a) update probability and (b) temperature parameter on training.

3.6.5. Handwritten digit recognition

For the entire datasets, each feature value was rescaled to the range 0 – 1. A chosen input dataset (28 × 28 pixels each of which has an 8-bit value) was converted to an input vector $\boldsymbol{u}_1$ $\left(\in \mathbb{R}^{784}; 0 \leq u_1[i] \leq 1\right)$. A write vector $\boldsymbol{v}$ $(\in \mathbb{Z}^{LH};\ v[i] \in \{-1, 0, 1\})$ was then generated with regard to the desired label of the chosen digit and RN r ($1 \leq r \leq H$). L and H are the number of total labels (here 10) and bucket size, respectively. A bucket of H elements is assigned to each label in the $\boldsymbol{v}$ vector so that $\boldsymbol{v}$ is a $10H$-long vector as illustrated in Fig. 3.4(a). Accordingly, the matrix $\boldsymbol{w}$ is partitioned into 10 sub-matrices. One of the H elements (rth element) in the bucket of the correct label is chosen at random and set to 1, the rth elements in the other buckets (9 in total) to -1, and the rest elements [$10(H - 1)$ in total] to 0. Therefore, in the matrix $\boldsymbol{w}$, the elements in only one row (rth row in the partition for the correct label) are potentially subject to potentiation, those in the 9 rows to depression (rth rows in the partitions for the incorrect labels), and the rest are invariant. The update is therefore sparse.

The weight matrices were initially filled with zeros. The update direction and probability were determined by (14). Each *ad hoc* update needs total $784LH$ RNs (one for each $w[i, j]$). The protocol was repeated for the next epoch with a randomly chosen digit. For accuracy evaluation, a vector $\boldsymbol{z}$ (= $\boldsymbol{w}\boldsymbol{u}_1$) was calculated after every *ad hoc* update and fed into the output neurons that are also partitioned according to the bucket configuration in the write vector and weight matrix. Note that this accuracy evaluation no longer needs stochastic neurons since their probabilistic behaviour rather limits the accuracy. Thus,

they are switched to sigmoid deterministic neurons only for accuracy evaluation, which follows $u_2[i] = \left[1 + \mathrm{e}^{-2z[i]/\tau}\right]^{-1}$. Finally, the output from each label n ($O[n]$) is evaluated. The maximum component of the output vector designates the estimated label for a given input. The recognition accuracy was evaluated with regard to agreement between the desired and estimated labels. The sequence of the MCHL algorithm application is elaborated in Table 3.2.

A network with a hidden layer is trained in a greedy layer-wise manner as for deep belief networks [25]. $\boldsymbol{w}_1$ in Fig. 3.4(a) was first fully trained following the protocol above. Subsequently, $\boldsymbol{w}_2$ was subject to training with input vector $\boldsymbol{u}_2$ ($\in \mathbb{Z}^{LH_1}$; $u_2[i] \in \{0, 1\}$) that is the output from the LH_1 hidden deterministic neurons taking $\boldsymbol{z}_1$ as input. The write vector $\boldsymbol{v}_2$ was chosen applying the same protocol as $\boldsymbol{w}_1$ training. Accuracy evaluation was conducted with deterministic sigmoid output neurons in line with the network without HL.

Table 3.2. MCHL algorithm for handwritten digit classification

Pre-arrangement of memory: Load the bucket of each label in write vector $\boldsymbol{v}$ with H elements. Matrix $\boldsymbol{w}$ partitioned accordingly
Update: update the matrix $\boldsymbol{w}$ given each input $\boldsymbol{u}_1$ and write vector $\boldsymbol{v}$
1. Write vector $\boldsymbol{v}$ generation: $\boldsymbol{v} \in \mathbb{Z}^N$; $N = LH$. L is the number of total labels
For a given input $\boldsymbol{u}$ and its label l, generate an RN r ($1 \leq r \leq H$)
$v[i] = 1$ for $i = l \cdot H + r$ -1 for $i = j \cdot H + r; j \neq l$ 0 otherwise
2. Evaluation of z: $z = \boldsymbol{w}\boldsymbol{u}_1$ given $\boldsymbol{w}$ and $\boldsymbol{u}_1$
3. Update of each component: updating $w[i, j]$ at P in (4)
Repeat

3.6.6. MCHL accelerator in detail

A block diagram of the MCHL accelerator (Virtex-7 XC7VX485T) is depicted in Fig. 3.14. The accelerator employs parallel structure such that L partitions, e.g. one indexed Partition 1 in Fig. 3.14, are deployed and operate in parallel. A sub-matrix $w_1[(n\text{-}1)H_1\text{+}1\text{:}nH_1,\cdot]$ for the nth label is accommodated in an SRAM array in Partition n, e.g. $w_1[1\text{:}H_1,\cdot]$ in Partition 1 as in Fig. 3.14. The entire M entries in each row of the SRAM array are simultaneously accessed at a time (one clock cycle).

For each training epoch (TRAIN=1 in Fig. 3.14), a random number generator RNG_1 produces a pseudo-random number r ($1 \le r \le H_1$), and accordingly the row subject to update in the sub-matrix in Partition n is chosen (see Appendix C). Note that such a pseudo-random number is generated using a linear feedback shift register.

The accessed row $w_1[(n\text{-}1)H_1\text{+}r,\cdot]$ is then multiplied by the input vector $\boldsymbol{u}_1$ to produce $z[(n\text{-}1)H_1\text{+}r]$ according to (1) (see the red-shaded box in Fig. 3.14 for n=1). Subsequently, the activation function module computes the deterministic neuron activity $a_2[(n\text{-}1)H_1\text{+}r]$ in the range 0-255 from $z[(n\text{-}1)H_1\text{+}r]$ using (7). For simplicity, this module approximates the sigmoid function in (7) to a linear function with a particular slope (matching that of (7) at $z = 0$) within a certain z window and zero otherwise. $u_2[(n\text{-}1)H_1\text{+}r]$ is then evaluated by comparing $a_2[(n\text{-}1)H_1\text{+}r]$ with a random number (0 – 255) from RNG_2. The $w_1[(n\text{-}1)H_1\text{+}r,\cdot]$, $\boldsymbol{u}_1$, $u_2[(n\text{-}1)H_1\text{+}r]$, and $v[(n\text{-}1)H_1\text{+}r]$ (generated for each partition using (8)) are then passed to the "Δw module" (blue-shaded box in Fig. 3.14 for n=1) that determines a Δw for each entry of $w_1[(n\text{-}1)H_1\text{+}r,\cdot]$ using the update

probability in (9) in parallel. This process is executed in a single clock cycle. The partition-wise parallel structure of the MCHL accelerator enables an update on $w_1[(n\text{-}1)H_1+r,\cdot]$ for all relevant partitions in parallel in a single clock cycle.

The same holds for an update on $\boldsymbol{w}_2$ except that the deterministic activity vector $\boldsymbol{a}_2$ given the fully trained $\boldsymbol{w}_1$ matrix should be acquired beforehand. The $\boldsymbol{a}_2$ vector is distributed over partitions such that $a_2[(n\text{-}1)H_1+1{:}nH_1]$ is stored in the serial-in-parallel-out (SIPO) buffer of Partition n (see Fig. 3.14 for n=1). Given the partition-wise parallel structure, the evaluation of $\boldsymbol{a}_2$ in response to each input data $\boldsymbol{u}_1$ simultaneously takes place over the n partitions so that it takes H_1/f_{clk}. Therefore, each $\boldsymbol{w}_2$-training epoch takes $(H_1+1)/f_{\text{clk}}$.

Likewise, when training a neural network with two HLs, each $\boldsymbol{w}_3$-training epoch consumes $(H_2+1)/f_{\text{clk}}$. For a neural network including n ($\geq$1) HLs, and thus n+1 weight matrices ($\boldsymbol{w}_1$, …, $\boldsymbol{w}_{n+1}$), the total (intrinsic) training runtime is given by $\sum_{i=2}^{n+1}(H_{i-1}+1)E_i/f_{\text{clk}} + E_1/f_{\text{clk}}$, where E_i denotes the total number of epochs for training the matrix $\boldsymbol{w}_i$.

Inference needs to evaluate the deterministic activity for all $(H_1+H_2)L$ neurons ($\boldsymbol{a}_2$ and $\boldsymbol{a}_3$) in the network using (1) and (7). For a given input digit ($\boldsymbol{u}_1$), $\boldsymbol{a}_2$ is first evaluated as follows. Each row of a sub-matrix $w_1[(n\text{-}1)H_1+r,\cdot]$ is sequentially addressed using an address counter (TRAIN = 0 in Fig. 3.14) in descending order and multiplied by $\boldsymbol{u}_1$, resulting in $a_2[(n\text{-}1)H_1+1{:}nH_1]$ through the red-shaded and activation function modules in Fig. 3.14. The array is The finally evaluated $\boldsymbol{a}_2$ vector for this partition, i.e. $a_2[(n\text{-}1)H+1{:}nH]$ where n = 0, is stored in a serial-in-parallel-out (SIPO) buffer (see Fig. 3.14). Given the partition-wise parallel structure, this process simultaneously takes place for the

other partitions so that it takes H_1/f_{clk} to evaluate the deterministic activities $\boldsymbol{a}_2$ of the hidden neurons in response to input data $\boldsymbol{u}_1$.

The same process holds for the $\boldsymbol{a}_3$ evaluation following the $\boldsymbol{a}_2$ evaluation. Thus, the time-consumption is H_2/f_{clk}. The only difference is that $\boldsymbol{a}_2$ in the SIPO buffers distributed over the partitions is taken as the input.

All elements of $a_3[(n\text{-}1)H_2\text{+}1\text{:}nH_2]$ in Partition n (label n) are added up in the accumulator module (see Fig. 3.14 for n = 1), resulting in $O[n]$ for Partition n (label n). The comparator module in Fig 3.11 compares the O's and consequently provides the index of the highest O value, which corresponds to the inferred label. This comparison is performed in a sequential manner, i.e. $O[0]$ is first compared with $O[1]$, the winner is then compared with $O[2]$, and so forth. The priority encoder finally encodes the address of the "final" winner. Note that the comparison is performed in parallel with the $\boldsymbol{a}_3$ evaluation process so that it does not consume additional time. Consequently, inference for each input digit consumes $(H_1+H_2)/f_{clk}$ in total.

Therefore, inference (intrinsic) runtime for each input through a network with n ($\geq$1) HLs (n+1 weight matrices) is $\sum_{i=1}^{n+1} H_i/f_{clk}$.

