UNIVERSITY OF CALIFORNIA SAN DIEGO

Neuro-inspired Computing Using Emerging Non-Volatile Memories

A dissertation submitted in partial satisfaction
of the requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Machine Learning and Data Science)

by

Yuhan Shi

Committee in charge:

Professor Duygu Kuzum, Chair
Professor Gert Cauwenberghs
Professor Mingu Kang
Professor Piya Pal
Professor Paul H. Siegel

2023

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SUPPLEMENTARY VIDEOS

ACKNOWLEDGEMENTS

This thesis signifies my scientific and personal growth that would not have been possible without many people. I would like to thank my advisor, Professor Duygu Kuzum, for her guidance and support throughout my Ph.D. program. I met her in the ECE recruitment event in 2016 and appreciate the opportunity she offered for me to enter neuroelectronics lab and pursue a graduate degree at University of California San Diego (UCSD). I am especially grateful for her patience, research insights, and continuous support for my study. She sets high standard for me to perform scientific research and her persistence always motivates me to keep thinking and trying no matter my experiments succeed or failed. She also trains me extensively in explaining and writing research findings or results with greater clarity and effectiveness, which significantly advance my writing skills as well as communication efficiency with broader audience.

I would like to thank my current and former committee members, Professor Gert Cauwenberghs, Professor Mingu Kang, Professor Paul H. Siegel, Professor Piya Pal, Professor Kenneth Kreutz-Delgado, and Professor Yu-Hwa Lo for their time, patience and advice during several milestones in my Ph.D.

I came to US pursuing my undergraduate degree and this year also marks my 10th year in this country. I would like to thank Professor Richard J. Radke, Professor Robert F. Karlicek, and Professor Jian Shi from Rensselaer Polytechnic Institute (RPI) offer me first-hand research experience in their labs during my undergraduate study. These valuable experiences not only excite my interests to apply for graduate school but also layout a good fundamental for me to continue performing research at UCSD.

Additionally, I am grateful for the amazing collaborators who worked with me and provided significant contributions to my research over the past years. I would like to thank Dr. John R. Jameson and Dr. Foroozan Koushan from Adesto Technologies for providing wafers with cutting -edge subquantum CBRAM technology. I would like to thank Dr. Seung H. Kang and Xiao (Bill) Lu from Qualcomm Inc for giving me an opportunity to work on STT-MRAM technology. I would like to thank Professor Gert Cauwenberghs and his students from Bioengineering Department at UCSD for collaborating on CMOS-RRAM projects. I would like to thank Professor Ivan K. Schuller and his postdocs from Department of Physics at UCSD for collaborating on Mott-ReLU projects.

I also felt extremely lucky to have a chance interning with Micron Technology and performing R&D development in DRAM characterization group led by Christopher Cardon and Yosuke Irie in summer 2022. I truly appreciate their dedication, advising and help during my internship. It is an eye-opening industry experience for me to understand how difficult it is to engineer semiconductor devices at advanced technology node into real products.

I am very fortunate to have excellent lab mates and colleagues. I appreciate them for their help in my research and bringing joys to my life. I want to thank Sangheon Oh for collaborating with me on numerous projects, transferring his exceptional fabrication skills to me and his great contributions in device fabrication, characterization and simulation. I want to thank Leon Nguyen for working with me on my first project and helping me to understand machine learning algorithms deeper. I want to thank Zhisheng Huang for assisting me with simulations and paper revisions. I want to thank Dr. Akshay Ananthakrishnan for his continuous support and discussions in solve challenging problems in my research. I want to thank Dr. Xin Liu for sharing life stories to each other and providing helpful discussion on almost every problem I got confused. I want to thank Jeong-Hoon (Jimmy) Kim for sharing novel research ideas and happy conversations for relieving the stress. I want to thank Madison Wilson for organizing lab gathering events each quarter and providing a warm lab environment. Finally, I want to thank Dr. Yichen Lu, Ruoyu Zhao, Mehrdad Ramezani, Dr. Jungwoo Song, Nathan Kaslan, Dr. Ashwani Kumar Yucheng Zhou, Jaeseoung Park, Ziqi Zhou, Gopabandhu Hota and Soumil Jain for their discussions and help in my research.

Finally, I want to thank all my close friends since high school. The monthly calls and annual trips with them always cheer me up and let me keep positiveness during downtime. Last but most importantly, I want to thank my family for their unconditional love and endless support throughout the way. They have given me genuine advice in every turning point in my life. They have also given me courage and strength to face and overcome all the obstacles in my life. This journey would not be possible without them. This dissertation is dedicated to them.

Chapter 1 is a reprint of **Yuhan Shi**, Leon Nguyen, Sangheon Oh, Xin Liu, Foroozan Koushan, John R. Jameson, and Duygu Kuzum. "Neuroinspired unsupervised learning and pruning with subquantum CBRAM arrays." Nature communications 9, no. 1 (2018): 1-11. The dissertation author was the first author of this paper.

Chapter 2 is a reprint of **Yuhan Shi**, Leon Nguyen, Sangheon Oh, Xin Liu, and Duygu Kuzum. "A soft-pruning method applied during training of spiking neural networks for in-memory computing applications." Frontiers in neuroscience 13 (2019): 405. The dissertation author was the first author of this paper.

Chapter 3 is a reprint of **Yuhan Shi**, Zhisheng Huang, Sangheon Oh, Nathan Kaslan, Jungwoo Song, and Duygu Kuzum. "Adaptive quantization as a device-algorithm co-design approach to improve the performance of in-memory unsupervised learning with SNNs." IEEE Transactions on Electron Devices 66, no. 4 (2019): 1722-1728. The dissertation author was the first author of this paper.

Chapter 4 is a reprint of **Yuhan Shi**, Sangheon Oh, Zhisheng Huang, Xiao Lu, Seung H. Kang, and Duygu Kuzum. "Performance prospects of deeply scaled spin-transfer torque magnetic random-access memory for in-memory computing." IEEE Electron Device Letters 41, no. 7 (2020): 1126-1129.The dissertation author was the first author of this paper.

Chapter 5 is a reprint of **Yuhan Shi**, Akshay Ananthakrishnan, Sangheon Oh, Xin Liu, Gopabandhu Hota, Gert Cauwenberghs, and Duygu Kuzum. "A Neuromorphic Brain Interface Based on RRAM Crossbar Arrays for High Throughput Real-Time Spike Sorting." IEEE Transactions on Electron Devices 69, no. 4 (2021): 2137-2144. The dissertation author was the first author of this paper. Chapter 5, in part, is also a reprint of **Yuhan Shi**, Akshay Ananthakrishnan, Sangheon Oh, Xin Liu, Gopabandhu Hota, Gert Cauwenberghs, and Duygu Kuzum. "High Throughput Neuromorphic Brain Interface with CuOx Resistive Crossbars for Real-

time Spike Sorting." In 2021 IEEE International Electron Devices Meeting (IEDM), pp. 16-5. IEEE, 2021. The dissertation author was the first author of this paper.

Chapter 6, is a reprint of the material that is under submission and to appear in publication as **Yuhan Shi**, Sangheon Oh, Javier del Valle, Pavel Salev, Ivan K. Schuller, Duygu Kuzum. “Integration of Ag-CBRAM Crossbars and Mott-ReLU Neurons for Efficient Implementation of Deep Neural Networks in Hardware.” The dissertation author was the first author of this paper. Chapter 6, in part, is also a reprint of Sangheon Oh, **Yuhan Shi**, Javier Del Valle, Pavel Salev, Yichen Lu, Zhisheng Huang, Yoav Kalcheim, Ivan K. Schuller, and Duygu Kuzum. "Energy-efficient Mott activation neuron for full-hardware implementation of neural networks." Nature nanotechnology 16, no. 6 (2021): 680-687. The dissertation author was the primary author of this paper.

VITA

2016	Bachelor of Science in Electrical Engineering, Rensselaer Polytechnic Institute
2018	Master of Science in Electrical Engineering (Applied Physics), University of California San Diego
2023	Doctor of Philosophy in Electrical Engineering (Machine Learning and Data Science), University of California San Diego

ABSTRACT OF THE DISSERTATION

Neuro-inspired Computing Using Emerging Non-Volatile Memories

by

Yuhan Shi

Doctor of Philosophy in Electrical Engineering (Machine Learning and Data Science)

University of California San Diego, 2023

Professor Duygu Kuzum, Chair

Data movement between separate processing and memory units in traditional von Neumann computing systems is costly in terms of time and energy. The problem is aggravated by the recent explosive growth in data intensive applications related to artificial intelligence. In-memory computing has been proposed as an alternative approach where computational tasks can be performed directly in memory without shuttling back and forth between the processing and memory units. Memory is at the heart of in-memory computing. Technology scaling of mainstream memory technologies, such as static random-access memory (SRAM) and Dynamic random-access memory (DRAM), is increasingly constrained by fundamental technology limits. The recent research progress of various emerging nonvolatile memory (eNVM) device technologies, such as resistive random-access memory (RRAM), phase-change memory (PCM), conductive bridging random-access memory (CBRAM), ferroelectric random-access memory (FeRAM) and spin-transfer torque magnetoresistive random-access memory (STT-MRAM), have drawn tremendous attentions owing to its high speed, low cost, excellent scalability, enhanced storage density. Moreover, an eNVM based crossbar array can perform in-memory matrix vector multiplications in analog manner with high energy efficiency and provide potential opportunities for accelerating computation in various fields such as deep learning, scientific computing and computer vision. This dissertation presents research work on demonstrating a wide range of emerging memory device technologies (CBRAM, RRAM and STT-MRAM) for implementing neuro-inspired in-memory computing in several real-world applications using software and hardware co-design approach.

Chapter 1 presents low energy subquantum CBRAM devices and a network pruning technique to reduce network-level energy consumption by hundreds to thousands fold. We showed low energy (10×-100× less than conventional memory technologies) and gradual switching characteristics of CBRAM as synaptic devices. We developed a network pruning algorithm that can be employed during spiking neural network (SNN) training to further reduce the energy by 10×. Using a 512 Kbit subquantum CBRAM array, we experimentally demonstrated high recognition accuracy on the MNIST dataset for digital implementation of unsupervised learning.

Chapter 2 presents the details of SNN pruning algorithm that used in Chapter1. The pruning algorithms exploits the features of network weights and prune weights during the training

based on neurons' spiking characteristics, leading significant energy saving when implemented in eNVM based in-memory computing hardware.

Chapter 3 presents a benchmarking analysis for the potential use of STT-MRAM in in-memory computing against SRAM at deeply scaled technology nodes (14nm and 7nm). A C++ based benchmarking platform is developed and uses LeNet-5, a popular convolutional neural network model (CNN). The platform maps STT-MRAM based in-memory computing architectures to LeNet-5 and can estimate inference accuracy, energy, latency, and area accurately for proposed architectures at different technology nodes compared against SRAM.

Chapter 4 presents an adaptive quantization technique that compensates the accuracy loss due to limited conductance levels of PCM based synaptic devices and enables high-accuracy SNN unsupervised learning with low-precision PCM devices. The proposed adaptive quantization technique uses software and hardware co-design approach by designing software algorithms with consideration of real synaptic device characteristics and hardware limitations.

Chapter 5 presents a real-world neural engineering application using in-memory computing. It presents an interface between eNVM based crossbar with neural electrodes to implement a real-time and high-energy efficient in-memory spike sorting system. A real-time hardware demonstration is performed using CuOx based eNVM crossbar to sort spike data in different brain regions recorded from multi-electrode arrays in animal experiments, which further extend the eNVM memory technologies for neural engineering applications.

Chapter 6 presents a real-world deep learning application using in-memory computing. We demonstrated a direct integration of Ag-based conductive bridge random access memory (Ag-CBRAM) crossbar arrays with Mott-ReLU activation neurons for scalable, energy and area efficient hardware implementation of DNNs.

Chapter 7 is the conclusion of this dissertation. The future directions of in-memory computing system based on eNVM technologies are discussed

Chapter 1. Neuro-inspired Unsupervised Learning and Pruning with Subquantum CBRAM Arrays

1.1 Abstract

Resistive RAM crossbar arrays offer an attractive solution to minimize off-chip data transfer and parallelize on-chip computations for neural networks. Here, we report a hardware/software co-design approach based on low energy subquantum conductive bridging RAM (CBRAM®) devices and a network pruning technique to reduce network level energy consumption. First, we demonstrate low energy subquantum CBRAM devices exhibiting gradual switching characteristics important for implementing weight updates in hardware during unsupervised learning. Then we develop a network pruning algorithm that can be employed during training, different from previous network pruning approaches applied for inference only. Using a 512 kbit subquantum CBRAM array, we experimentally demonstrate high recognition accuracy on the MNIST dataset for digital implementation of unsupervised learning. Our hardware/software co-design approach can pave the way towards resistive memory-based neuro-inspired systems that can autonomously learn and process information in power-limited settings.

1.2 Introduction

Inspired by the biological neural networks giving rise to human intelligence, artificial neural networks [1] have revolutionized numerous computer vision [2, 3] and speech recognition [4, 5] tasks. Their near-human performance has been widely leveraged in various applications, including automated systems [6], aerospace and defense [7], health care [8], and home assistance devices [9]. However, training of neural networks requires substantial computing power and time due to the iterative updates of massive number of network parameters. For example, today's advanced neural network algorithms require training times ranging from days to weeks and use carefully organized datasets consisting of millions of images to recognize objects such as animals or vehicles [10-12], while it only takes a few repetitions for a two-year-old toddler to identify these accurately and effortlessly [13]. Another example is AlphaGo, an advanced neural network trained for playing the board game Go against world champions, requiring 1920 CPUs and 280 GPUs and consuming hundreds of kilowatts per game [14]. The human brain, which can perform the exact same task, is 30,000 times more efficient, only consuming power on the order of 10W [13, 15]. High energy consumption and extensive training time have been the major limitations for widespread adoption of neural networks at every scale – from mobile devices to data centers. The need for back-and-forth data transfer between the memory and processor in conventional computing systems based on von Neumann architecture is one of the major causes of high energy consumption during neural network computations. To address this major architectural drawback, on-chip memory storage and in-memory computing solutions using resistive switching memory arrays have been proposed to perform storage and computing at the same location. Non-volatile memory-based synaptic devices such as phase change synapses (PCM) [16, 17], Ag-based conductive bridging synapses (CBRAM) [18], and resistive RAM synapses (RRAM) [19-21] have been investigated for implementing synaptic weight updates during neural network operation. The synaptic arrays using memristors have also been widely used in energy efficient implementation of unsupervised learning [22-25] and MNIST classification [26-34] in the past.

On a separate front, the pruning algorithm [35, 36] inspired from neuroscience [37] has been suggested towards reducing network level energy consumption and time by settings the low valued weights to zero. However, these methods were mostly applied on the trained networks [35, 36]. Pruning during training by backpropagation was previously employed in literature to prevent overfittings [38, 39]. Yet, there is no systematic study showing how pruning can address the energy consumption and excessive training time problems during the training in hardware.

In order to overcome the energy consumption challenge, incremental improvements in devices or algorithms alone will not be sufficient. Therefore, in this work, we focus on a hardware/software co-design approach that combines the advances in low-power device technologies with algorithmic methods to reduce the energy consumption during neural network training. First, we experimentally investigate and characterize the gradual conductance change characteristics of subquantum CBRAM devices, targeting implementation of neural network training in hardware. We show that the subquantum CBRAM devices can achieve gradual switching using stepwise programming and they can be directly programmed into any arbitrary level by controlling wordline (WL) voltage. Then we develop a spiking neural network (SNN) model for unsupervised learning and evaluate its performance by simulations for both analog and digital hardware implementations. In order to improve network level efficiency, we introduce a pruning algorithm carried out during the training and investigate its limits and performance through software simulations. Different from previous algorithmic approaches employing pruning on already trained networks [35, 36], our neuro-inspired pruning method is applied during the network training to minimize the energy consumption and training time. Combining the energy-efficient subquantum CBRAM devices and the pruning technique, we experimentally demonstrate highly energy efficient unsupervised learning using a large-scale (512kbit) subquantum CBRAM array. The hardware/software codesign approach presented in this work can open up new avenues for applications of unsupervised learning on low-power and memory-limited hardware platforms.

1.3 Results

1.3.1 Subquantum synaptic device characteristics

In this section, we investigate device characteristics of subquantum CBRAM relevant to the general context of neural network operation. We explore gradual switching capability of subquantum CBRAM for implementation of different biological or non-biological weight update rules. For CBRAM devices, the 1-atom conductance (G_{1atom}), which corresponds to the conductance (G) of a filament just one atom "wide" at its thinnest point, is a critical parameter affecting energy consumption and filament stability (retention) [40]. G_{1atom} is on the order of the fundamental conductance $G_0 = 2e^2/h \approx 80\mu S$ for CBRAM cells based on filament metals such as Ag and Cu, so typical programming voltages of about 1-3V yield a minimum programming current (i.e., to form a filament just 1 atom “wide”) of $I_{\text{prog}} \approx G_0$ (1V – 3V) = 80-240μA, resulting in high energy consumption in the range from about 1pJ to 100pJ for commonly used programming pulse durations (10ns to 100ns) (Supplementary Table1.S1). Subquantum CBRAM cells reduce programming energy and improve filament stability (Figure 1.1a) by utilizing filaments comprising a semiconductor or semimetal (at least at their thinnest spot, which dominates the resistance) [40]. A subquantum CBRAM memory cell utilizing tellurium (Te), an elemental semiconductor with a band gap of 0.3eV [41], which has a 1-atom conductance deduced [40] to be $G_{1atom} = 0.03G_0$, is shown in Figure 1.1b. With a much lower G_{1atom} than Ag or Cu and with write/erase speeds as low as about 10ns (Supplementary Figure 1.S1), such subquantum CBRAM cells can consume as little as about 0.2pJ ($I_{prog} \approx 0.03G_0 \times (1V - 3V) \approx 2.4\text{-}7\mu A$ and $E = I_{prog} \times V_{prog} \times \textit{pulse duration} = 7\mu A \times 3V \times 10ns = 0.2pJ$) when programmed to their 1-atom limit. This is an order of magnitude lower than for metal filament-based devices programmed to their corresponding 1-atom limit (Supplementary Table1.S1). The retention of the subquantum CBRAM device is shown in Supplementary Figure 1.S2 and is discussed in Supplementary Note 1.

Figure 1.1b shows a cross-section TEM of a subquantum CBRAM cell, fabricated using Ta as the cathode material, sputtered amorphous Al_2O_3 as the insulating layer, and sputtered amorphous ZrTe as the anode material. The array (Figure 1.1b) containing the subquantum CBRAM device has one-transistor one-resistor (1T1R) structure, which provides access to individual cells. I-V characteristics of subquantum CBRAM cells measured by a typical double DC sweep exhibit bipolar characteristic (Figure 1.1c). In the positive regime, a voltage bias is applied to the anode and swept from 0V to +3V with step size 5mV. The resistance of the cell was switched from a high resistance state to a low resistance ON-state. This process is suggested [40] as inducing an electrochemical replacement reaction wherein Te is liberated from the anode by O from the oxide layer. In the negative regime, reversing the polarity of the voltage will break the filament and switch the cell back to a high resistance OFF-state. The resistance can be read without disturbing the state of the cell by applying a small voltage (~100mV) of either polarity. These two distinct states are utilized in memory applications to store binary information. On the other hand, a gradual, analog-like conductance change has been suggested as a requirement for implementation of synaptic plasticity and learning [42]. Gradually increasing and decreasing device conductance is equivalent to long-term potentiation (LTP) and long-term depression (LTD) of synapses in the brain, which are two major forms of synaptic plasticity. LTP and LTD allow for fine synaptic weight updates during network training. Subquantum CBRAM cells can potentially provide more gradual changes in conductance than metal filament-based cells since during programming G tends to increase in increments of ~G_{1atom}, which for Te is an order of magnitude smaller than for metals.

We investigate general gradual programming characteristics of subquantum CBRAM cells using two different methods. Controlling WL voltage allows to change programming current values to program the CBRAM devices to different conductance levels, as this property of resistive memories has been studied before. Figure 1.2a shows gradual switching of a subquantum CBRAM cell by application of stepwise voltage pulses applied to the WL with an increasing step of 10mV for conductance increase and 4mV for conductance decrease over many cycles. Subquantum CBRAM cells can provide linear weight tuning for both LTP and LTD (Figure 1.2a, as shown by linear trend lines). The linearity of the weight tuning was previously reported to be important for implementation of various operations and achieving high accuracy in artificial neural network implementations with resistive memory devices [43, 44]. Stepwise gradual programming of subquantum CBRAM synapses (Figure 1.2a) can be used to implement various forms of learning and plasticity. As representative examples, Supplementary Figure 1.S3 shows two different forms of biological spike-timing-dependent plasticity (STDP) [16, 42, 45] implemented with subquantum CBRAM synapses. Symmetric plasticity Supplementary Figure 1.S3a can be employed for associative learning and recall [16], and asymmetric plasticity (Supplementary Figure 1.S3b) can be used to transform temporal information into spatial information for sequence learning [16]. The STDP implementation is discussed in Supplementary Note 2.

Alternative to stepwise programming, the subquantum CBRAM cells can also be directly programmed into an arbitrary conductance state by controlling the WL voltage without being bound to a particular sequence of states. Figure 1.2b shows a sequence of programming operations in which the WL voltage increases with step size 20mV followed each time by an erase operation. This offers flexibility for implementing weight update rules of greater complexity. Supplementary Figure 1.S4 shows that the nonlinear weight update rule we used can be greatly represented by the device conductance change using this WL voltage modulation.

In order to implement neural network training with 1T1R resistive memory arrays, synaptic weights can be represented in either binary (digital) or analog manners [46]. For digital implementation, N binary 1T1R cells are grouped to represent one synaptic weight (Figure 1.2c) and each cell is programmed to high or low conductance states, providing N-bit weight precision in a binary format. For analog implementation, the cells can be arranged into a pseudo-crossbar array and synaptic weights are stored in the form of multi-level conductances (Figure 1.2d) [46]. As shown in the measurement results presented in this section, the subquantum CBRAM devices are capable of both digital and analog implementations. The tradeoff between analog and digital implementations in terms of energy consumption, latency and area will be further discussed in the context of our neural network model in the Neural network algorithm for unsupervised learning section.

1.3.2 Neural Network Algorithm for Unsupervised Learning

Here, we investigate neuro-inspired spiking neural network (SNN) configurations and implement unsupervised learning on 1T1R CBRAM synaptic arrays to classify MNIST handwritten digits, which consists of 60,000 training samples and 10,000 test samples. Different from other neural networks trained using back propagation, neuro-inspired SNNs use event-based and data-driven updates to reduce redundant information processing to gain efficiency and minimize energy consumption, making them ideal for hardware implementations [47-49]. Neuromorphic hardware platforms based on SNNs have already been demonstrated and employed in various applications of neural networks [48-50]. To reduce the network size, we crop some black background pixels from the full image of 784 (28 × 28) pixels. Therefore, our network contains 397 input neurons with a bias term and 500 output neurons, resulting in 199,000 synaptic weights (Figure 1.3a). SNNs encode information between input and output neurons using spike trains. The firing frequency of the Poisson spike trains generated by the input neurons scales linearly with respect to the pixel intensity (0 Hz for intensity value of 0 and 200 Hz for intensity value of 1). The output neurons integrate all the inputs to generate output spike trains based on a probabilistic winner-take-all (WTA) mechanism (See Methods section for more details) [51, 52]. The synaptic weights of the firing output neuron are updated by a simplified STDP rule shown in Figure 1.3b during training. STDP rule that modulates weights based on the timing of input and output spikes: if the time difference between the post-spike and pre-spike is less than 10ms, the synaptic weight is updated via the LTP rule, otherwise, it is updated via the LTD rule. Here, the LTD update is a constant weight decrease and the LTP update depends on the current weight state of the synapse with an exponentially decaying function shown in Figure 1.3c. Exponential LTP updates will guarantee that the weights converge to the upper bound of 1. For LTD updates, the lower bound of the weight is clipped to -1. Overall, these rules result in weight values that are in the range of -1 to 1, allowing for a feasible and practical hardware implementation. During the training, the weights are adjusted incrementally based on the STDP rule so that output neurons fire selectively for a certain class in the dataset. Before training, output neurons exhibit random spiking response to the presented digits (Figure 1.3a). However, after training, output neurons fire selectively during the presentation of specific samples learned during the training (Figure 1.3a). Figure 1.3d and e show MNIST digit classification accuracy as a function of training epoch and neuron number. Training more than 3 epochs (Figure 1.3d) or increasing the output neuron number beyond 500 (Figure 1.3e) do not result in noticeable increase in accuracy, similar to what has been reported for single layer spiking neural networks in literature [53]. Therefore, we choose to use 500 neurons and 3 epochs for the training in our analysis. The algorithm we used for unsupervised learning is summarized in Supplementary Figure 1.S5. After training is complete, the training dataset is presented again to assign neuron labels to the output neurons by determining which digits provoked the highest average firing rate for each of the output neurons [53]. We predict the labels from the test set, which consists of 10,000 new samples from the MNIST test set, based on the same framework used during training to find the output neuron with the highest average firing rate for each sample (See Methods section for more details). We simulate our network for the ideal software (64-bit), and our proposed digital (Figure 1.2c) and analog implementations (Figure 1.2d). Table1.1 summarizes classification accuracy for all three cases. For the ideal software implementation, it is important to point out that ~94% accuracy is already very high for unsupervised learning with SNN [53]. Increasing the accuracy further to the levels of deep neural networks will definitely require introducing supervision to the SNN [54-56]. For digital implementation, we use 8-bit digital synapses and the weights are quantized to 256 levels distributed evenly between [-1, 1-2/256]. For analog implementation, we directly use conductance values (Figure 1.2a) from device characteristic in our simulation to perform weight update during training. Neural network weights in the range of [-1, 1] can be mapped to device conductance using a linear transformation, as explained in the Methods section. Our results suggest that 8-bit digital implementation achieves comparable recognition accuracies with ideal software case and analog implementation has slightly lower accuracy due to the limited conductance states exhibited by each CBRAM synapse.

In order to compare the digital (Figure 1.2c) and analog synaptic core (Figure 1.2d), we develop a SNN platform for NeuroSim [46] (SNN+NeuroSim). NeuroSim is a C++ based simulator with hierarchical organization starting from experimental device data and extending to array architectures with peripheral circuit modules and algorithm-level neural network models^{46}. We use SNN+NeuroSim to perform circuit-level simulations (Table1.2) to estimate the energy, latency and area for the digital and analog implementations using the experimental data measured from subquantum CBRAM devices (Figure 1.2). The left two columns of Table1.2 show benchmarking results for analog synaptic core and 6-bit digital synaptic core. 6-bit precision is chosen to match the number of levels that can be achieved by gradual programing of subquantum CBRAM devices for the analog implementation. However, in order to achieve a recognition accuracy above 90%, 8-bit precision is required. Therefore, we include the third column, showing the results for 8-bit digital case, which is also used in the hardware demonstration (Hardware demonstration of pruning during training section). The best performing metrics are highlighted in yellow. As shown in the Table, the 6-bit digital scheme has better accuracy, shorter latency and lower energy consumption. On the other hand, the analog scheme occupies smaller chip area. Therefore, the benchmarking results suggest that digital implementation could be more advantageous in terms of energy consumption and latency for hardware implementation of on-line learning using subquantum CBRAM array.

1.3.3 Pruning During the Training

Neural network pruning algorithms have been very effective to reduce the time and energy consumption during inference by removing unimportant weights. Conventional pruning methods [35, 36], which we also refer to as pruning in this work, set the low valued weights to zero. However, these methods are not suitable to be directly applied to the network learning algorithms that can produce non-zero centered weight distributions. In such situations, zero-valued weights are also important so that arbitrarily setting pruned weights to zero may affect accuracy. Additionally, conventional pruning mostly targets the networks which have already been trained. Therefore, the issues of excessive time and energy consumption during training remain unaddressed. To address both of these, we develop a method as an extension of pruning, which we refer to as soft-pruning [57]. Instead of completely removing the weights from network by setting them to zero, soft-pruning sets the values of pruned weights to a constant non-zero value and prevents them from being updated during the rest of the training while allowing them to still participate in the inference step after the training. Therefore, pruning weights during training helps to significantly reduce the number of weight updates, minimizing computation and energy consumption. To decide when to prune weights during the training, we determine if the output neurons are trained enough to recognize a class from the dataset. We quantify this by counting the occurrences of consecutive output spikes (Supplementary Figure 1.S6) from a single output neuron. The corresponding time interval between consecutive output spikes follows a Poisson distribution. Once an output neuron sees p occurrences of consecutive spikes during the training, a certain percentage of its weights are pruned to their lowest possible value (in our case, W_{min} = -1). The pruning algorithm is summarized in Supplementary Figure 1.S7. Potential hardware implementations of this pruning algorithm are discussed in Supplementary Note 3 and associated overheads estimation in area, energy and latency via simulation (SNN+NeuroSim) are shown in Supplementary Figure 1.S8 and Supplementary Table1.S2. We investigate the distribution of weights in the SNN before and after soft-pruning along with a baseline control case, where pruning is not employed (no pruning) (Figure 1.4a). Simulation of recognition accuracy for different *p* values in Figure 1.4b suggests that *p* = 10 provides the highest accuracy even for very large pruning percentages (up to 80%). Visualization of weights from ten representative output neurons (bottom row of Figure 1.4a) shows that foreground pixels (the digits) correspond to higher weight values on the distributions, and background pixels (background of the digits) correspond to lower weight values for no pruning case (weights visualization for all output neurons can be found in Supplementary Figure 1.S9). The Supplementary Videos 1.1 and 1.2 show the development of the output neurons' weights during the training for both soft-pruning and no pruning cases. Before pruning, the distributions indicate that the weight updates have been the same for both cases. Figure 1.4c compares recognition accuracy for as a function of pruning percentage for soft-pruning and pruning during the training, in comparison to pruning at the end of training for both cases. The recognition accuracy for pruning falls below ~90% for ~40% pruning percentage. In contrast, soft-pruning maintains high classification accuracy (~90%) even up to ~75% pruning percentage (Figure 1.4c) The accuracy improvement achieved by the soft-pruning algorithm can be understood from the following two perspectives. First, since the pruned weights are set to -1 instead of being completely removed from the network, they still participate in the inference. Pruning the unimportant weight to -1 effectively decreases the membrane potential of output neurons, which helps to prevent false positive spikes. Second, the soft-pruning algorithm preserves the original weight distribution. As shown in Figure 1.4a, the final distribution of learned weights clearly consists of two distinct parts which correspond to the foreground and background pixels of the image. The weights concentrated at -1 are associated with the background pixels, while the remaining weights centered around zero accounts for the foreground pixels. Soft-pruning sets pruned weights to -1, grouping them with the background pixels. On the contrary, pruning sets pruned weights to 0, which is in the range of weights that are associated with foreground pixels; this significantly changes the shape of foreground weight distributions, which leads to the accuracy degradation. Our soft-pruning method achieves high recognition accuracy for extensively pruned networks, offering superior energy efficiency during training for hardware implementations of unsupervised learning.

1.3.4 Hardware Demonstration of Pruning During Training

In order to implement unsupervised learning and pruning during the training on the hardware, we used a 512kbit subquantum CBRAM chip fabricated in a 130nm Cu back end of line (BEOL) process (Figure 1.1b). The array has a 1T1R architecture, which provides access to individual cells. Although each individual cell in our array has gradual conductance switching capabilities as demonstrated in Figure 1.2 a and b, the digital implementation offers smaller energy consumption and shorter latency which is important for online learning as shown in Table1.2. Furthermore, analog approach with varying amplitude pulses requires peripheral neuron circuits to produce non-identical pulses with fine grained duration [58],[59]. Therefore, we choose to use digital implementation for hardware demonstration. We uniformly quantize the weights and map them onto the CBRAM array using an 8-bit digital representation between W_{min} = -1 and W_{max} = 1 (Details are explained in the Methods section), as our simulations have shown high recognition accuracy for 8-bit representation. Each weight is approximated to its closest quantized level when updating. Using our proposed network size to implement 10-digits MNIST classification requires at least 199,000 $\times$ 8 = 1.5 Mbit array. Given our array size limitation of 512kbit, we reduce the network size to 395 input and 10 output neurons to classify three classes (“0”, “3”, and “4”) from MNIST. Figure 1.5a shows recognition accuracy as a function of bit precision in the range of 5 to 12 bits, corresponding to quantization to 2^5 and 2^{12} discrete levels. The recognition accuracy stays relatively constant down to 8 bits but shows a steep decrease for bit precisions less than 7 bits. For hardware implementation of online unsupervised learning, the weights are updated on the subquantum CBRAM array at run-time. Figure 1.5b shows experimentally obtained weight maps from the subquantum CBRAM array for the 10 output neurons for the no pruning and 50% soft-pruning cases after unsupervised online training with 1,000 MNIST samples. Weight update history during the online training process is investigated. Supplementary Figure s1.S10a and b show the number of switching cycles of every bit in CBRAM cells for no pruning and 50% soft-pruning, respectively. Least significant bits (LSB) update more frequently than the most significant bits (MSB) in both cases. For the no pruning case, all bits are constantly updated throughout training, causing extensive energy consumption through programming and erasing of the subquantum CBRAM devices. In contrast, pruning reduces the number of switching cycles for all of the individual bits and the number of cumulative switching cycles as shown in Supplementary Figure 1.S10b and Supplementary Figure 1.S10c, respectively. Figure 1.5c shows the accuracy for the pruning and no pruning cases for the experimental results obtained with the subquantum CBRAM array as a function of training set size. This hardware implementation achieves 93.19% accuracy, which is very close to the accuracy for no pruning (93.68%) and the 8-bit and 64-bit ideal software implementations. Figure 1.5d shows the number of bit updates by device updates vs. training set size, where the data for the first 1,000 samples are obtained from the hardware implementation, and the rest is computed using software simulations. The number of bit updates for both cases is identical until pruning starts. After all output neurons are pruned, the 50% pruned network has around twofold reduction in the number of bit updates compared to the no pruning case. Although our hardware demonstration focuses on 50% pruning, our simulations suggest that pruning percentages up to 80% can be implemented to further increase energy savings.