Practically, both inference and training rates are dominantly dictated by the rate of input data transfer from the computer to the MCHL accelerator. Each handwritten digit image was 2 bits/pixel (downsized from 8 bits/pixel in the original MNIST dataset), and thus 1,568 bits (2×28×28) per image. The MCHL accelerator was interfaced with the computer through 16 general-purpose input-output (GPIO) lines, yielding a data transfer bandwidth of ca. 300 kb/s. Therefore, transferring one image to the accelerator consumes approximately

5.2 ms, outweighing the intrinsic training and inference runtimes. We did not count this delay in data transfer as training and inference runtimes because the delay is not an intrinsic characteristic of the MCHL algorithm.

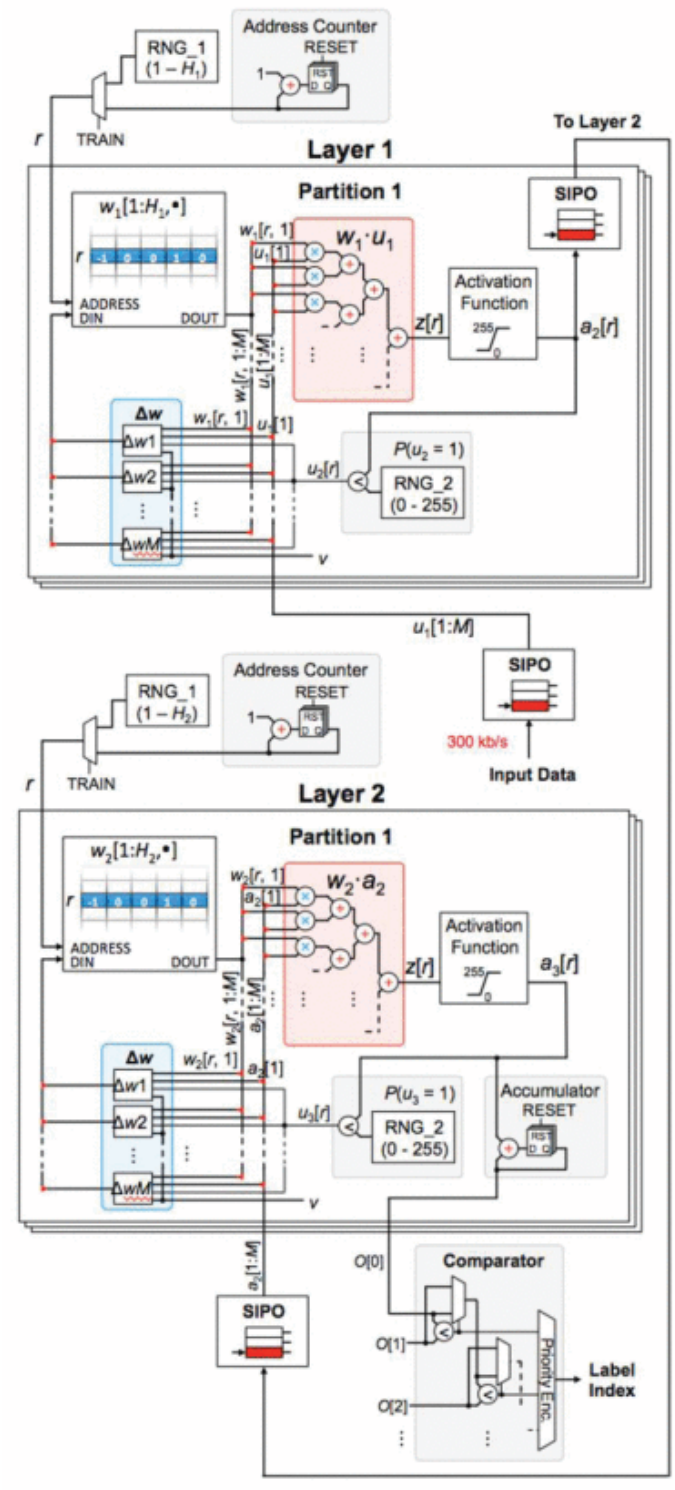


Figure 3.14. Block diagram of the MCHL accelerator.

3.6.7. Multiplication table memorization

Training was fully deterministic in that the output neurons were frozen and the update no longer required RNs. Integers ($\leq M$) were expressed as one-hot vectors of M elements; a pair of factors ($\leq M$) were put together to give an input vector $\boldsymbol{u}_1$ $\left(\in \mathbb{Z}^{2M};\ u_1[i] \in \{0, 1\}\right)$. The product of the factors serves as a label among M^2 labels, each of which has a bucket of H elements. Therefore, a write vector $\boldsymbol{v}$ has M^2H elements in total $\left(\boldsymbol{v} \in \mathbb{Z}^{M^2H};\ v[i] \in \{0, 1\}\right)$. For factors of a and b ($a \times b = c$), the hth element in the cth label, i.e. $v[(c\text{-}1), H+h]$, is set to the only one in the write vector. h is determined in the order of training; the first pair of factors resulting in a particular label during training takes $h = 1$ in the corresponding bucket. Thus, allocating h for each multiplication depends on the entire training sequence over the $M \times M$ multiplication table. The weight matrix $\boldsymbol{w}$ $\left(\in \mathbb{Z}^{M^2H \times 2M};\ w[i, j] \in \{0, 1\}\right)$ was trained in an ascending order of n in the n-times table ($n \times$) from 1 to M, and within the n-times table ($n \times m$), m was also taken in ascending order: 1×1, 1×2, …, $1 \times M$, 2×1, 2×2, …, $2 \times M$, … $M \times 1$, $M \times 2$, … $M \times M$. Upon training completion, final h ($\leq H$) for label i (i.e. h_i) is acquired, which defines vector $\boldsymbol{A}$ $\left(\in \mathbb{Z}^{M^2};\ A[i] = h_i\right)$. In fact, $A[i]$ reveals the number of multiplications producing label i, for instance, $A[6] = 4$ given that 1×6, 2×3, 3×2, and 6×1 result in 6 (see Fig. 3.8(a)). Notably, this number is identical to the number of factors for a given label: 1, 2, 3, and 6 for 6. The sequence of the MCHL algorithm application is tabulated in Table 3.3.

$z[i]$ in $\boldsymbol{z}$ ($=\boldsymbol{w}\boldsymbol{u}_1$) was integrated over elements in the bucket of each label,

which was subsequently fed into an output sigmoid neuron, resulting in output vector $\boldsymbol{O}$ as illustrated in Fig. 3.8(a).

Table 3.3. MCHL algorithm for multiplication table memorization

Pre-arrangement of memory: Load the bucket of each label in write vector $\boldsymbol{v}$ with H elements. Matrix $\boldsymbol{w}$ partitioned accordingly. $A[l] = 1$ for all l's ($1 \leq l \leq L$). L is the number of total labels (products).
Update: update matrix $\boldsymbol{w}$ given each input $\boldsymbol{u}$ (a pair of one-hot vectors) and write vector $\boldsymbol{v}$
1. Write vector $\boldsymbol{v}$ generation: $\boldsymbol{v} \in \mathbb{Z}^N$; $N = LH$.
For a given input $\boldsymbol{u}_1$ and its label l,
$v[i] = 1$ for $i = (l - 1) \cdot H + A[l]$ 0 otherwise
2. Update of each component: updating $w[i, j]$ at P in (4)
3. $A[l] = A[l] + 1$
Repeat

3.6.8. Prime factorization

As such, the aliquot parts of number n are in parallel retrieved using the transpose of $\boldsymbol{w}$ $\left[\boldsymbol{w}^{\mathrm{T}} \in \mathbb{Z}^{2M\times M^2H}\right]$ memorizing the $M\times M$ multiplication table and input vector $\boldsymbol{u}$ $\left(\in \mathbb{Z}^{M^2H};\ u[i] \in \{0, 1\}\right)$ whose nth bucket is filled with H 1's—insofar as n's largest aliquot part is not larger than M. However, for prime factorization of n, aliquot parts other than 1 and itself (if they exist) are of concern, so that it is desirable to avoid retrieving $1 \times n$ and $n \times 1$. With the aid of vector $\boldsymbol{A}$, a pair of proper factors can be chosen selectively. As shown in Fig. 3.8(a), for 6 ($M \geq 6$), h = 1, 2, 3, and 4 indicate 1×6, 2×3, 3×2, and 6×1, respectively. For a prime number, e.g. 7, h=1 and 2 indicate 1×7 and 7×1, respectively. Only the kth multiplication is retrieved, $k = \max(A[i] - 1, 1)$ for each label i, e.g. for i = 6 ($M \geq 6$), 3 × 2, and for i = prime number ($M \geq n$), $1 \times n$. Thus, operator $\boldsymbol{T}_1$ is a $M^2H\times M$ matrix:

$$T_1[i,j] = \begin{cases} 1 & \text{if } i = (n-1)H + k \text{ and } j = n \text{ for } n = 1,\ldots, M^2 \\ 0 & \text{otherwise} \end{cases}.$$

For instance, n = 840 (M = 50) is initially represented by vector $\boldsymbol{a}_0$ whose 840th element is the only one while the rest are zero. $\boldsymbol{u}$ ($=\boldsymbol{T}_1\boldsymbol{a}_0$) is subsequently fed into $\boldsymbol{w}^{\mathrm{T}}$, resulting in $\boldsymbol{z}$ ($=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{u}$) in which z[40] = 1 and z[50 + 21] = 1—denoting 40 and 21, respectively. These two vectors are merged through operator $\boldsymbol{T}_2$ into $\boldsymbol{a}_1$ $(\in \mathbb{Z}^M; \boldsymbol{a}_1 = z[1{:}M] + z[M+1{:}2M])$. $\boldsymbol{T}_2$ is, therefore, an $M\times 2M$ matrix:

$$T_2[i,j] = \begin{cases} 1 & \text{if } j = i \text{ for } i = 1,\ldots, M \\ 1 & \text{if } j = i + \mathrm{M} \text{ for } i = 1,\ldots, M \\ 0 & \text{otherwise} \end{cases}.$$

This operation confers 1 on a_1[21] and a_1[40] in $\boldsymbol{a}_1$. The address of each element represents a factor, and the element values its exponent so that the result of the

first factorization is $21^1 \times 40^1$. Insofar as $\boldsymbol{a}_1$ differs from $\boldsymbol{a}_0$, the same cycle is repeated. Note that $a_1[1]$ (exponent of 1) is set to zero because a factor of 1 is redundant in factorization. The following cycle factorizes 21 and 40 in parallel, providing $\boldsymbol{a}_2$ in which $a_2[2] = 1$, $a_2[3] = 1$, $a_2[7] = 1$, and $a_2[20] = 1$, i.e. $2^1 \times 3^1 \times 7^1 \times 20^1$.