1.4 Discussion

The performance of our hardware implementation for unsupervised learning is far superior to the previous state-of-the-art unsupervised learning of MNIST dataset with synaptic devices in terms of recognition accuracy, energy consumption per programming, number of weight updates in training, and network size (Supplementary Table1.S3). For energy consumption per programming event, subquantum CBRAM is two to three orders of magnitude more efficient than transistor-based devices (Supplementary Table1.S1) and shows the lowest energy consumption among RRAM based synaptic devices (Supplementary Table1.S3). Our pruning algorithm can reduce the number of parameter updates significantly and lead to ~20× less number of parameter updates compared to previous reports (Supplementary Table1.S3). Combining device level energy savings provided by subquantum CBRAM with network level energy savings by pruning may lead up to two orders of magnitude reduction in total energy consumption for hardware implementation of weight updates during unsupervised learning.

Compared to other software simulations in the literature (Supplementary Table1.S4), our network achieves a high classification accuracy on MNIST dataset using the lowest number of neurons and synapses and a low-complexity one-layer architecture that can be easily mapped onto 1T1R or crossbar arrays. Supplementary Table1.S5 compares hardware demonstration of our pruning method with other software approaches of pruning in terms of energy savings and accuracy loss. Our method provides comparable energy savings with minimal accuracy loss, while being the only method, which can be applied during the training. Last but not least, our work presents the demonstration of mapping of pruning onto a hardware platform.

We demonstrate unsupervised learning using an energy efficient subquantum CBRAM array. Synaptic pruning is implemented during the training and mapped onto hardware to reduce energy consumption while maintaining a classification accuracy close to ideal software simulations. We show that subquantum CBRAM cells are capable of gradual and linear conductance changes desirable for implementing online training in hardware and can be directly programmable into different conductance states indicating their potential for implementing a broad range of weight update rules for neuromorphic applications. Following a software/hardware co-design approach, we develop a neuro-inspired synaptic pruning method to significantly reduce the number of parameter updates during neural network training. Low-energy subquantum CBRAM devices combined with the network-level energy savings achieved by pruning can provide a promising path towards realizing AI hardware based on spiking neural networks that can autonomously learn and handle large volumes of data. Our hardware/software co-design approach can also be adapted to other network models to reduce the energy cost in implementing network training in low-power mobile applications.

1.5 Methods

1.5.1 Neural Network Algorithm

Here we describe the network architecture of the SNN including the input and output layers. Then, we explain our training, labeling, and classification procedure for the MNIST dataset. Supplementary Table1.S6 summarizes the parameters used in simulations.

A. Network Architecture

Our SNN is a one-layer network defined by the number of inputs neurons *m*, the number of outputs neurons *n*, and an *m* by *n* weight matrix. Each output neuron is fully-connected to every input neuron. Our SNN has 398 input and 500 output neurons. Our output neurons do not have refractory periods and there is no lateral inhibition between them.

B. Input Layer

We crop each training sample by removing pixels that represent the background in at least 95% of the training samples. Because the pixels have intensity values in the range [0, 1], those with a value of 0 correspond to the background and are thus candidates for removal. After this step, we have 397 input neurons in total by including an additional bias term, which has an input value of 1. The weights associated with this bias input neuron are learned via the same learning rule as the other weights. Each input neuron generates a Poisson spike train X_i whose mean firing rate is determined linearly by the pixel intensity, where a pixel of value 0 corresponds to 0 Hz and a pixel of value 1 leads to 200 Hz. The timing of each spike that is generated by the Poisson process is rounded towards the nearest millisecond, which is the time step of the simulation.

C. Output Layer

The SNN fires an output spike from any given output neuron according to a Poisson process with the specified frequency. The output neuron that fires is chosen from a softmax distribution of the output neurons' membrane potentials as (1) [52]:

$$P(u_k) = \frac{e^{u_k}}{\sum_{k=1}^{N} e^{u_k}} \qquad (1)$$

, where $P(u_k)$ is the softmax probability distribution of the membrane potentials u_k (*k* = 1, …, *N*). *N* is the number of output neurons. We calculate membrane potentials u_k using (2)

$$u_k = \sum_i W_{ki} X_i + b_k \qquad (2)$$

W_{ki} is the weight between input neuron *i* and output neuron *k*. X_i is the spike train generated by input neuron *i* and b_k is the weight of the bias term.

D. Training

The SNN displays each input sample for the first 40 ms of a 50 ms presentation period, and thus the input spikes for a given sample only occurs in this 40 ms window. Figure 1.3a shows an example of the input spiking activity for the duration of four training samples. We use the whole training set, which contains 60,000 samples, and train for three epochs. It is important to note that 50 ms is a virtual simulation parameter along with the firing frequency chosen for generating input spikes. In the real hardware implementation, the presentation time of one image can be much shorter than 50 ms as long as enough number of input spikes are generated. The weights are updated via STDP rule shown in Figure 1.3b. The LTP and LTD rules are detailed in (3) and (4) respectively,

$$\Delta W_{LTP} = a \times e^{-b\,(W+1)} \qquad (3)$$

, where *a* and *b* are parameters that control the scale of the exponential, and W is the current weight value. The result ΔW is the amount of weight update of LTP and it is dependent on current W. LTD is a constant depression in terms of c in (4),

$$\Delta W_{LTD} = -c \qquad (4)$$

E. Labeling

After training is done, we fix the trained weights and assign a class to each neuron by the following steps: First, we present the whole training set to the SNN and record the cumulative number of output spikes N_{ij}, where $i = 1, ..., N$ (N is number of output neurons) and $j = 1, \ldots, M$ (M is number of classes). Then, for each output neuron i, we calculate its response probability Z_{ij} to each class j using (5). Finally, each neuron i is assigned to the class that gives the highest response probability Z_{ij}.

$$Z_{ij} = \frac{N_{ij}}{\sum_{j=1}^{M} N_{ij}} \qquad (5)$$

F. Classification

We use the standard test set which contains 10,000 images. We use equation (6) to predict the class of each sample, where S_{jk} is the number of spikes for the *k*th output neuron that are labeled as class *j* and N_j is the number of output neurons labeled as class *j* [53].

$$J = \underset{j}{argmax} \frac{\sum_{k=1}^{N_j} S_{jk}}{N_j} \qquad (6)$$

G. Weight Mapping for Analog Synapse Implementation

The network weights (W) ranging from -1 to 1 are mapped to the device conductance data range from ~1µS to 200 µS, we map the device conductance to the weight range [-1, 1] by using below linear transformation (7),

$$G_{NORM} = \frac{G - \frac{G_{max} + G_{min}}{2}}{\frac{G_{max} - G_{min}}{2}} \quad (7)$$

In Eq. (7), we denote this normalized conductance as G_{NORM}. *G*, G_{max} and G_{min} are extracted from experimental data (Figure 1.2).

1.5.2 Hardware Implementation

For the hardware demonstration of unsupervised learning and pruning shown in Figure 1.5 CBRAM devices are employed as binary synapses. The network contains 395 input neurons (crop using the same method explained in B. Input Layer) and 10 output neurons to classify three classes from MNIST. In 3-digits classification, out of the ~20,000 samples that represent the digits “0”, “3”, or “4” in the entire MNIST dataset, we randomly sample 5,000 to create our training set. We present this training set for one epoch to train our SNN. We form the test set by drawing 10,000 samples from the remaining 15,000 samples. Neurons are implemented using a custom software to program the digital peripheral circuitry of the chip. Weight summation is performed by this program to implement the integrate-and-fire neuron. Weight update values are converted into programming pulses by the peripheral circuitry to update binary weights in the digital implementation. Fixed wordline voltages are used for binary programming of CBRAM devices. We use 8 bits to represent a synaptic weight in the network, where 1 bit is used to represent the sign of the weight value and the other 7 bits stores the absolute weight value. Bit 1 is MSB and bit 7 is LSB. The weight range [-1,1] is first uniformly divided into 256 (2^8) discrete intervals $[-1 + \frac{i}{128}, -1 + \frac{i+1}{128})$, where *i* = 0, …, 255. Then we map the weight whose value lies in the *i*th interval to the *i*th discrete values. For example, the weights between [-1, -0.9921875) are mapped to

00000000, whereas the weights between [-0.9921875, -0.984375) are mapped to 00000001, etc. For the boundary case where the weight takes the value of 1, we map it to 11111111. The weights are updated on the hardware at run-time. We track the weight update history during the online training process (Supplementary Figure 1.S10).

1.6 Acknowledgments

The authors acknowledge support from the Office of Naval Research Young Investigator Award (N00014161253), National Science Foundation (ECCS-1752241, ECCS-1734940) and the UC San Diego Frontiers of Innovation Scholars Program for funding this research. Adesto and CBRAM are Trademarks of Adesto Technologies Corp.

Chapter 1 is a reprint of Yuhan Shi, Leon Nguyen, Sangheon Oh, Xin Liu, Foroozan Koushan, John R. Jameson, and Duygu Kuzum. "Neuroinspired unsupervised learning and pruning with subquantum CBRAM arrays." Nature communications 9, no. 1 (2018): 1-11. The dissertation author was the first author of this paper.

1.7 Figures

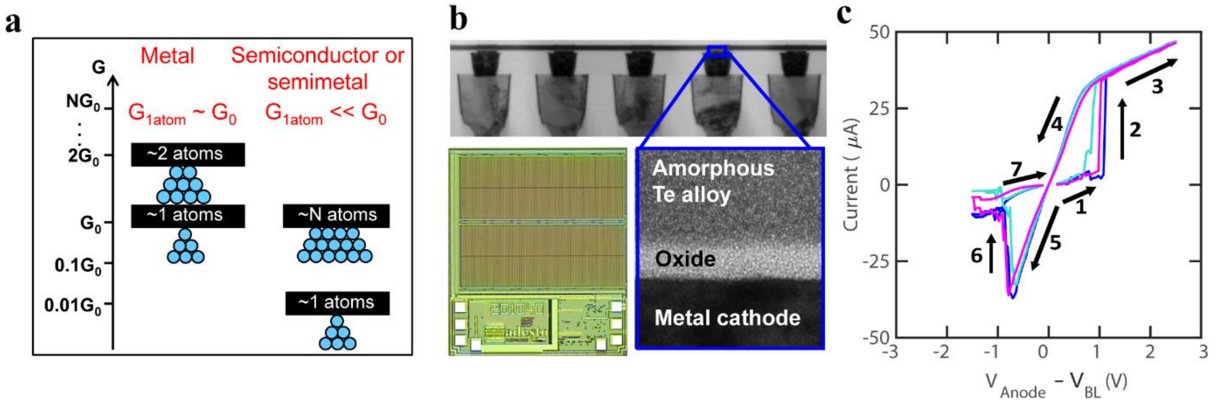


Figure 1. 1 Subquantum CBRAM Characteristics

(a) Semiconductor or semimetal filaments can yield lower conductance than metal filaments of comparable “width”. (b) Subquantum conductive bridging RAM (CBRAM) cell fabricated in a standard 130 nm logic process. Photograph shows 512 kbit subquantum CBRAM chip with one-transistor one-resistor (1T1R) array architecture. Cell cross-section shows amorphous Te alloy as anode, metal as cathode and oxide as switching layer. (c) Example of bipolar current-voltage characteristic of a subquantum CBRAM cell. Directionality of switching is shown in arrows.

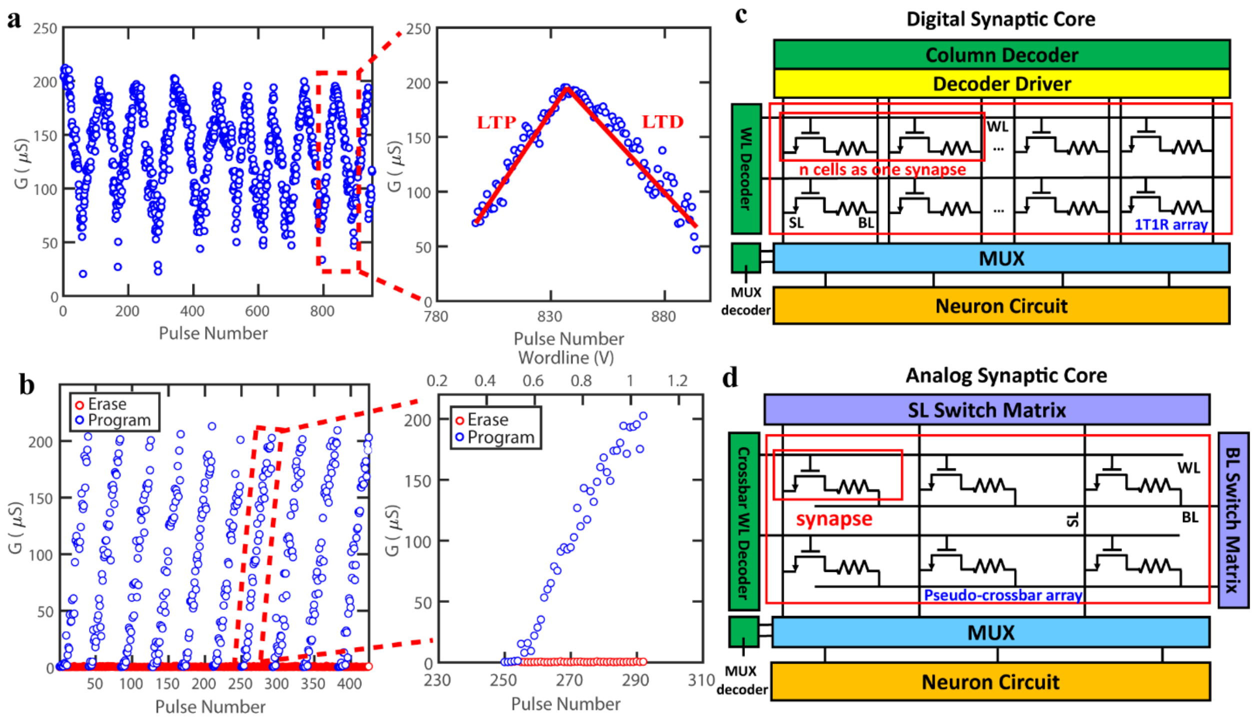


Figure 1. 2 Subquantum CBRAM Gradual Switching and Synaptic Core Architecture

(a) Gradual switching in a subquantum conductive bridging RAM (CBRAM) synapse using stepwise voltage pulses applied to the wordline (WL) (left). Callout window (right) shows one cycle of long-term potentiation (LTP) and long-term depression (LTD). Red lines are added to emphasize linearity of the conductance change. For LTP, anode (AN) =3V, bitline (BL)=0, and WL stepped from 0.8V in increments of 10mV. For LTD, AN=0, BL=WL, and WL stepped from 1.6V in increments of 4mV. (b) Gradual switching in a subquantum CBRAM synapse by WL voltage modulation. The subquantum CBRAM cells are directly programmed into the conductance state by controlling the WL voltage. The Figure (left) shows a sequence of programming operations in which the WL voltage increases with step size 20mV followed each time by an erase operation. Callout window (right) shows conductance versus pulse number and WL voltage for a representative cycle. (c) Digital synaptic core design groups multiple binary one-transistor one-resistor (1T1R) cells along the row as one synapse to represent a synaptic weight with higher precision. WL decoder is used to activate the WL in a row-by-row fashion. Column decoder can select a group of synapses to perform the weight update. The weighted sum is implemented using mux and neuron circuit. The mux is used to share the read periphery circuitry [46]. The neuron circuit which contains sense amplifier, adder and shift register can be used to read out the memory array and accumulate partial weight sum to get the final weighted sum. d, Analog synaptic core uses a single cell with multi-level conductance states to represent one synaptic weight. The crossbar WL decoder can activate all WLs, BL read out the weighted sum results and neuron circuit contains analog-to-digital (ADC) converters convert current to digital outputs. Source line (SL) can be used to perform weight update [46].

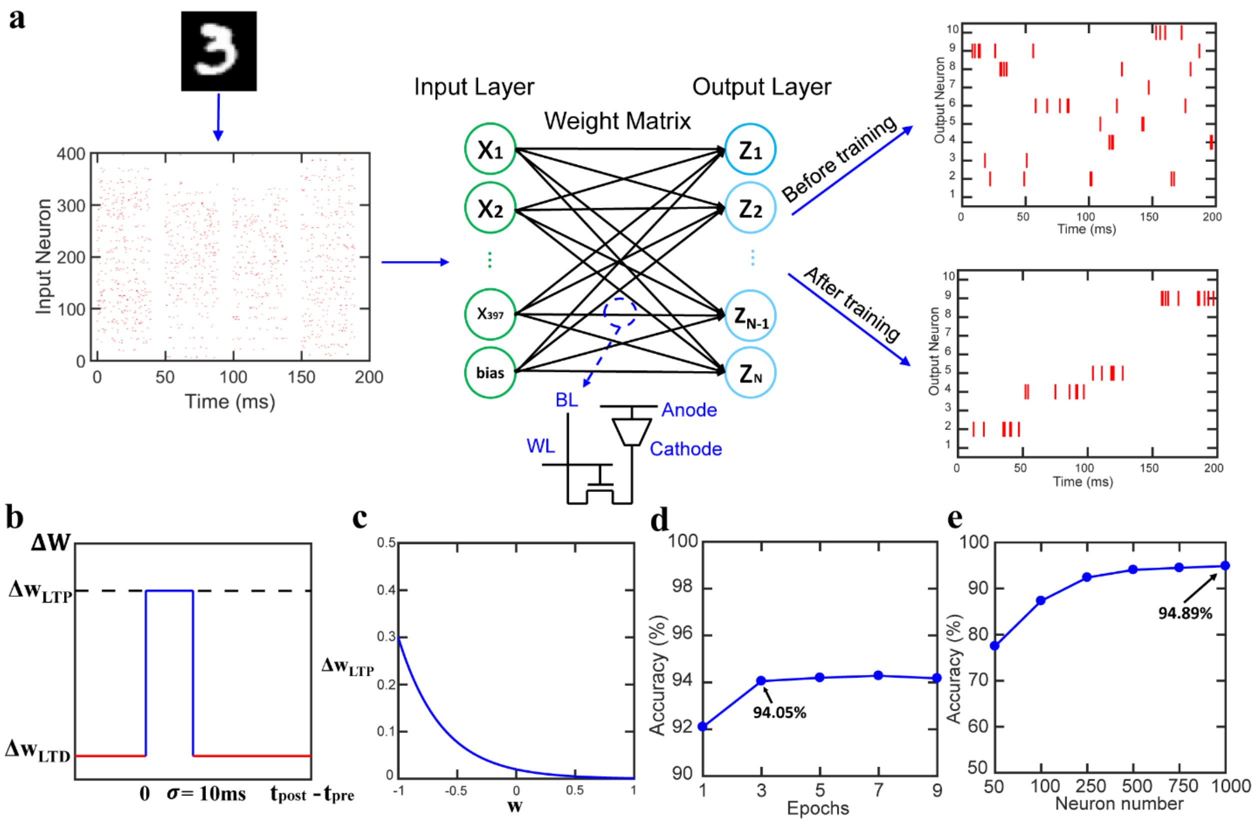


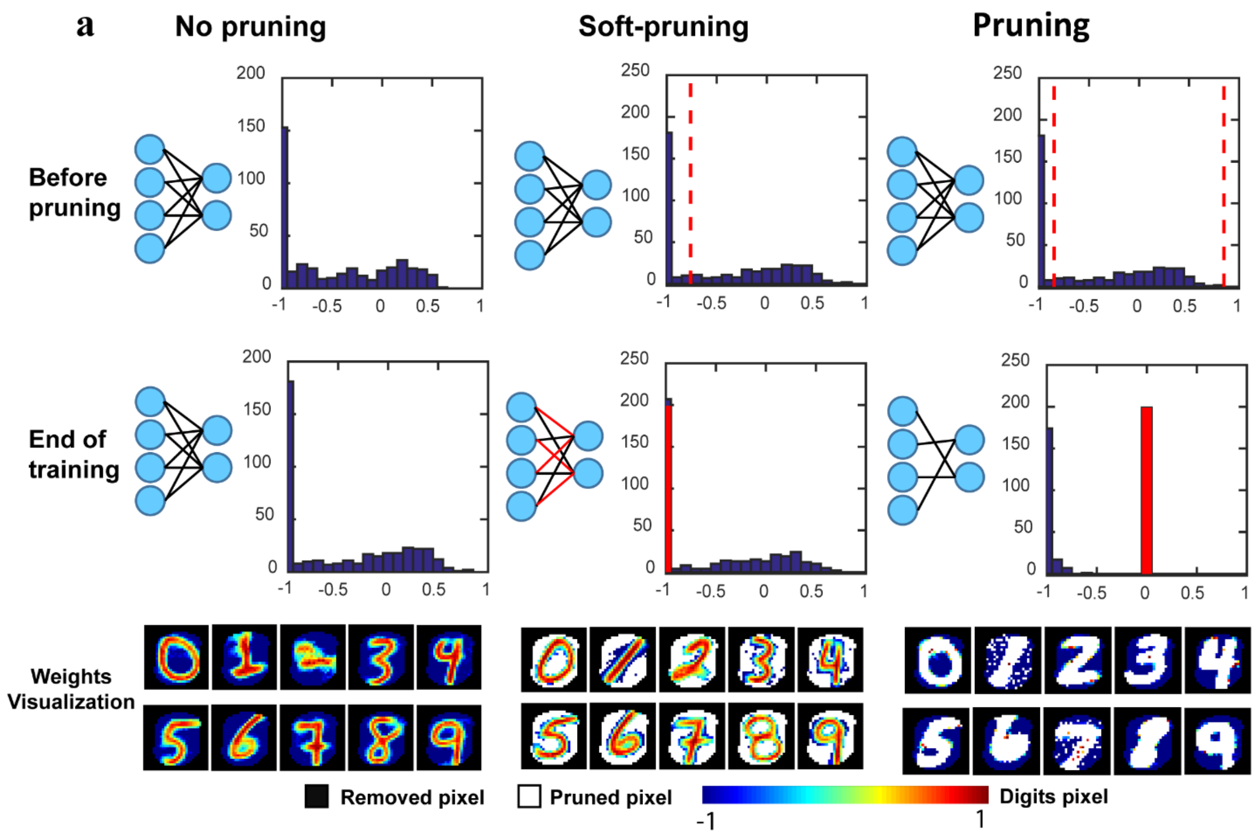
Figure 1. 3 Spiking Neural Network for Unsupervised Learning.

(a) Each input digit contains 28 x 28 = 784 pixels and has been cropped and reduced to 397 pixels. The neural network has 397 input neurons with a bias term and 500 output neurons. Input spike trains of input neurons are generated according to pixel density (from 0 to 1) and then fed to the neural network. Synaptic devices represent weights in the network. top (before training): Random spike activity from representative 10 out of 500 output neurons before learning. bottom (after training): Output spike trains after learning show coordinated selective firing activity as a result of unsupervised learning of digits. (b) Spike-timing-dependent plasticity (STDP) rule showing the 10 ms window for an post-pre spike time difference (t_{post} – t_{pre}) that determines whether a long-term potentiation (LTP) or a long-term depression (LTD) update is performed. If the firing time of an output neuron (t_{post}) is within 10 ms of the firing time of an input neuron (t_{pre}), the weight (synapse) between this input-output neuron pair is updated via LTP. Otherwise, the weight is updated via LTD. (c) The LTP update is an exponentially decaying function that depends on the current weight, and the LTD update is a constant. The exponential LTP update depending on the current weight keeps the weight values within the range [-1, 1]. (d) Recognition accuracy vs. number of training epochs. 3 epochs are used in our network training. (e) Recognition accuracy vs. neuron number. Recognition accuracy does not have noticeable increase when number of output neurons is larger than 500. Therefore, 500 output neurons are used in our network model.

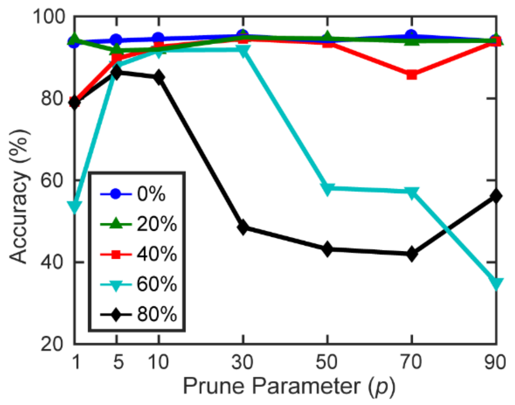
Figure 1. 4 Spiking Neural Network Pruning During Training.

(a) Schematics compare no pruning, soft-pruning and pruning cases. Top two row shows weight histograms of a representative output neuron. For no pruning, the spike-timing-dependent plasticity (STDP) rule results in weights ranging from -1 to 1 at the end of training. For 50% soft-pruning, it prunes weights smaller than the dashed line (weights on the left of the dashed line) to the lowest value -1. 50% Pruning prunes the weights between the two dashed lines, which represent the 50% of the weights that are centered around 0 and sets their values to 0 (red bar). Only unpruned weights continue to be updated until end of training. Bottom row shows weight visualization of all representative 10 out of 500 output neurons for no pruning, 50% soft-pruning and pruning. Soft-pruning allows for the weights to still learn the foreground and background pattern of the input samples while reducing weight update computations during training. Pruning causes the pruned weights to overwhelm the learned weights and results in inaccuracy. (b) Recognition accuracy vs. prune parameter (*p*) for varying pruning percentages. Prune parameter is the criterion to decide when to prune for each neuron during training. c, Recognition accuracy vs. pruning percentage for soft-pruning and pruning performed during training. Soft-pruning during the training performs significantly better than pruning especially for high pruning percentages. The baseline accuracy (no pruning) is 94.05%. The data points are taken in steps of 10%. The parameters used in the simulation are specified in Supplementary Table1.S6.

a


No pruning
Soft-pruning
Pruning
Before pruning
End of training
Weights Visualization
Removed pixel
Pruned pixel
Digits pixel
-1
1

b


Accuracy (%)
Prune Parameter (p)
0%
20%
40%
60%
80%

c

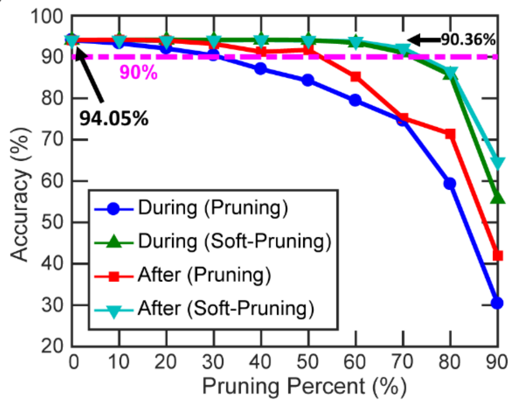

90.36%
90%
94.05%
Accuracy (%)
Pruning Percent (%)
During (Pruning)
During (Soft-Pruning)
After (Pruning)
After (Soft-Pruning)

Figure 1. 5 Hardware Implementation of Unsupervised Learning and Pruning

(a) Recognition accuracy vs. Bit precision. The bit precision levels include 1 bit for representing the sign. 64 corresponds to 64-bit floating point. The accuracy drops below 90% after 8 bits. Test dataset has 10k images. (b) Experimentally measured binary weights from subquantum conductive bridging RAM (CBRAM) synaptic array as a result of training with 1k MNIST digits. Binary weight 1 corresponds to black pixel, which is high resistance state (~1MΩ). Binary weight 0 corresponds to white pixel, which is low resistance state (~10kΩ). The bit precision per weight is 8 bits with one bit used for the sign (+/-). During the training, there is a total of 8,959 weight updating events for output neurons. For no pruning (top), there were 833,889-bit updates. For training with soft-pruning at a 50% pruning rate (bottom), there were 481,921-bit updates. During the training, weights of different neurons are pruned at different times based on their learning level. At the end of training, all 10 neurons' weights have been pruned. Bits corresponding to pruned weights are marked in blue. (c) Recognition accuracy vs. training digits for 50% soft-pruning and no pruning calculated using experimental data from hardware implementation of unsupervised learning with subquantum CBRAM array. The accuracy for pruning is comparable to no pruning. Test dataset has 10k images. (d) Number of bit updates by device updates vs. Training digits/Timesteps with a 50% pruning rate (blue) and without pruning (red). First 1k samples are from hardware implementation of spiking neural network (SNN) using CBRAM array.

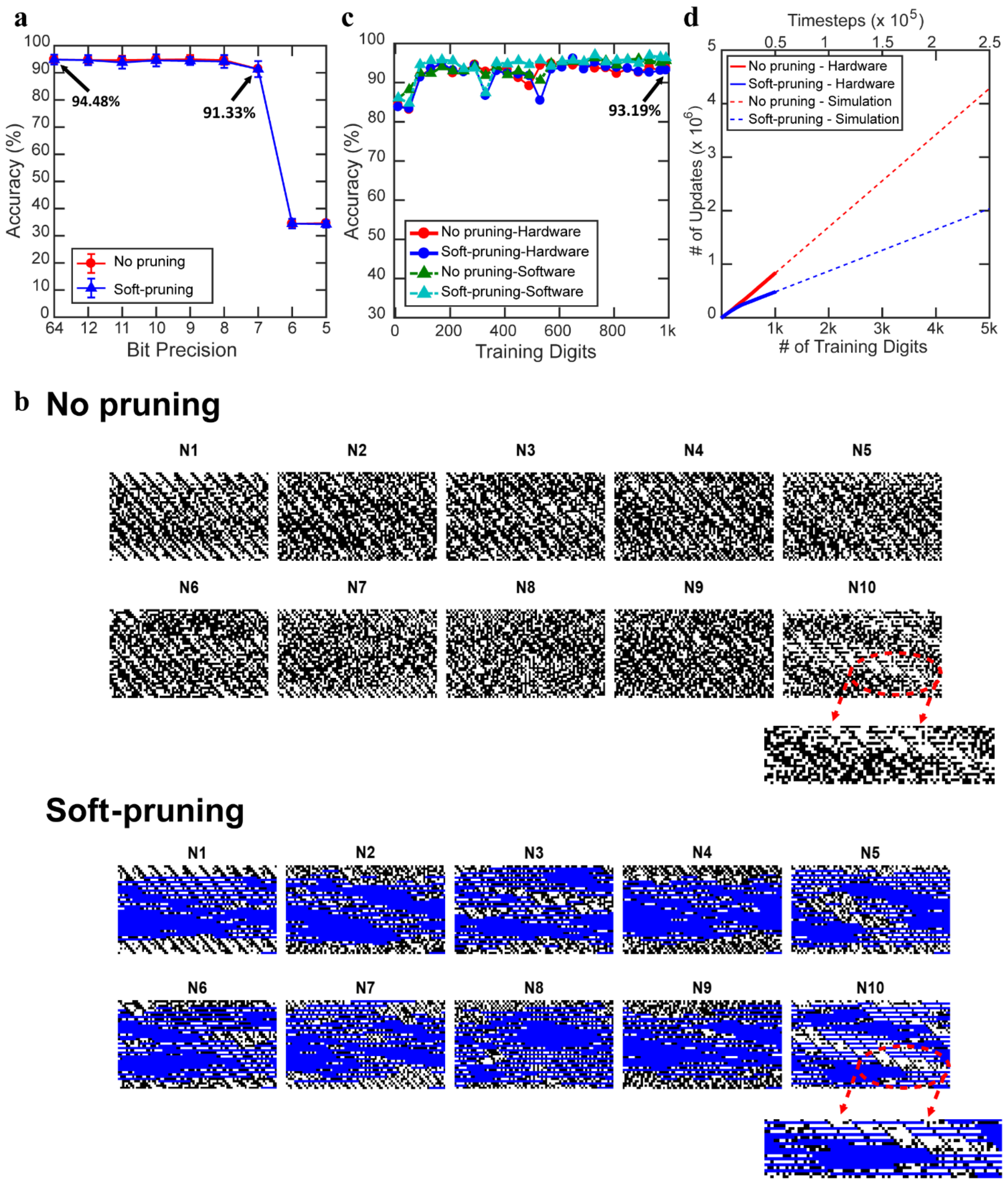

a
Accuracy (%)
Bit Precision
94.48%
91.33%
No pruning
Soft-pruning
c
Accuracy (%)
Training Digits
93.19%
No pruning-Hardware
Soft-pruning-Hardware
No pruning-Software
Soft-pruning-Software
d
Timesteps (x 10^5)
# of Updates (x 10^6)
# of Training Digits
No pruning - Hardware
Soft-pruning - Hardware
No pruning - Simulation
Soft-pruning - Simulation
b
No pruning
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10
Soft-pruning
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10

Table 1. 1 Network Accuracy

Table summarizes the recognition accuracy of 64-bit ideal software simulation, 8-bit digital implementation and analog CBRAM synapses implementation evaluated using our network.

Precision	Accuracy
64-bit	94.05%
CBRAM (Analog)	82%
8-bit (Digital)	92.02%

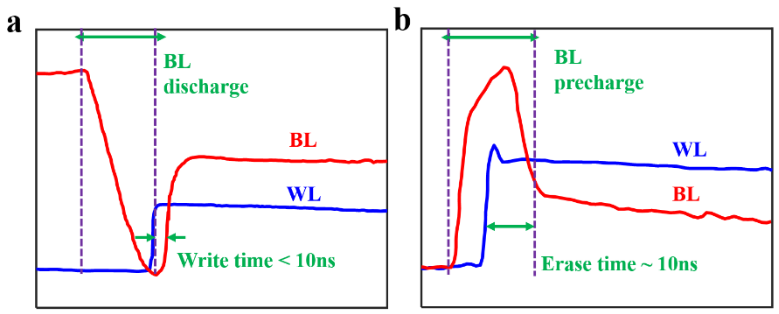
Table 1. 2 Circuit-Level Benchmark Results

Table summarizes circuit-level benchmark results using SNN+NeuroSim for analog synaptic core and digital synaptic core with 6-bit and 8-bit. The simulations are performed for 14nm technology node.

	Analog	**Digital (6-bit)**	**Digital (8-bit)**
Conductance levels	57 levels (~6 bit)	64 levels	256 levels
LTP pulse	0.8V-1.32V/10mV/1us	2V/1us	2V/1us
LTD pulse	1.6V-1.84V/4mV/10us	2V/1us	2V/1us
Accuracy*	82%	85.87%	92.02%
Area (μm^2)	122,77.05	353,97.34	47233.8
Latency* (s)	516	129.72	401.1
Energy* (mJ)	149.4097	62.911	151.977
Leakage Power (μW)	53.78	54.14	58.99

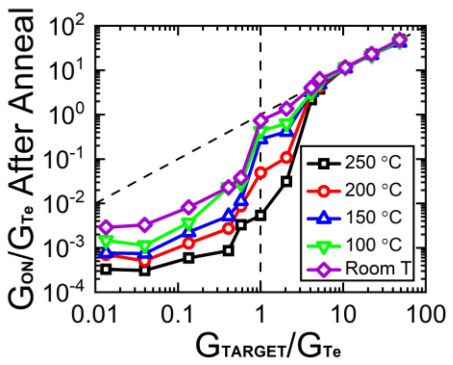
*For 60,000 training images.

1.8 Supplementary Information



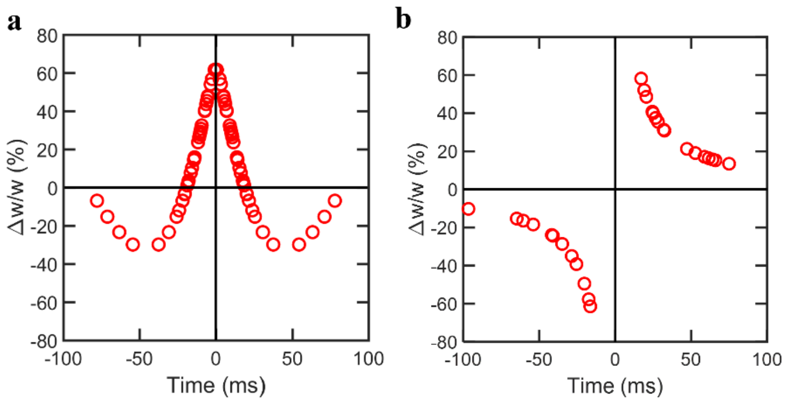
Supplementary Figure 1. S1 CBRAM Write/Erase Speed.

(a) Bitline (BL) and wordline (WL) during a 3V program operation. The anode voltage is fixed at the 3V BL voltage. After the WL is enabled, the cell programs in <10ns. b, Bitline (BL) and wordline (WL) during an erase operation. After the WL is enabled, the cell erases in ~10ns [1]. For programming, the voltage to be applied to the cell is established when the BL discharges (red curve). After that, the WL (blue curve) is enabled. When the cell programs, the BL voltage increases towards the anode voltage (which is high). The programming time is the offset between the time when the WL is enabled and the time when the BL voltage is seen to increase, which is shown to be <10 ns in (a). The situation is similar for erase operation, where only the polarity is reversed (BL is high). The erase time is the offset between the time when WL is enabled and the time when the BL pulls down towards the anode (which is low). Erase time is measured as ~10ns as seen in (b).



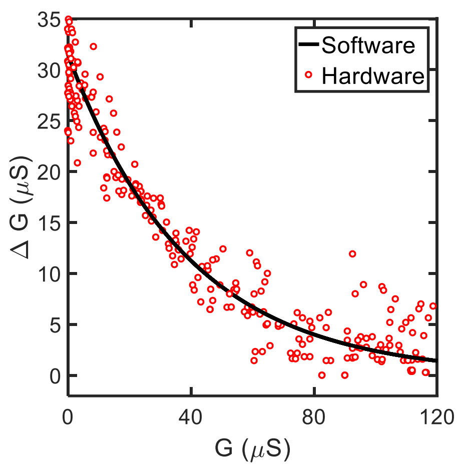
Supplementary Figure 1. S2 CBRAM Retention Characteristic.

Excellent retention is achieved by the subquantum cells for 10 min annealing at high temperature with the ON-state conductance a few times greater than G_{Te} are targeted [2].



Supplementary Figure 1. S3 STDP.

(a) Symmetric spike-timing-dependent plasticity (STDP) and (b) Asymmetric STDP learning rules modeled using the gradual programming data of 1T1R subquantum CBRAM cells in Figure 1.2a.



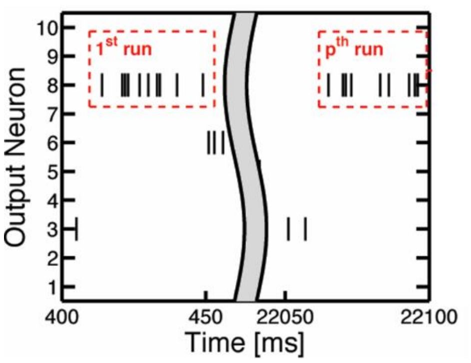
Supplementary Figure 1. S4 STDP Fitting.

The measured data from Figure 1.2b is fitted into the neural network weight updating rule (Figure 1.3c).

```
Algorithm 1: Unsupervised SNN Training
  Input: Training digits
  Initialization: STDP parameters, synaptic weights
    for Each simulation time step t = 1ms do
      STEP 1   Present a training sample for 50ms and generate input Poisson spike train $X_i$
               from 0 – 200 Hz according to pixel intensity $P_i$
      STEP 2   Compute membrane potential for each output neuron
               $U_k(t) = \sum_i W_{ki} X_i(t) + b_k$
               $X_i(t) = 1$ if $X_i$ fired within past 10 ms; else $X_i(t) = 0$
      STEP 3   Generate output Poisson spike train via probabilistic firing model
      STEP 4   Update corresponding neuron's weights using STDP
               $W_{ki} = W_{ki} + \Delta W_{ki}$
               $\Delta W_{ki}$ is updated via LTP rule if $X_i$ fired within past $\sigma = 10$ ms;
               else $\Delta W_{ki}$ is updated via LTD rule
    end for
```

Supplementary Figure 1. S5 Unsupervised Learning Algorithm.

Unsupervised spiking neural network (SNN) learning algorithm used in software neural network simulation and hardware demonstration.



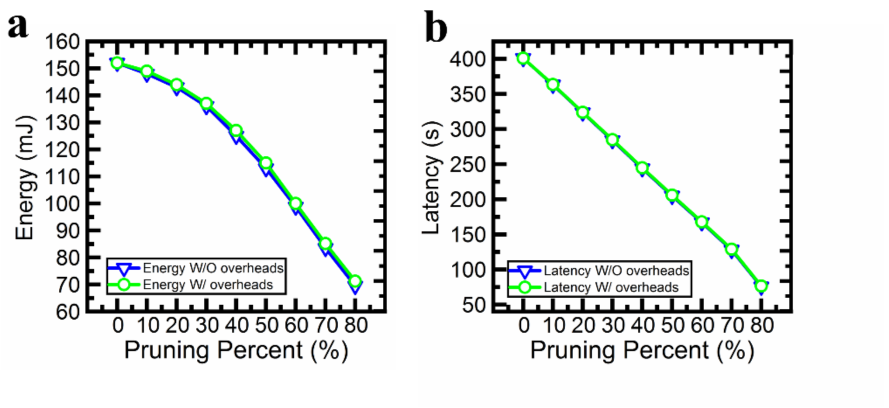
Supplementary Figure 1. S6 Consecutive Spikes.

The illustration of consecutive output spikes of 10 output neurons as a representative example. The consecutive output spikes of Neuron 8 are boxed in red. The consecutive spikes can be measured using integrate-and-fire neuron circuits which contain capacitors [3] or memristor [4] to store the information about how many spikes they received within a time interval representing using charge or resistance.

```
Algorithm 2: Soft-pruning during unsupervised learning
  Input: pruning percentage r, valid runs p, spike count c, number of weights N
    if there is an output spike then
      if this neuron has yet to be pruned then
        STEP 1   Count p for each neuron
        STEP 2   Check each sample present time window
                 1. if an output neuron has accumulated p runs
                 2. if each of those run has at least c consecutive output spikes
        STEP 3   If both criteria in STEP 2 are met
                 1. Set the weight threshold to the pre-determined threshold which represents
                    ≈ rth percentile of AAthe neuron's weights
                 2. Set all weights below the threshold to the lowest possible values
                    $W_{min} = -1$.
      end if
    end if
```

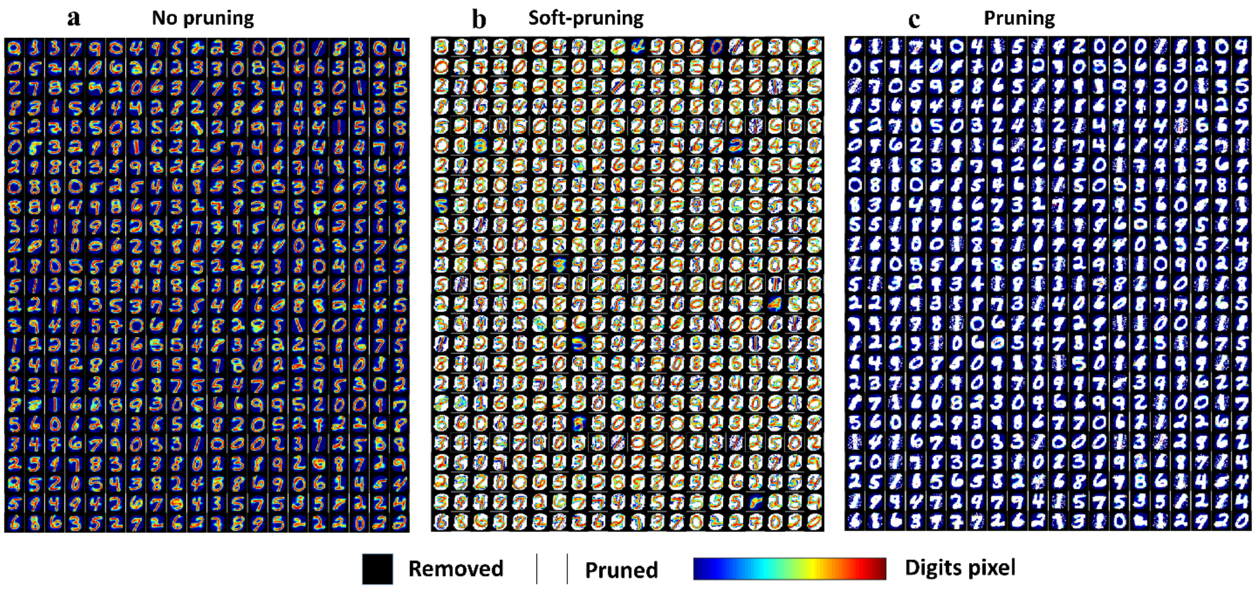
Supplementary Figure 1. S7 Pruning Algorithm.

Soft-pruning during the training algorithm for hardware implementation.



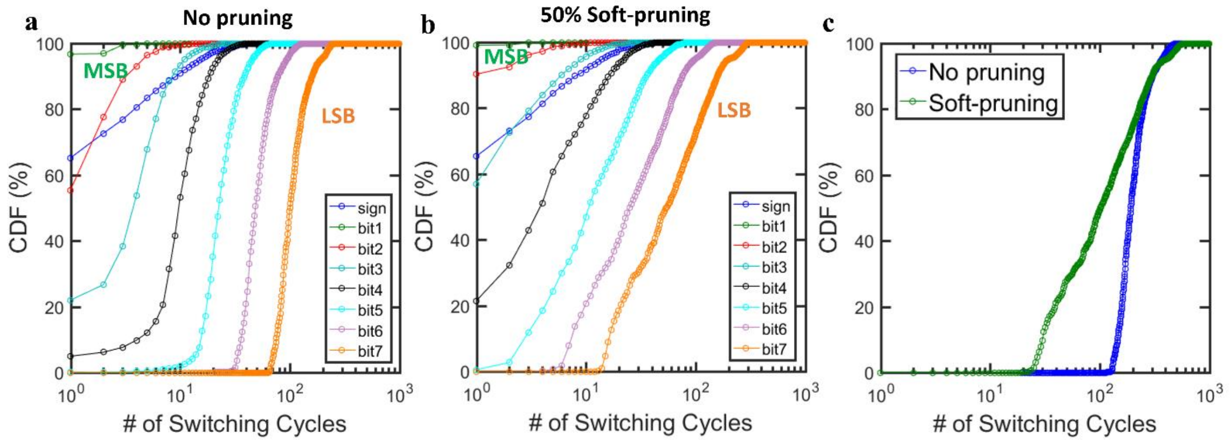
Supplementary Figure 1. S8 Pruning Overheads.

(a) Energy and (b) Latency without and with overheads estimation for soft-pruning from 10% to 80% with a step of 10% using SNN+NeuroSim. Overheads include hardware flag and setting pruned weights to -1. Without overheads (W/O overheads) results mean that flagging mechanism is implemented in software and overhead associated with setting pruned weights to -1 is not considered. With overheads (W/ overheads) results mean that flagging mechanism is implemented in hardware and overhead associated with setting pruned weights to -1 is considered.



Supplementary Figure 1. S9 Classification and Pruning Visualization**.**

Weights visualization of all 500 output neurons for (a) no pruning, (b) 50% soft-pruning and (c) pruning after training.



Supplementary Figure 1. S10 Device Switching Cycles during Training.

(a) (b) Empirical cumulative distribution of the switching cycles of each bit in the weight matrix during training (a) no pruning and (b) with 50% soft-pruning. We use one bit for the sign. Bit 1 is MSB and bit 7 is LSB. LSB updates more frequently than MSB in both cases. 50% pruning method effectively reduces the weight updates in every bit. (c) Cumulative distribution of the switching cycles of all bits. Pruning significantly reduces the number of switching cycles for all the bits during training.

Supplementary Table 1. S1 Device Energy Profile

Energy consumption in subquantum conductive bridging RAM (CBRAM), metal filament-based CBRAM cells and floating gate flash [5]. Subquantum CBRAM is 10× more energy efficient than metal filament CBRAM and 100× more energy efficient than floating gate flash, even for the maximum energy consumption cases.

	Subquantum CBRAM Synapses	**Metal Filament CBRAM Synapses**	**Floating Gate Flash**
G_{1atom}	$0.03\ G_0$	$1\ G_0$	-
Read voltage	1V (for WL)	1V (for WL)	3-5 V (for WL)
Program voltage	1–3 V	1–3 V	|6| – |9| V
Program time	0.01 – 0.1 μs/cell	0.01 – 0.1 μs/cell	1 – 10 μs/cell
Program energy	0.1 – 10 pJ/cell	1-100 pJ/cell	1000 pJ/cell
Erase voltage	1-3 V	1-3 V	|6| – |9| V
Erase time	0.01 – 0.1 μs/cell	0.01 – 0.1 μs/cell	1 ms/cell
Erase energy	0.1 – 10 pJ/cell	1-100 pJ/cell	1000 pJ/cell

Supplementary Table 1. S2 Pruning Overheads Estimation

Area, energy and latency estimation of no pruning, 80% soft-pruning without and with overheads. Without overheads (W/O overheads) results mean that flagging mechanism is implemented in software and overhead associated with setting pruned weights to -1 is not taken into account. With overheads (W/ overheads) results mean that flagging mechanism is implemented in hardware and overhead associated with setting pruned weights to -1 is taken into account. The numbers inside of the parentheses show the energy and latency increase due to overheads associated with [a]hardware flag and [b]setting pruned weights to -1, respectively.

Accuracy (%)	**No Pruning (8-bit)**	**80% soft-Pruning W/O overheads (8-bit)**	**80% soft-Pruning W/ overheads* (8-bit + 1-bit flag)**
Area (μm^2)	47233.8	47233.8	53218.5
Energy (mJ)	151.9	69.5	71.3 (+1.09[a], +0.68[b])
Latency (s)	401.1	75.6	76.5 (+0.42[a],+0.50[b])

* Overheads include [a] hardware flag and [b] setting pruned weights to -1.

Supplementary Table 1. S3 State-of-the-Art Unsupervised Learning Demonstration with Synaptic Devices on MNIST

Table compares the overall performance of this work with the state-of-the-art unsupervised learning demonstration with synaptic device on MNIST dataset. All the references report recognition performance simulated using single device data, while this work reports recognition accuracy for hardware implementation. The synaptic device energy consumption per programming is calculated by multiplying the pulse amplitude with the current flowing across the device and the programming pulse width. The number of updates is calculated by multiplying number of iterations in training with number of weights needed to be updated per iterations. The numbers of neurons are counted by summing up input and output neurons. If the first cell of a row contains multiple citations, subsequent values may have been taken from any one of the cited works, which are written or cited by the same authors.

Architecture	Preprocessing	Learning-rule	# Neurons/ # Plastic synapses	Performance	Hardware Type	Energy Consumption	# of Updates (10^6)
Two layer network[6,7,8]	None	Exponential STDP	1,184/313,600	91.6%	HfO*x* based RRAM (simulation)	~0.85 – 24 pJ	~141
One layer network[9,10]	None	Exponential STDP	1,084/235,200	87.4%	STT-MRAM (simulation)	~0.09 – 96.9 pJ	~47
One layer network[11,12]	None	Probabilistic prespike rule	834/39,200	84%	WO*x* based RRAM (simulation)	~1.68 nJ	~47
One layer network[13]	None	Exponential STDP	864/62,720	70%	CNT synaptic transistors (simulation)	~40 – 800 nJ	~47
One layer network[14]	None	Exponential STDP	794/7,840	60%	CNT synaptic transistors (simulation)	~50 nJ	~47
One layer network[15]	yes	Exponential STDP	206/~1,980	59.8%	TiO*x* based RRAM (simulation)	~25 – 750 nJ	~8.8
This work	**yes**	**Exponential STDP**	**405/3,950**	**93.19%**	**Subquantum CBRAM (Hardware implementation)**	**~0.1 – 10 pJ**	**No prune: ~4.2 50% prune: ~2**

Supplementary Table 1. S4 State-of-the-Art Software Demonstration of Unsupervised Learning Demonstration on MNIST

Table compares the performance (recognition accuracy) of this work with the state-of-the-art software demonstrations of unsupervised learning on MNIST dataset. The numbers of neurons are counted by summing up input and output neurons.

Architecture	Preprocessing	Learning-rule	# Neurons/ # Plastic synapses	Performance	Hardware Type
Spiking Deep neural network[16]	None	Simplified STDP	N/A	98.4%	No
Two layer network[8]	None	Exponential STDP	7,184/5,017,600	95%	No
Two layer network[17]	yes	Exponential STDP	~600/~50,000	80.14%	No
This work	**yes**	**Exponential STDP**	**~898/~199,000**	**94.05%**	**Subquantum CBRAM**

Supplementary Table 1. S5 Pruning Techniques

This Table summarizes pruning methods. The first four can only be applied after training as reported by the references. The method described in this work is the first to be implemented in hardware and also can be applied either during or after training. Energy savings are relative to the result with no pruning. Energy savings for this work is obtained from Supplementary Table1.S2. Accuracy loss is based on the result with 50% pruning.

Pruning Method	**Machine Learning Tasks**	**Network Structure**	**During Training**	**After Training**	**Hardware Implementation of Pruning**	**Simulation System**	**Energy Savings**	**Accuracy Loss (50% pruning)**
Deep Compression[18]	ImageNet pattern recognition	AlexNet	no	yes	no	EIE[22] (SRAM)	53%	~ -0.1% – +0.1%
SIMD-Aware Pruning[19]	ImageNet pattern recognition	AlexNet	no	yes	no	CPU,GPU	~53%	~ -1% – +1%
Energy-Aware Pruning[20]	ImageNet pattern recognition	AlexNet	no	yes	no	CPU	73%	< 1%
Structured Pruning[21]	CIFAR10 pattern recognition	CNN	no	yes	no	CPU	N/A	~ -1% – +2%
This work	**MNIST pattern recognition**	**Fully Connected**	**yes**	**yes**	**yes**	**CBRAM synaptic array**	**54.2%**	**During: +0.9% After: -0.5%**

Supplementary Table 1. S6 Simulation Parameters

All the parameters used for the simulations are listed below.

Parameters		**10-Digits**		
		Training	**Labeling**	**Testing**
# of Neuron	Input	398		
	Output	500		
Firing Rate (Hz)	Input	200	200	200
	Output	200	200	600
Image Presenting Time (ms)		50	50	200
Pruning Threshold	Prune Parameter (p)	10	-	-
	Spike Count	8	-	-
STDP		a = 0.0667 b = 2.5 c = 0.0167		

1.9 Supplementary Notes

Supplementary Note 1

Supplementary Figure 1.S2 shows an experimentally quantified stability (retention) as a function of conductance. It can be seen that the subquantum CBRAM has robust thermal stability. In this Figure, the x-axis is the target cell conductance during a program operation. Note that the x-axis is normalized so that the value 1 corresponds to the conductance of a filament with a 1-atom constriction. The y-axis is the actual conductance measured after annealing at temperatures ranging from room temperature to 250 °C. The plot shows that stability (retention) is poor if the targeted conductance level is lower than the conductance of an incomplete filament whose thinnest spot is less than 1 atom thick. However, once the target conductance is above the value of the 1-atom thick filament, the post-anneal conductance quickly approaches the targeted value. Therefore, the filament is increasingly stable as it becomes thicker. Note that filaments can be stable even at the highest temperature used in the study (250 °C).

Supplementary Note 2

Asymmetric and symmetric STDP are implemented using the same spike scheme described by *Kuzum. et al.* [23]. Pre and post spikes are implemented to the word line and bit line of the 1T1R array. Time overlap of the pre and post spikes allows programming of the CBRAM synapse. The spike timing differences between the pre and post spikes are translated to the amplitude of voltage pulses applied to the word line. Integrate-and-fire neurons are implemented using a computer program and pulse generators, and the pulses are applied to the WL and BL of the device to modulate the conductance change.

Supplementary Note 3

The overhead costs of the pruning algorithm can be estimated using our SNN platform for NeuroSim. The first overhead is that the pruned weights need to be flagged to prevent them from further updating. This can be implemented by adding an additional bit with an initial value of 0 to serve as a hardware flag for pruning. We update the pruning flags of an output neuron's weights to ‘1’ when they have been pruned during the training. Note that since the weights are only pruned once during the entire training, each hardware flag is just written once. Before weight update, we read the hardware flag of the winner neuron’s weights and the weight will not be updated if its flag is ‘1’. Another overhead is setting the pruned weights to -1. We take these two overheads into account in our simulation for pruning using digital hardware implementation (Figure 1.2c). In Supplementary Table1.2, both overhead costs are estimated in terms of area, energy and latency based on the peripheral programming circuitry shown in Figure 1.2c for no pruning, 80% soft-pruning without (W/O overheads) and with overheads (W/ overheads). As can be seen from this Table, the area is increased by ~12.7% because the flag only takes up one extra bit for each synapse. The hardware flag increases energy and latency by ~1.6% (1.09mJ) and ~0.6% (0.42s), respectively. Setting pruned weights to -1 increases energy and latency by ~0.98% (0.68mJ) and ~0.66% (0.5s), respectively. In summary, total energy and latency are increased by ~2.5% (~1.7mJ) and ~1.2% (~0.92s) due to the overheads of pruning implementation. This is significantly smaller and hence negligible compared to the energy and latency gains from pruning.

1.10 Supplementary References

[1] J. Jameson, P. Blanchard, C. Cheng, J. Dinh, A. Gallo, V. Gopalakrishnan, C. Gopalan, B. Guichet, S. Hsu and D. Kamalanathan, "Conductive-bridge memory (CBRAM) with excellent high-temperature retention," in 2013 IEEE International Electron Devices Meeting, 2013: IEEE, pp. 30.1. 1-30.1. 4.

[2] J. R. Jameson and D. Kamalanathan, "Subquantum conductive-bridge memory," Applied Physics Letters, vol. 108, no. 5, p. 053505, 2016.

[3] G. Indiveri, "A low-power adaptive integrate-and-fire neuron circuit," in Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03., 2003, vol. 4: IEEE, pp. IV-IV.

[4] X. Zhang, W. Wang, Q. Liu, X. Zhao, J. Wei, R. Cao, Z. Yao, X. Zhu, F. Zhang and H. Lv, "An artificial neuron based on a threshold switching memristor," IEEE Electron Device Letters, vol. 39, no. 2, pp. 308-311, 2017.

[5] J. R. Jameson, P. Blanchard, J. Dinh, N. Gonzales, V. Gopalakrishnan, B. Guichet, S. Hollmer, S. Hsu, G. Intrater and D. Kamalanathan, "Conductive bridging RAM (CBRAM): then, now, and tomorrow," ECS Transactions, vol. 75, no. 5, p. 41, 2016.

[6] A. M. Tosson, S. Yu, M. H. Anis and L. Wei, "A study of the effect of RRAM reliability soft errors on the performance of RRAM-based neuromorphic systems," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 11, pp. 3125-3137, 2017.

[7] D. Kuzum, S. Yu and H. P. Wong, "Synaptic electronics: materials, devices and applications," Nanotechnology, vol. 24, no. 38, p. 382001, 2013.

[8] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," Frontiers in computational neuroscience, vol. 9, p. 99, 2015.

[9] D. Zhang, L. Zeng, Y. Zhang, W. Zhao and J. O. Klein, "Stochastic spintronic device based synapses and spiking neurons for neuromorphic computation," in 2016 IEEE/ACM

International Symposium on Nanoscale Architectures (NANOARCH), 2016: IEEE, pp. 173-178.

[10] A. F. Vincent, J. Larroque, N. Locatelli, N. B. Romdhane, O. Bichler, C. Gamrat, W. S. Zhao, J.-O. Klein, S. Galdin-Retailleau and D. Querlioz, "Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," IEEE transactions on biomedical circuits and systems, vol. 9, no. 2, pp. 166-174, 2015.

[11] P. Sheridan, W. Ma and W. Lu, "Pattern recognition with memristor networks," in 2014 IEEE International Symposium on Circuits and Systems (ISCAS), 2014: IEEE, pp. 1078-1081.

[12] T. Chang, S.-H. Jo, K.-H. Kim, P. Sheridan, S. Gaba and W. Lu, "Synaptic behaviors and modeling of a metal oxide memristive device," Applied physics A, vol. 102, no. 4, pp. 857-863, 2011.

[13] S. Kim, B. Choi, M. Lim, J. Yoon, J. Lee, H.-D. Kim and S.-J. Choi, "Pattern recognition using carbon nanotube synaptic transistors with an adjustable weight update protocol," ACS nano, vol. 11, no. 3, pp. 2814-2822, 2017.

[14] S. Kim, J. Yoon, H.-D. Kim and S.-J. Choi, "Carbon nanotube synaptic transistor network for pattern recognition," ACS applied materials & interfaces, vol. 7, no. 45, pp. 25479-25486, 2015.

[15] F. Zahari, M. Hansen, T. Mussenbrock, M. Ziegler and H. Kohlstedt, "Pattern recognition with TiO x-based memristive devices," AIMS Materials Science, vol. 2, no. 3, pp. 203-216, 2015.

[16] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," Neural Networks, vol. 99, pp. 56-67, 2018.

[17] B. Nessler, M. Pfeiffer, L. Buesing and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," PLoS computational biology, vol. 9, no. 4, p. e1003037, 2013.

[18] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," ACM SIGARCH Computer Architecture News, vol. 44, no. 3, pp. 243-254, 2016.

[19] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," ACM SIGARCH Computer Architecture News, vol. 45, no. 2, pp. 548-560, 2017.

[20] T.-J. Yang, Y.-H. Chen and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5687-5695.

[21] S. Anwar, K. Hwang and W. Sung, "Structured pruning of deep convolutional neural networks," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 13, no. 3, pp. 1-18, 2017.

[22] S. Han, J. Pool, J. Tran and W. Dally, "Learning both weights and connections for efficient neural network," Advances in neural information processing systems, vol. 28, 2015.

[23] D. Kuzum, R. G. D. Jeyasingh, S. Yu and H.-S. P. Wong, "Low-energy robust neuromorphic computation using synaptic devices," IEEE Transactions on Electron Devices, vol. 59, no. 12, pp. 3489-3494, 2012.

1.11 References

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, p. 436, 2015.

[2] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097-1105.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla and M. Bernstein, "Imagenet large scale visual recognition challenge," International Journal of Computer Vision, vol. 115, no. 3, pp. 211-252, 2015.

[4] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in Proceedings of the 25th international conference on Machine learning, 2008: ACM, pp. 160-167.

[5] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen and T. N. Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82-97, 2012.

[6] C. Chen, A. Seff, A. Kornhauser and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2722-2730.

[7] S. Vishwakarma and A. Agrawal, "A survey on activity recognition and behavior understanding in video surveillance," The Visual Computer, vol. 29, no. 10, pp. 983-1009, 2013.

[8] D. C. Cireşan, A. Giusti, L. M. Gambardella and J. Schmidhuber, "Mitosis detection in breast cancer histology images with deep neural networks," in International Conference on Medical Image Computing and Computer-assisted Intervention, 2013: Springer, pp. 411-418.

[9] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in

Proceedings of the 2015 International Workshop on Internet of Things towards Applications, 2015: ACM, pp. 7-12.

[10] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Technical report, University of Toronto, vol. 1, p. 7, 2009.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, 2009: IEEE, pp. 248-255.

[12] A. Asuncion and D. Newman, "UCI machine learning repository," 2007.

[13] G. Salelanonda. (2016). Learning how to learn: Toddlers vs. neural networks [Online]. Available: https://www.eetimes.com/author.asp?section_id=36&doc_id=1330538.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam and M. Lanctot, "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484-489, 2016.

[15] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum and A. J. Hudspeth, Principles of neural science. McGraw-hill New York, 2000.

[16] D. Kuzum, R. G. Jeyasingh and H.-S. P. Wong, "Energy efficient programming of nanoelectronic synaptic devices for large-scale implementation of associative and temporal sequence learning," in Electron Devices Meeting (IEDM), 2011 IEEE International, 2011: IEEE, pp. 30.3. 1-30.3. 4.

[17] S. B. Eryilmaz, D. Kuzum, R. G. Jeyasingh, S. Kim, M. BrightSky, C. Lam and H.-S. P. Wong, "Experimental demonstration of array-level learning with phase change synaptic devices," in Electron Devices Meeting (IEDM), 2013 IEEE International, 2013: IEEE, pp. 25.5. 1-25.5. 4.

[18] D. Mahalanabis, M. Sivaraj, W. Chen, S. Shah, H. J. Barnaby, M. N. Kozicki, J. B. Christen and S. Vrudhula, "Demonstration of spike timing dependent plasticity in CBRAM devices with silicon neurons," in Circuits and Systems (ISCAS), 2016 IEEE International Symposium on, 2016: IEEE, pp. 2314-2317.

[19] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu and H. Qian, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in Electron Devices Meeting (IEDM), 2016 IEEE International, 2016: IEEE, pp. 16.2. 1-16.2. 4.