3.6.9. Direct search factorization

Integer n is repeatedly divided by a series of divisors (decreasing by one) until zero remainders. The first divisor is $\lfloor\sqrt{n}\rfloor$. If the remainder is nonzero, $\lfloor\sqrt{n}\rfloor - 1$ is taken as the next divisor. With zero remainder, two factors (divisor and quotient) are obtained, and each factor is separately subject to the same factorization as above.

3.7. Bibliography

[1] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, in *IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, 2014, pp. 1701-1708.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Nature*, Article vol. 529, no. 7587, pp. 484-489, 2016.

[4] R. Raina, A. Madhavan, and A. Y. Ng, in *26th Annual International Conference on Machine Learning*, 2009, pp. 873-880.

[5] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, and C. S. Hwang, *Advanced Electronic Materials*, vol. 2, no. 9, p. 1600090, 2016.

[6] J. Y. Seok, S. J. Song, J. H. Yoon, K. J. Yoon, T. H. Park, D. E. Kwon, H. Lim, G. H. Kim, D. S. Jeong, and C. S. Hwang, *Advanced Functional Materials*, vol. 24, no. 34, pp. 5316-5339, 2014.

[7] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, *Nature*, Letter vol. 521, no. 7550, pp. 61-64, 2015.

[8] G. W. Burr, P. Narayanan, R. M. Shelby, S. Sidler, I. Boybat, C. di Nolfo, and Y. Leblebici, in *International Electron Devices Meeting*, 2015, pp. 4.4.1-4.4.4.

[9] L. Gao, P. Y. Chen, and S. Yu, *IEEE Electron Device Letters*, vol. 37, no. 7, pp. 870-873, 2016.

[10] P. Smolensky, in *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, E. R. David, L. M. James, and C. P. R. Group Eds., 1 ed.: MIT Press, 1986, sec. 104290, pp. 194-281.

[11] Y. Freund and D. Haussler, in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann Eds.: Morgan-Kaufmann, 1992, pp. 912-919.

[12] G. E. Hinton, *Neural Computation*, vol. 14, no. 8, pp. 1771-1800, 2002.

[13] G. E. Hinton, in *Neural Networks: Tricks of the Trade: Second Edition*, 2012, pp. 599-619.

[14] S. Nagpal, M. Singh, R. Singh, and M. Vatsa, *IEEE Access*, vol. 3, pp. 3010-3018, 2015.

[15] K. Zhang and X.-W. Chen, *IEEE Access*, vol. 2, pp. 395-403, 2014.

[16] R. Q. Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried, *Nature*, vol. 435, no. 7045, pp. 1102-1107, 2005.

[17] R. Quian Quiroga, A. Kraskov, C. Koch, and I. Fried, *Current Biology*, vol. 19, no. 15, pp. 1308-1313, 2009.

[18] R. Q. Quiroga, *Nature Reviews Neuroscience*, vol. 13, no. 8, pp. 587-597, 2012.

[19] M. Courbariaux, Y. Bengio, and J.-P. David, *Advances in neural information processing systems*, 2015.

[20] C. Baldassi, A. Braunstein, N. Brunel, and R. Zecchina, *Proceedings of the National Academy of Sciences*, vol. 104, no. 26, pp. 11079-11084, 2007.

[21] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, *Machine learning*, vol. 50, no. 1-2, pp. 5-43, 2003.

[22] R. Bellman, *Journal of mathematics and mechanics*, pp. 679-684, 1957.

[23] S. Thrun, in *Advances in neural information processing systems*, 2000, pp. 1064-1070.

[24] Z.-H. Zhou and M.-L. Zhang, in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 2006: MIT Press, pp. 1609-1616.

[25] Q. Wu, M. K. Ng, and Y. Ye, *Knowledge and information systems*, vol. 37, no. 1, pp. 83-104, 2013.

[26] N. Brunel, F. Carusi, and S. Fusi, *Network: Computation in Neural Systems*, vol. 9, no. 1, pp. 123-152, 1998.

[27] G. L. Barrows, in *IEEE International Joint Conference on Neural Networks*, Anchorage, AK, 1998, pp. 525-530.

[28] W. Senn and S. Fusi, *Physical Review E*, vol. 71, no. 6, p. 061907, 2005.

[29] G. E. Hinton, S. Osindero, and Y.-W. Teh, *Neural Computation*, vol. 18, no. 7, pp. 1527-1554, 2006.

[30] T. Gokmen and Y. Vlasov, *Frontiers in Neuroscience*, vol. 10, no. 333, 2016.

[31] K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics: an introduction*, Third ed. Springer Berlin Heidelberg, 1997.

[32] Z. Lin, R. Memisevic, and K. Konda, *arXiv:1511.02580*, 2015.

[33] S. A. Hecht, *Memory & Cognition*, vol. 27, no. 6, pp. 1097-1107, 1999.

[34] J. I. D. Campbell and Q. Xue, *Journal of Experimental Psychology: General*, vol. 130, pp. 299-315, 2001.

[35] D. DeStefano and J. A. LeFevre, *European Journal of Cognitive Psychology*, vol. 16, no. 3, pp. 353-386, 2004.

[36] M. A. Zidan, Y. Jeong, J. H. Shin, C. Du, Z. Zhang, and W. D. Lu, *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 698-710, 2017.

[37] S. Long, X. Lian, T. Ye, C. Cagli, L. Perniola, E. Miranda, M. Liu, and J. Suñé, *IEEE Electron Device Letters*, vol. 34, no. 5, pp. 623-625, 2013.

[38] J. H. Yoon, S. J. Song, I. H. Yoo, J. Y. Seok, K. J. Yoon, D. E. Kwon, T. H. Park, and C. S. Hwang, *Advanced Functional Materials*, vol. 24, no. 32, pp. 5086-5095, 2014.

[39] H. Lim, H.-W. Ahn, V. Kornijcuk, G. Kim, J. Y. Seok, I. Kim, C. S. Hwang, and D. S. Jeong, *Nanoscale*, vol. 8, no. 18, pp. 9629-9640, 2016.

[40] S. Yu, Y. Wu, and H.-S. P. Wong, *Applied Physics Letters*, vol. 98, no. 10, p. 103514, 2011.

[41] D. Ielmini, *IEEE Transactions on Electron Devices*, vol. 58, pp. 4309-4317, 2011.

[42] M. Suri, O. Bichler, D. Querlioz, B. Traore, O. Cueto, L. Perniola, V. Sousa, D. Vuillaume, C. Gamrat, and B. DeSalvo, *Journal of Applied Physics*, vol. 112, no. 5, pp. 054904-10, 2012.

[43] J. Hajnal and M. Bartlett, in *Mathematical Proceedings of the Cambridge Philosophical Society*, 1958, vol. 54, no. 2: Cambridge University Press, pp. 233-246.

4. Combination-encoding content addressable memory

4.1. Introduction

Content-addressable memory (CAM) is a type of memory accessed based on contents instead of memory addresses as opposed to random access memory (RAM) [1]. Upon receiving an input data word to search (search key), CAM simultaneously searches all memory entries for search-key-relevant contents in one clock cycle and returns the addresses of the contents. Therefore, CAM has a significant advantage over RAM in searching speed. Its main application domains include lookup tables (LUTs) in network routers [2]-[7]. The network router decides the forwarding direction of a data packet between networks. The LUT in the network router including the hierarchical addresses is searched for the best route or port for the data packet to be forwarded. Additionally, the CAM storing the LUT is a critical component for digital communications among neurons in neuromorphic hardware [8]. The LUT including the topology of a neural network is searched for the postsynaptic neuron addresses upon the occurrence of an event from a presynaptic neuron. A fast search of the CAM significantly accelerates event routing processes, enabling real-time inference and learning. It also applies to vector-quantization [9], decomposing an input image to a set of vectors, and information retrieval [10], finding the information relevant to the desired information from big data.

CAM is categorized as binary CAM (BCAM) and ternary CAM (TCAM). As the names indicate, each unit cell in BCAM represents either '0' or '1' whereas that in TCAM has an additional 'don't care' (or 'X') state [1]. For instance, '1X1' in TCAM is matched to search keys '101' and '111' because 'X' matches both '0' and '1'. This high flexibility of TCAM is the key to packet forwarding tasks [2]-[7].

Static RAM (SRAM)-based CAM is the most popular form of CAM [1], [11]. The SRAM-based CAM leverages fast searching speed and high compatibility with well-established complementary metal-oxide-semiconductor (CMOS) technologies. Nevertheless, significant disadvantages are its low areal density due to the use of many transistors (≥8) to represent a single bit and high static power consumption due to the leakage current of SRAM [12]-[14]. As alternatives to the SRAM-based CAM, CAMs based on emerging non-volatile memories (NVMs) such as phase-change memory [14-16], magnetic tunnel junction [14], [17]-[19], ferroelectric memory [20], and resistance switch [21-25] have been proposed to date. Such NVM-based CAMs highlight their high data density and zero-static energy consumption due to the non-volatility. They also offer solutions to TCAM by appropriately configuring the non-volatile memory elements [14]-[25].

Among the candidates, resistance switch-based CAM (RCAM) is a front runner; two-transistor two-resistor (2T2R)-based RCAM has been prototyped using a 4 kb resistive RAM (RRAM) [25]. Additionally, RCAM may be realized in a passive crossbar array that highlights its ideal $8F^2$ cell size [21], [23]. Nevertheless, RCAM has a lower content density than RRAM because it uses a

pair of resistance switches as a single bit of content, i.e., 0.5 bit/switch. A further increase in data density needs a new content-encoding framework. To this end, we propose a new type of resistance switch-based CAM, named combination-encoding CAM (CECAM).

Section II outlines the working principle of the CECAM including a search key-encoding algorithm (Section II.A) and its implementation in a digital circuit (Section II.B). Section III explains parallel searches of multiple CECAM domains to realize TCAM with coarse granularity. Reading contents from the CECAM needs to decode them using an appropriate decoding algorithm because the contents in the CECAM are encoded, which is addressed in Section IV. Finally, Section V highlights the general application of the CECAM scheme to various CAM designs.

4.2. Combination-encoding content addressable memory

Fig. 4.1 illustrates a schematic of unit cells of active and passive RCAMs (voltage- and current-reading schemes, respectively). RCAM takes a pair of resistance switches as a single unit of single-bit capacity. Each switch is set to one of the binary states: high resistance state (HRS) and low resistance state (LRS). Specifically, the two switches are complementary; they are in different resistance states. This yields two distinguishable configurations, HRS-LRS (HL) and LRS-HRS (LH), representing one bit of content. For both schemes in Fig. 4.1, LH corresponds to '0' and HL to '1'. Each bit of a search key is represented by voltage signals on complementary search lines (SL and $\overline{\text{SL}}$) such that '0' pulls SL low and $\overline{\text{SL}}$ high while '1' pulls SL high and $\overline{\text{SL}}$ low. When a search bit of

‘0’ is applied to a stored content of ‘0’ (LH), the RCAM units in Figs. 4.1(a) and 4.1(b) notify matching signals on the match lines (black V_{ML} and I_{ML} in Figs. 1c and 1d, respectively). These signals are contrasted with a mismatching case, where a search bit of ‘1’ is applied to the same content ‘0’, resulting in mismatch signals on the match lines indicated by the red V_{ML} and I_{ML} in Figs. 4.1(c) and 4.1(d), respectively. Thus, searching N-bit keys commonly needs $2N$ switches per match line, i.e., 0.5 bit/switch per match line.

In contrast, N-CECAM ($N = 1, 2, 3, \ldots$) uses a chunk of $2N$ switches per match line as a single unit of multi-bit capacity, boosting the memory capacity per switch far beyond a content density of 0.5 bit/switch. Its key difference from RCAM is that the N-CECAM harnesses the large number of possible combinations of $2N$ switches to boost the content density in contrast to RCAM using complementary pairs of switches and search lines to store and search a single bit of contents. We regard a passive array of nonvolatile resistance switches with current reading as a model system of the CECAM. This model system leverages its high memory density and fast content reading. Nevertheless, the sneak current disturbing current read-out processes is a critical downside. However, the CECAM concept is fully compatible with other types of CAM including active arrays of resistance switches freer from the sneak current issue, which will be addressed in Section V.

The N-CECAM consists of a resistance switch array and a search key encoder as illustrated in Fig. 4.2(a). In the array, $2N$ resistance switches are placed at the crossing points between each match (horizontal) line and $2N$ search (vertical) lines. The search key encoder converts an n-bit search key to a $2N$-digit binary

key, and each digit is applied to each of the 2*N* search lines such that '1' and '0' pull the search line high and low, respectively. Assuming *m* switches are in the HRS and the other (2*N*-*m*) switches in the LRS, the minimum current response to a single 2*N*-digit binary key exists only if the key includes *m* 1's and (2*N*-*m*) 0's, and each of the *m* 1's in the key is matched to each of the *m* HRS switches. To maximize the number of possible configurations of 2*N* switches $\binom{2N}{m}$, *m* is set to *N*. Thus, the search key encoder maps an *n*-bit search key to a 2*N*-digit key with *N* 1's and *N* 0's in a bijective manner. When matching *N* 1's in the encoded key to the *N* HRS switches, the current response through the match line is minimal as shown in Fig. 4.2(b). Otherwise, some 1's in the encoded key are inevitably associated with LRS switches, and thus the current response through the match line becomes high, indicating a mismatch. Note that the current response scales with the degree of mismatch, i.e., the number of mismatched bits in the encoded key. The worst mismatch regarding a sensing margin is due to two mismatched bits as depicted in Fig. 4.2(b).

The total number of 2*N*-digit encoded keys is $\binom{2N}{N}$, and so is the number of 2*N* switch configurations per match line. Given the use of *n*-bit search keys, 2^n of $\binom{2N}{N}$ configurations are associated with the total *n*-bit keys, satisfying $2^n \le \binom{2N}{N} < 2^{n+1}$. This inequality yields

$$n = \left\lfloor \log_2 \binom{2N}{N} \right\rfloor \tag{1}$$

where $\lfloor \cdot \rfloor$ denotes a floor function. Therefore, the content density (content bit per switch) is given as $\left\lfloor \log_2 \binom{2N}{N} \right\rfloor / 2N$, which is plotted in Fig. 4.3. Notably,

for N's (>2), the content density exceeds the density of the conventional RCAM designs (0.5 bit/switch) and approaches the density of RRAM (1 bit/switch) asymptotically.

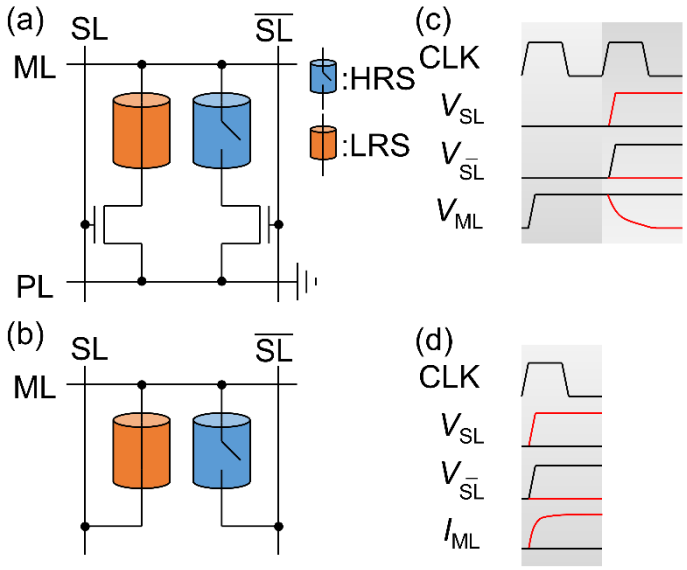


Figure 4.1. Schematic of the conventional RCAM in (a) active and (b) passive crossbar arrays. ML, SL, $\overline{SL}$, and PL denote a match line, search line, complementary search line, and plate line, respectively. A timing diagram for active and passive arrays is illustrated in (c) and (d), respectively. CLK, V_{SL}, $V_{\overline{SL}}$, and V_{ML} denote a clock cycle, search line voltage, complementary search line voltage, and match line voltage, respectively. I_{ML} in (d) means the current through the match line. The red lines in (c) and (d) indicate V_{SL}, $V_{\overline{SL}}$, and the CAM responses when mismatching.

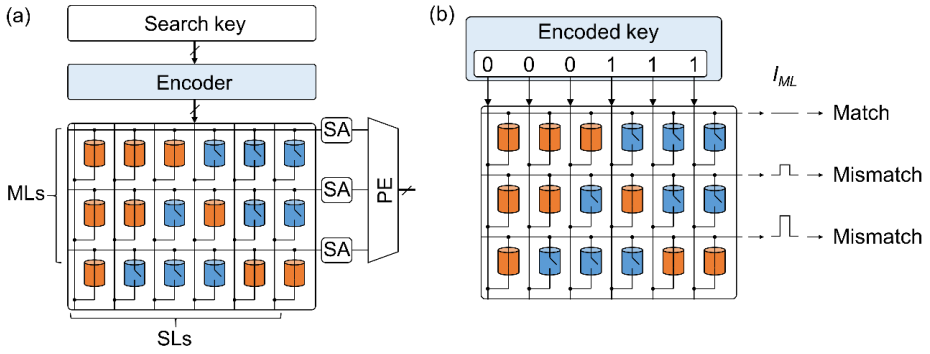


Figure 4.2. (a) Schematic of 3-CECAM (N = 3). A single unit consists of N HRS and N LRS switches. SA and PE mean a sense amplifier and priority encoder, respectively. (b) Current responses to a given encoded key upon a match and mismatches. Matching allows the minimal current response (first row).

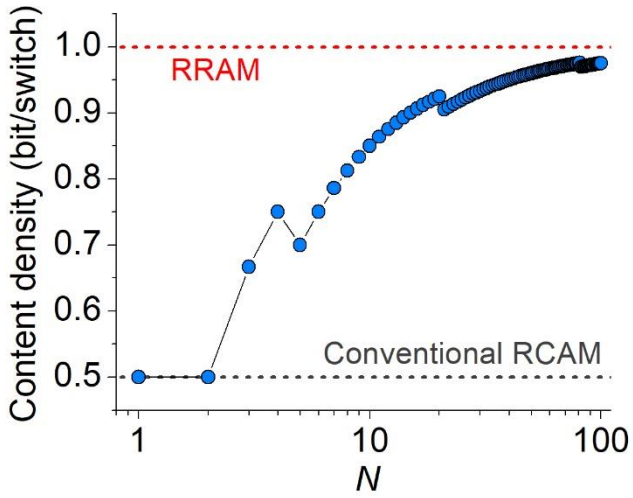


Figure 4.3. Content density of N-CECAM with N in comparison with the conventional RCAM and RRAM. The kinks arise from the floor function in (1).

4.2.1. Algorithm for combination encoding

The key to the *N*-CECAM is the bijective mapping of the total *n*-bit search keys to 2^n 2*N*-digit keys ($2^n \le \binom{2N}{N} \le 2^{n+1}$) using an appropriate encoding function. We propose the encoding function E_N for an *n*-bit search key *a* as follows:

function $E_N(a)$

set *b* **to** 2*N*-digit binary number 0

for $i = 0$ **to** N-1 **do**

if there is *c* satisfying $\binom{c}{N-i} \le a < \binom{c+1}{N-i}$ **then**

set the (*c*+1)th digit of b **to** 1

set *a* **to** $a - \binom{c}{N-i}$

end if

end for

return *b*

end function.

Note that E_N is bijective when $\boldsymbol{A}_t = \left\{0, 1, \ldots, \binom{2N}{N} - 1\right\}$ and $\boldsymbol{B}_t = \{b \mid b$: 2*N*-digit binary numbers with of *N* 1's and *N* 0's$\}$ are taken as the domain and codomain of E_N, respectively (**Theorem 1** in Appendix). Therefore, E_N is also a bijective function for domain $\boldsymbol{A}=\{0, 1, \ldots, 2^n-1\}$ $\left(\subset \boldsymbol{A}_t\right)$ and codomain $\boldsymbol{B}=\{E_N(0), E_N(1), \ldots, E_N(2^n-1)\}$ $\left(\subset \boldsymbol{B}_t\right)$.

Table 4.1 shows the encodings of 4-bit search keys as 16 distinguishable 6-digit binary numbers with three 1's and three 0's using the encoding function E_3 ($N = 3$). The encoded data are subsequently programmed in 2*N* switches such

that a ‘1’ and ‘0’ in the encoded data are written as ‘H’ and ‘L’, respectively. ‘H’ and ‘L’ denote HRS and LRS, respectively. The last configuration ‘HHHHHH’ indicates ‘don’t care’ for TCAM.