[20] V. Milo, G. Pedretti, R. Carboni, A. Calderoni, N. Ramaswamy, S. Ambrogio and D. Ielmini, "Demonstration of hybrid CMOS/RRAM neural networks with spike time/rate-dependent plasticity," in Electron Devices Meeting (IEDM), 2016 IEEE International, 2016: IEEE, pp. 16.8. 1-16.8. 4.

[21] S. Park, H. Kim, M. Choo, J. Noh, A. Sheri, S. Jung, K. Seo, J. Park, S. Kim, W. Lee, J. Shin, D. Lee, G. CHoi, J. Woo, E. Cha, J. Jang, C. Park, M. Jeon, B. Lee, B. H. Lee and H. Hwang, "RRAM-based synapse for neuromorphic system with pattern recognition function," in Electron Devices Meeting (IEDM), 2012 IEEE International, 2012: IEEE, pp. 10.2. 1-10.2. 4.

[22] A. Serb, J. Bill, A. Khiat, R. Berdan, R. Legenstein and T. Prodromakis, "Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses," Nature communications, vol. 7, p. 12611, 2016.

[23] S. Choi, P. Sheridan and W. D. Lu, "Data clustering using memristor networks," Scientific reports, vol. 5, p. 10492, 2015.

[24] Z. Wang, S. Joshi, S. Savel'ev, W. Song, R. Midya, Y. Li, M. Rao, P. Yan, S. Asapu, Y. Zhuo, H. Jiang, P. Lin, C. Li, J. H. Yoon, N. K. Upadhyay, J. Zhang, M. Hu, P. J. Strachan, M. Barnell, Q. Wu, H. Wu, R. S. Williams, Q. Xia and J. J. Yang, "Fully memristive neural networks for pattern classification with unsupervised learning," Nature Electronics, vol. 1, no. 2, p. 137, 2018.

[25] Y. Jeong, J. Lee, J. Moon, J. H. Shin and W. D. Lu, "K-means data clustering with memristor networks," Nano letters, 2018.

[26] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Wu Qing, R. S. Williams, J. J. Yang and Q. Xia, "Efficient

and self-adaptive in-situ learning in multilayer memristor neural networks," Nature Communications, vol. 9, no. 1, p. 2385, 2018.

[27] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia and J. P. Strachan, "Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine," Advanced Materials, vol. 30, no. 9, p. 1705914, 2018.

[28] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha, B. Killeen, C. Cheng, Y. Jaoudi and G. W. Burr, "Equivalent-accuracy accelerated neural-network training using analogue memory," Nature, vol. 558, no. 7708, p. 60, 2018.

[29] I. Kataeva, F. Merrikh-Bayat, E. Zamanidoost and D. Strukov, "Efficient training algorithms for neural networks based on memristive crossbar circuits," in Neural Networks (IJCNN), 2015 International Joint Conference on, 2015: IEEE, pp. 1-8.

[30] F. M. Bayat, M. Prezioso, B. Chakrabarti, H. Nili, I. Kataeva and D. Strukov, "Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits," Nature communications, vol. 9, no. 1, p. 2331, 2018.

[31] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee and W. D. Lu, "Reservoir computing using dynamic memristors for temporal information processing," Nature communications, vol. 8, no. 1, p. 2204, 2017.

[32] I. Boybat, M. Le Gallo, S. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian and E. Eleftheriou, "Neuromorphic computing with multi-memristive synapses," Nature communications, vol. 9, no. 1, p. 2514, 2018.

[33] S. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian and E. Eleftheriou, "Mixed-precision architecture based on computational memory for training deep neural networks," in Circuits and Systems (ISCAS), 2018 IEEE International Symposium on, 2018: IEEE, pp. 1-5.

[34] C. Liu, Q. Yang, B. Yan, J. Yang, X. Du, W. Zhu, H. Jiang, Q. Wu, M. Barnell and H. Li, "A memristor crossbar based computing engine optimized for high speed and accuracy," in VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on, 2016: IEEE, pp. 110-115.

[35] S. Han, H. Mao and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv:1510.00149, 2015.

[36] T.-J. Yang, Y.-H. Chen and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," arXiv preprint arXiv:1611.05128, 2016.

[37] J. Graham, "Children and brain development: What we know about how children learn," Cooperative Extension Publication, 2011.

[38] R. Reed, "Pruning algorithms-a survey," IEEE transactions on Neural Networks, vol. 4, no. 5, pp. 740-747, 1993.

[39] Y.-S. Goh and E.-C. Tan, "Pruning neural networks during training by backpropagation," in TENCON'94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994, 1994: IEEE, pp. 805-808.

[40] J. R. Jameson and D. Kamalanathan, "Subquantum conductive-bridge memory," Applied Physics Letters, vol. 108, no. 5, p. 053505, 2016.

[41] V. A. Vis, "Photoconductivity in Single‐Crystal Tellurium," Journal of Applied Physics, vol. 35, no. 2, pp. 360-364, 1964.

[42] D. Kuzum, R. G. Jeyasingh, B. Lee and H.-S. P. Wong, "Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing," Nano letters, vol. 12, no. 5, pp. 2179-2186, 2011.

[43] S. Yu, P.-Y. Chen, Y. Cao, L. Xia, Y. Wang and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in Electron Devices Meeting (IEDM), 2015 IEEE International, 2015: IEEE, pp. 17.3. 1-17.3. 4.

[44] P.-Y. Chen and S. Yu, Impact of Nonideal Resistive Synaptic Device Behaviors on Implementation of Sparse Coding Algorithm (Neuro-inspired Computing Using Resistive Synaptic Devices). Springer, 2017, pp. 233-251.

[45] D. Kuzum, R. G. D. Jeyasingh, S. Yu and H.-S. P. Wong, "Low-energy robust neuromorphic computation using synaptic devices," IEEE Transactions on Electron Devices, vol. 59, no. 12, pp. 3489-3494, 2012.

[46] P.-Y. Chen, X. Peng and S. Yu, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[47] J. H. Lee, T. Delbruck and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," Frontiers in neuroscience, vol. 10, 2016.

[48] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," Frontiers in neuroscience, vol. 7, p. 272, 2014.

[49] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo and Y. Nakamura, "A million spiking-neuron integrated circuit with a scalable communication network and interface," Science, vol. 345, no. 6197, pp. 668-673, 2014.

[50] 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, "Neuromorphic silicon neuron circuits," Frontiers in neuroscience, vol. 5, 2011.

[51] B. Nessler, M. Pfeiffer, L. Buesing and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," PLoS computational biology, vol. 9, no. 4, p. e1003037, 2013.

[52] B. Nessler, M. Pfeiffer and W. Maass, "STDP enables spiking neurons to detect hidden causes of their inputs," in Advances in neural information processing systems, 2009, pp. 1357-1365.

[53] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," Frontiers in computational neuroscience, vol. 9, p. 99, 2015.

[54] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization," Neural Networks, vol. 103, pp. 118-127, 2018.

[55] J. H. Lee, T. Delbruck and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," Frontiers in neuroscience, vol. 10, p. 508, 2016.

[56] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in Neural Networks (IJCNN), 2015 International Joint Conference on, 2015: IEEE, pp. 1-8.

[57] B. Kijsirikul and K. Chongkasemwongse, "Decision tree pruning using backpropagation neural networks," in Proceedings of IEEE Int. Conf. on Neural Networks, 2001, vol. 3, pp. 1876-1880.

[58] P.-Y. Chen, B. Lin, I. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J.-s. Seo, Y. Cao and S. Yu, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 2015: IEEE Press, pp. 194-199.

[59] J. Zhang, Z. Wang and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," J. Solid-State Circuits, vol. 52, no. 4, pp. 915-924, 2017.

Chapter 2. A Soft-pruning Method Applied during Training of Spiking Neural Networks for In-memory Computing Applications

2.1 Abstract

Inspired from the computational efficiency of the biological brain, spiking neural networks (SNNs) emulate biological neural networks, neural codes, dynamics, and circuitry. SNNs show great potential for the implementation of unsupervised learning using in-memory computing. Here, we report an algorithmic optimization that improves energy efficiency of online learning with SNNs on emerging nonvolatile memory (eNVM) devices. We develop a pruning method for SNNs by exploiting the output firing characteristics of neurons. Our pruning method can be applied during network training, which is different from previous approaches in the literature that employ pruning on already-trained networks. This approach prevents unnecessary updates of network parameters during training. This algorithmic optimization can complement the energy efficiency of eNVM technology, which offers a unique in-memory computing platform for the parallelization of neural network operations. Our SNN maintains ~90% classification accuracy on the MNIST dataset with up to ~75% pruning, significantly reducing the number of weight updates. The SNN and pruning scheme developed in this work can pave the way towards applications of eNVM based neuro-inspired systems for energy efficient online learning in low power applications.

2.2 Introduction

In recent years, brain-inspired spiking neural networks (SNNs) have been attracting significant attention due to their computational advantages. SNNs allow sparse and event-driven parameter updates during network training [1-4]. This results in lower energy consumption, which is appealing for hardware implementations [5-8]. Emerging non-volatile memory (eNVM) arrays have been proposed as a promising in-memory computing platform to implement SNN training in an energy efficient manner. eNVM devices can implement spike-timing-dependent plasticity (STDP) [9, 10], which is a commonly used weight update rule in SNNs. Most demonstrations utilize eNVM crossbar arrays to parallelize computation of the inner product [11-16]. In addition, there are several works focus on using eNVM hardware such as spintronic devices or crossbars with additional algorithmic optimization of STDP learning rules to perform hardware implementation of SNN [17-21]. While eNVM crossbar arrays improve energy efficiency at a device level for SNN training, network level algorithmic optimization is still important to further improve energy efficiency for wide adoption of SNNs in low power applications.

Pruning network parameters, i.e., synaptic weights, is a recent algorithmic optimization [22] that is widely used for compressing the network to improve the energy efficiency for the inference operation of deep neural networks. Although synaptic pruning has been demonstrated in many biophysical SNN models [23-27], how the pruning can be used for non-biophysical SNN has not been fully explored yet. Moreover, this method is applied on already-trained networks and it does not address the high-energy consumption during training, which requires iterative weight updates. A new approach towards network training that improves the energy efficiency of SNNs is crucial to develop online learning systems that can learn and perform inference in real world scenarios.

Here, we develop an algorithm to prune during training for SNNs with eNVMs to improve network level energy efficiency for in-memory computing applications. Although Rathi et al. [28] has showed pruning in SNN before, there are several key innovations and differences of the pruning method in this work compared to Rathi et al.' work. Our method considers the spiking activity of the output neurons to decide when to prune during the training while Rathi et al. performs the pruning at regular intervals for every batch without considering the characteristics of the output neurons. In addition, once the weights have been pruned during the training, we do not update the pruned weights for the rest of the training while Rathi et al. only temporally removes the pruned weights and they can still be updated when new batches present to the network. Finally, we develop soft-pruning as an extension of pruning. Soft-pruning sets the pruned weights to a constant non-zero values. Therefore, it is novel in terms of treating pruned weights. Rathi et al. only implement pruning.

Our paper is organized as follows: first, we describe our unsupervised SNN model and the weight update rule. Then, we introduce a pruning method that exploits spiking characteristics of the SNN to decrease the number of weight updates and thus energy consumption during training. Finally, we discuss how our SNN training and pruning algorithm can potentially be realized using eNVM crossbar arrays and perform circuit-level simulations to confirm the feasibility for online unsupervised learning to reduce the energy consumption and training time.

In Section 2.1 to Section 2.5, we discuss our SNN model and the algorithms relating to weight updates. In Section 2.6, we discuss methods to prune during training. In Section 3, we discuss our software simulation results, compare our SNN with state-of-the-art unsupervised SNN algorithms on MNIST and explore the method to implement our SNN model and pruning algorithm using the eNVM crossbar array through circuit-level simulations.

2.3 Results

Inspired by the information transfer in biological neurons via precise spike timing, SNNs temporally encode the inputs and outputs of a neural network layer using spike trains. The weights of the SNN are updated via a biologically plausible STDP, which modulates weights based on the timing of input and output spikes [3, 4]. This can be easily implemented on an eNVM crossbar array [9], making it ideal for online learning in hardware.

Our SNN performs unsupervised classification of handwritten digits from the MNIST dataset. It is a single layer network defined by the number of inputs neurons n, the number of outputs neurons m, and an m by n weight matrix. The number of input neurons can vary depending on preprocessing, but by default there are 784 input neurons to account for each grayscale pixel in a training sample. The output layer consists of 500 neurons to classify the 10 classes of the MNIST dataset (60,000 training images and 10,000 testing images). Figure 2.1 describes the fully connected network architecture.

As an overview of the pipeline, we first train the SNN by sequentially presenting samples from the training set. The purpose of training is to develop the weights of each output neuron so that they selectively fire for a certain class in MNIST. Afterwards, we present the training set for a second time to label each trained output neuron with the class of training samples that has the highest mean firing rate. This organizes the output neurons into populations that each respond to one of the classes. Finally, we test the SNN by predicting the label of each of the test samples based the class of output neurons with the highest mean firing rate.

2.3.1 Input Layer

We first remove the pixels that are used to represent the background in at least 95% of the training samples to reduce the number of input layer neurons. Because the grayscale pixels have intensity values in the range [0, 1], the pixels with a value of 0 correspond to the background and are thus checked for removal. After this step, we retain 397 of the original 784 pixels, reducing the complexity of the SNN. Therefore, we have 398 input neurons for a given training sample after accounting for an additional bias input neuron, which has a value of 1. Our output neurons do not have refractory periods and there is no lateral inhibition between them.

We encode each of these inputs as a Poisson spike train at a frequency of 200 times its value, leading to a maximum input firing rate of 200 Hz. We round the timing of each spike that is generated by the Poisson process to the nearest millisecond, which is the time of one-time step in the SNN. The SNN displays each training sample for the first 40 ms of a 50 ms presentation period, and thus the input spikes for a given training sample can only occur in this 40 ms window. Figure 2.2a shows an example of the input spiking activity for the duration of three training samples.

2.3.2 Output Layer

For output spikes, we use the Bayesian winner-take-all (WTA) firing model [3]. Unlike traditional integrate-and-fire models [29, 30], this model is shown to demonstrate Bayes’ rule [3], which is a probabilistic model for learning and cognitive development [31]. The SNN fires an output spike from any given output neuron according to a 200 Hz Poisson process. The output neuron that fires is chosen from a softmax distribution of the output neurons’ membrane potentials:

$$\mathrm{p}(u_k) = \frac{\exp(u_k)}{\sum_{i=1}^{m} \exp(u_i)} \qquad (1)$$

, where $\{\mathrm{p}(u_k)\}_{k=1,\dots,m}$ is the softmax probability distribution of the membrane potentials $\{u_k\}_{k=1,\dots,m}$. *m* is the number of output neurons. Our firing mechanism is probabilistic instead of hard thresholding the membrane potentials. Therefore, the neuron with higher membrane potential means that it has higher chance to fire. We calculate membrane potentials u_k using (2)

$$u_k = \sum_i W_{ki} X_i + b_k \qquad (2)$$

, where W_{ki} is the weight between input neuron *i* and output neuron *k*, X_i is the spike train generated by input neuron *i* and b_k is the weight of the bias term. Eq. (2) calculates an output neuron's membrane potential as the inner product between the input spikes at a given time step and the output neuron's weights, but this does not need to be integrated with each time step. Instead, we only calculate the membrane potentials at time steps when an output neuron fires because it is only used to determine which output neuron to fire. This removes additional parameters and resources needed with typical integrate-and-fire neuron models, which use the membrane potential to also find when to fire output neurons, allowing for a more efficient hardware implementation.

2.3.3 Weight updates: STDP Rule

When an output neuron fires, a simple STDP rule determines which weights to update via long-term potentiation (LTP) or long-term depression (LTD). As shown in Figure 2.3a, if an input neuron's most recent spike is within σ = 10 ms of the output spike, then the weight for this input-output synapse is increased (LTP). Otherwise, if it is beyond this 10 ms window of the output spike, then the weight is decreased (LTD).

This 10 ms window is in accordance with the fact that training samples are not displayed during the final 10 ms of their presentation period—they are only displayed for the first 40 ms of the 50 ms presentation period. Thus, there are no input spikes in the final 10 ms of each presentation, as seen in Figure 2.2a. Therefore, this STDP window prevents LTP weight updates that are potentially caused by the input spiking activity of the previous training sample. For example, when a new training sample is inputted to the SNN, an output spike occurring at simulation time *t* = 50 ms cannot have a spike-timing difference with an input spike occurring from *t* = 41 ms to *t* = 49 ms, since this is within the 10 ms window for LTP weight updates.

Figure 2.2b shows an example of the output spiking activity for 10 representative output neurons with randomly initialized weights, illustrating the random spiking activity of an untrained SNN. The effect of performing weight updates is to train the network to selectively fire to certain classes of inputs. At the start of training, we randomly initialize all weight values between [-1, 1], and the LTP and LTD update rules keep the weight values within the range [-1, 1]. The LTP weight update is an exponential function of the form $\Delta w_{LTP}(w) = ae^{-b(w+1)}$ (Figure 2.3b), where $a \in \{\mathbb{R}: 0 < a < 1\}$ and $b \in \mathbb{R}_{>0}$ are parameters that control the scale of the exponential, and *w* is the current weight value. For LTP updates to keep weight values within the upper bound of 1, we pick the parameters such that the weight update decays towards 0 as the current weight approaches 1. As a result, exponential LTP updates will guarantee that the weights converge to the upper bound of 1.

Unlike LTP, the LTD weight update is a constant function that disregards the current weight value: $\Delta w_{LTD} = -c$, where $c \in \{\mathbb{R}: 0 < c < 1\}$ is a parameter that controls the magnitude of the weight decrease. Because there is no guarantee of convergence as with the exponential LTP update, the SNN clips weights to the lower bound of -1. Alternatively, we can have an exponential LTD update that is mirrored about $w = 0$ from the exponential LTP update, i.e., $\Delta w_{LTD}(w) = -ae^{b(w-1)}$, and choose parameters to have weight convergence as in the case of LTP. However, the constant LTD update is easier to implement in hardware since there are less parameters to tune. The specific parameter choices of a, b and c are shown in Table2.1 and they come from cross validation of the parameter set to optimize the classification accuracy. Several previously published papers have proposed probabilistic synapses to perform STDP weight update [21, 32]. It is worth to note that the synapses in our network is deterministic and only the firing mechanism of output neurons is probabilistic as explained in Section 2.2.

2.3.4 Scaling Weight Updates as a Normalization Method

To perform a weight update, we add to the current weight *w_t* the weight update, which is scaled by an additional factor depending on whether the update is LTP or LTD:

$$w_{t+1} = \begin{cases} w_t + \frac{d}{n}\Delta w_{LTP}(w_t), \ LTP \\ w_t + \frac{p}{n}\Delta w_{LTD}, \ LTD \end{cases} \tag{3}$$

, where *d* is the number of weights to undergo LTD, *p* is the number of weights to undergo LTP, and *n* is the total number of weights for an output neuron, which also corresponds to the number of input neurons. Because of the STDP rule, all *n* weights of an output neuron are updated at any given output neuron firing event, which means that $d + p = n$. Because the number of LTP updates is often disproportionate with that of LTD due to the probabilistic spike firing, the scaling factors *d* and *p* keep the net weight change of both types of updates proportional so that for all output neurons, the distribution of weight values have roughly the same mean and variance. With this, an overview of the SNN training method is outlined in Figure 2.4.

This scaling of LTP and LTD weight updates is used to prevent certain output neurons from firing more than others. It effectively normalizes the weight distributions of each output neuron so that they fire according to the correlation between their weights and the training sample, rather than firing because the magnitude of their weights artificially increases their membrane potential. This foregoes the need to normalize the weight distributions of each output neuron through calculating the mean and standard deviation, which requires additional resources when implementing the weight update in hardware.

2.3.5 Testing

After training is done, we fix the trained weights and assign a class to each neuron by the following steps: First, we present the whole training set to the SNN and record the cumulative number of output spikes N_{kj}, where k = 1, ..., m (m is number of output neurons) and j = 1, …, n (n is number of classes, for MNIST, n = 10). Then, for each output neuron i, we calculate its response probability Z_{kj} to each class j using Eq. (4). Finally, each neuron k is assigned to the class that gives the highest response probability Z_{kj}.

$$Z_{kj} = \frac{N_{kj}}{\sum_{j=1}^{n} N_{kj}} \tag{4}$$

After training and labeling are done, we fix the weights and present test set to our network. We use Eq. (5) to predict the class of each sample, where S_{jk} is the number of spikes for the kth output neuron that are labeled as class j and N_j is the number of output neurons labeled as class j.

$$J = \underset{j}{argmax} \frac{\sum_{k=1}^{N_j} S_{jk}}{N_j} \tag{5}$$

2.3.6 Pruning During Training

Pruning is a concept in machine learning that removes redundant branches from a decision tree to reduce complexity and improve accuracy of the classifier. It prevents overfitting by learning the general structure of the input data instead of learning minute details. *Han et al*. implement pruning on trained convolutional neural networks to remove unimportant weights that have low contribution to the output [22]. For example, weights with values close to 0 can be removed since their inner product with their respective inputs will yield low output values. This removal effectively sets the weight values to 0, allowing for a sparser representation of the network for mobile applications while still retaining the same classification performance. Instead of pruning after training, we propose a method to prune during training on SNNs to reduce the number of weight updates.

Our implementation of pruning removes unimportant weights belonging to each output neuron, and each output neuron is only pruned once during training. When an output neuron fires, its weights can potentially be pruned based on the level of development in its weights. There is a tradeoff in choosing when to prune an output neuron. If we prune weights early during training, we save computation by not having to update these weights later on. However, by pruning early, the weights might not be trained enough to recognize a certain class in the dataset at the time of pruning, and this early pruning can hamper the future development of the weights. Conversely, pruning late better insures that the weights are trained at the expense of computing more weight updates.

To determine when to prune the weights of an output neuron, we refer to the spiking activity of the output neurons. The output neuron spiking activity is an inherent feature of SNNs that indicates the level of development in an output neuron's weights. Once an output neuron is trained enough to recognize a certain class from the dataset, it will start to fire more consistently, as in Figure 2.2c, due to its high membrane potential. To quantify this consistent output neuron firing behavior, we accumulate a count of the occurrences where there are at least 8 consecutive output spikes (Table2.1) from a specific output neuron during the 40 ms presentation period of a training sample. This count is kept for each output neuron as shown in Figure 2.5, and once an output neuron accumulates *r* (*r* = 10 in our case as shown in Table2.1) such counts during training, the SNN prunes a user-defined percentage of its weights. We choose to look for 8 consecutive output spikes based on the 200 Hz output firing rate, and the 10-count threshold is a hyperparameter to control how early or late to prune an output neuron. It is worth noting that the pruning percentages are set externally in our method and they can be chosen according to the dataset, the accuracy requirement and power/latency budget of the specific applications.

We explore two different methods of pruning in this work. We use the conventional pruning method [22] to prune the weights by setting their values to 0, which we also refer to pruning in this work. We also investigate a soft-pruning method [33] as an extension of conventional pruning.

Instead of completely removing the weights by setting them to 0, soft-pruning keeps the pruned weights constant at their current values for the remainder of training, or even keeping certain weights constant at the lowest or highest weight values allowed. This allows for more flexible criteria in regard to which weights are pruned, and what values they take as a result of pruning. In this work, we set the pruned weights to the lowest possible weight values, which is -1 for our network. The advantage of pruning is in reducing the representation of the weight matrix by introducing more sparsity. Figure 2.6 demonstrates this by the physical removal of synapses. However, depending on the dataset, the number of weights that will be close enough to 0 to comfortably prune without losing important information can vary. While soft-pruning does not necessarily introduce more sparsity, it can allow for more weights to be pruned, thus saving computation by preventing more weight updates without drastically altering the weight distribution. Figure 2.6 shows the pruned weights via soft-pruning as dashed lines to indicate that they still need to be stored in memory and participate in the testing. Soft-pruning does not increase the sparsity of weight matrix. However, since these weights are no longer updated, this can reduce energy consumption in the hardware implementation.

The usage of these two different pruning methods is dependent on the dataset to be classified. For example, the features of an image from MNIST can be separated into binary categories, i.e., the foreground and the background. In such a case, an example of soft-pruning is to prune a percentage of the lowest-valued weights of an output neuron by keeping these weight values at the lowest possible value, which for our SNN is -1. This variant of soft-pruning is analogous to learning a weight representation where the pixels representing the background take a single value, but the pixels representing the foreground can take on a range of values. Intuitively, soft-pruning results in a weight representation that does not waste resources to encode the black background pixels in MNIST in order to learn the details of the foreground, which can have varying levels of intensity due to the stroke weight of the handwriting. The top row of Figure 2.7 shows an example of the learned weight visualizations of 10 representative output neurons when the SNN is trained on the MNIST dataset in three cases: without pruning, with pruning, and with soft-pruning. By the seeding of the random number generator, we control the spiking activity of all three cases so that the third output neuron (N3) is the first to meet the pruning criteria. Therefore, up to the point before N3 is pruned, the SNNs for each of the three cases have the exact same spiking activity and weight update history for all output neurons. For example, the middle row of Figure 2.7 shows that N3's weight distribution is the same for all three cases. After this point, the different pruning methods between the three cases cause the weights of the output neurons between each case to develop differently.

Comparing the weight distributions for N3 in the final row of Figure 2.7, we can verify that soft-pruning is more reasonable than pruning for the MNIST dataset because it better preserves the shape of the original weight distribution, without pruning, in Figure 2.7a. In this example, we use both pruning methods to prune half of an output neuron's weights to clearly demonstrate the effect of each pruning method on the weight distribution. For pruning in Figure 2.7b, pruning 50% of the weights centered about the value 0 results in compressing a wide range of weights, shown by the space between the two dashed lines in the middle panel. Effectively, these pruned weights, most of which represent the foreground features of the MNIST dataset, are set to 0. Although the final panel of Figure 2.7b shows a somewhat binary weight distribution, which matches the binary foreground and background features of the MNIST dataset that we want to learn, the problem is that the shape of this weight distribution is drastically different than that of the weight distribution when the weights develop without pruning, as seen in the final panel of Figure 2.7a. In contrast, the effect of soft-pruning on the shape of the weight distribution, as seen in the final panel of Figure 2.7c, is minimal when compared to the case without pruning. Therefore, the pruned output neurons will produce comparable membrane potentials to the unpruned output neurons during training, resulting in balanced training between all output neurons.

With more complex datasets, e.g., color images, we might want to prune weights by setting weights around 0 to 0, or by setting weights to their current value. *Han et al.* demonstrate the former [22]. In the latter case, an interpretation can be that we set unimportant weights to their current value with the assumption that their current representation is already satisfactory for learning. Another approach is to freeze important, high-valued weights, which is a recently explored neuro-inspired concept called consolidation [34].

2.4 Discussion

We simulate our SNN model, pruning and soft-pruning in MATLAB. To determine a suitable size for the training dataset, we find via Figure 2.8a that three epochs (60,000 training samples per epoch) is sufficient to reach ~94% classification accuracy. Additionally, from Figure 2.8b, we use a 50 ms presentation period per training sample because longer presentation times show diminishing improvements in classification accuracy. Figure 2.8c shows the accuracy increases as the number of output neurons increase. However, adding output neurons will significantly increase the simulation time. Therefore, we choose to use 500 output neurons.

Following the pruning methods described in Section 2.6 Pruning During Training, we investigate the performance through software simulations. Simulation of classification accuracy for different *p* values in Figure 2.9a suggests that *r* = 10 provides the high accuracy even for very large pruning percentages (up to 80%). Figure 2.9b shows the performance of pruning and soft-pruning for varying pruning percentages when applied after training and during training. When applied after training, pruning and soft-pruning are comparable with each other until ~50% pruning rate. After this point, the accuracy for the regular pruning method falls below ~90% at ~60% pruning rate, but with soft-pruning, the accuracy stays at ~90% until ~75% pruning rate. When each method is applied during training to save on computation of weight updates, the accuracy with pruning falls below ~90% at around a ~40% of pruning rate, and the accuracy with soft-pruning falls below this mark at a ~75% of pruning rate. The performance of pruning drops much earlier than soft-pruning because pruning compresses the representation of important weights and causes uneven firing between output neurons, as mentioned in Section 2.6 Pruning During Training. Soft-pruning during training provides comparable accuracy to pruning after training for up to 75% pruning rate while preventing excess computation on weight updates. Additionally, when soft-pruning is applied during training, the classification accuracy is maintained at ~94% with a pruning rate up to 60%. The aim of our work is mainly energy optimization during SNN training. Therefore, soft-pruning is chosen to maintain high accuracy with larger pruning percentage, while providing significant energy reduction during training. Since soft-pruning does not completely remove synaptic weights, it is not the best way to achieve memory optimization. Alternatively, conventional pruning [22] presented in this work completely removes synaptic weights and it can be used to reduce the size of memory array used for inference with a little loss in accuracy (Figure 2.9b).

We also compare the number of weight updates of conventional STDP [35], STDP used in this work and STDP used in this work with 50% soft-pruning in Table2.2. Since conventional STDP demonstrated by Song et al. bound the number of weight update of excitatory synapses ($\overline{g_a}$) between 0 and $\overline{g_{max}}$ while our STDP bound the weights between -1 and 1, the number of weight updates of conventional STDP and our STDP are almost the same as shown in the Table2.2. On the other hand, STDP+Soft-pruning significantly reduces the number of device updates for 50% soft pruning. In addition, soft-pruning is conceptually similar to stop learning that has been proposed in semi-supervised models [36, 37]. However, there are two major differences between soft-pruning and stop-learning. Our SNN training is unsupervised. Therefore, the criterion for our soft-pruning to stop updating the synapses is when an output neuron can generate enough count of consecutive spikes to a specific class of MNIST digits (See Section 2.6 in the manuscript). *Brader et al.* [36] use a semi-supervised model. Therefore, stop-learning will happen when the total current h of an output neuron is in agreement with instructor signal (target). The threshold θ is chosen to determine if the output neuron satisfies the criterion. Furthermore, our soft-pruning stops updating part of the synapses of an output neuron depending on the pruning percentage the user set. This means that the un-pruned synapses still can be updated for the rest of the training. However, *Brader et al.* stop updating all the synapses of an output neuron once the stop-learning criterion is satisfied.

Our classification accuracy is comparable to previous software implementations of unsupervised learning for the MNIST dataset with SNNs (Table2.3). As can be seen from the Table, multilayer SNNs [29, 38-40] generally have higher accuracy than single layer SNNs. However, the works with accuracy higher than 95% [38-40] all require using multiple convolution and pooling layers, and other complex processing techniques, which are difficult to implement in hardware. Compared to the SNNs without convolution layers, our classification accuracy is much higher than previous single layer SNNs [3, 41] and achieves performance very close to [29] with much fewer neurons and synapses. Our single layer SNN architecture does not require complex processing and is particularly suitable for easy hardware implementation. Differing from all previous approaches, we present a novel pruning method to reduce the number of updates to network parameters during SNN training. Hence, despite only part of the synapses in our network needing to be updated during training, our SNN still maintains a high classification accuracy with up a 75% pruning rate. Therefore, our pruning scheme can potentially reduce the energy consumption and training time in hardware implementation. The simple one-layer SNN architecture and STDP rule proposed in our work mainly focus on demonstrating the idea of pruning during the training. Scaling our SNN algorithm to larger datasets can be achieved by modifying the network architecture in several approaches such as by adding more fully connected layers [42-44] or convolutional layers [1, 40, 42, 43], adjusting learning rule and involving the supervision [1].

Our single layer SNN network (Figure 2.10a) can be directly mapped to a crossbar array based on eNVM devices (Figure 2.10b) to perform online learning. The input of the network is decoded into a Poisson spike train based on the pixel intensity and it can be mapped to the input voltage spikes of the crossbar array (Figure 2.10A). There are many demonstrations showing that eNVM devices can have multilevel conductance states to emulate analog weight tuning [9, 10].

Therefore, the weights in the SNN can be represented using the conductance of eNVM devices. Since the weights in our network is ranging from -1 to 1, there are two ways to use device conductance to represent the weights. One approach could be using a single device to represent a synaptic weight. The weights in the network are linearly transformed to the conductance range as shown in Eq. 6 for the hardware implementation [45-49].

$$G = W\frac{(G_{max} - G_{min})}{2} + \frac{(G_{max} + G_{min})}{2} \quad (6)$$

An alternative approach could be using of two devices as one synaptic weight as shown in previous literature [47, 50]. Both positive and negative weights can be represented by taking the difference between conductance of two devices ($G = G^{+} - G^{-}$). The weighted sum operation for calculating membrane potential (see Section 2.2 for details) can be calculated in a single step by accumulating the current flowing through each column in the crossbar array [11]. Our STDP weight update rule can be realized by overlapping of the pre-spike and post-spike pulses (Figure 2.10b) to program the device to different conductance levels, as shown in previous demonstrations [9, 51].