Table 4.1. Truth table of encodings of 4-bit integers as resistor configurations

Integer	Configuration	Search key	Integer	Configuration	Search key
0	LLLHHH	000111	9	LHHHLL	011100
1	LLHLHH	001011	10	HLLLHH	100011
2	LLHHLH	001101	11	HLLHLH	100101
3	LLHHHL	001110	12	HLLHHL	100110
4	LHLLHH	010011	13	HLHLLH	101001
5	LHLHLH	010101	14	HLHLHL	101010
6	LHLHHL	010110	15	HLHHLL	101100
7	LHHLLH	011001	0 – 15	HHHHHH	000000
8	LHHLHL	011010			

(N = 3)

4.2.2. Implementation of encoding circuit

A search key is encoded as a $2N$-digit key iteratively, which needs to be implemented in circuitry in a way to reduce a delay in decoding at the cost of memory usage. To this end, the LUT P of $\binom{c}{N-i}$ for $0 \le c \le 2N$ and $0 \le i < N$ is stored in a memory, which is referred to as a combination table. The LUT P is an $N \times (2N+1)$ matrix whose element $P[i, c]$ is $\binom{c}{N-i}$. A main advantage of employing the combination table is that the comparison $\binom{c}{N-i} \le a < \binom{c+1}{N-i}$ in the encoding function E_N can be accelerated considerably by retrieving $\binom{c}{N-i}$ and $\binom{c+1}{N-i}$ from the LUT P rather than evaluating them for every comparison. A block diagram of the encoding circuit including the LUT P is shown in Fig. 4.4(a). When RESET is 1, the encoding circuit receives search key a that is encoded (a_0). Simultaneously, a $2N$-long array b is initialized as $b_0[k] = 0$ for $0 \le k < 2N$, where $b_0[k]$ denotes the $(k+1)$th digit of b_0. We note that RESET is synchronized with clock signals (CLK) to avoid a metastability problem as shown in Fig. 4.4(b). The circuit first addresses the first row of the LUT P, i.e., $P[0, :]$. The NEXT block finds c in $P[0, :]$, satisfying $P[0, c] \le a_0 < P[0, c+1]$, using parallel comparators, resulting in c_0. a_1 is consequently evaluated as $a_1 = a_0 - P[0, c_0]$. b_1 is identical to b_0 except its (c_0+1)th digit that is set to one, $b_1[c_0] = 1$. c_1 is subsequently evaluated for the next row of the LUT P, i.e., $P[1, :]$, as for the first row. This evaluation is repeated for all remaining rows, eventually resulting in a $2N$-digit encoded key b ($=b_N$). Therefore, the delay in encoding is caused by iteratively addressing each row of the LUT P, which

scales with N (Fig. 4.4(b)). Given the relationship in (1), the encoding delay is associated with the bit number of a search key (n) as plotted in Fig. 4.5 (blue line); the delay tends to increase with the bit number. To address the memory overhead for the LUT P, the number of its entries was also evaluated with the bit number of a search key and co-plotted in Fig. 4.5 (red line).

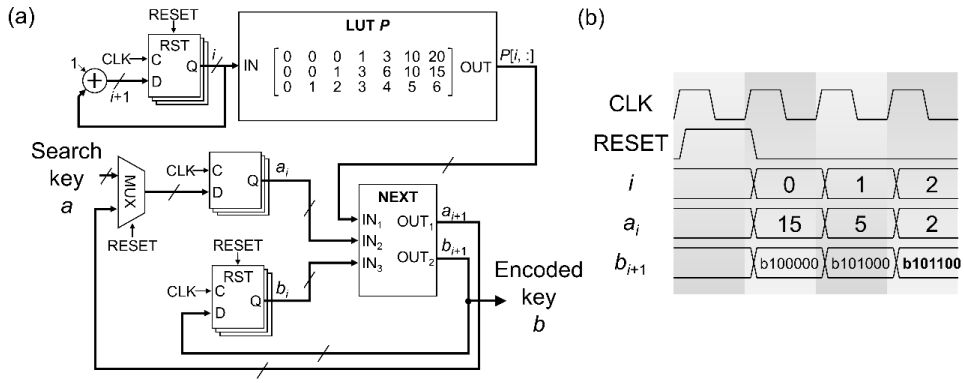


Figure 4.4. (a) Block diagram of an encoding circuit for 3-CECAM. (b) Timing diagram for encoding a search key of 15 as a 6-digit key of 101100.

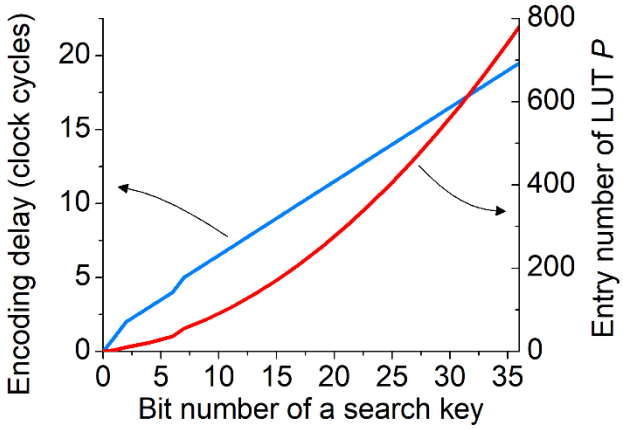


Figure 4.5. Encoding delay and number of entries in the LUT P with the bit number of a search key (n).

4.3. Parallel search of *N*-CECAM domains

A search of a single *N*-CECAM domain can be extended to parallel searches of multiple *N*-CECAM domains (partitions). Such parallel searches are useful, particularly, when a search key is so lengthy that *N* becomes large according to (1). To this end, n_p partitions are given to each match line, where each partition is a single N_p-CECAM domain loaded with $2N_p$ resistance switches in total. All partitions can share a single LUT *P* whose component $P[i, c]$ is $\binom{c}{N_p\text{-}i}$ for $0 \leq c \leq 2N_p$ and $0 \leq i < N_p$ as in Fig. 4.6. Therefore, each match line holds *n* bits expressed as

$$n = \left\lfloor \log_2 \binom{2N_p}{N_p} \right\rfloor \cdot n_p. \tag{2}$$

Each of n_p partitions is responsible for each n/n_p bit chunk of the total *n*-bit search key. Notably, this method reduces content-memory density. For instance, for a 60-bit search key, n_p = 1, 4, 10, 15, and 60 (respectively corresponding to N_p = 32, 9, 4, 3, and 1) yields approximately 0.94, 0.83, 0.75, 0.67, and 0.5 bit/switch, respectively. N_p = 32 and 1 indicate the single domain CECAM and the conventional 2R-based RCAM, respectively. Despite the reduction in content density, the advantage of the partitioning is threefold: reductions in the encoding delay and memory usage for the LUT *P*, and granularity of 'don't care' bits. Regarding the first advantage, the encoding delay is proportional to N_p as considered in Section II.B. Therefore, reducing N_p results in a reduction in the encoding delay. Regarding the second advantage, the LUT *P* is an $N_p \times (2N_p+1)$ matrix where the largest component is $\binom{2N_p}{N_p}$ which reaches 1.83×10^{18} for N_p

= 1, requiring 61-bit memory. Thus, reducing N_p (i.e., introducing partitions) reduces memory usage for the LUT *P* considerably.

For TCAM application, a configuration of All $2N_p$ resistance switches in the HRS represents 'don't care' bits in an N_p-CECAM domain. Therefore, the granularity of 'don't care' bits in the N_p-CECAM equals n/n_p bits. As shown in Table 4.1, 3-CECAM offers a 'don't care' granularity of 4 bits; when all six resistance switches in a domain are set to the HRS, the switch configuration is matched to any of 4-bit search keys between 0 and 15. Setting N_p = 32 for 60-bit search keys yields the coarsest granularity (60 bits), unsuitable for TCAM applications whereas N_p = 1, equivalent to the conventional RCAM, yields the finest granularity (1 bit). Therefore, introducing partitions is a viable method to decrease data granularity. Nevertheless, because this comes at the cost of a reduction in content density, the granularity should be reconciled with content density.

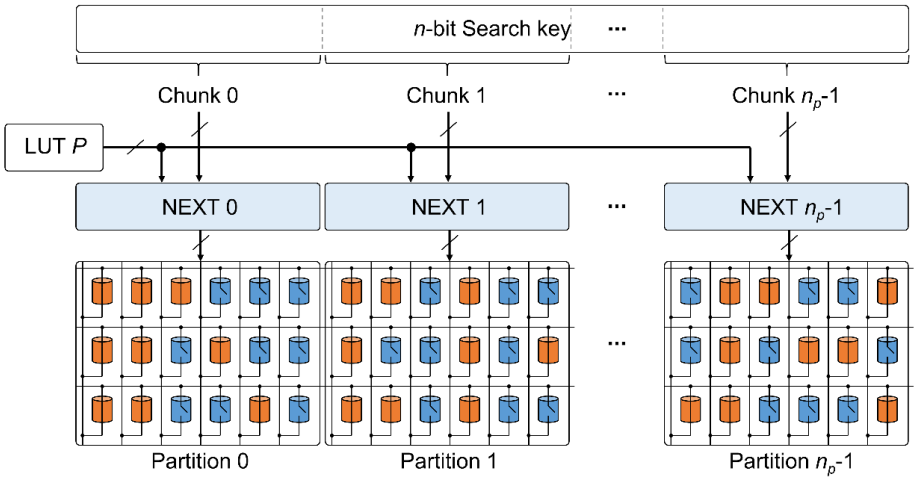


Figure 4.6. Schematic of parallel searches of N_p-CECAM partitions. NEXT in the figure means NEXT block in the encoding circuit. The n-bit search key is divided into n_p chunks, and each chunk applies to the NEXT block of each partition. All partitions share a single LUT P.

4.4. Algorithm for content decoding and circuit implementation

Contents in the state-of-the-art 2T2R-RCAM illustrated in Fig. 4.1(a) are read bitwise such that each bit (a pair of resistance switches) is iteratively examined for matching with the same key applied to the complementary search lines (SL and $\overline{\text{SL}}$)[14]. Therefore, a delay in reading is proportional to the bit number of contents. An advantage of a passive array of resistance switches shown in Fig. 4.1(b) over the active array is that the total contents per match line can simultaneously be read by pulling the match line high and simultaneously measuring the current response on all search lines. Either design employing the CECAM should be able to decode the 2*N*-digit contents as the original *n*-bit contents by an appropriate decoding function. The decoding function D_N is the reverse of the encoding function E_N, which is implemented as follows:

function $D_N(b)$

set a, i, c **to** 0

while $i < N$ **do**

if [the (c+1)th digit of b] = 1 **then**

set i **to** i+1

set a **to** $a+\binom{c}{i}$

end if

set c **to** c+1

end while

return a

end function.

A 2*N*-digit content in the CECAM is decoded as an *n*-bit key iteratively.

The decoding function D_N is implemented in a digital circuit as shown in Fig. 4.7(a). The circuit first initializes an *n*-long array a_0 to 0. Upon receiving a 2*N*-digit content *b* that is decoded ($b = b_0$), the ADR block in Fig. 4.7(a) searches for the address of the right-most '1' in b_0 and returns it, which corresponds to c_0 ($b_0[c_0] = 1$). $P[N\text{-}1, c_0] = \binom{c_0}{1}$ is subsequently retrieved from the LUT *P*, and a_1 is evaluated as $a_1 = a_0 + P[N\text{-}1, c_0]$. b_1 is identical to b_0 other than its right-most '1' switched to '0'. Subsequently, c_1 is evaluated as the address of the right-most 1 in b_1, and then a_2 and b_2 are evaluated as $a_2 = a_1 + P[N\text{-}2, c_1]$ and $b_2 = b_1$ except that $b_2[c_1] = 0$, respectively. This evaluation is repeated *N* times, resulting in an *n*-bit decoded content *a*. Similar to encoding, a delay in decoding is caused by iteratively addressing each row of the LUT *P*. Therefore, the delay also scales with *N* (Fig. 4.7(b)).

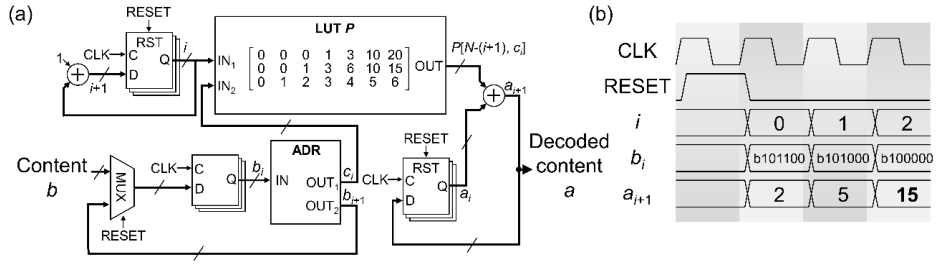


Figure 4.7. (a) Block diagram of a decoding circuit for 3-CECAM. (b) Timing diagram for decoding an encoded search key of 101100 as its original search key (15)

4.5. Discussion

The CECAM scheme was applied to a passive array of resistance switches as a model system. This model system allows the ultimate integration density, i.e., when used as RAM, $4F^2$ cell size per bit. In this case, a current-sensing scheme is suitable for bitwise reading. However, the reading process is significantly prone to error because of the notorious sneak current issue due to the lack of bit-selection devices[26]. Employing transistors as active selectors significantly keeps the sneak current sufficiently low for reliable reading as for the 2T2R-based RCAM design[27]. The CECAM scheme applies to an active array of resistance switches with a voltage-sensing scheme as shown in Fig. 4.8. In the array, a single unit consists of 2*N* transistors and 2*N* resistance switches. A one-transistor and one-resistor (1T1R) unit is placed at a crossing point between each match (horizontal) line and each of the 2*N* search (vertical) lines. Specifically, the gates of 2*N* transistors are wired to the 2*N* search lines, and thus the encoded 2*N*-digit key determines the channel conductance of the 2*N* transistors during a searching period. For all transistors, the source is connected to a common plate line which is grounded during searching.

When searching, the match lines are pre-charged simultaneously. Then, a 2*N*-digit encoded key is applied to the search lines such that 1's and 0's pull the search lines up and down, respectively. When matching *N* 1's in the encoded key to the transistors paired with the *N* HRS switches, the voltage on the match line remains high because none of the pull-down paths are activated. Otherwise, some 1's in the encoded key are inevitably associated with transistors paired with LRS switches, indicating the activation of pull-down paths (Fig. 4.8).

Therefore, the voltage on the pre-charged match line decays rapidly, which is noticed by a sense amplifier as a mismatch. This search process is identical to the 2T2R-based conventional RCAM.[25]

Regarding the sense amplifier design for the CECAM, a current- and a voltage-sensing amplifier are suitable for a passive and active array of switches, respectively, as for RCAM. Compared with RCAM, the CECAM does not impose additional requirements on its sense amplifiers, so that previously developed sensing technologies[16, 28, 29] are compatible with the CECAM. In this regard, the CECAM can make full use of previous RCAM technologies given a subtle difference between the CECAM and RCAM. The subtle difference lies in content- and search key-encoding, which is the key of our present study. Nevertheless, the subtle difference remarkably enhances the content density.

The application domain of the proposed CECAM fully covers other resistance-based NVMs with two-terminal switches, e.g., phase-change memory[14, 15] and magnetic tunnel junction[14, 17], in both active and passive arrays. Moreover, the CECAM concept is compatible with three-terminal NVMs, for instance, ferroelectric transistors[20] and NOR Flash memory.

Table 4.2 compares the CECAM with previous CAM designs. The 4-CECAM was considered with reference to the 4 kb 1T1R RRAM prototype[25] and simulation results of a 2R-TCAM[23]. The 128 bit word width in [25] allows 16 × (4-CECAM domain), i.e., $N_p = 4$ and $n_p = 16$. According to (2), each match line holds 96 bit contents; instead, the conventional RCAM allows

64 bit contents per match line only. Therefore, the cell area per content bit is approximately 0.67 times that of the conventional RCAM as shown in Table 4.2. The same holds for the 2R-TCAM with 64 bit word width in [23]. Regretfully, the actual size of the 1T1R RRAM prototype[25] is unavailable so that the cell area per bit for the CECAM is expressed as its area relative to that of the RCAM (0.67). The additional delay in searching due to the encoding of a search key is N_p/f_{clk}, where is f_{clk} clock speed. The cell area per bit and search delay aside, the CECAM is identical to the RCAM.

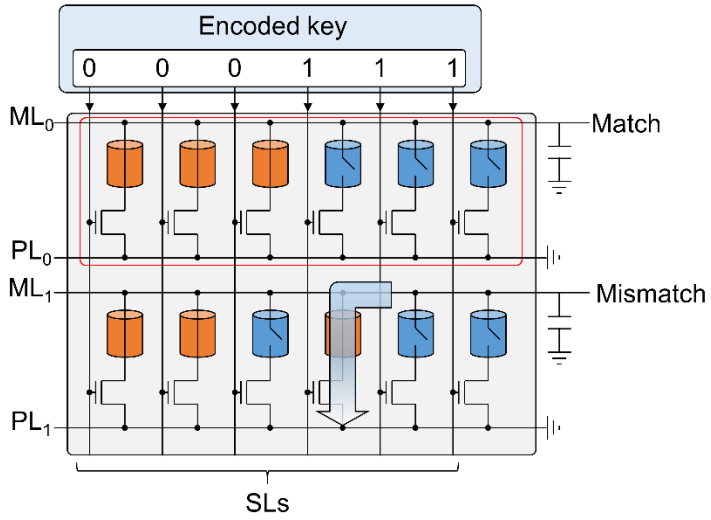


Figure 4.8. Schematic of CECAM with a voltage-reading scheme. The blue arrow in the second row illustrates activated pull-down path.

Table 4.2. Comparison to previous work

	16T-TCAM [11]	6T2R-TCAM [18]	11T3R-TCAM [19]	2T2R-TCAM [16]	2T2R-TCAM [25]	2R-TCAM [23]	4-CECAM referenced to [25]	4-CECAM referenced to [23]
Memory type	SRAM	MTJ	MTJ	PCM	RS	RS	RS	RS
Reading scheme	Voltage	Voltage	Voltage	Voltage	Voltage	Current	Current	Current
Technology (nm)	65	90	180	90	130	90	130	90
Word width (bit)	72	32	144	64	128	64	128	64
Cell area (μm^2/bit)	1.69	10.35	42	0.41	NA (1×)[a]	NA (1×)[b]	NA (0.66×)[a]	NA (0.66×)[b]
Supply voltage (V)	1	1.2	1	1.2	0.9	0.2	0.9V	0.2
Search delay	1.9ns	0.29ns	8ns	1.9ns	2ns	0.5ps	0.5ps	0.5ps
Search energy (fJ/bit/search)	1.98	1.04	7.4	NA	NA (1×)[a]	0.23	NA (0.66×)[a]	0.15

*RS denotes resistance switch

*[a,b] denotes markers comparing normalized values between previous work and CECAM

4.6. Conclusion

A new type of CAM, referred to as CECAM, was proposed to improve the content density in a memory array. The *N*-CECAM employs a group of 2*N* resistance switches as a single memory unit with multi-bit (*n*-bit) capacity, which enhances its content density far beyond that of the conventional RCAM (0.5 bit/switch). For instance, 10-CECAM ($N = 10$; 20 resistance switches) has 17-bit content capacity ($n = 17$) in contrast to the conventional RCAM that needs 34 resistance switches for 17-bit content capacity. The key to the CECAM is an algorithm for *n*-to-2*N* encoding and its decoding. The proposed encoding and decoding algorithms were proven to match *n*-bit search keys to 2*N*-digit keys bijectively. Additionally, the algorithms are readily implemented in digital circuits with a combination table, which results in an encoding (decoding) delay of *N* clock cycles for the *N*-CECAM. The proposed CECAM concept is compatible with various NVM-based CAM designs including active and passive RCAM, other two-terminal resistance switch-based CAM, e.g., phase-change memory and magnetic memory, and NVM transistors, e.g., ferroelectric transistor and NOR Flash memory.

4.7. Appendix

Theorem 1. E_N: $\boldsymbol{A}_t \rightarrow \boldsymbol{B}_t$ is a bijective function for $\boldsymbol{A}_t = \left\{0, 1, \ldots, \binom{2N}{N} - 1\right\}$ and $\boldsymbol{B}_t = \{b \mid b$: 2*N*-bit binary numbers, each with *N* 1's and *N* 0's$\}$.

Proof. Nonnegative integer a_i is defined as

$$a_{i+1} = a_i - \binom{c_i}{N-i} \quad \text{for} \quad 0 \le i < N-1, \tag{3}$$

and $a_0 = a$ (≥ 0). c_i satisfies the following inequality:

$$\binom{c_i}{N\text{-}i} \le a_i < \binom{c_i+1}{N\text{-}i} \text{ for } 0 \le i < N. \quad (4)$$

Define the number of elements in set $\boldsymbol{X}$ as $n(\boldsymbol{X})$.

Lemma 1: $n(\boldsymbol{A}_t) = n(\boldsymbol{B}_t) = \binom{2N}{N}$.

Lemma 2: $c_i > c_{i+1}$ for $0 \le i < N\text{-}1$.

Proof. For $a_i = 0$, the only c_i satisfying the inequality in (4) is $c_i = N\text{-}i\text{-}1$ that yields $a_{i+1} = 0$ according to (1). From (4), $c_{i+1} = N\text{-}i\text{-}2$. Therefore, $c_i > c_{i+1}$. For $a_i > 0$, using (3) and the fact that $a_i < \binom{c_i+1}{N\text{-}i}$ in (4), the following inequality is acquired:

$$0 \le a_{i+1} < \binom{c_i+1}{N\text{-}i} - \binom{c_i}{N\text{-}i} = \binom{c_i}{N\text{-}(i+1)}. \quad (5)$$

Equation (4) for a_{i+1} is $\binom{c_{i+1}}{N\text{-}(i+1)} \le a_{i+1} < \binom{c_{i+1}+1}{N\text{-}(i+1)}$. a_{i+1} should simultaneously satisfy this equation and (3), which is true if $\binom{c_i}{N\text{-}(i+1)} > \binom{c_{i+1}}{N\text{-}(i+1)}$. Therefore, $c_i > c_{i+1}$. Consequently, $c_i > c_{i+1}$ holds for nonnegative a_i.