In order to implement pruning in hardware, the pruned cells need to be flagged to prevent them from being updated further. One solution is to use an extra binary device associated with each eNVM synaptic weight to serve as a hardware pruning flag. This binary device is initially programmed to ‘0’ (the lowest conductance state), to indicate that the cell has not been pruned. We update the pruning flag of an output neuron’s weights to ‘1’ (the highest conductance state) when it has been pruned during training. Before the weight update, we read the hardware flag of the winning neuron’s weight to decide whether or not to update. The weights are only pruned once during the entire training. As a result, each hardware flag is just written once and hence the energy overhead will be negligible. However, the hardware pruning flag will slightly increase the area of the array. If the size of the array is crucial for a system, an alternative way can be used to implement the hardware flag without area overhead. The pruned cells can be reset to a very low conductance state with additional reset current [52, 53]. Such cells generally require reforming to be programmed to a multi-level conductance state regime again [54]. Therefore, the pruned cells will not be further updated during training and we can use its very low conductance state as pruning flag.

In order to confirm the feasibility of the proposed hardware implementation of pruning during SNN training. We perform circuit-level benchmarking simulations with NeuroSim [49] to evaluate the performance of a full system of analog synaptic core as shown in Figure 2.11a. NeuroSim is a C++ based simulator with hierarchical organization starting from experimental device data and extending to array architectures with peripheral circuit modules and algorithm-level neural network models [49]. We develop a SNN platform for NeuroSim (SNN+NeuroSim). SNN+NeuroSim can simulate circuit-level performance metrics (area, energy and latency) at run-time of online learning using eNVM arrays. We implement the hardware flagging mechanism of pruning in SNN+NeuroSim and estimate energy and latency overheads caused by flagging mechanism. Figure 2.11b and c show energy and latency without and with overheads due to pruning. The results show that the energy and latency can be significantly decreased as the pruning percentages increase. The results also suggest that energy consumption and latency do not significantly increase due to the overheads associated with the hardware flag for the pruning percentages from 10% to 80%.

2.5 Conclusion

In this work, we first demonstrate a low-complexity single layer SNN training model for unsupervised learning on MNIST. We then develop a new method to prune during training for SNNs. Our pruning scheme exploits the output spike firing of the SNN to reduce the number of weight updates during network training. With this method, we investigate the impact of pruning and soft-pruning on classification accuracy. We show that our SNN can maintain high classification accuracy (~90%) on the MNIST dataset and the network can be extensively pruned (75% pruning rate) during training. We also discuss and simulate the possible hardware implementation of our SNN and pruning algorithm with eNVM crossbar arrays using SNN+NeuroSim. Our algorithmic optimization approach can be applied to improve network level energy efficiency of other SNNs with eNVM arrays for in-memory computing applications, enabling online learning of SNNs in power-limited settings.

2.6 Acknowledgments

The authors acknowledge support from the Office of Naval Research Young Investigator Award (N00014161253), National Science Foundation (ECCS-1752241, ECCS-1734940) and Qualcomm FMA Fellowship for funding this research.

Chapter 2 is a reprint of Yuhan Shi, Leon Nguyen, Sangheon Oh, Xin Liu, and Duygu Kuzum. "A soft-pruning method applied during training of spiking neural networks for in-memory computing applications." Frontiers in neuroscience 13 (2019): 405. The dissertation author was the first author of this paper.

2.7 Figures

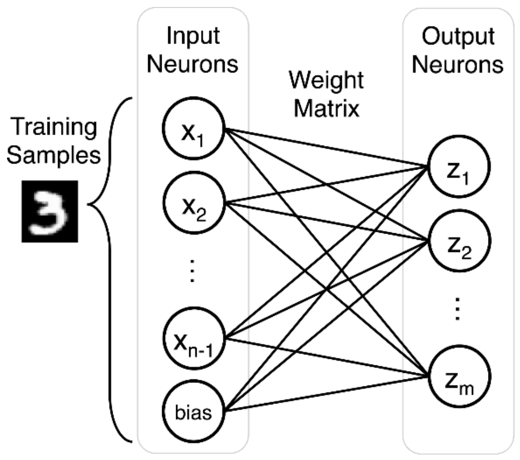


Figure 2. 1 Spiking Neural Network Architecture

Each training sample is represented by *n* input neurons (*n*-1 input neurons and a bias term). The weight matrix is an *m* by *n* array composed of the weights that connect input neurons to output neurons in the single layer network.

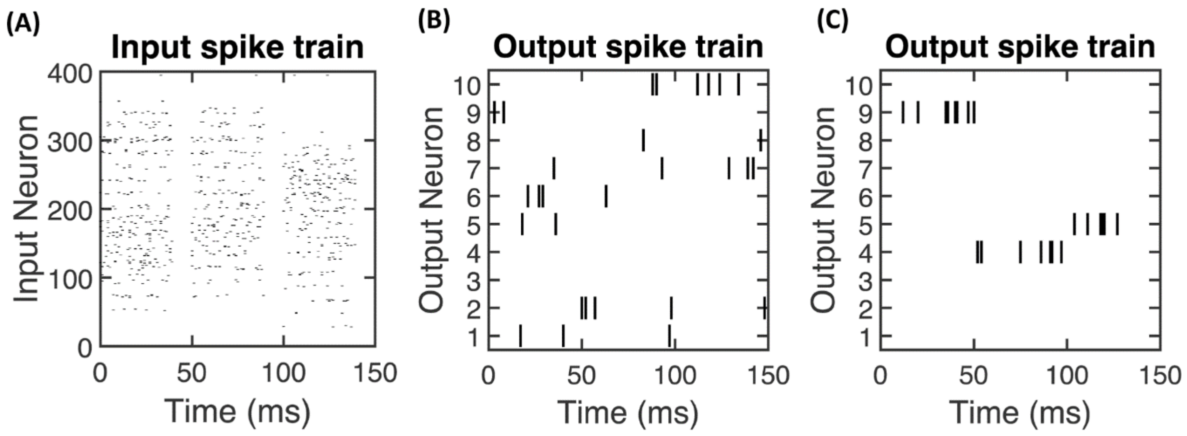


Figure 2. 2 Spiking Neural Network Spiking Raster

Spike raster plots showing examples of (a) input spiking activity, (b) output spiking activity for an untrained SNN, and (c) output spiking activity for a trained SNN. For (b) and (c), 10 output neurons' spiking activities are selected as a representative example. After the SNN is trained, the output spike firing activity is more coordinated, which is indicated by the output neurons selectively firing to certain input stimuli. The time duration on the *x* axis indicates the presentation of training samples. Since the output neuron firing rate is 200Hz, therefore there are around 10 spikes (# of spikes = presentation time × frequency = 50ms × 0.001 × 200Hz = 10) will be generated within 50ms presentation time.

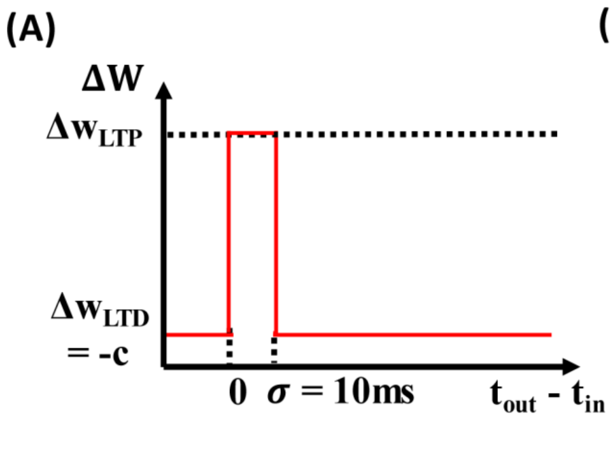
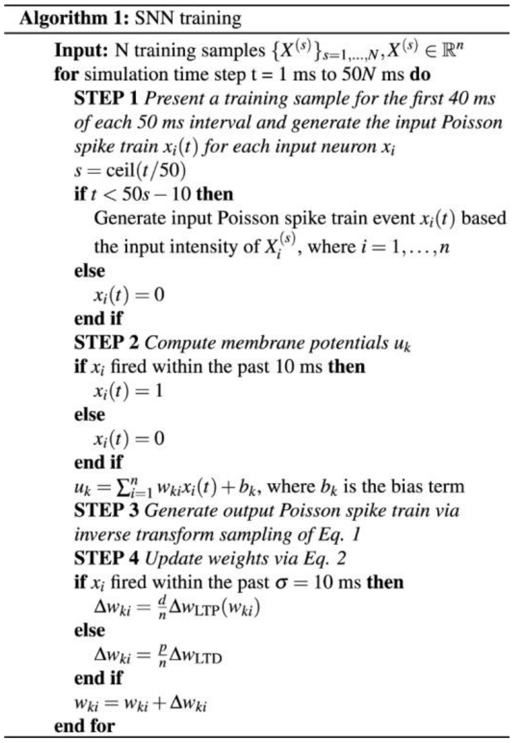


Figure 2. 3 Spiking Neural Network STDP and Weight Update Rule

(a) STDP rule showing the 10 ms window for an output-input spike time difference (*t*$_{out}$ – *t*$_{in}$) that determines whether an LTP or an LTD update is performed. If the output-input spike time difference (*t*$_{out}$ – *t*$_{in}$) is within 10 ms, the weight corresponding to this input-output synapse is updated via LTP. Otherwise, the weight is updated via LTD. The LTP update is an exponential function that depends on the current weight, and the LTD update is a constant. (b) The exponential LTP update is dependent on the current weight *w* and it helps keep the weight values within the range [-1, 1].



```
Algorithm 1: SNN training
  Input: N training samples $\{X^{(s)}\}_{s=1,\dots,N}, X^{(s)} \in \mathbb{R}^n$
  for simulation time step t = 1 ms to 50N ms do
    STEP 1 Present a training sample for the first 40 ms
    of each 50 ms interval and generate the input Poisson
    spike train $x_i(t)$ for each input neuron $x_i$
    $s = \text{ceil}(t/50)$
    if $t < 50s - 10$ then
      Generate input Poisson spike train event $x_i(t)$ based
      the input intensity of $X_i^{(s)}$, where $i = 1, \dots, n$
    else
      $x_i(t) = 0$
    end if
    STEP 2 Compute membrane potentials $u_k$
    if $x_i$ fired within the past 10 ms then
      $x_i(t) = 1$
    else
      $x_i(t) = 0$
    end if
    $u_k = \sum_{i=1}^{n} w_{ki} x_i(t) + b_k$, where $b_k$ is the bias term
    STEP 3 Generate output Poisson spike train via
    inverse transform sampling of Eq. 1
    STEP 4 Update weights via Eq. 2
    if $x_i$ fired within the past $\sigma = 10$ ms then
      $\Delta w_{ki} = \frac{d}{n} \Delta w_{\text{LTP}}(w_{ki})$
    else
      $\Delta w_{ki} = \frac{p}{n} \Delta w_{\text{LTD}}$
    end if
    $w_{ki} = w_{ki} + \Delta w_{ki}$
  end for
```

Figure 2. 4 Spiking Neural Network Training Algorithm

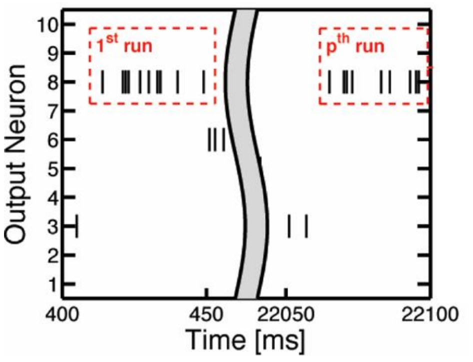


Figure 2. 5 Spiking Neural Network Output Spike and Pruning Criteria

The illustration of consecutive output spikes of 10 output neurons as a representative example. The consecutive output spikes of Neuron 8 are boxed in red.

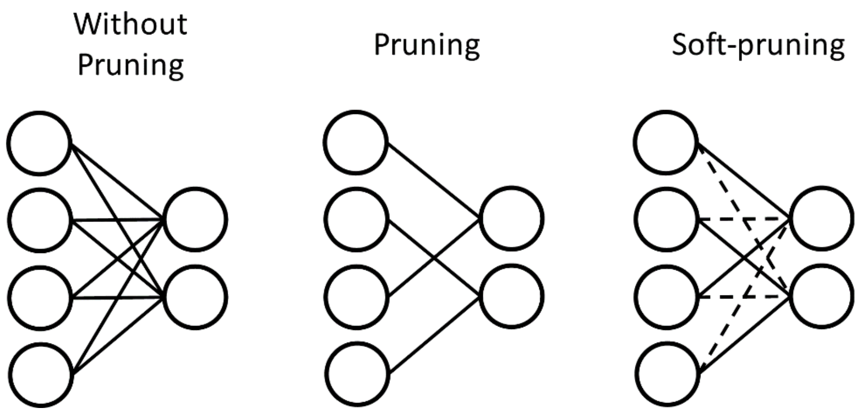


Figure 2. 6 Pruning Schemes

Pruning prunes weights with values around 0 by setting these weights to 0. Much like a sparse matrix, these weights do not have to be stored in memory if their index is stored. Therefore, the synapses are physically removed to represent pruning. In contrast, soft-pruning prunes weights with values meeting certain criteria by keeping these weights constant at a certain value for the rest of training. Therefore, the pruned weights still need to be stored because they can be nonzero, but they are represented by dashed lines to indicate that they no longer need to be updated during training of the SNN.

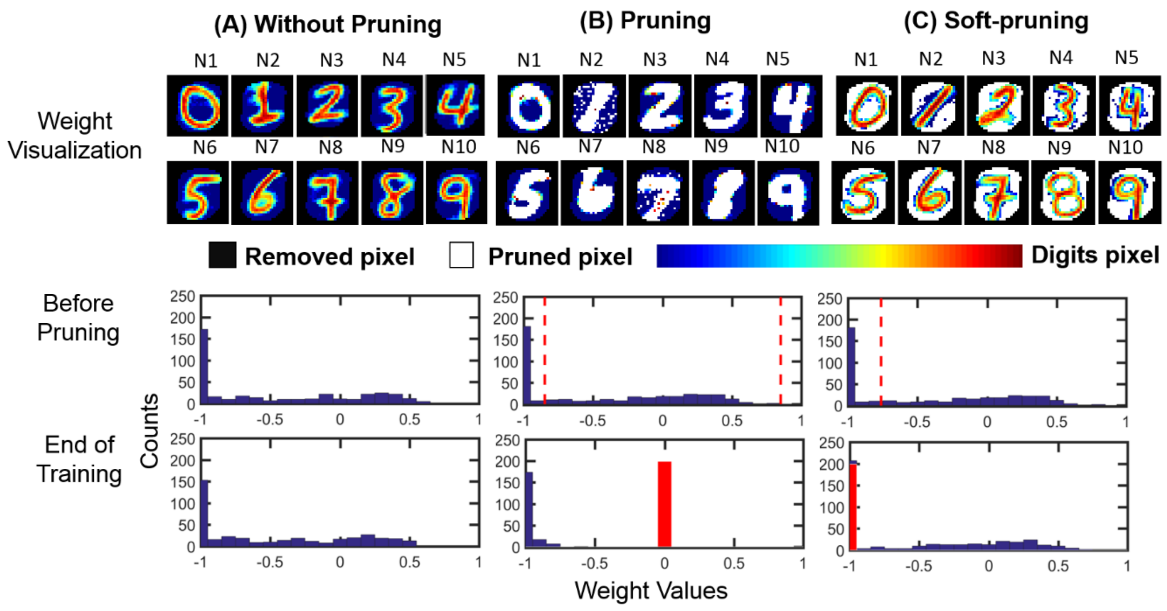


Figure 2. 7 Pruning Weight Visualization

Pruning example. (Top row) Weight visualization of the 10 representative output neurons after the SNN is trained in 3 different cases: (1) without pruning, (2) with pruning (while pruning 50% of the weights), and (3) with soft-pruning (while pruning 50% of the weights). (Middle row) Weight distribution of a representative output neuron (N3) before it is about to be pruned during training. (Bottom row) Weight distribution of the same output neuron at the end of training. It is worth to note that we only recover the removed pixels (black pixels) for visualizing the learned weights. Since those pixels are not used during training, histograms only include corresponding digit pixels (color bar) and pruned pixels (white pixels). (a) Without pruning, the STDP rule causes the weights for the MNIST dataset to follow a distribution where many of the weight values saturate at the lowest possible value. These low weights represent the background of MNIST training samples that are being learned by the SNN. (b) Pruning prunes the weights between the two dashed lines, which represent the 50% of the weights that centered about 0, and sets their values to 0 (red bar). (c) Soft-pruning prunes the weights to the left of the dashed line, which represent the 50% of the weights that are the lowest-valued weights, to the lowest possible value, which is at -1 (red bar). For both pruning and soft-pruning, the weights that are not pruned continue to develop for the rest of training.

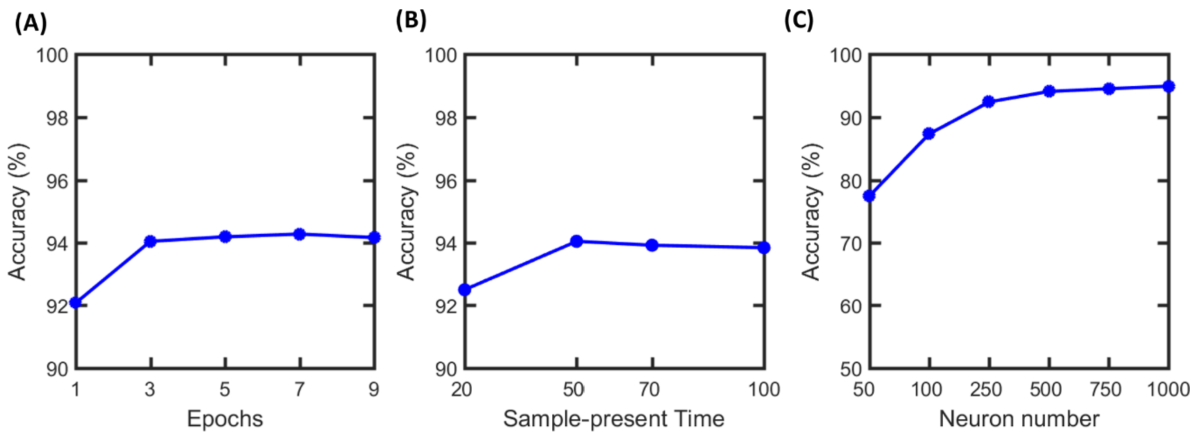


Figure 2. 8 Classification Accuracy vs. Epochs, Sample-present Time and Neuron Number

Classification accuracy vs. (a) number of training epochs, (b) sample-present time and (c) output neuron numbers. Classification accuracy does not have noticeable increase after 3 epochs and 50ms present time. Therefore, 3 epochs and 50ms are used in the training. Although the accuracy can be further improved if neuron number increases, it will significantly increase the simulation time. Therefore, we choose to use 500 output neurons in our simulation.

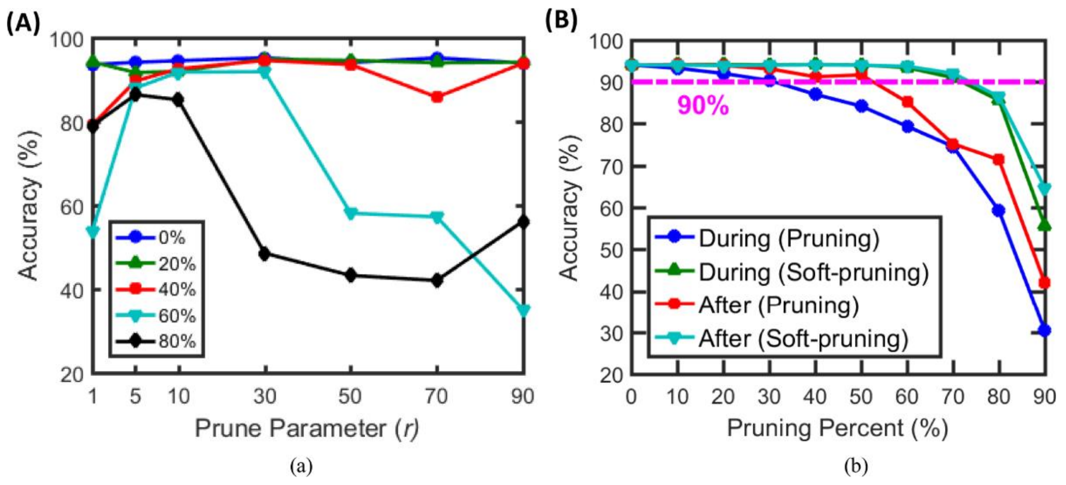


Figure 2. 9 Classification Accuracy vs. Pruning Parameter

(a) Classification accuracy vs. prune parameter (*r*) for varying pruning percentages. Prune parameter is the criterion to decide when to prune for each neuron during training. (b) Classification accuracy vs. pruning percentage for pruning and soft-pruning when applied during training and after training. The data points are taken in steps of 10%. The dashed line represents classification accuracy of 90%. Soft-pruning during the training performs better than pruning especially for high pruning percentages. Soft-pruning maintains > 90% up to 75% pruning percentage while pruning falls below 90% at only 40% pruning. Although we focus on pruning during training, we also present results from pruning weights after training as a baseline for previously established pruning methods from the literature. The parameters used in the simulation are specified in Table2.1.

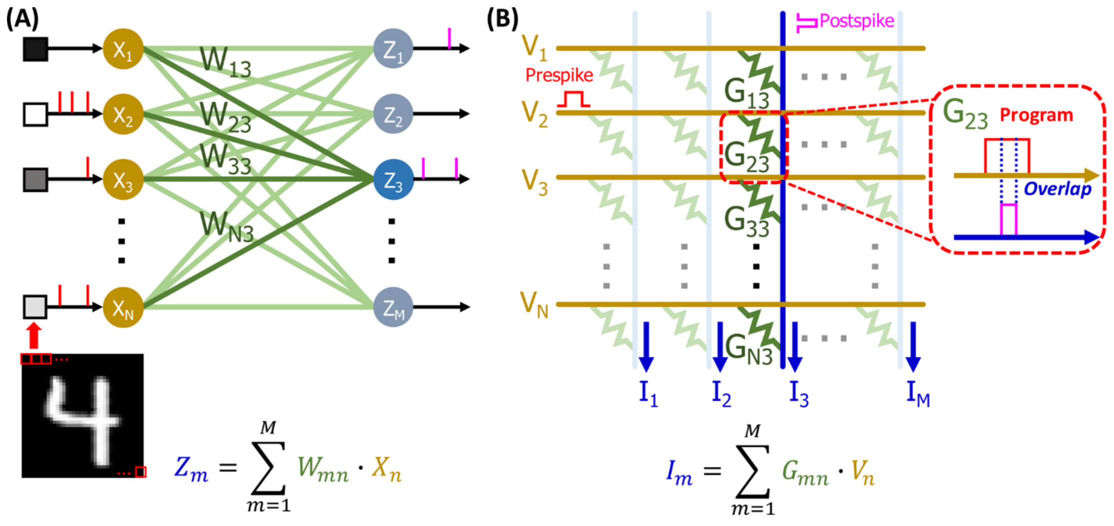


Figure 2. 10 Hardware Mapping to Crossbar Array

Schematic of SNN with *n* input neurons and *m* output neurons. The pixel intensities of input image are decoded into passion spiking training and fed to the input of the network. The weights (W_{13}, W_{23}, …, W_{N3}) of output neuron Z_3 has been highlighted. (b) Schematic of a crossbar array based on eNVM devices. The input of (a) can be mapped to the voltage. The weights (W_{13}, W_{23}, …, W_{N3}) are mapped to the conductance (G_{13}, G_{23}, …, G_{N3}) of the devices. The weighted sum can be obtained by measuring the current at the end of each column. The post-spike pulses are generated based on the weighted sums (*I*). The overlap of pre and post spike pulses as shown in callout window programs the device to different conductance states.

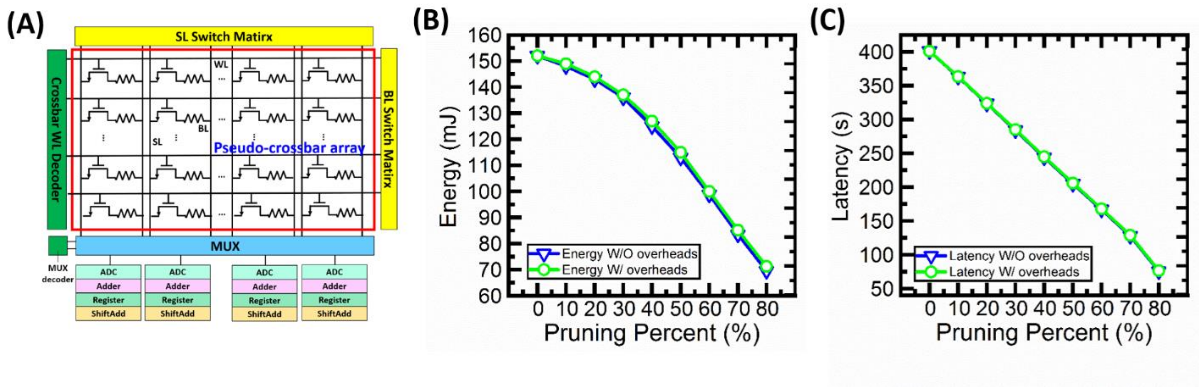


Figure 2. 11 Hardware Energy and Latency Overhead

(a) Analog synaptic core uses a single cell with multi-level conductance states to represent one synaptic weight. One transistor is added to each cell in order to avoid sneak path problem. The crossbar wordline (WL) decoder can activate all WLs, bitline (BL) read out the weighted sum results, and source line (SL) can be used to perform weight update. Multiplexer (MUX) is used to share the neuron circuitry. The neuron circuit contains analog-to-digital converters (ADCs), adders, registers and shift adders, which are used to perform weighted sum. (b) The energy and (c) latency without and with overhead estimation for soft-pruning from 10% to 80% with a step of 10% using SNN+NeuroSim. Without overheads (W/O overheads) results mean that flagging mechanism is implemented in software. With overheads (W/ overheads) results mean that flagging mechanism is implemented in hardware.

Table 2. 1 Network Simulation Parameters

Simulation parameters used in training, labeling and testing for this work.

Parameters		**10-Digits**		
		Training	**Labeling**	**Testing**
# of Neuron	Input	398		
	Output	500		
Firing Rate (Hz)	Input	200	200	200
	Output	200	200	600
Image Presenting Time (ms)		50	50	200
Neuron Removal Threshold		-	0.75	-
Pruning Threshold	Prune Parameter (*r*)	10	-	-
	Spike Count	8	-	-

STDP		a = 0.0667
		b = 2.5
		c = 0.0167		

Table 2. 2 Number of Weight Updates in Different Works

The number of weight updates of conventional STDP (Song et al., 2000), STDP used in this work with and without 50% soft-pruning.

	# of Weight Updates
Conventional STDP (Song et al., 2000)	649289638
STDP (this work)	648669156
STDP (this work) + 50% Soft-pruning	357656929

Table 2. 3 Benchmarking Comparison with Other Works

Classification accuracy comparison between this work and the state-of-the-art software demonstrations of unsupervised learning of SNNs on the MNIST dataset. The Table lists the complex processing techniques used, the learning rule, and the #Neurons/synapses used in each work. The Table also indicates if pruning during training is involved in the work. The numbers of neurons are counted by summing the input and output neurons.

Architecture	**Complex Processing**	**Learning rule**	**#Neurons/synapses**	**Pruning during training**	**Performance**
Spiking deep neural network (Kheradpisheh et al., 2017)	Convolution, DOG filter, Pooling	Simplified STDP	N/A	None	98.4%
Multi layer (Ferré et al, 2018)	Convolution, Pooling, Dropout	Binary STDP	N/A	None	98.49%
Three layer (Tavanaei et al, 2017)	Convolution, Pooling	Probabilistic STDP	N/A	None	98.36%
Two layer (Diehl et al., 2015)	None	Exponential STDP	7,184/5,017,600	None	95%
One layer (Al-Shedivat et al., 2015)	Population Coding	Probabilistic STDP	1,696/200,704	None	78.4%
One layer (Nessler et al., 2013)	Population Coding	Exponential STDP	808/70,800	None	80.14%
One layer (this work)	**None**	**Simplified STDP**	**898/~199,000**	**Yes (~75%)**	**94.05%**

2.8 References

[1] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization," Neural Networks, vol. 103, pp. 118-127, 2018.

[2] W. Maass, "Networks of spiking neurons: the third generation of neural network models," Neural networks, vol. 10, no. 9, pp. 1659-1671, 1997.

[3] B. Nessler, M. Pfeiffer, L. Buesing and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," PLoS computational biology, vol. 9, no. 4, p. e1003037, 2013.

[4] A. Tavanaei, T. Masquelier and A. S. Maida, "Acquisition of visual features through probabilistic spike-timing-dependent plasticity," in Neural Networks (IJCNN), 2016 International Joint Conference on, 2016: IEEE, pp. 307-314.

[5] Y. Cao, Y. Chen and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," International Journal of Computer Vision, vol. 113, no. 1, pp. 54-66, 2015.

[6] J. M. Cruz-Albrecht, M. W. Yung and N. Srinivasa, "Energy-efficient neuron, synapse and STDP integrated circuits," IEEE transactions on biomedical circuits and systems, vol. 6, no. 3, pp. 246-256, 2012.

[7] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, K. S. Esser, R. Appuswamy, B. Taba, A. Amir, D. M. Flickner, P. W. Risk, R. Manohar and S. D. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," Science, vol. 345, no. 6197, pp. 668-673, 2014.

[8] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," Frontiers in neuroscience, vol. 7, p. 272, 2014.

[9] D. Kuzum, R. G. Jeyasingh, B. Lee and H.-S. P. Wong, "Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing," Nano letters, vol. 12, no. 5, pp. 2179-2186, 2011.

[10] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," Nano letters, vol. 10, no. 4, pp. 1297-1301, 2010.

[11] S. B. Eryilmaz, E. Neftci, S. Joshi, S. Kim, M. BrightSky, H.-L. Lung, C. Lam, G. Cauwenberghs and H.-S. P. Wong, "Training a probabilistic graphical model with resistive switching electronic synapses," IEEE Transactions on Electron Devices, vol. 63, no. 12, pp. 5004-5011, 2016.

[12] S. Choi, P. Sheridan and W. D. Lu, "Data clustering using memristor networks," Scientific reports, vol. 5, p. 10492, 2015.

[13] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," Nature, vol. 521, no. 7550, p. 61, 2015.

[14] H.-S. P. Wong, "The End of the Road for 2D Scaling of Silicon CMOS and the Future of Device Technology," in 2018 76th Device Research Conference (DRC), 2018: IEEE, pp. 1-2.

[15] R. Ge, X. Wu, M. Kim, J. Shi, S. Sonde, L. Tao, Y. Zhang, J. C. Lee and D. Akinwande, "Atomristor: Nonvolatile Resistance Switching in Atomic Sheets of Transition Metal Dichalcogenides," Nano letters, vol. 18, no. 1, pp. 434-441, 2017.

[16] F. Alibart, E. Zamanidoost and D. B. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," Nature communications, vol. 4, p. 2072, 2013.

[17] A. Ankit, A. Sengupta, P. Panda and K. Roy, "Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks," in Proceedings of the 54th Annual Design Automation Conference 2017, 2017: ACM, p. 27.

[18] P. Panda, G. Srinivasan and K. Roy, "Convolutional spike timing dependent plasticity based feature learning in spiking neural networks," arXiv preprint arXiv:1703.03854, 2017.

[19] P. Panda, G. Srinivasan and K. Roy, "EnsembleSNN: Distributed assistive STDP learning for energy-efficient recognition in spiking neural networks," in 2017 International Joint Conference on Neural Networks (IJCNN), 2017: IEEE, pp. 2629-2635.

[20] A. Sengupta, M. Parsa, B. Han and K. Roy, "Probabilistic deep spiking neural systems enabled by magnetic tunnel junction," IEEE Transactions on Electron Devices, vol. 63, no. 7, pp. 2963-2970, 2016.

[21] G. Srinivasan, A. Sengupta and K. Roy, "Magnetic tunnel junction based long-term short-term stochastic synapse for a spiking neural network with on-chip STDP learning," Scientific reports, vol. 6, p. 29545, 2016.

[22] S. Han, J. Pool, J. Tran and W. Dally, "Learning both weights and connections for efficient neural network," in Advances in neural information processing systems, 2015, pp. 1135-1143.

[23] M. Deger, M. Helias, S. Rotter and M. Diesmann, "Spike-timing dependence of structural plasticity explains cooperative synapse formation in the neocortex," PLoS computational biology, vol. 8, no. 9, p. e1002689, 2012.

[24] M. Deger, A. Seeholzer and W. Gerstner, "Multicontact Co-operativity in Spike-Timing–Dependent Structural Plasticity Stabilizes Networks," Cerebral Cortex, vol. 28, no. 4, pp. 1396-1415, 2017.