For given a, vector $\boldsymbol{c}(a)$ is defined as $\boldsymbol{c}(a)=[c_0, c_1, \ldots, c_{N\text{-}1}]$ where the components are sorted in descending order according to Lemma 2. Associating b with $\boldsymbol{c}(a)$ such that '1' is placed on each (c_i+1)th digit of b, and '0's on the other digits, it is proven that $\boldsymbol{c}(a)$ is bijectively mapped to b. If a is also bijectively mapped to $\boldsymbol{c}(a)$, a is eventually proven to be mapped to b in a bijective manner. Also, using Lemma 1 and bijective mapping of $\boldsymbol{c}(a)$ to b, the following equation is acquired:

$$n(\boldsymbol{A}_t) = n(\boldsymbol{B}_t) = n(\boldsymbol{C}) = \binom{2N}{N}, \quad (6)$$

where $\boldsymbol{C} = \{c \mid c = \boldsymbol{c}(a)\}$.

Lemma 3: if $x \neq y$, then $\boldsymbol{c}(x) \neq \boldsymbol{c}(y)$.

Proof. If Lemma 3 is true, its contraposition (if $\boldsymbol{c}(x) = \boldsymbol{c}(y)$, then $x = y$) is also true. Given (1), a (= a_0) is expressed as

$$a = a_{N\text{-}1} + \sum_{i=0}^{N-2} \binom{c_i}{N\text{-}i}. \tag{7}$$

Equation (4) for $i = N$-1 yields $\binom{c_{N-1}}{1} \leq a_{N-1} < \binom{c_{N-1}+1}{1}$, i.e., $c_{N-1} \leq a_{N-1} < c_{N-1}+1$. Thus, $a_{N-1} = c_{N-1} = \binom{c_{N-1}}{1}$. Therefore, (7) is rewritten by $a = \sum_{i=0}^{N-1} \binom{c_i}{N\text{-}i}$. This equation indicates a unique a for a given vector $\boldsymbol{c}(a)$, so that if $\boldsymbol{c}(x) = \boldsymbol{c}(y)$, then $x = y$. Lemma 3 therefore holds true, identifying injective mapping of a to $\boldsymbol{c}$.

Equation (6) and Lemma 3 identify bijective mapping of a to $\boldsymbol{c}$. Given the bijective mapping of a to $\boldsymbol{c}$ and $\boldsymbol{c}$ to b, the encoding function E_N is a bijective function.

4.8. Bibliography

[1] K. Pagiamtzis and A. Sheikholeslami, *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712-727, 2006.

[2] H. J. Chao, *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1518-1558, 2002.

[3] T.-B. Pei and C. Zukowski, in *IEEE International Conference on Computer Communications*, 1991, pp. 515-524.

[4] T.-B. Pei and C. Zukowski, *IEEE Network*, vol. 6, no. 1, pp. 42-50, 1992.

[5] N.-F. Huang, W.-E. Chen, J.-Y. Luo, and J.-M. Chen, in *IEEE Blogal Communications Conference*, 2001, vol. 3, pp. 1877-1881.

[6] G. Qin, S. Ata, I. Oka, and C. Fujiwara, in *IEEE International Conference on Computer Communications*, 2002, vol. 3, pp. 2350-2354.

[7] A. J. McAuley and P. Francis, in *IEEE International Conference on Computer Communications*, 1993, pp. 1382-1391.

[8] V. Kornijcuk, J. Park, G. Kim, D. Kim, I. Kim, J. Kim, J. Y. Kwak, and D. S. Jeong, *Advanced Materials Technologies*, vol. 4, no. 1, p. 1800345, 2019.

[9] S. Panchanathan and M. Goldberg, *IEEE Transactions on Signal Processing*, vol. 39, no. 9, pp. 2066-2078, 1991.

[10] C. Lee and M. Paull, *Proceedings of the IEEE*, vol. 51, no. 6, pp. 924-932, 1963.

[11] I. Hayashi, T. Amano, N. Watanabe, Y. Yano, Y. Kuroda, M. Shirata, K. Dosaka, K. Nii, H. Noda, and H. Kawai, *IEEE Journal of Solid-State Circuits*, vol. 48, no. 11, pp. 2671-2680, 2013.

[12] O. Tyshchenko and A. Sheikholeslami, *IEEE Journal of Solid-State Circuits*, vol. 43, no. 9, pp. 1972-1981, 2008.

[13] Y. Yang, J. Mathew, R. S. Chakraborty, M. Ottavi, and D. K. Pradhan, *IEEE Transactions on Nanotechnology*, vol. 15, no. 3, pp. 527-538, 2016.

[14] Q. Guo, X. Guo, Y. Bai, and E. Ipek, in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 339-350.

[15] B. Rajendran, R. W. Cheek, L. A. Lastras, M. M. Franceschini, M. J. Breitwisch, A. G. Schrott, J. Li, R. K. Montoye, L. Chang, and C. Lam, in *2011 3rd IEEE International Memory Workshop*, 2011, pp. 1-4.

[16] J. Li, R. K. Montoye, M. Ishii, and L. Chang, *IEEE Journal of Solid-State Circuits*, vol. 49, no. 4, pp. 896-907, 2013.

[17] S. Matsunaga, M. Natsui, K. Hiyama, T. Endoh, H. Ohno, and T. Hanyu, *Japanese Journal of Applied Physics*, vol. 49, no. 4S, p. 04DM05, 2010.

[18] S. Matsunaga, A. Katsumata, M. Natsui, S. Fukami, T. Endoh, H. Ohno, and T. Hanyu, in *2011 Symposium on VLSI Circuits-Digest of Technical Papers*, 2011: IEEE, pp. 298-299.

[19] W. Xu, T. Zhang, and Y. Chen, *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 1, pp. 66-74, 2009.

[20] I. Bayram and Y. Chen, in *Non-Volatile Memory Systems and Applications Symposium*, 2014, pp. 1-6.

[21] B. Chen, Y. Zhang, W. Liu, S. Xu, R. Cheng, R. Zhang, and Y. Zhao, *IEEE Electron Device Letters*, vol. 39, no. 9, pp. 1294-1297, 2018.

[22] L.-Y. Huang, M.-F. Chang, C.-H. Chuang, C.-C. Kuo, C.-F. Chen, G.-H. Yang, H.-J. Tsai, T.-F. Chen, S.-S. Sheu, and K.-L. Su, in *2014 Symposium on VLSI Circuits Digest of Technical Papers*, 2014, pp. 1-2.

[23] R. Han, W. Shen, P. Huang, Z. Zhou, L. Liu, X. Liu, and J. Kang, *Japanese Journal of Applied Physics*, vol. 57, no. 4S, p. 04FE02, 2018.

[24] D. Ly, B. Giraud, J. Noel, A. Grossi, N. Castellani, G. Sassine, J. Nodin, G. Molas, C. Fenouillet-Beranger, and G. Indiveri, in *2018 IEEE International Electron Devices Meeting*, 2018, pp. 20.3.1-20.3.4.

[25] A. Grossi, E. Vianello, C. Zambelli, P. Royer, J.-P. Noel, B. Giraud, L. Perniola, P. Olivo, and E. Nowak, *IEEE Transactions on Very Large Scale Integration Systems*, pp. 1-9, 2018.

[26] A. Chen, *IEEE Transactions on Electron Devices*, vol. 60, no. 4, pp. 1318-1326, 2013.

[27] P.-Y. Chen and S. Yu, *IEEE Transactions on Electron Devices*, vol. 62, no. 12, pp. 4022-4028, 2015.

[28] I. Arsovski and A. Sheikholeslami, *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1958-1966, 2003.

[29] N. Mohan, W. Fung, D. Wright, and M. Sachdev, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 3, pp. 566-573, 2008.

5. Conclusion

As mentioned above, binary resistance switch array can be applied as synapse array in synapse block or lookup-table in topology block. Therefore, we have studied on three subjects, which are a new simulation method for binary resistance switch array, a new learning algorithm with ternary synaptic weight, and a new type of resistance switch-based content addressable memory with high content density.

In the first part, multi-layer perceptrons with different structures (depth and the number of perceptrons in each layer) was successfully trained to infer the current response of a random crossbar array to a randomly applied voltage vector. The trained network predicted exact current response with appropriate network structure and sufficient training examples. Additionally, this neural network is 8 times faster than Newton-Raphson method for 10×9 resistance switch array.

Secondly, a new learning algorithm, reffered to as Markov Chain Hebbian Learning, was proposed. MCHL uses ternary synaptic weight. Therefore, MCHL is appropriate to use when using binary resistance switch array as synapse array. Another distinct feature of MCHL is that it does not use backpropagation and synaptic units are stochastically updated. This feature is similar to restricted Boltzmann machine, but MCHL is discriminative with write vector. The potentiation or depression of synaptic units are governed by write vector $\boldsymbol{v}$ and exact update probability was controlled by activation of input and output neuron. MCHL was applied to hand-written digit recognition and it

have shown 92% accuracy. This accuracy is much lower than that of conventional backpropagation algorithm, 98%. MCHL, however, uses much less memory and is faster than backpropagation algorithm. This aspect stands out when MCHL is implemented in FPGA board. MCHL was also applied to prime factorization and it needs much less steps than direct search factorization.

At last, a new type of CAM, referred to as CECAM, was proposed to improve the content density in a memory array. The *N*-CECAM uses a group of *N* HRS resistance switches and *N* LRS resistance switches as a single memory unit. As a result, CECAM's content density is far beyond that of the conventional RCAM (0.5 bit/switch). For instance, 10-CECAM (N = 10; 20 resistance switches) has 17-bit content capacity (n = 17) in contrast to the conventional RCAM that needs 34 resistance switches for 17-bit content capacity. The encoding and decoding algorithm for CECAM were also proposed. They have been proven to convert a *n*-bit search keys to 2*N*-digit keys with *N* 1's and *N* 0's bijectively. Additionally, the algorithms are readily realized in digital circuits with a combination table. The combination table is implemented to minimize calculation costs from binomial factor. It results in an encoding (decoding) delay of *N* clock cycles for the *N*-CECAM. The proposed CECAM concept is compatible with various NVM-based CAM designs including active and passive RCAM, other two-terminal non-volatile memory-based CAM, e.g., phase-change memory and magnetic memory, and NVM transistors, e.g., ferroelectric transistor and NOR Flash memory.

Curriculum Vitae

Guhyun Kim

Department of Materials Science and Engineering
College of Engineering
Seoul National University
1 Gwanak-ro, Gwanak-gu, Seoul 151-742, Korea

E-mail:
kgh920507@snu.ac.kr
Tel.: +82-2-880-8923
Fax.: +82-2-874-6414

I. Educations

2011. 3. - 2015. 2. B.S.

Department of Materials Science and Engineering

Seoul National University, Seoul, Korea

2015. 3. – 2020. 2. Ph.D

Department of Materials Science and Engineering

Seoul National University, Seoul, Korea

II. Research Areas

1. Thin Film Materials and Devices

- Characterization of electronic properties of thin films
- Thin films deposition technique
- Resistance switch array simulation technique

2. Neuromorphic Engineering

- Learning algorithm for SNN and ANN
- Routing algorithm for spike transmission and synaptic unit update
- Binarized neural network

III. Experimental Skills

1. Deposition methods

- DC & RF sputtering (in-situ deposition of Pt, HfO_2, TiN)

2. Sample preparation

- Photo-lithography

3. Analysis methods

- Pulse/pattern generator and digital oscilloscope for pulse switching measurement of resistance switches

4. Programs apprentice

- MATLAB (Mathworks) 2013
- Python
- C++

List of publications

1. Refereed Journal Articles (SCI)

1.1 Domestic

1.2. International

[1] Hyungkwang Lim, Hyung-Woo Ahn, Vladimir Kornijcuk, **Guhyun Kim,** Jun Yeong Seok, Inho Kim, Cheol Seong Hwang, and Doo Seok Jeong, "Relaxation oscillator-realized artificial electronic neurons, their responses, and noise," *Nanoscale*, vol. 8, no. 18, pp. 9629-9640, 2016.

[2] Hyungkwang Lim, Rohit Soni, Dohun Kim, **Guhyun Kim**, Vladimir Kornijcuk, Inho Kim, Jong-Keuk Park, Cheol Seong Hwang, and Doo Seok Jeong, "Chameleonic electrochemical metallization cells: dual-layer solid electrolyte-inducing various switching behaviours," *Nanoscale*, vol. 8, no. 34, pp. 15621-15628, 2016.

[3] Vladimir Kornijcuk, Hyungkwang Lim, Jun Yeong Seok, **Guhyun Kim**, Seong Keun Kim, Inho Kim, Byung Joon Choi, and Doo Seok Jeong, "Leaky integrate-and-fire neuron circuit based on floating-gate integrator," *Frontiers in neuroscience*, vol. 10, p. 212, 2016.

[4] Vladimir Kornijcuk, Jongkil Park, **Guhyun Kim**, Dohun Kim, Inho Kim, Jaewook Kim, Joon Young Kwak, and Doo Seok %J Advanced Materials Technologies Jeong, "Reconfigurable Spike Routing Architectures for On-Chip Local Learning in Neuromorphic Systems," *Advanced Materials Technologies*, vol. 4, no. 1, p. 1800345, 2019.

[5] **Guhyun Kim**, Vladimir Kornijcuk, Dohun Kim, Inho Kim, Jaewook Kim, Hyo Cheon Woo, Jihun Kim, Cheol Seong Hwang, and Doo Seok Jeong, "Markov chain Hebbian learning algorithm with ternary synaptic units," *IEEE Access*, vol. 7, pp. 10208-10223, 2019.

[6] **Guhyun Kim**, Vladimir Kornijcuk, Dohun Kim, Inho Kim, Cheol Seong Hwang, and Doo Seok Jeong, "Artificial Neural Network for Response Inference of a Nonvolatile Resistance-Switch Array," *Micromachines*, vol. 10, no. 4, p. 219, 2019.

[7] **Guhyun Kim**, Vladimir Kornijcuk, Jeeson Kim, Dohun Kim, Cheol Seong Hwang, and Doo Seok Jeong, "Combination-Encoding Content-Addressable Memory With High Content Density," *IEEE Access*, vol. 7, pp. 137620-137628, 2019.

2. CONFERENCES

2.1 Domestic

[1] Guhyun Kim, Cheol Seong Hwang, and Doo Seok Jeong, "Artificial Neural Network for Response Inference of a Nonvolatile Resistance-Switch Array," in *Nano Korea*, 2019, oral

2.2. International

[1] Guhyun Kim, Cheol Seong Hwang, and Doo Seok Jeong, "Stochastic Learning with Back Propagation," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019: IEEE, pp. 1-5, oral

Abstract (in Korean)

저항 변화 소자는 차세대 메모리의 선두주자 중 하나이다. 저항 변화 소자는 높은 저항 상태와 낮은 저항 상태를 가지고 있으며, 상태의 변화는 전압 혹은 전류를 가해주는 전기적인 자극에 의해서 발생한다. 크로스바 어레이 구조를 통해, 저항 변화 소자 어레이는 $4F^2$ (F: minimum feature size)의 매우 높은 집적도를 나타낸다. 아날로그 저항 변화 소자 또한 개발되고 있으나, 대부분은 매우 정밀한 저항 컨트롤이 필요하고 저항 변화가 가해주는 펄스 수에 비 선형적이라는 단점이 있다.

저항 변화 소자 어레이의 가장 큰 특징은 행렬-벡터 곱을 구현할 수 있다는 점이다. 즉, 저항 변화 소자 어레이의 출력 전류는 전도도 행렬과 입력 전압 벡터의 곱으로 표현된다.

저항 변화 소자 어레이를 시뮬레이션 하는 것은 저항 변화 소자 어레이의 성질을 분석하는데 매우 유용하다. 가장 대중적인 시뮬레이션 방법은 Newton-Raphson 방법을 사용하는 것이다. 하지만 이 방법은 계산을 위해 많은 리소스가 필요하다. 이에 대한 대안으로, 본 논문에서는 인공신경망을 활용하였다. 본 논문에서는 (10 × 9 또는 28 × 27의 크기를 가지는) 임의의 이진 저항 변화 소자 어레이와 임의의 입력 전압 벡터에 의한 출력 전류를 유추하는 인공신경망을 구성하였다. 인공 신경망은 leaky rectified

linear units을 사용하는 multilayer perceptron (MLP)를 활용하였다. 이 인공신경망은 500,000개 혹은 1,000,000개의 예제를 통해 학습되었다. 각각의 예제마다, 인공신경망의 입력 벡터는 저항 변화 소자 어레이의 전도도 행렬과 입력 전압 벡터의 합으로 구성되었다. 즉, M개의 행에 전압이 가해지는 $M \times N$ 어레이에 대하여, 입력 벡터의 크기는 $M \times (N+1)$을 가진다. 각각의 예제에 대해 Newton-Raphson 방법을 사용해 계산한 출력 전류가 지도 학습의 데이터 레이블로 활용되었다. 이 시도는 정확한 출력 전류를 예측하였으며, 28 × 27 어레이의 경우 상관계수값이 0.9995에 이르렀다. 또한 이 방법은 기존의 Newton-Raphson 방법에 비해 약 8배 빠른 계산속도를 나타내었다.

저항 변화 소자 어레이의 병렬 작동에 기반하여, 저항 변화 소자 어레이는 뉴로모픽 하드웨어의 다양한 부분에 활용될 수 있다. 가장 널리 알려진 것은 저항 변화 소자 어레이로 인공 시냅스 어레이를 구성하는 것이다. 저항 변화 소자 어레이의 행렬-벡터 곱은 인공신경망 내부의 행렬-벡터 곱과 유사하기 때문에, 저항 변화 소자 어레이를 인공 시냅스 어레이로 활용하는 것은 뉴로모픽 하드웨어의 작동을 가속화할 수 있다.

따라서 본 논문에서는, Markov chain Hebbian learning이라고 불리는 이진 저항 변화 소자 어레이에 적합한 학습 알고리즘을 개발하였다. 이 학습 알고리즘은 메모리 측면에서 효율성을

나타내는데 이는 1) 시냅스 가중치가 −1, 0, 1의 ternary 값을 가지고 2) 시냅스 가중치의 업데이트가 마코프 체인—현 시점의 업데이트는 이전 시점의 가중치 값이 필요 없다—을 따르기 때문이다. 또한 −1, 0, 1의 ternary 값은 한 쌍의 저항 변화 소자를 쉽게 구현할 수 있기 때문에, 이진 저항 변화 소자 어레이에도 적합한 알고리즘이라고 볼 수 있다. 이 알고리즘은 이미지 인식과 곱셈표 암기 두가지 분야로 검증되었다. 특히 후자의 경우 사람의 암산과 같은 메모리 기반 곱셈에 기반하였다. 또한 메모리 기반 곱셈에 기반한 방식이기에 인수 분해에도 활용할 수 있음을 증명하였다.

저항 변화 소자 어레이의 또다른 응용 분야는 topology block의 lookup table로 사용될 수 있는 내용 주소화 기억장치 (content-addressable memory, CAM)이다. 이 lookup table은 뉴런 사이의 모든 연결 정보를 저장하고 있어, 스파이크가 발생하였을 시 스파이크가 전달될 뉴런들과 업데이트 해야 할 시냅스들을 검색하는 역할을 하고 있다. 저항 소자 기반 CAM은 빠른 검색 능력과 높은 집적도, 낮은 정적 에너지 소모량을 가지고 있기 때문에 lookup table로 활용하기 적합하다고 볼 수 있다.

그러나 RCAM은 한 쌍의 저항 변화 소자로 하나의 bit를 표현하기 때문에 (0.5bit/switch) resistive random access memory (1bit/ switch)에 비해 낮은 컨텐츠 밀도를 가지고 있다.

본 논문에서는 combination-encoding CAM (CECAM)이라 불리는 새로운 종류의 RCAM을 제시하였다. N-CECAM은 N개의 높은 저항 상태를 가지는 소자와 N개의 낮은 저항 상태를 가지는 소자를 하나의 유닛으로 구성하고. 이 소자들의 조합을 통해 높은 컨텐츠 밀도를 달성할 수 있었다. (N=10일 경우 0.85 bit/switch). CECAM의 핵심은 n-bit의 search key를 $2N$ 자리의 이진 key로 인코딩하는 알고리즘과 반대로 디코딩 알고리즘을 구성하는 것이다. 본 논문에서는 CECAM에 적합한 인코딩 알고리즘과 디코딩 알고리즘 및 이 알고리즘들에 대한 회로 역시 구성하였다.

주요어: 뉴로모픽 엔지니어링, 저항 변화 소자 어레이, 다층 퍼셉트론, 마코프 체인, 내용 주소화 기억장치

학번: 2015-20801

김 구 현