[25] J. Iglesias and A. E. Villa, "Effect of stimulus-driven pruning on the detection of spatiotemporal patterns of activity in large neural networks," Biosystems, vol. 89, no. 1-3, pp. 287-293, 2007.

[26] D. Kappel, S. Habenschuss, R. Legenstein and W. Maass, "Synaptic sampling: A Bayesian approach to neural network plasticity and rewiring," in Advances in Neural Information Processing Systems, 2015, pp. 370-378.

[27] R. Spiess, R. George, M. Cook and P. U. Diehl, "Structural plasticity denoises responses and improves learning speed," Frontiers in computational neuroscience, vol. 10, p. 93, 2016.

[28] N. Rathi, P. Panda and K. Roy, "STDP based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[29] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," Frontiers in computational neuroscience, vol. 9, p. 99, 2015.

[30] A. Gupta and L. N. Long, "Character recognition using spiking neural networks," in Neural Networks, 2007. IJCNN 2007. International Joint Conference on, 2007: IEEE, pp. 53-58.

[31] A. Perfors, J. B. Tenenbaum, T. L. Griffiths and F. Xu, "A tutorial introduction to Bayesian models of cognitive development," Cognition, vol. 120, no. 3, pp. 302-321, 2011.

[32] A. F. Vincent, J. Larroque, W. Zhao, N. B. Romdhane, O. Bichler, C. Gamrat, J.-O. Klein, S. Galdin-Retailleau and D. Querlioz, "Spin-transfer torque magnetic memory as a stochastic memristive synapse," in 2014 IEEE International Symposium on Circuits and Systems (ISCAS), 2014: IEEE, pp. 1074-1077.

[33] B. Kijsirikul and K. Chongkasemwongse, "Decision tree pruning using backpropagation neural networks," in Proceedings of IEEE Int. Conf. on Neural Networks, 2001, vol. 3, pp. 1876-1880.

[34] 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, "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, p. 529, 2015.

[35] S. Song, K. D. Miller and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," Nature neuroscience, vol. 3, no. 9, p. 919, 2000.

[36] J. M. Brader, W. Senn and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," Neural computation, vol. 19, no. 11, pp. 2881-2912, 2007.

[37] H. Mostafa, C. Mayr and G. Indiveri, "Beyond spike-timing dependent plasticity in memristor crossbar arrays," in 2016 IEEE International Symposium on Circuits and Systems (ISCAS), 2016: IEEE, pp. 926-929.

[38] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," Neural Networks, vol. 99, pp. 56-67, 2018.

[39] P. Ferré, F. Mamalet and S. J. Thorpe, "Unsupervised Feature Learning With Winner-Takes-All Based STDP," Frontiers in computational neuroscience, vol. 12, p. 24, 2018.

[40] A. Tavanaei and A. S. Maida, "Multi-layer unsupervised learning in a spiking convolutional neural network," in Neural Networks (IJCNN), 2017 International Joint Conference on, 2017: IEEE, pp. 2023-2030.

[41] M. Al-Shedivat, R. Naous, G. Cauwenberghs and K. N. Salama, "Memristors empower spiking neurons with stochasticity," IEEE journal on Emerging and selected topics in circuits and systems, vol. 5, no. 2, pp. 242-253, 2015.

[42] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in 2015 International Joint Conference on Neural Networks (IJCNN), 2015: IEEE, pp. 1-8.

[43] J. H. Lee, T. Delbruck and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," Frontiers in neuroscience, vol. 10, p. 508, 2016.

[44] P. O'Connor and M. Welling, "Deep spiking networks," arXiv preprint arXiv:1602.08323, 2016.

[45] A. Serb, J. Bill, A. Khiat, R. Berdan, R. Legenstein and T. Prodromakis, "Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses," Nature communications, vol. 7, p. 12611, 2016.

[46] S. Oh, Y. Shi, X. Liu, J. Song and D. Kuzum, "Drift-Enhanced Unsupervised Learning of Handwritten Digits in Spiking Neural Network With PCM Synapses," IEEE Electron Device Letters, vol. 39, no. 11, pp. 1768-1771, 2018.

[47] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves, Z. Li, J. P. Strachan, P. Lin, Z. Wang, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang and Q. Xia, "Analogue signal and image processing with large memristor crossbars," Nature Electronics, vol. 1, no. 1, p. 52, 2018.

[48] H. Kim, T. Kim, J. Kim and J.-J. Kim, "Deep neural network optimized to resistive memory with nonlinear current-voltage characteristics," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 14, no. 2, p. 15, 2018.

[49] P.-Y. Chen, X. Peng and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 12, pp. 3067-3080, 2018.

[50] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," IEEE Transactions on Electron Devices, vol. 62, no. 11, pp. 3498-3507, 2015.

[51] D. Kuzum, R. G. D. Jeyasingh, S. Yu and H. S. P. Wong, "Low-Energy Robust Neuromorphic Computation Using Synaptic Devices," IEEE Transactions on Electron Devices, vol. 59, no. 12, pp. 3489-3494, 2012.

[52] L. Xia, M. Liu, X. Ning, K. Chakrabarty and Y. Wang, "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in Proceedings of the 54th Annual Design Automation Conference 2017, 2017: ACM, p. 33.

[53] M. Arita, A. Takahashi, Y. Ohno, A. Nakane, A. Tsurumaki-Fukuchi and Y. Takahashi, "Switching operation and degradation of resistive random access memory composed of tungsten oxide and copper investigated using in-situ TEM," Scientific reports, vol. 5, p. 17103, 2015.

[54] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen and M.-J. Tsai, "Metal–oxide RRAM," Proceedings of the IEEE, vol. 100, no. 6, pp. 1951-1970, 2012

Chapter 3. Performance Prospects of Deeply Scaled Spin-transfer Torque Magnetic Random-access Memory for In-memory Computing

3.1 Abstract

In recent years, Spin-Transfer-Torque Magnetic Random Access Memory (STT-MRAM) has been considered as one of the most promising non-volatile memory candidates for in-memory computing. However, system-level performance gains using STT-MRAM for in-memory computing at deeply scaled nodes have not been assessed with respect to more mature memory technologies. In this letter, we present perpendicular magnetic tunnel junction (pMTJ) STT-MRAM devices at 28nm and 7nm. We evaluate the system-level performance of convolutional neural network (CNN) inference with STT-MRAM arrays in comparison to Static Random Access Memory (SRAM). We benchmark STT-MRAM and SRAM in terms of area, leakage power, energy, and latency from 65nm to 7nm technology nodes. Our results show that STT-MRAM keeps providing ~5× smaller synaptic core area, ~20× less leakage power, and ~7× less energy than SRAM when both devices are scaled from 65nm to 7nm. With the emerging need for low power computation for a broad range of applications such as internet-of-things (IoT) and neural network (NN), STT-MRAM can offer energy-efficient and high-density in-memory computing.

3.2 Introduction

Neural networks (NNs) have revolutionized artificial intelligence (AI) and achieved remarkable advances in image recognition and classification. However, transferring a massive amount of NN parameters between memory and processor has become the major bottleneck dominating the system-level energy consumption with conventional von Neumann approach due to its insufficient on-chip memory resources. In-memory computing architectures have been proposed to minimize the need for data transfer between memory and processor by integrating computation units inside the memory. As one of the mainstream CMOS based memories, Static Random Access Memory (SRAM), has been demonstrated for implementing in-memory computing of NN [1-3]. On the other hand, emerging non-volatile memory (eNVM) devices [4-7] attract a lot of interest due to its compact cell size (4-12F^2) compared to SRAM (150-840F^2) for in-memory computing among various emerging memories Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) [8-11] has been considered as one of the most promising candidates owing to its unique features such as non-volatility, near-zero standby leakage, high write/read speed, CMOS compatibility, scalability, excellent endurance and high integration density [8, 10]. It has been shown that large-scale NN models can be loaded to high-density STT-MRAM chips at 22nm [12] to perform near-memory computing and are more energy-efficient than SRAM. Several works have shown that STT-MRAM can perform in-memory logic operations [13] and in-memory NN operations at 45nm [14]. However, since the state-of-the-art integration of STT-MRAM with CMOS is at 22nm [12] or 28nm [11, 15], there is no study on performance gains of using STT-MRAM for in-memory computing at deeply scaled technology nodes (<14nm). Moreover, there are no systematic and comprehensive evaluations on the system-level performance of STT-MRAM against SRAM for in-memory computing.

In this letter, we present 28nm and 7nm perpendicular magnetic tunnel junction (pMTJ) STT-MRAM technology for a comprehensive investigation of the hardware implementation of a NN model using STT-MRAM and SRAM. We develop a CNN framework (CNN+NeuroSim) for NeuroSim, which is an integrated device-to-algorithm simulation platform [16-19]. We incorporate the measured device data into our simulator and use a LeNet-5 model to perform inference on MNIST dataset to benchmark the area, leakage power, energy consumption, and latency for STT-MRAM and SRAM scaled from 65nm to 7nm.

3.3 Results

3.3.1 STT-MRAM Fabrication and Characterization

Co/Pt-based pMTJ stacks with dual MgO interfaces (Figure 3.1a, b) were deposited by an Applied Materials Endura® sputtering system on 300mm wafers. pMTJ was patterned using 193nm dry lithography and advanced etching tools to define pMTJ arrays with 40-50nm in diameter and 130-200nm in pitch. Transmission electron microscopy (TEM) and scanning electron microscopy (SEM) images of the array after patterning are shown in Figure 3.1b, c [11]. Figure 3.1d shows a packaged STT-MRAM chip fabricated by 28nm CMOS technology.

To achieve high accuracy for the inference with STT-MRAM array, high tunneling magnetoresistance (TMR) for enough sensing margin, low write error rate (WER) for accurate weight transfer, low read disturbance rate (RDR) for accurate weighted sum operation are required [14]. Figure 3.2a shows the typical 28nm STT-MRAM switching loops, which represents that the device has a good TMR ratio ($(R_{AP} - R_P)/R_P = 116\%$) [11]. High resistance state (R_{AP}) and low resistance stage (R_P) distributions with variation $\sigma R/R = 0.074$ are shown in Figure 3.2b. Since MTJ is a two-terminal device, high read voltage can disturb the programmed data of a cell during read operations. To perform an accurate weighted sum operation in memory, RDR close to 10^{-6} is required [14]. Our device can meet this requirement with 0.16V/10ns pulse scheme while maintaining a relatively high thermal stability factor of $\Delta = 56$ (Figure 3.2c) [20]. The probabilistic switching nature of a ferromagnetic layer can cause write errors. To guarantee that the trained weights can be accurately loaded into STT-MRAM array, the WER close to 10^{-6} is needed. Our device meets this requirement with a write pulse from ±0.4V (100ns) to ±0.6V (5ns) (Figure 3.2d) [20]. As the pulse width increases, the WER slopes increase, indicating the smaller temporal variation.

Although the aforementioned characteristics are for 28nm MTJ, we have shown that the MTJ can be further scaled to be compatible with 7nm CMOS [8, 20]. The bitcell is built on 7nm FinFET transistors (2-Fin NFET and 2-Fin PFET) and the target MTJ resistance and the cell and array switching margins were controlled by tuning a combination of MTJ diameter of 30-35 nm and resistance-area product (RA) of 4-5 Ωmm^2. Table3.1 summarizes the device characteristics of both nodes, which are used for the simulations discussed in Section III.

3.3.2 Circuit-level Performance Benchmark

We construct a LeNet-5 model for supervised learning of MNIST dataset (Figure 3.3a). Figure 3.3b explains the mapping of the network to a synaptic array. The input has dimension

W×W×D, where D is the number of channels. The network has N 2D kernels (k×k). We unrolled each filter into a vector and concatenated D copies of it to form a column (k×k×D×1) in the array. As the filters slide over the input, the part of input overlaps with the filters is also unrolled to a 1D vector (k×k×D×1) and feed into the array to convolve with the kernel in each column. Each pixel value in the receptive field of the input image is multiplied by the corresponding value in the kernel. The weighted sum is accumulated at the end of each column as the pixel value in the feature map (M×M×N).

In order to investigate the system-level performance of STT-MRAM array and benchmark against SRAM array, we develop a CNN framework (CNN+NeuroSim) for NeuroSim, which is a C++ based simulator with a hierarchical organization starting from the experimental device data and extending to array architectures with peripheral circuit modules and algorithm level NN models [16-19]. The architectures used for NN inference with STT-MRAM and SRAM are shown in Figure 3.3c and d, respectively. One-transistor one-resistor (1T1R) pseudo-crossbar array is used for STT-MRAM (Figure 3.3c), while conventional SRAM memory array with modified neuron peripheral circuit is used for the SRAM implementation (Figure 3.3d). Since the cells in both technologies have binary states, we map n-bit weights by grouping n cells to represent one synaptic weight. The major difference between STT-MRAM and SRAM implementation is that STT-MRAM can perform parallel readout using Kirchhoff's current law by turning on all the wordlines (WLs) and reading the weighted sum current from each bitline (BL), while SRAM requires a row-by-row readout to get weighted sum.

To simulate LeNet-5 inference with the proposed architectures, we train the network with full precision and quantize the weights that load onto the memory array. To decide the required bit precision, we investigate the inference accuracy with various bit precision. The accuracy can reach above 98% with 6-bit precision (Figure 3.4a). Hence, we use 6-bit quantization (weights and activations) for the rest of the analysis. The study of the impact of bit precision on system level performance can be found in our previous work [17].

We also explore how the accuracy changes with variations in R_{AP} and R_P. We study three different cases of MTJ variation (σR/R); both R_{AP} and R_P have variations, or either only R_P or R_{AP} has variations. The accuracy is above 98% if the variation is less than 0.05 for all three cases (Figure 3.4b). Variations in R_{AP} degrades accuracy more compared to variations in R_P. It is because when the same amount of percentage variation is applied to either R_{AP} or R_P, the deviation from its nominal value is larger with R_{AP} variation. (e.g. 10% σR/R on R_A (σ_{AP}) = 972Ω while 10% σR/R on R_P (σ_P) = 450Ω). The solid green star symbol is the result of our device with variation (σR/R = 0.074).

To compare the performance of STT-MRAM and SRAM with technology scaling, we simulate digital implementations of MNIST inference (10k test images) with SRAM under technology nodes from 65nm to 7nm (Figure 3.5). Since no hardware demonstration of MTJ devices integrated with CMOS beyond 14nm has been reported, we simulate technology nodes from 65nm to 14nm for STT-MRAM using scaling roadmap from [21]. Our 28nm STT-MRAM is shown as solid green star symbols. For deeply scaled (<14nm) STT-MRAM, we use our 7nm device data (solid blue star symbols). Simulated performance using STT-MRAM results published by other works [9, 14, 22] are also shown using hollow stars symbols for comparison. We extract R_P, R_{AP}, read voltage, and read pulse width of these devices and NeuroSim + CNN computes energy and latency based on these parameters.

Figure 3.5a shows that STT-MRAM synaptic core area is ~5× less than SRAM due to the smaller cell size for technology nodes from 65nm to 32nm and is ~3× less for nodes smaller than 32nm due to the scaling challenge at advanced nodes [8]. The total dynamic energy, leakage power, and total latency that include both memory arrays and peripheral circuits of STT-MRAM during inference are ~7×, ~20× and ~3× less than SRAM from 65nm to 14nm Figure 3.5b-d). For the dynamic energy, the array energy of STT-MRAM is slightly higher than SRAM due to the higher current of STT-MRAM in read operation. However, peripheral circuitry energy dominates the total energy consumption of the system for both STT-MRAM and SRAM case. STT-MRAM has lower total dynamic energy because WL switch matrix can turn on all the WLs simultaneously to achieve parallel read operation while WL decoders in SRAM turn on the WLs one by one. The significant reduction of system-level leakage power (Figure 3.5c) is the most beneficial aspect of using STT-MRAM for NN inference due to its zero leakage current. STT-MRAM has lower overall latency than SRAM (Figure 3.5d) is also due to the parallel read operation enabled by WL switch matrix. Our 28nm device is greatly matched with the predicated scaling trend. Our 7nm device provides 3.7× smaller synaptic core area, 3× less energy and latency, and 16× less leakage power as compared to 7nm SRAM (Figure 3.5). Although the hardware integration of MTJ with CMOS beyond 14nm is yet to be demonstrated, the projected performance gain using our 7nm STT-MRAM data indicates the great potential as an in-memory computing platform. Moreover, the optimized mapping methods such as array partitioning [23] or 3D array architectures [24] can further speed up the inference and reduce energy to achieve higher performance gains.

3.4 Conclusion

We studied and evaluated the system-level performance gains of using STT-MRAM for in-memory computing by incorporating our device data and other literature on STT-MRAM in comparison to SRAM. We showed that STT-MRAM could offer significant performance gains at the system level over SRAM down to 7nm. The performance gains of deeply scaled STT-MRAM support that STT-MRAM is a very promising path for achieving high area and energy efficiency in-memory computing for IoT and NN applications.

3.5 Acknowledgements

This work was supported in the part by Qualcomm FMA fellowship.

Chapter 3 is a reprint of Yuhan Shi, Sangheon Oh, Zhisheng Huang, Xiao Lu, Seung H. Kang, and Duygu Kuzum. "Performance prospects of deeply scaled spin-transfer torque magnetic

random-access memory for in-memory computing." IEEE Electron Device Letters 41, no. 7 (2020): 1126-1129.The dissertation author was the first author of this paper.

3.6 Figures

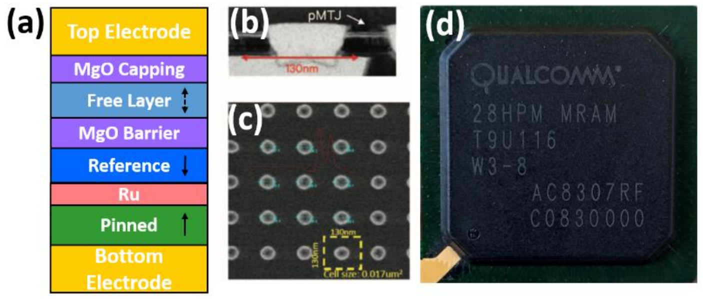


Figure 3. 1 STT-MRAM Device Stack and Chip Image.

(a) pMTJ stack structure [11]. (b) TEM image after pMTJ patterning [11]. (c) SEM image of pMTJ array after Ta hardmask opening [11]. The 130nm pMTJ pitch has a bitcell size of 0.017 μm^2 (d) A packaged perpendicular STT-MRAM chip in 28nm.

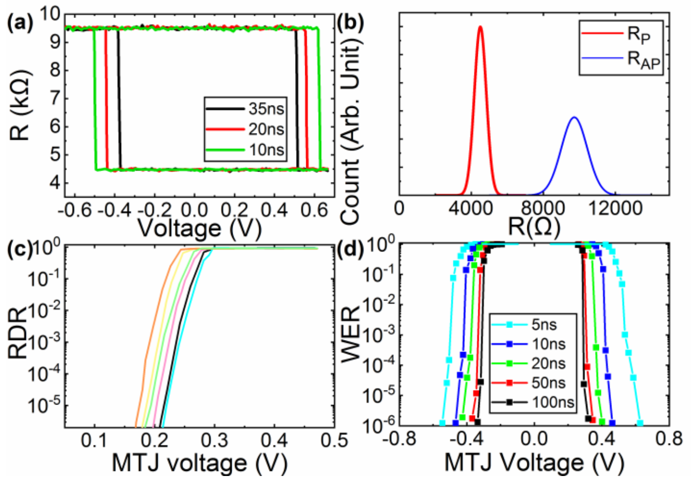


Figure 3. 2 STT-MRAM Switching Characteristics.

(a) Typical STT-MRAM switching loops with different pulse width [11]. (b) RP and RAP distribution. (c) RDR plots with averaged Δ = 56 [20]. (d) WER plot as a function of write voltage at pulse width 5-100ns [20].

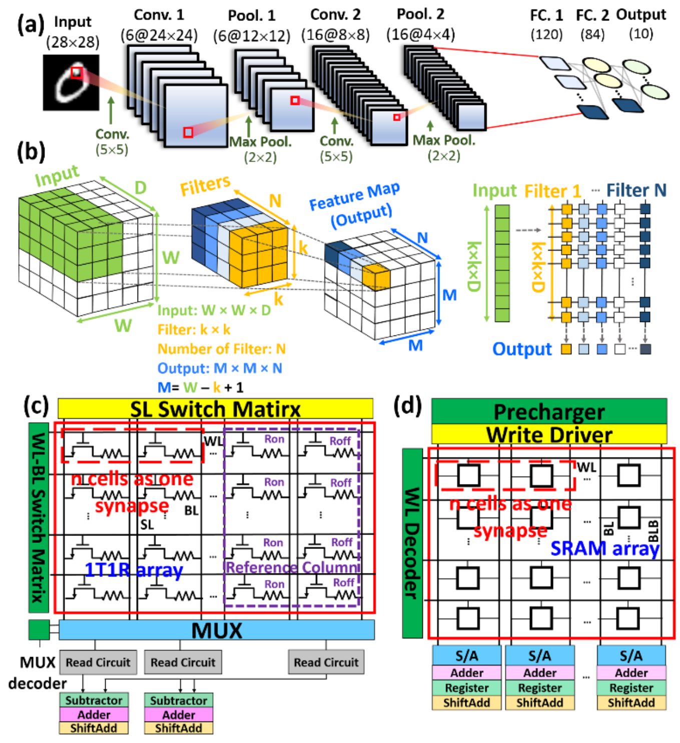


Figure 3. 3 LeNet-5 Network Architecture and Synaptic Core (1T1R and Crossbar) Architecture.

(a) A schematic of LeNet-5. (b) An illustration shows how to map LeNet-5 to a synaptic array, where M (output size) = W (input size) – k (Filter size) + 1 (stride). (c) A schematic of a STT-MRAM 1T1R pseudo-crossbar array. Two reference columns are added for reading the states of STT-MRAM cells. (d) A schematic of SRAM hardware implementation.

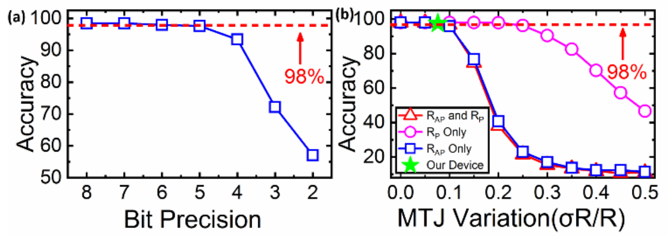


Figure 3. 4 Classification Accuracy vs. Bit Precision and MTJ variation.

(a) Inference accuracy with weights quantized by various bit precision. (b) Inference accuracy with MTJ variation with three cases. The star symbol is based on experimental data (σR/R = 0.074) of our STT-MRAM.

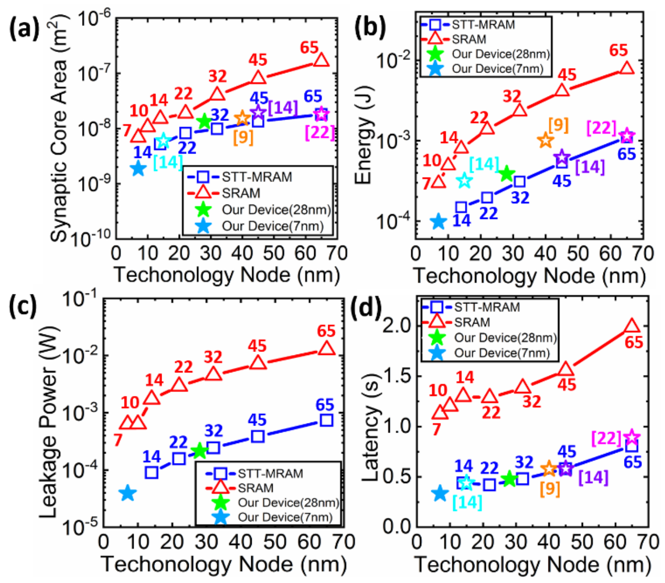


Figure 3. 5 Technology Node vs. Synaptic Core Area, Energy, Leakage Power and Latency.

(a) Synaptic core area. (b) Total dynamic Energy (Memory array energy + peripheral circuitry energy), (c) Leakage power, and (d) Total Latency (Memory array latency + peripheral circuitry latency) with various technology nodes of SRAM and STT-MRAM.

Table 3. 1 Device Characteristics of 28nm and 7nm MTJ

Node	**28nm (chip)**	**7nm (MTJ)**
Bitcell Size	0.017μm^2	0.00684μm^2
R_P	4500Ω	5000Ω
R_{AP}	9720Ω	14000Ω
Read Voltage	0.16V-0.2V (MTJ) 0.9V (cell)	0.1V on (MTJ) 0.75V on (cell)
Read Pulse Width	10ns	4ns
Read Energy	0.36pJ (per cell)	0.003pJ (per cell)

3.7 References

[1] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, and S. Yu, "A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors," in Proc. IEEE Int. Solid-State Circuits Conf., Feb. 2018, pp. 496-498, DOI: 10.1109/ISSCC.2018.8310401.

[2] X. Si, W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Sun, R. Liu, S. Yu, H. Yamauchi, Q. Li, and M.-F. Chang, "A dual-split 6T SRAM-based computing-in-memory unit-macro with fully parallel product-sum operation for binarized DNN edge processors," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 66, no. 11, pp. 4172-4185, Aug. 2019, DOI: 10.1109/TCSI.2019.2928043.

[3] Z. Jiang, S. Yin, M. Seok, and J.-s. Seo, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," in Symp. VLSI Technol., Jun. 2018, pp. 173-174, DOI: 10.1109/VLSIT.2018.8510687.

[4] B. Chen, F. Cai, J. Zhou, W. Ma, P. Sheridan, and W. D. Lu, "Efficient in-memory computing architecture based on crossbar arrays," in IEDM Tech. Dig., Dec. 2015, pp. 17.5. 1-17.5. 4, DOI: 10.1109/IEDM.2015.7409720.

[5] P. Kumbhare, I. Chakraborty, A. Khanna, and U. Ganguly, "Memory performance of a simple Pr 0.7 Ca 0.3 MnO 3-based selectorless RRAM," IEEE Trans. Electron Devices, vol. 64, no. 9, pp. 3967-3970, Jul. 2017, DOI: 10.1109/TED.2017.2725900.

[6] S. B. Eryilmaz, D. Kuzum, R. G. Jeyasingh, S. Kim, M. BrightSky, C. Lam, and H.-S. P. Wong, "Experimental demonstration of array-level learning with phase change synaptic devices," in IEDM Tech. Dig., Dec. 2013, pp. 25.5. 1-25.5. 4, DOI: 10.1109/IEDM.2013.6724691.

[7] S. Oh, Y. Shi, X. Liu, J. Song, and D. Kuzum, "Drift-enhanced unsupervised learning of handwritten digits in spiking neural network with PCM synapses," IEEE Electron Device Lett., vol. 39, no. 11, pp. 1768-1771, Sep. 2018, DOI: 10.1109/LED.2018.2872434

[8] S. H. Kang and C. Park, "MRAM: Enabling a sustainable device for pervasive system architectures and applications," in IEDM Tech. Dig., Dec. 2017, pp. 38.2. 1-38.2. 4, DOI: 10.1109/IEDM.2017.8268514.

[9] Y. Lu, T. Zhong, W. Hsu, S. Kim, X. Lu, J. Kan, C. Park, W. Chen, X. Li, X. Zhu, P. Wang, M. Gottwald, J. Fatehi, L. Seward, J. P. Kim, N. Yu, G. Jan, J. Haq, S. Le, Y. J. Wang, L. Thomas, J. Zhu, H. Liu, Y. J. Lee, R. Y. Tong, K. Pi, D. Shen, R. He, Z. Teng, V. Lam, R. Annapragada, T. Torng, P.-K. Wang, and S. H. Kang, "Fully functional perpendicular STT-MRAM macro embedded in 40 nm logic for energy-efficient IOT applications," in IEDM Tech. Dig., Dec. 2015, pp. 26.1. 1-26.1. 4, DOI: 10.1109/IEDM.2015.7409770.

[10] S. H. Kang, "Embedded STT-MRAM for energy-efficient and cost-effective mobile systems," in Symp. VLSI Technol., Sep. 2014, pp. 1-2, DOI: 10.1109/VLSIT.2014.6894354.

[11] C. Park, J. Kan, C. Ching, J. Ahn, L. Xue, R. Wang, A. Kontos, S. Liang, M. Bangar, H. Chen, S. Hassan, M. Gottwald, X. Zhu, M. Pakala, and S. H. Kang, "Systematic optimization of 1 Gbit perpendicular magnetic tunnel junction arrays for 28 nm embedded STT-MRAM and beyond," in IEDM Tech. Dig., Dec. 2015, pp. 26.2. 1-26.2. 4, DOI: 10.1109/IEDM.2015.7409771.

[12] B. Sun, D. Liu, L. Yu, J. Li, H. Liu, W. Zhang, and T. Torng, "Mram co-designed processing-in-memory cnn accelerator for mobile and iot applications," arXiv:1811.12179, Nov. 2018. [Online]. Available: https://arxiv.org/abs/1811.12179.

[13] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-based In-Memory Accelerator," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Mar. 2019, DOI: 10.1109/TCAD.2019.2907886.

[14] N. Xu, Y. Lu, W. Qi, Z. Jiang, X. Peng, F. Chen, J. Wang, W. Choi, S. Yu, and D. S. Kim, "STT-MRAM design technology co-optimization for hardware neural networks," in IEDM Tech. Dig., Dec. 2018, pp. 15.3. 1-15.3. 4, DOI: 10.1109/IEDM.2018.8614560.

[15] S. Aggarwal, H. Almasi, M. DeHerrera, B. Hughes, S. Ikegawa, J. Janesky, H. Lee, H. Lu, F. Mancoff, K. Nagel, G. Shimon, J. J. Sun, T. Andre, and S. Alam, M, "Demonstration of a Reliable 1 Gb Standalone Spin-Transfer Torque MRAM For Industrial Applications," in IEDM Tech. Dig., Dec. 2019, pp. 2.1. 1-2.1. 4, DOI: 10.1109/IEDM19573.2019.8993516.

[16] P.Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 12, pp. 3067-3080, Jan. 2018, DOI: 10.1109/TCAD.2018.2789723.

[17] Y. Shi, Z. Huang, S. Oh, N. Kaslan, J. Song, and D. Kuzum, "Adaptive quantization as a device-algorithm co-design approach to improve the performance of in-memory unsupervised learning with SNNs," IEEE Trans. on Electron Devices, vol. 66, no. 4, pp. 1722-1728, Feb. 2019, DOI: 10.1109/TED.2019.2898402.

[18] Y. Shi, L. Nguyen, S. Oh, X. Liu, and D. Kuzum, "A Soft-Pruning Method Applied During Training of Spiking Neural Networks for In-memory Computing Applications," Frontiers in neuroscience, vol. 13, p. 405, Apr. 2019, DOI: 10.3389/fnins.2019.00405.

[19] Y. Shi, L. Nguyen, S. Oh, X. Liu, and D. Kuzum, "A Soft-Pruning Method Applied During Training of Spiking Neural Networks for In-memory Computing Applications," Frontiers in neuroscience, vol. 13, p. 405, 2019.

[20] C. Park, H. Lee, C. Ching, J. Ahn, R. Wang, M. Pakala, and S. Kang, "Low RA magnetic tunnel junction arrays in conjunction with low switching current and high breakdown voltage for STT-MRAM at 10 nm and beyond," in in Symp. VLSI Technol., Oct. 2018, pp. 185-186, DOI: 10.1109/VLSIT.2018.8510653.

[21] K. C. Chun, H. Zhao, J. D. Harms, T.-H. Kim, J.-P. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular MTJ based STT-MRAMs for high-density cache memory," IEEE Journal of Solid-State Circuits, vol. 48, no. 2, pp. 598-610, 2012.

[22] K. Tsuchida, T. Inaba, K. Fujita, Y. Ueda, T. Shimizu, Y. Asao, T. Kajiyama, M. Iwayama, K. Sugiura, and S. Ikegawa, "A 64Mb MRAM with clamped-reference and adequate-reference schemes," in Proc. IEEE Int. Solid-State Circuits Conf., Mar. 2010, pp. 258-259, DOI: 10.1109/ISSCC.2010.5433948.

[23] X. Peng, R. Liu, and S. Yu, "Optimizing Weight Mapping and Data Flow for Convolutional Neural Networks on Processing-In-Memory Architectures," IEEE Trans. Circuits Syst. I, Reg. Papers, May. 2019, DOI: 10.1109/ISCAS.2019.8702715.

[24] B. Li, J. R. Doppa, P. P. Pande, K. Chakrabarty, J. X. Qiu, and H. Li, "3D-ReG: A 3D ReRAM-based Heterogeneous Architecture for Training Deep Neural Networks," ACM Journal on Emerging Technol. in Comput. Sys., vol. 16, no. 2, pp. 1-24, Jan. 2020, DOI: 10.1145/3375699.

Chapter 4. Adaptive Quantization as a Device-algorithm Co-design Approach to Improve Performance of In-memory Unsupervised Learning with SNN

4.1 Abstract

Off-chip memory access is the primary bottleneck towards accelerating neural network operations and reducing energy consumption. In-memory training and computation using eNVMs have been proposed to address this problem. However, small number of conductance states limit in-memory online learning performance. Here we introduce a device-algorithm co-design approach and its application to PCM for improving learning accuracy. We present an adaptive quantization method, which compensates the accuracy loss due to limited conductance levels and enables high-accuracy unsupervised learning with low-precision eNVM devices. We develop an SNN framework for NeuroSim platform to compare online learning performance of PCM arrays for analog and digital implementations and benchmark the trade-offs in energy consumption, latency and area.

4.2 Introduction

Neural networks (NNs) have revolutionized artificial intelligence (AI) and led to remarkable advances across diverse applications. However, high level of parallelism required by neural network operations necessitates continuous shuffling of massive amount of NN parameters between memory and processor. This causes substantial computing power and time for conventional von Neumann based computation systems such as CPUs/GPUs [1]. To eliminate delay and power consumption problems due to data-transfer between CPU and memory, in-memory training and computing using emerging nonvolatile memory devices (eNVM) has been identified as a promising non-von Neumann approach [2-4]. Advances in new materials and emerging nonvolatile memory devices offer new approaches to very low. energy computing with scalable devices [5-8]. Mapping NN training to eNVM arrays requires quantization of weight values into discrete conductance levels. Unfortunately, most of the synaptic devices demonstrated so far have limited conductance levels and cannot represent NN weights in ideal high precision (64-bit) as shown in Figure 4.1.

All previous demonstrations of learning with synaptic arrays have adopted uniform quantization, which maps continuous NN weights into discrete device conductance values uniformly, leading to steep decrease in accuracy for precisions less than 6-bits [5-8] for online learning. Various different quantization schemes have been proposed in the literature [9-11]. However, this work mainly focuses on development of adaptive quantization to be applied onto the eNVM devices. To overcome this accuracy degradation, we propose an adaptive quantization technique, which maps NN weights to the hardware conductances based on the distribution and importance of the weights. We apply this device-algorithm co-design approach to phase change memory (PCM) synapses for online unsupervised learning with a spiking neural network (SNN). SNNs allow sparse and event-driven parameter updates for energy efficient implementation of online learning in hardware. Therefore, SNNs have been widely explored for neuromorphic circuits in the past. In addition, SNNs are particularly suitable for unsupervised learning using unlabeled data, offering complementary skills to widely-adopted artificial NNs using supervised learning based on back-propagation.

In this paper, we investigate adaptive quantization methodologies to train SNNs with low bit precision synaptic devices for online learning in hardware. We also study the impact of adaptive quantization on abruptness and asymmetry of device conductance. Then, we develop a SNN framework for NeuroSim integrated device-to-algorithm simulator (SNN+NeuroSim). Using SNN+NeuroSim, we explore system-level performance of implementation of unsupervised learning with PCM arrays for analog and digital architectures in various technology nodes.

4.3 Results

4.3.1 PCM Characterization

In this work, we use $Ge_2Sb_2Te_5$ (GST), a phase change material, to implement PCM-based synaptic devices. 200 nm thick GST is deposited between a bottom electrode (75 nm diameter) and a top electrode. Cross-section TEM images of a PCM synaptic device programmed into low conductance (7 V, 50 ns) and high conductance (1.2 V, 50 ns) states are shown in Figure 4. 2a and b. At high conductance state the GST is poly-crystalline. At low conductance state, an amorphous cap starts to form at the bottom electrode interface, determining the resistance of the PCM device.

We investigate gradual programing in PCM synapses. When identical amplitude pulses (2V, 50ns) are used, PCM synapses exhibit gradual programing only for conductance increase (Figure 4.3a). Figure 4.3b and c show gradual conductance change of our PCM device in both of high and low conductance (G) regime. To achieve both gradual set and reset in our PCM device, we need to apply pulses with increasing amplitude. In high-G regime (Figure 4.3b), gradual set (increasing conductance) programming of the PCM devices is achieved by using staircase pulses (20 pulses per each voltage step of 0.1 V starting from 0.5 V to 0.9 V) and gradual reset (decreasing conductance) is achieved using pulses with increasing amplitude from 2 V to 4 V with 20 mV voltage steps. In low-G regime (Figure 4.3c), gradual set is performed by stair-case pulses with an increasing step of 50 mV in the range of 1 V to 1.7 V (four pulses for each step) and gradual reset is performed by pulses with increasing amplitude from 5.7 V to 7.3 V with 25 mV voltage steps. The current for gradual set ranges from 0.04 mA to 0.25 mA and the current for gradual reset ranges from 2 mA to 2.4 mA in low-G regime. 0.1 V and 40 ns pulse is used to read the device conductance. We plot the callout window in Figure 4.3c to clearly show the abruptness during gradual reset. If we implement large synaptic core array, IR drop across metal lines can affect the accuracy. To avoid the accuracy drop due to the IR drop, ON resistance of memory cell needs to be higher than 10 kΩ for on-line learning case [12]. Since ON state resistance of high-G regime is 5 kΩ, we use low-G PCM data (ON state: 200 kΩ) for in-memory NN training in this work. Our PCM device exhibits ~55 levels for gradual conductance increase and decrease, corresponding to ~6-bit precision.

4.3.2 Neural Network Model

SNNs have been extensively investigated by neuromorphic circuits community since they offer sparse and event-driven parameter updates for energy efficient implementation of online learning in hardware [13]. In this work, we use an SNN model to investigate unsupervised online learning with PCM arrays. Our SNN model for unsupervised learning is summarized in Figure 4.4a and b. For the training, synaptic weights are updated using a timing and weight-based learning rule (Figure 4.4c and d). The iterative training cycle consists of first converting all input digits to Poisson pre-spike trains, computing the membrane potentials for the output layer, generating a post-spike using a probabilistic firing mechanism, and finally updating the synaptic weights using the simplified spike-timing dependent plasticity (STDP) rule shown in Figure 4.4c and d.

According to this rule, if the time difference between the post-spike and pre-spike is within a 10ms window, the synaptic weight is increased by ΔWLTP according to the long-term potentiation (LTP) rule in Eq. (1). Otherwise, the synaptic weight is decreased by ΔWLTD using the long-term depression (LTD) rule in Eq. (2).

$$\Delta W_{LTP} = \alpha \times \exp(-\beta (W + 1)) \qquad (1)$$

$$\Delta W_{LTD} = -\gamma \qquad (2)$$

The parameters α and β control the LTP strength. W is the current weight value. The parameter γ determines the depression scale. The network is trained in an unsupervised fashion with 60,000 MNIST digits. After training is done, we assign labels to the output neurons and perform inference with MNIST test set of 10,000 handwritten digits. The classification accuracy for our SNN is 94.05% for ideal 64-bit floating point. This accuracy is already high for unsupervised learning and can be further increased up to 98.17% if supervision is introduced into the SNN [13].

There are two ways to use our PCM devices for implementing on-line training of our SNN model. First, since our device can only achieve gradual SET using identical pulses (Figure 4.3a), we can use 2-PCM configuration [14] for on-line training. An alternative way is to use the device characteristics shown in Figure 4.3c. Since we use non-identical pulses for gradual switching of the devices, an additional read step is required before updating the weights in hardware as suggested by [15]. Figure 4.5 shows how this test scheme can be applied to on-line training. Before the weight update, we read the device conductance from the PCM array. The peripheral neuron circuit then calculates the weight update (ΔW), which is converted into ΔG to calculate the number of programming pulses (ΔP) and amplitudes based on ΔG and the current conductance state. Finally, the programming circuitry will apply the pulses to update the conductance of the PCM devices in the array.

4.3.3 Adaptive Quantization for Low Precision Synapses

Although low-precision weights can be used for inference, online learning requires high-precision representation of weights to achieve high accuracy [12]. Therefore, mapping network training to eNVM arrays requires quantization of weight values with high precision. Uniform quantization ignores the distribution and evolution of weights during training and treats all the weights with equal importance. However, every weight does not equally contribute to learning outcome and hence, unimportant weights do not require high precision. To address that, we develop adaptive quantization for quantizing weights based on their distribution during training using Lloyd maximum quantization [16].

To train an adaptive quantizer, we use the evolution of weights in the first 5,000 training samples (Figure 4.6a). We investigate Medium-W, Low-W and High-W quantizers (Figure 4.6b), allocating more levels to intermediate, negative and positive weights, respectively. Figure 4.7a and b show weight visualizations of output neurons for ideal 64-bit software simulation and 4-bit Low-W quantize We also explore the performance of different quantizers for training SNN using PCM data (Figure 4.3c) with lower precision. To implement adaptive quantization with the PCM data, we first choose the number of quantized levels to distribute in positive ([0,1]) and negative ([-1,0]) region according to the bit precision and the type of quantizer. Then we use Lloyd-Max quantization [16] to obtain quantization intervals and their corresponding quantization values for the quantizers. Since our neural network weights range from -1 to 1 while the device conductance data ranges from 0.4µS-5.5µS, we use linear transformation to map the quantized weight (W_quan) to the closest PCM conductance values in Eq. (3):

$$G = W_{quan} \frac{(G_{max} - G_{min})}{2} + \frac{(G_{max} + G_{min})}{2} \quad (3)$$

Figure 4.8a shows that quantized weight levels are mapped to PCM conductance values for Low-W quantization as a representative example. PCM gradual programing data from Figure 4.3c is subsampled to 32 levels (5-bit quantization). For Low-W quantization, a larger number of levels were allocated to low conductance values. Similar mappings are performed for uniform, Med-W and High-W quantizers. Figure 4.8b shows theoretically simulated classification accuracies of different quantization techniques without using device data, shown by solid lines. Figure 4.8b also includes adaptive quantization applied directly to PCM data, shown by star symbols. Low-W adaptive quantization boosts classification accuracy by ~60% for 5-bit and ~75% for 4-bit precisions. r respectively, as representative examples.

Table4.1 summaries the performance of 5-bit adaptive quantization against 5-bit uniform quantization and ideal 64-bit software simulation. 5-bit (A.Q.) is the software simulation result based on 5-bit adaptive quantization without using the device data and PCM 5-bit (A.Q.) directly uses subsampled device conductance from Figure 4.8a to perform adaptive quantization. Our results suggest that adaptive quantization can enable use of eNVM devices with limited conductance levels.

To better understand the effect of different adaptive quantizations on the weight development, we plot weight distribution at the beginning (after presenting 100th sample to the network) and the end of the training for no quantization (64-bit) along with all four quantizers (4-bit) in Figure 4.9. To achieve high accuracy, the distribution of the trained weights should well represent the input features of MNIST digits. In MNIST case, the distribution of the trained weights can be divided into two distinct parts, namely the foreground pixels (green, yellow, and red;

positive weights [0,1]) and background pixels (blue; negative weights [-1,0]) as shown in Figure 4.7a. Therefore, both positive and negative weights are important for creating the contrast between foreground and background pixels. As shown in Figure 4.9, no quantization represents both foreground and background pixels very well by distributing the weights in [-1,1]. Among the four quantizers, Low-W and Med-W adaptive quantization have weights distributed in similar range with no quantization case. Moreover, compared to Med-W quantization, Low-W quantization has more positive weights and its maximum weight value is closer to 1. This indicates that Low-W provides better contrast between foreground and background pixels than Med-W. Therefore, Med-W shows slightly lower accuracy than Low-W (Figure 4.8b) while Low-W achieves a more accurate representation of input features than other quantizers and shows the highest accuracy (Figure 4.8b and Table4.1). However, for uniform and High-W quantization, the weights get stuck at the negative range or positive range and do not develop properly during training. Hence, the accuracy with uniform or High-W quantizations is lower than Low-W and Med-W. It is important to note that the choice of different quantizers depends on the network algorithm.

We have shown that adaptive quantization can effectively boost the accuracy for low precision devices. Furthermore, we investigate its effects on abruptness of conductance change and asymmetry of weight update. Abruptness and asymmetry are the two non-ideal effects, which could be impacted by different adaptive quantization schemes. Other non-ideal characteristics such as non-linearity and variation of PCM devices have been extensively studied in the literature [12, 14, 15, 17, 18], previously.

First, we investigate the effectiveness of adaptive quantization on abruptness of conductance change. As shown in Figure 4.8b, PCM uniform quantization (cyan stars) performs slightly worse than theoretically simulated uniform quantization (purple line) because there are no conductance levels to represent weights in LTP part due to the abruptness (Figure 4.3c callout window). However, PCM Low-W quantization (green stars) achieves reasonable accuracy in low bit precision and suffers less from the abruptness. This suggests that the Low-W quantization could help to mitigate the effect of the abruptness in device conductance on accuracy.

In addition to abruptness, symmetry of the weight update is another important consideration of the on-line training [17, 19]. Here, we characterize the symmetry of our device using the Gaussian process regression (GPR) method presented in [17]. We extract the noise-free curve for our PCM data (Figure 4.3c) as shown in Figure 4.10a. We vary the σK value in the range between 0 and 50 (Figure 4.10b) to find the optimum value for GPR fitting (σK = 31.6). Based on the fitting, we then characterize the symmetry factor (SF) of our device. SF of our device is presented along with SF of device from [20, 21] in Figure 4.10c. The device data from [20] shows good switching symmetry according to symmetry requirement specified in [19]. Our device is less symmetric than the device from [20] but more symmetric than the device from [21]. Therefore, we use the most asymmetric device [21] to investigate the impact of adaptive quantization on accuracy. We incorporate this device data (~6-bit) directly into our simulation. In addition, we implement 6-bit Low-W quantization based on this data. Table4.2 shows that the Low-W quantization improves the accuracy to 81.13% while the device data only shows 64.39% accuracy due to asymmetry of the weight update. These results suggest that adaptive quantization could be helpful to improve the accuracy for the devices that exhibit asymmetric weight update.

4.3.4 Circuit-Level Performance Benchmark

In order to investigate performance gains as a result of adaptive quantization, we develop an SNN framework for NeuroSim [12]. NeuroSim is a C++ based simulator with hierarchical organization starting from experimental device data and extending to array architectures with peripheral circuit modules and algorithm level neural network models. SNN+NeuroSim can simulate circuit-level performance metrics (energy, area, latency and leakage power) at run-time of online learning, while providing instruction-accurate classification accuracy for the SNN using experimental PCM data. For implementation of neural network training with PCM arrays, synaptic weights can be represented in either analog formats or binary (digital). For analog implementation, the cells can be arranged into a pseudo-crossbar array and synaptic weights are stored in the form of multi-level conductances (Figure 4.11a). For digital implementation, n binary 1T1R cells are grouped to represent one synaptic weight (Figure 4.11b) and each cell is programmed to high or low conductance states.

We apply adaptive quantization to both analog and digital approaches to reduce bit precision for in-memory online learning. With SNN+NeuroSim, we simulate analog synaptic core (Figure 4.11a) mapping network weights into discrete conductance levels of the PCM device data (Figure 4.3c) and digital synaptic core (Figure 4.11b) using binary states of memory cells. Figure 4. 12 and Figure 4.13 show total energy consumption and chip area for analog and digital architectures as a function of technology node and bit precision, respectively. Note that the technology node used in our simulation refers to the transistors of peripheral circuit.

The analog implementation consumes more energy than the digital (Figure 4.12a and 13a) mainly due to the voltage levels used in the write operation of pseudo-crossbar array (Figure 4.11a) [12]. On the other hand, analog implementation always occupies less chip area than digital implementation because smaller number of devices are used (Figure 4.12b and 13b). To make a fair comparison between analog and digital synaptic core, we plot energy vs. area (under different techonology node) for both cases in Figure 4.14. As can be seen in the Figure , analog occupies less area while consumes more energy than digital. As shown in Figure 4.12a and b, energy and area increase with the technology node since transistors for larger technology require higher Vdd and larger area. Figure 4.13a shows that the energy consumption continues increasing as the bit precision increases, indicating that it is critical to reduce bit precision to significantly improve the energy efficiency. Figure 4.13b shows that the total neurosynaptic core area does not change for the analog implementation with different bit precisions since single devices are used for all cases. On the other hand, use of higher bit precision for digital case increases the chip area. Therefore, adaptive quantization can help to reduce bit precision while substantially decreasing energy consumption, chip area and latency.

Table4.3 summarizes the benchmarking results for online learning with SNN for analog and digital architectures using PCM device data (left two columns) and device-algorithm co-design approach using adaptive quantization (right two columns). The best performance metrics are highlighted in yellow and blue. Our device-algorithm co-design approach applies 4-bit Low-W quantizers, which allocate more levels for negative weights. We use the simulation results for 14nm technology node in this Table. As can be seen in Analog (1st column) and Analog 4-bit (3rd column) cases in Table4.3, adaptive quantization allows the use of 16 conductance levels to reduce energy and latency while achieving better accuracy (86.11%). As shown in Digital (2nd column) and Digital 4-bit (4th column) cases in Table4.3, 4-bit precision enabled by adaptive quantization achieves a ~10-fold decrease in latency (red dash boxes), while also decreasing the energy consumption and chip area, and providing a higher classification accuracy. For both PCM device data and device/algorithm co-design cases, our benchmarking results suggest that analog implementation provides better latency than the digital while digital has lower energy consumption. However, it is important to note that the use of analog or digital implementation to achieve best performance strongly depends on the device characteristics and programming pulse parameters. Adaptive quantization enables both lower energy and shorter latency. Particularly for digital implementation, adaptive quantization provides substantial decrease in latency by enabling 4-bit precision.

4.4 Conclusion

This work demonstrated that accuracy loss due to limited conductance levels can be compensated by adaptive quantization. We also showed that abruptness and asymmetry in device conductance can be mitigated by the adaptive quantization. Benchmarking results with our SNN+NeuroSim platform showed that digital PCM architecture achieves lower energy consumption than the analog one, while the analog PCM is preferred for smaller chip area and lower latency. Our device-algorithm co-design solutions suggested that energy consumption, chip area and latency can be significantly reduced by lowering bit precision with adaptive quantization and engineering the eNVM characteristics.

4.5 Acknowledgments

The authors acknowledge Office of Naval Research (N000141612531), National Science Foundation (ECCS-1752241, ECCS-1734940) and Samsung Electronics for funding.

Chapter 4 is a reprint of Yuhan Shi, Zhisheng Huang, Sangheon Oh, Nathan Kaslan, Jungwoo Song, and Duygu Kuzum. "Adaptive quantization as a device-algorithm co-design approach to improve the performance of in-memory unsupervised learning with SNNs." IEEE Transactions on Electron Devices 66, no. 4 (2019): 1722-1728. The dissertation author was the first author of this paper.

4.6 Figures

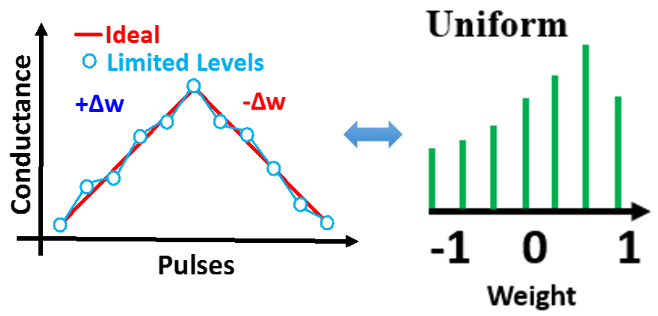


Figure 4. 1 Illustration of Uniform Quantization.

Illustration of limited conductance levels of the device. Uniform quantization maps weights to conductance changes linearly.

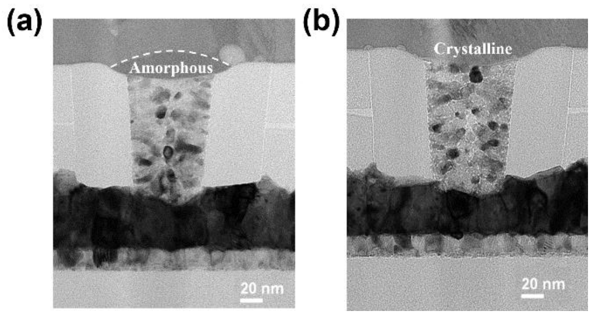


Figure 4. 2 PCB TEM Image in Amorphous and Crystalline States.

A cross-section TEM image of an electronic synapse made of GST. (a) Low conductance amorphous state. (b) High conductance poly-crystalline state.

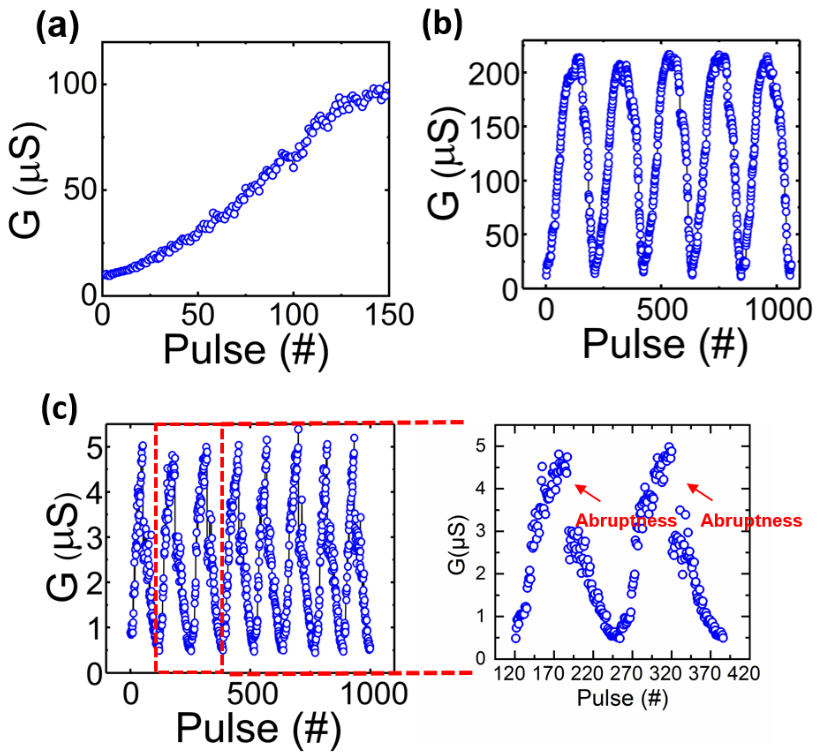


Figure 4. 3 PCM Pulse Characterization.

(a) Measured conductance increase programmed by same amplitude pulses. We use pulses with 50 ns of pulse width, 5 ns of rise time, and 5 ns of fall time. (b) and (c) are the gradual switching characteristics of the device in high-G and low-G regimes, respectively. For both high-G and low-G regimes, we use pulses with 10 ns of pulse width, 5 ns of rise time, and 5 ns of fall time for gradual set and 20 ns of pulse width, 5 ns of rise time, and 5 ns of fall time for gradual reset. The callout window in (c) shows abrupt conductance change during gradual reset.

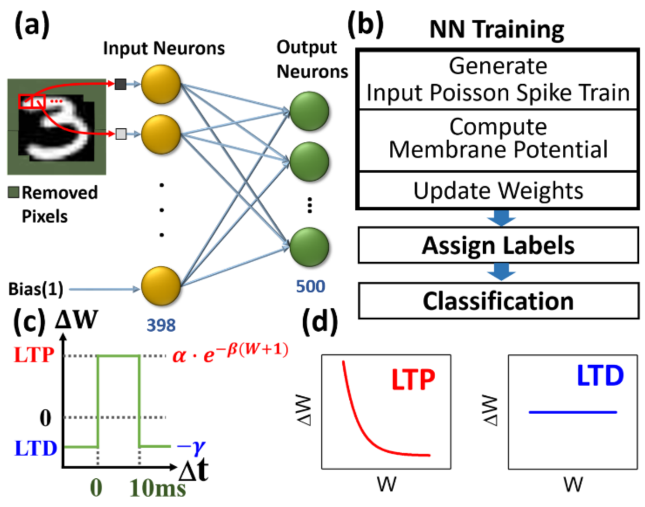


Figure 4. 4 SNN Network Architecture and STDP Rule.

(a) SNN architecture with fully connected structure. Each pixel of a MNIST image is corresponding to one of the input neurons. The number of output layer neurons ranges from 100 to 500. (b) The algorithm used for training of the SNN. (c) A simplified STDP rule used for SNN. (d) The LTP update is an exponential decaying function that depends on the current weight, and the LTD update is a constant.

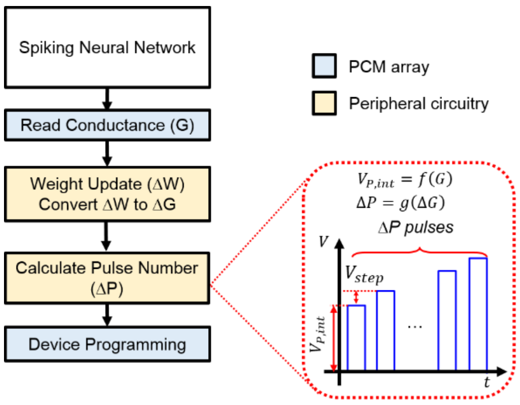


Figure 4. 5 Flowchart of Training Using Non-Identical Pulses.

Schematic of on-line training using non-identical pulse scheme for SNN.

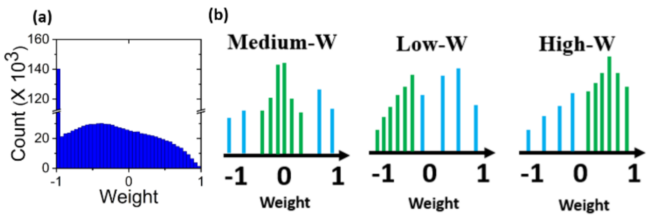


Figure 4. 6 Quantization Types (Medium, Low and High Quantization).

(a) The weight distribution during the training of the first 5,000 samples. (b) Illustrations of Medium-W, Low-W and High-W quantization.

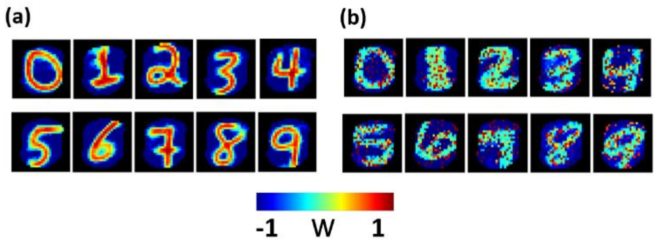


Figure 4. 7 Weight Visualization from Trained Network.

The weight visualizations of ten representative output neurons at the end of training. (a) Ideal software 64-bit. (b) Low-W 4-bit adaptive quantization.

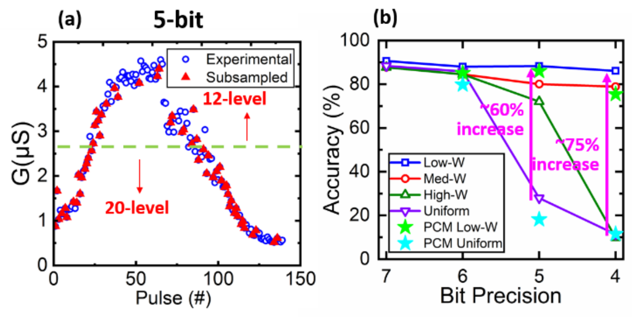


Figure 4. 8 Adaptive Quantization Results with Different Quantization Methods.

(a) PCM gradual programming data (blue circle) is sub-sampled (red triangle) to 32 levels (5-bit) to perform adaptive quantization. (b) Comparison of quantization methods. Low-W achieves ~60% and ~75% increase in 5-bit and 4-bit over uniform. Lines represent accuracies of different quantization techniques without using device data. Stars represent accuracies of different quantization techniques using PCM device data from Figure 4.3c.

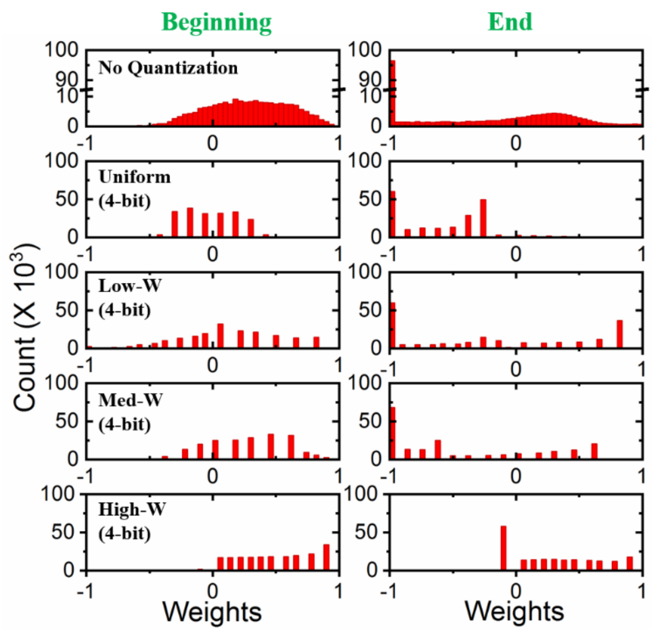


Figure 4. 9 Weight Distribution in Beginning and End of Quantization.

Weights at the beginning and the end of training of no quantization (64-bit), 4-bit uniform, Low-W, Med-W and High-W quantization.

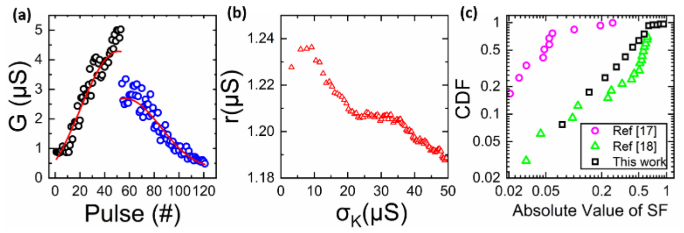


Figure 4. 10 Variations Impact in Adaptive Quantization.

(a) Noise-free curve of our device data (Figure 4.3c) using GPR method. (b) r against varying σK values for our device. r represents the absolute difference between predicted and observed G values [17]. (c) Plot of cumulative distribution function (CDF) of absolute values of SF for our data and device data from [20, 21].

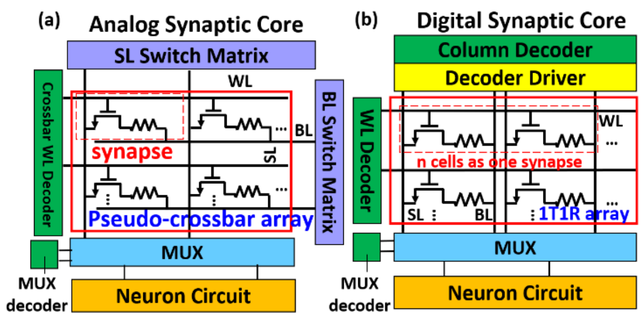


Figure 4. 11 Analog and Digital Synaptic Core Hardware Architectures.

(a) Analog synaptic core with pseudo-crossbar 1T1R array and peripheral circuitry. Each eNVM represents one synapse. (b) Digital synaptic core consists of 1T1R eNVM array with peripheral circuitry. n eNVM cells represent one synapse.

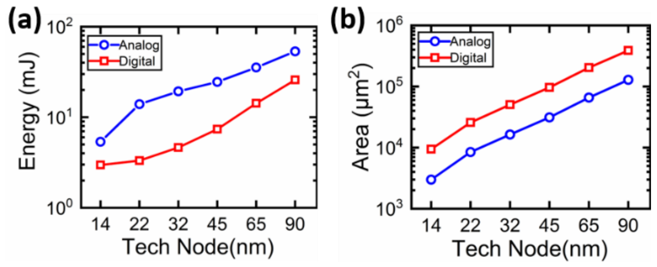


Figure 4. 12 Energy Consumption vs. Technology Node.

(a) Energy consumption and (b) chip area vs. technology node (nm) for analog and digital synaptic cores.

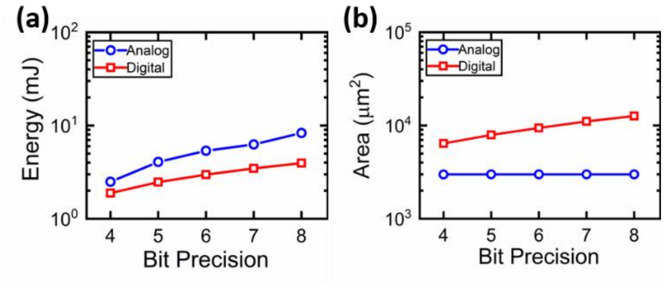


Figure 4. 13 Energy Consumption vs. Bit Precision.

(a) Energy consumption and (b) chip area vs. bit precision for analog and digital synaptic cores.

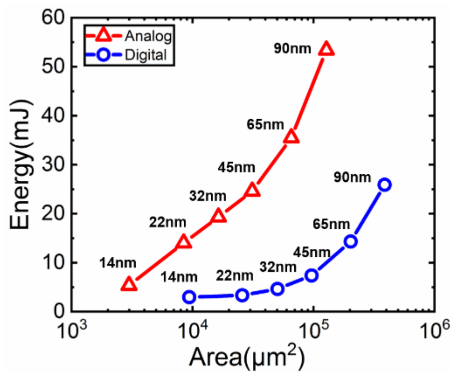


Figure 4. 14 Energy Consumption vs. Area.

Energy vs. Area (under different technology node: 14, 22, 32, 45, 65, 90nm for analog and digital synaptic cores.

Table 4. 1 Classification Accuracy for Different Quantization Schemes and PCM Data.

Precision	**Accuracy**
64-bit	94.05 %
5-bit	27.96 %
5-bit (A.Q.)	88.31 %
PCM 5-bit (A.Q.)	86.05 %

Table 4. 2 Classification Accuracy for Asymmetric Device.

Precision	**Accuracy**
Device [21] (~6-bit)	64.39 %
Device [21] 6-bit (A.Q.)	81.13 %

Table 4. 3 Benchmark Results of PCM Device Data and Algorithm/Device Co-design for Analog and Digital Architectures (14nm)

	PCM Device Data (Fig. 3b)		**Device/Algorithm Co-design**	
	Analog	**Digital**	**Analog 4-bit (A.Q.[#])**	**Digital 4-bit(A.Q.[#])**
Bit precision	50 levels (~6)	6	16 levels (~4)	4
R_{on}	200kΩ	200kΩ	200kΩ	200kΩ
ON/OFF ratio	10	10	10	10
LTP pulse	1-1.7V/10ns	1.2V/50ns	1-1.7V/10ns	1.2V/50ns
LTD pulse	5.7V-7.3V/20ns	7V/50ns	5.7V-7.3V/20ns	7V/50ns
Accuracy*	85.12%	85.87%	86.11%	86.11%
Area(μm^2)	2990	9420.94	2990	6420
Latency* (s)	**3.29**	12.2	**0.22**	1.32
Energy* (mJ)	5.36	**2.97**	2.49	**1.89**
Leakage Power (μW)	53.8	54.1	53.8	49.3

*for 3 training epochs (60k images/epoch) [#] A.Q.: adaptive quantization.

4.7 References

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, Nov. 2017.

[2] S. B. Eryilmaz, D. Kuzum, R. G. Jeyasingh, S. Kim, M. BrightSky, C. Lam, and H.-S. P. Wong, "Experimental demonstration of array-level learning with phase change synaptic devices," in Proc. IEEE Int. Electron Devices Meeting, pp. 25.5. 1-25.5. 4, Dec. 2013.

[3] D. Kuzum, R. G. Jeyasingh, and H.-S. P. Wong, "Energy efficient programming of nanoelectronic synaptic devices for large-scale implementation of associative and temporal sequence learning," in Proc. IEEE Int. Electron Devices Meeting, pp. 30.3. 1-30.3. 4, Dec. 2011.

[4] Y. Jeong and W. Lu, "Neuromorphic Computing Using Memristor Crossbar Networks: A Focus on Bio-Inspired Approaches," IEEE Nanotechnology Magazine, vol. 12, no. 3, pp. 6-18, Jul. 2018.

[5] R. Ge, X. Wu, M. Kim, J. Shi, S. Sonde, L. Tao, Y. Zhang, J. C. Lee, and D. Akinwande, "Atomristor: Nonvolatile Resistance Switching in Atomic Sheets of Transition Metal Dichalcogenides," Nano letters, vol. 18, no. 1, pp. 434-441, Dec. 2017.

[6] M. Kim, R. Ge, X. Wu, X. Lan, J. Tice, J. C. Lee, and D. Akinwande, "Zero-static power radio-frequency switches based on MoS 2 atomristors," Nature communications, vol. 9, no. 1, p. 2524, Jun. 2018.

[7] D.-H. Kang, W.-Y. Choi, H. Woo, S. Jang, H.-Y. Park, J. Shim, J.-W. Choi, S. Kim, S. Jeon, S. Lee, and J.-H. Park, "Poly-4-vinylphenol (PVP) and Poly (melamine-co-formaldehyde)(PMF)-Based Atomic Switching Device and Its Application to Logic Gate Circuits with Low Operating Voltage," ACS applied materials & interfaces, vol. 9, no. 32, pp. 27073-27082, Aug. 2017.

[8] S. Lashkare, N. Panwar, P. Kumbhare, B. Das, and U. Ganguly, "PCMO-based RRAM and NPN bipolar selector as synapse for energy efficient STDP," IEEE Electron Device Letters, vol. 38, no. 9, pp. 1212-1215, Jul. 2017.

[9] C. Louizos, M. Reisser, T. Blankevoort, E. Gavves, and M. Welling, "Relaxed Quantization for Discretized Neural Networks," in 2019 International Conference on Learning Representations (ICLR), May. 2019.

[10] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7197-7205, Jul. 2017.

[11] J. Wang, J. Lin, and Z. Wang, "Efficient Hardware Architectures for Deep Convolutional Neural Network," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 6, pp. 1941-1953, Nov. 2017.

[12] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jan. 2018.

[13] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization," Neural Networks, vol. 103, pp. 118-127, Jul. 2018.

[14] 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, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," IEEE Transactions on Electron Devices, vol. 62, no. 11, pp. 3498-3507, Jul. 2015.

[15] P.-Y. Chen, B. Lin, I. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J.-s. Seo, Y. Cao, and S. Yu, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in Proc. IEEE/ACM Int. Computer-Aided Design, pp. 194-199, Nov. 2015.

[16] S. Lloyd, "Least squares quantization in PCM," IEEE transactions on information theory, vol. 28, no. 2, pp. 129-137, Mar. 1982.

[17] N. Gong, T. Idé, S. Kim, I. Boybat, A. Sebastian, V. Narayanan, and T. Ando, "Signal and noise extraction from analog memory elements for neuromorphic computing," Nature communications, vol. 9, no. 1, p. 2102, May 2018.

[18] S. Oh, Y. Shi, X. Liu, J. Song, and D. Kuzum, "Drift-Enhanced Unsupervised Learning of Handwritten Digits in Spiking Neural Network With PCM Synapses," IEEE Electron Device Letters, vol. 39, no. 11, pp. 1768-1771, Nov. 2018.

[19] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: design considerations," Frontiers in neuroscience, vol. 10, p. 333, Jul. 2016.

[20] J. Woo, K. Moon, J. Song, S. Lee, M. Kwak, J. Park, and H. Hwang, "Improved synaptic behavior under identical pulses using AlO x/HfO 2 bilayer RRAM array for neuromorphic systems," IEEE Electron Device Letters, vol. 37, no. 8, pp. 994-997, Aug. 2016.

[21] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," Nano letters, vol. 10, no. 4, pp. 1297-1301, Mar. 2010.

Chapter 5. A Neuromorphic Brain Interface based on RRAM Crossbar Arrays for High Throughput Real-time Spike Sorting

5.1 Abstract

Real-time spike sorting and processing are crucial for closed-loop brain-machine interfaces and neural prosthetics. Recent developments in high-density multi-electrode arrays with hundreds of electrodes have enabled simultaneous recordings of spikes from a large number of neurons. However, the high channel count imposes stringent demands on real-time spike sorting hardware regarding data transmission bandwidth and computation complexity. Thus, it is necessary to develop a specialized real-time hardware that can sort neural spikes on the fly with high throughputs while consuming minimal power. Here, we present a real-time, low latency spike sorting processor that utilizes high-density CuOx resistive crossbars to implement in-memory spike sorting in a massively parallel manner. We developed a fabrication process which is compatible with CMOS BEOL integration. We extensively characterized switching characteristics and statistical variations of the CuOx memory devices. In order to implement spike sorting with crossbar arrays, we developed a template matching-based spike sorting algorithm that can be directly mapped onto RRAM crossbars. By using synthetic and in vivo recordings of extracellular spikes, we experimentally demonstrated energy efficient spike sorting with high accuracy. Our neuromorphic interface offers substantial improvements in area (~1000× less area), power (~200× less power), and latency (4.8μs latency for sorting 100 channels) for real-time spike sorting compared to other hardware implementations based on FPGAs and microcontrollers.

5.2 Introduction

Extracellular recordings of neuronal spikes using microelectrode arrays have been widely used in studying neural circuits involved in sensory [1], motor [2], and navigation [3] functions in the brain [4]. The recorded signals are a mix of activities from multiple neurons and a crucial processing step, called spike sorting, is required to separate the firing activities and assign the recorded spikes to individual neurons from the recordings. Spike sorting is an indispensable tool in neuroscience for studying neural circuits [5], connectivity, causality and decoding brain activities [6, 7]. It is also fundamental in decoding intentions from neural activity in brain-machine interfaces (BMIs) [8] and neural prosthetics [9]. Conventionally, spike sorting is performed offline by transmitting raw digitized signals recorded by neural electrodes to a nearby computer. However, the off-line processing approach becomes impractical for sorting neural recordings generated from advanced high-density microelectrode arrays (HDMEAs) that comprise hundreds or thousands of recording sites in a single probe, such as recently developed Neuropixels probe [10]. Transmitting vast amounts of neural recording data from HDMEAs to an off-line spike sorter leads to excessive power dissipation which poses a serious risk of damage for the surrounding tissues [11]. For example, a 100-channel microelectrode array with a 16-bit ADC operating at 30kHz sampling frequency generates 3MSamples/s and dissipates mW-level power to nearby tissues. More importantly, to enable the closed-loop BMIs for prosthetics with multiple degrees of freedom, hundreds of neurons distributed in multiple cortical areas need to be monitored in real-time with minimal delay [12]. An 8-hour recording experiment using a 100-channel microelectrode array would accumulate ~200GB of data [13], demanding at least a few hours to sort the recorded spikes off-line [14]. The high latency associated with spike sorting becomes a limiting factor for closed-loop applications requiring rapid feedback. These drawbacks highlight the need for developing compact, low-power and high throughput hardware that can be integrated with high density implantable microelectrode arrays to perform on-chip spike sorting in real-time.

Although there have been sustained efforts to develop real-time spike sorting in FPGAs, most implementations are inefficient in terms of area and power consumption. Want et al., demonstrated a single channel real-time spike sorting while using >90% FPGA resources [15]. Laszlo et al. implemented the “Osort” algorithm in FPGA for sorting 128 recording channels, using hundreds of block RAM and DSP units. However, this approach does not scale well with channel count [16]. On the other hand, resistive switching random access memory (RRAM) has been considered as a promising next-generation memory technology due to its low switching energy, non-volatility, high switching speed and small footprint [17]. In-memory computing based on RRAM arrays has been widely used in accelerating data intensive applications such as neural network inferences, computer vision, and compressed sensing [18]. A crossbar array consisting of thousands of RRAM devices offers large non-volatile memory storage and facilitates massive parallelization of matrix-vector multiplications. These advantages make RRAM crossbars uniquely poised to implement a large number of dot-products in real-time with high energy-efficiencies. However, to the best of our knowledge, no studies have yet shown RRAM-based brain interfaces for real-time spike sorting.

In this paper, we designed a compact, energy-efficient, and high throughput neuromorphic brain interface based on CuOx crossbar arrays that can perform spike sorting for extracellular neural recordings. On the hardware front, we developed a low-temperature fabrication process that is compatible with BEOL CMOS integration to fabricate high-density CuOx crossbars. We developed a template matching-based spike sorting algorithm that is a hardware-friendly and scalable for mapping onto crossbars. In our neuromorphic brain interface, low amplitude neural signals (few µVs) from an implanted neural probe were amplified and digitized using an Intan amplifier. The neural templates were encoded into device conductances and stored in columns of CuOx crossbars (Figure 5.1). Template matching was achieved by feeding neural signals to the wordlines (WLs) and using the crossbar architecture to compute their dot products with corresponding neural templates in each column. The sorting results were obtained parallelly by processing the weighted sum currents in the bitlines (BLs). We experimentally demonstrated the ability of our CuOx crossbar arrays to sort simulated synthetic spikes as well as extracellular recordings from in vivo animal experiments with high accuracy i.e. close to ideal software implementation. Based on experimental results, we also performed a system-level simulation and estimated that our approach can sort 100-channel recordings within 4.8µs with ~1000× reduction in chip area, ~200× reduction in power, and ~50× less energy per channel compared to the state-of-the-art FPGA and microcontroller implementations.

The rest of this paper is organized as follows. Section II presents device characterization results for CuOx devices, including DC switching, transient pulse responses, cycle-to-cycle, and device-to-device variations and retention. Section III describes the template matching algorithm and two datasets used in the hardware demonstration. Section IV explains how the algorithm is mapped to the hardware and the spike sorting in the crossbar. Section V discusses the system-level benchmarking results of our approach in comparison to other hardware implementations. Section VI summarizes this paper.

5.3 Results

5.3.1 CuOx Resistive Crossbars

We developed a wafer-scale process for fabricating 16×16 crossbar arrays of Au/CuOx/Au resistive switching devices (Figure 5.2a). The SEM image of the crossbar array and the cross-section schematic are shown in Figure 5.2b and d. The fabrication flow is illustrated in Figure 5.2c. First, Au with Cr adhesion layer (100 nm) is sputtered and patterned via photolithography and lift-off for bottom electrodes (or WLs) with 1µm linewidth and a 2um pitch. Then, 70 nm of CuOx switching layer is deposited and patterned with reactive sputtering of Cu and Ar/O_2 (95 %/5 %) gas. After that, top electrodes are deposited and patterned following the same fabrication steps as the bottom electrodes. Lastly, 300 nm of SiO_2 layer is deposited and patterned to passivate the device active region to ensure long-term stability. Since all the processes for the CuOx crossbars are low-temperature process, it can be built directly on the BEOL of CMOS circuits.

After fabricating Au/CuOx/Au resistive switching devices, we extensively characterized them (Figure 5.3 and Figure 5.4). The Au/CuOx/Au devices displayed consistent bipolar switching in response to 30 DC voltage sweeps (Figure 5.3a). They could be set to a low resistance state of ~100Ω at VSET = ~1.5V whereas applying VRESET = ~-0.7V increased device resistances to as high as ~1GΩ with low cycle-to-cycle variations (Figure 5.3b). The high ON/OFF ratio (~10^7) of the device resistances (Figure 5.3c) provides a sufficiently large window for implementing the neuromorphic brain interface. Furthermore, the relatively low SET and RESET voltages (Figure 5.3b) is desirable for future integration with peripheral CMOS circuitry.

Low device-to-device variations are important to ensure accurate mapping templates to the crossbar. To quantify this, we randomly selected 120 Au/CuOx/Au devices from different regions of the wafer. Cumulative distribution (CDF) of switching voltage and resistance is shown in Figure 5.4a and b respectively. The measured SET and RESET latencies are presented in [19]. The RESET transition (~80μs) was significantly faster than the SET process, highlighting the scope for further device optimization. Non-volatility of low resistance state (LRS) and high resistance state (HRS) was characterized by reading the device (V_{read} = 0.1V) at regular time intervals immediately after a successful SET or RESET process. The Au/CuOx/Au devices could retain their LRS and HRS for more than >10000 seconds, indicating these devices can faithfully store the neuron templates needed for real-time spike sorting and periodic refresh operations could be utilized if experiments taking longer than this time period (Figure 5.4c).

5.3.2 Template Matching Algorithm

A. Algorithm Overview

Spike sorting is a challenging clustering problem and many algorithms have been developed over the past years such as principal component analysis [20], template matching [21], Bayesian statistical frameworks [22], and hidden Markov models [23]. Among these, template matching is the most efficient approach to sort neural spikes [24]. It assumes a pre-existing database of neuron templates; the goal is to assign the best-fit templates to the detected spike waveform, hence clustering the spikes to specific neuron units. Motivated by this, we developed a template matching algorithm that can be directly mapped to the crossbars to achieve real-time spike sorting.

Figure 5.5 outlines the algorithm (Step1-Step4) by showing a simplified example for classifying two neurons (n=2) from three-channel recordings (m=3). The same methodology can be used to classify a larger number of neurons recorded across hundreds of channels. Each neuron had a template matrix $T_n = [T_{n,1}, T_{n,2}, \ldots, T_{n,m}]$, where column $T_{i,j}$ represented the template for neuron i corresponding to channel j (Figure 5.5a). The $T_{i,j}$ is a vector with S samples with $S = f_s \times k$, where f_s is the sampling frequency and k is the user-define window that determines the duration of templates. In this example, f_s = 30kHz and k = 3ms. T_n is built by horizontally concatenating these templates across m electrodes (m=3). The template matrix T_n was normalized by its Frobenius Norm ($T_n/\|T_n\|_F$) to maintain the amplitude of the spikes in the same range (Figure 5.5a). Similarly, we defined the neural signal $V(t) = [V_1(t), V_2(t)\ldots, V_m(t)]$, where $V_j(t)$ is the recorded signal from channel j. Figure 5.5b shows an example of recording in three channels at 30kHz. To perform the template matching, we first computed the waveform similarity $C_{n,m}(t)$, which is the convolution between signals from channel m and the template of neuron n on channel m measured at time t. The convolution can be expressed as $C_{n,m}(t)=V_m(t)*T_{n,m}$, which is simply a sliding dot product between the signal and template. Then, the resulting waveform similarities from all m channels were summed up for each neuron (Figure 5.5c) to give $C_n(t) = \sum_1^m C_{n,m}(t)$, the overall activation of neuron n at time t. In the final step (Figure 5.5d), we applied a threshold, which is ~3 standard deviation of the $C_n(t)$ to identify the spike times. After that, we assigned the spikes to the neuron having the largest $C_n(t)$. Note that these templates are typically obtained offline through a semi-automatic algorithm with human curation to ensure accuracy. The details of mapping templates to the hardware are discussed in Section IV.

B. Datasets

We implemented the aforementioned template matching algorithm on two neural recordings: (1) a synthetic “NeuroNexus-32” data [25] and (2) “real” spikes from *in vivo* animal experiments recorded with the NeuroFITM probe [6] for validating our spiking sorting hardware with different neural electrode technologies. In the “NeuroNexus-32” dataset, the extracellular spiking activities with ground truth were generated using MEArec [25]. MEArec generated data in two phases. In the template generation phase, biophysically realistic neuron models were positioned at different locations of the NeuroNexus-32 probe model to produce extracellular potentials to form a template library. In the recording generation phase, it convolved the templates selected from the library with randomly generated spike trains. Additive Gaussian noise was added to the convolution results to obtain the final recording data. Typically, a channel can record activities of ~1-3 neurons nearby. Our synthetic dataset contains extracellular recordings of twelve neurons from 32 channels sampled at 30kHz [19]. The "real" dataset contains 1-hour recordings sampled at 32kHz from an *in-vivo* animal experiment recorded with the 32-channel NeuroFITM probe (Figure 5.6a) [6], where spike sorting results from offline Kilosort algorithm [14] was considered as the ground truth. Figure 5.6b and c show representative neuron templates and the recordings in Ch4. Top of Figure 5.6c shows the predicted spike train as square symbols and the clustered neuron spike waveforms are presented in Figure 5.6d. As can be seen, for each neuron, the shape of the clustered spike waveforms closely matched their respective templates. A similar waveform example of NeuroNeuxus-32 and the complete template libraries of both probes can be found in our previous work [19].

C. Sorting Performance

The sorting outcome of our algorithm is determined against the ground truth spikes by comparing the spike time. To quantify the sorting performance, we employed the commonly-used F1 score (in %) given by 2TP/ (2TP+FP+FN), where TP, FP, and FN denote the true positive, false positive, and false-negative outcomes. A TP is defined as a spike that has been classified correctly by the algorithm. An FP is defined as a spike that is classified as spiking activity but does not exist in ground truth data. An FN is defined as a spike that exists in the ground truth data but is not detected by our algorithm. The spike predictions from our algorithm agree with the ground truth well. Eleven out of twelve neurons in the NeuroNexus-32 dataset have F1 score > 90% (Figure 5.7a), whereas all the two neurons in the NeuroFITM "real" dataset have F1 score > 85% (Figure 5.7b). The F1 score of the "real" dataset is slightly less than the synthetic dataset due to higher noise and probe drifting [26] during the recording, making the classification more difficult. To map the templates to the hardware, we investigated how quantization impacts the F1 score.

The template was quantized to 2N discrete levels between the min and max amplitude range of the normalized template library. After quantization, we followed the same sorting pipeline to obtain the F1 score. Figure 5.7c shows that the performance could be retained if the templates are quantized to at least 4-bit resolution for NeuroNeuxus-32 dataset, which is also applied for NeuroFITM dataset.

5.3.3 Hardware Implementation of Spike Sorting

A. Hardware Mapping

To process hundreds of spikes per second, it would be necessary to adopt a multi-core architecture (Figure 5.8a) where each core consists of a crossbar that stores the templates for a specific set of neurons (Figure 5.8b). Figure 5.8c illustrates how a set of templates could be mapped on to a crossbar core. In the illustration, we assume that three channels (m=3) record spike activities of two neurons (n=2), resulting in a total of 6 templates. The templates from the same channel are mapped to the adjacent columns in the crossbar. The devices in the crossbar can store the templates using multi-level for analog implementation or binary (HRS or LRS) conductance states for digital implementation [27]. A column of devices with 16 (4-bit) multi-level can be used to map a template directly as shown in this example (i.e. templates of N1-Ch1 and N2-Ch2 are mapped to the first two columns of the crossbar respectively). Similarly, templates from other channels are mapped to the rest of the columns to achieve the maximum usage of the array (Figure 5.8c). If binary conductance state is used, four columns are required to map a template from MSB to LSB. Although device with multi-level states can achieve maximum area efficiency, it has been shown that these multi-level states may exhibit high device to device variations, non-linearity and resistance drift due to unstable filament formation [18]. In contrast, digital implementation is more robust against of variations [28], which makes it a better approach to realize high sorting accuracy for template matching task. In addition to conductance states, differential pair scheme is commonly used to represent both negative and positive values of the templates [29].

After all templates are mapped on a core, the voltage spike inputs on WLs (V_{WLi}) are convolved with the templates stored as cross point conductances (G_{ij}). The columns of the crossbar can perform template matching (BL currents $I_{BLj}=\sum G_{ij}V_{WLi}$) in parallel. Since a set of templates from each channel need to convolve with neural signals from the corresponding channel, recordings from Ch1-Ch3 are processed in a time-multiplexed manner, the matching results ($I_{BLn,j}$) for each channel are collected from the corresponding BLs in parallel (n: neuron; j: channel number). The final classification result is obtained by adding the BL currents for each neuron i.e., $\mathrm{I}_n = \sum_1^m I_{BLn,j}$ from all m channels and then assigning the spike to the neuron with the maximum I_n. For the sake of illustration, we show all templates mapped to a single crossbar. For practical applications involving large channel counts, a multicore architecture can be adopted, where each core is dedicated to a channel and stores all templates belonging to the assigned channel. As a result, all channels can be processed at the same time to achieve higher parallelism.

B. Hardware Demonstration

A custom PCB board was used to access the WLs and BLs of the wire-bonded CuOx crossbar (Figure 5.9a and b). Before mapping the templates, array read was performed to confirm the initial states of the crossbar. To read a single device, the selected WL was biased to V_{read} = 0.25V while all other lines were grounded. The as-fabricated devices had initial resistances greater than 500kΩ (Figure 5.9c). As explained in Subsection A. Hardware Mapping, digital implementation was adopted in our demonstration. Neuron templates were quantized, binarized, and mapped onto crossbar columns using differential pair scheme. To program the devices to different states, we used $V_{dd}/2$ write scheme, where the selected WL and BL were biased to $V_{dd}/2$ and $-V_{dd}/2$, and all other unselected lines were grounded to prevent sneak paths (SET: V_{dd} = 4V and RESET: V_{dd} = 3V).

Figure 5.9d-f shows four representative templates (F1-F4) of Neuro-FITM implemented in the crossbar. The templates were quantized to 4-bit and then binarized to two levels (“0”-black or

“1”-white) off-line (Figure 5.9d). “0” was mapped to HRS and “1” was mapped to LRS of the device respectively (Figure 5.9e). Since the crossbar was initially off, only “1” needs to be programmed accordingly. The patterns of the hardware templates match well with software templates, indicating precise write operation. To validate the accuracy of crossbar convolutions, we biased all WLs to high (V_{WLs}=0.25V) and measured the BL currents. As shown in Figure 5.9f, the weighted-sum BL currents (I_{sum}) increased proportionately with the number of LRS devices in the columns. Templates from NeuroNexus-32 dataset are mapped in the same way [19].

Using the programmed templates, we performed spike sorting on NeuroNexus-32 and NeuroFITM recordings. Neural data encoded as 8-bit voltage pulse trains were fed into the WLs and I_{sum} were measured on the BLs. Figure 5.10a and Figure 5.11a show the NeuroNexus-32 and NeuroFITM recordings and the hardware spike sorting results implemented to sort representative three neurons (N1-N3) from the NeuroNexus-32 data and two neurons (N1,N2) from the NeuroFITM data. The neural voltage traces from the recording channels (Ch1-3 in NeuroNexus-32 and Ch1-4 in NeuroFITM) are shown at the bottom. Hardware convolution trace generated by CuOx crossbar represents final current $\mathrm{I}_n = \sum_1^m I_{BLn,j}$ by adding weighted sum currents measured in each $I_{BLn,j}$ for “m” channels and “n” neurons (NeuroNexus-32: m = 3, n =3; NeuroFITM: m = 4, n = 2). The raster plots on the top of Figure 5.10a and Figure 5.11a show the spike train predicted in hardware compared with the ground truth spikes for Neuronexus-32 and NeuroFITM dataset, respectively.

Figure 5.10b and Figure 5.11b show the callouts for the spikes highlighted in rectangular boxes (Figure 5.10a and Figure 5.11a). Inside the boxes, the snippet spike waveform of each neuron is shown in the left. Channels are coded in different colors that match with the signal traces above. The template matching results in software and hardware are shown as convolution traces in the middle (SW) and right (HW) respectively. Different colors represent N1-N3 of NeuroNexus-32 and N1-N2 of NeuroFITM. The software convolution traces are shown as arbitrary units while hardware traces are shown as measured weighted sum currents. For each spike, the neuron with the highest peak in the convolution trace was assigned to the spike. The shapes of convolution traces produced by the CuOx crossbars matched closely with software, thereby confirming our hardware can reliably sort neural spikes. Note that the off-peak regions of the hardware convolution traces are slightly noisy compared with software mainly due to variations in the programmed device conductances across crossbar columns. This issue can be alleviated by adopting a more robust “program and verify” scheme in storing the templates in the crossbar [30].

C. System-level Performance Benchmarking

Based on the hardware spike sorting results obtained over a 100ms time window (Figure 5.10 and Figure 5.11), we evaluated F1 scores on the entire 30s-wide recordings in both neural data and compared them with software predictions. The hardware F1 scores were calculated by performing template matching between neural signals with hardware templates that contain measured device resistances. To evaluate sorting performance across multiple neurons, we averaged F1 score based on neuron number. Table5.1 shows neurons could be sorted with high mean accuracy (~92.5% for NeuroNexus-32, ~94.6% for NeuroFITM).

To project the sorting performance of multi-core architecture (Figure 5.8a) with our crossbar-based spike sorting hardware, we performed a system-level benchmarking to estimate area, power, and latency and compared it with the state-of-the-art FPGA and microcontroller implementations. All implementations included in Table5.2 use *in-vivo* experimental datasets and template matching based approach for a fair comparison. Our work and microcontroller implementation [31] demonstrated sorting for 32-channel probe while FPGA [15] implemented sorting for a single channel. The area per channel was estimated by the number of columns used in mapping a neuron template of a channel (i.e. ~8 columns are used for a channel template and it occupies 40 µm × 20 µm = 8×10^{-4} mm^2). Power per channel was calculated by averaging power consumption $P_{avg} = \sum_1^N I_{sum} \times V_{read}$ across a representative spike waveform snippet during template matching. Here, N is number of samples in the spike waveform (N=30), I_{sum} is the weighted sum current for processing a sample measured crossbar.

Overall, our crossbar-based spike sorting hardware promises ~1000× smaller (area/channel) [15] and ~200× reduction in power consumption [31] compared to state-of-the-art spike sorting hardware implementations that rely on FPGAs (Table5.2). To better understand the sorting latency in the multicore architecture, we assume one crossbar core can have size up to 256×256 and 10ns read latency. Unlike previous works that rely on sequential processing, each crossbar core in the multi-core architecture can process multiple recording channels in a highly parallelized manner. We estimated twelve CuOx crossbar (256×256) cores can sort 100 channel recordings within 4.8µs using the same mapping scheme of our hardware demonstration. As a result, it consumes ~30-50× less energy (energy = power × latency) [15, 31]. These performance gains make real-time spike sorting possible using our crossbars for high throughput BMI applications.

5.4 Conclusion

We presented a high throughput neuromorphic brain interface for real-time spike sorting based on resistive crossbar arrays. We fabricated CuOx crossbars using a simple low-temperature process enabling easy 3D BEOL integration with underlying CMOS circuits. In order to realize real time spike sorting, we developed a hardware compatible template matching algorithm and developed methods for mapping onto crossbar arrays. We demonstrated that hardware implementation of template matching using CuOx crossbars can accurately classify spikes from individual neurons recorded *in vivo*. Our neuromorphic approach offers substantial performance gains in area, power, latency, and energy for spike sorting hardware designed for processing recordings from neural probes with high channel counts. Our work paves the way towards in-memory computing-based real-time spike sorting and processing hardware for next-generation closed-loop brain interfaces.

5.5 Acknowledgments

This paper was submitted on Nov 2021 for review. This work was supported by Office of Naval Research (N000142012405), the National Science Foundation (ECCS-1752241, ECCS-2024776), and the National Institutes of Health (DP2 EB030992). The fabrication of the devices was performed at the San Diego Nanotechnology Infrastructure (SDNI) of the University of California San Diego, supported by the National Science Foundation (ECCS-1542148).

Chapter 5 is a reprint of Yuhan Shi, Akshay Ananthakrishnan, Sangheon Oh, Xin Liu, Gopabandhu Hota, Gert Cauwenberghs, and Duygu Kuzum. "A Neuromorphic Brain Interface Based on RRAM Crossbar Arrays for High Throughput Real-Time Spike Sorting." IEEE Transactions on Electron Devices 69, no. 4 (2021): 2137-2144. The dissertation author was the first author of this paper. Chapter 5, in part, is also a reprint of Yuhan Shi, Akshay Ananthakrishnan, Sangheon Oh, Xin Liu, Gopabandhu Hota, Gert Cauwenberghs, and Duygu Kuzum. "High Throughput Neuromorphic Brain Interface with CuOx Resistive Crossbars for Real-time Spike Sorting." In 2021 IEEE International Electron Devices Meeting (IEDM), pp. 16-5. IEEE, 2021. The dissertation author was the first author of this paper.

5.6 Figures

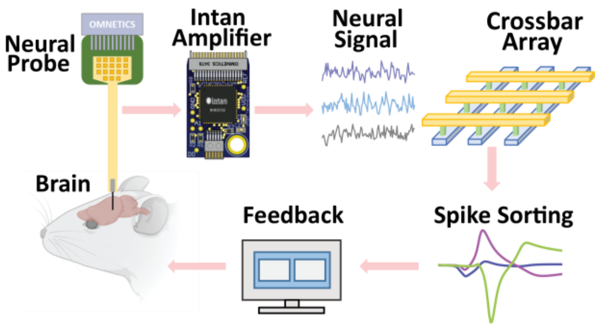


Figure 5. 1 The Schematic of Proposed Neuromorphic Brain Interface.

Proposed neuromorphic brain interface based on CuOx crossbar array for spike sorting. Neural signals recorded by the multichannel neural probe are amplified and digitized using an Intan amplifier and ADC respectively. CuOx crossbar array performs spike sorting in real-time. That can be used as real-time feedback for a closed-loop neural interface.

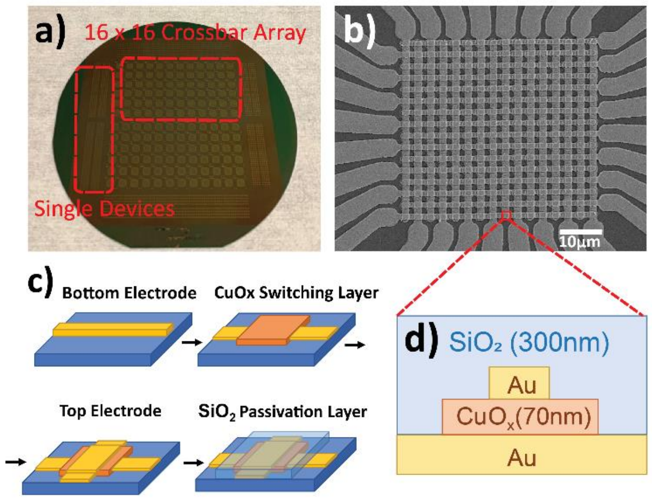


Figure 5. 2 CuOx Device and Crossbar Stack and SEM Image.

(a) Image of a wafer including fabricated 16×16 CuOx crossbar arrays and single devices for testing. (b) SEM images of 16×16 crossbar with 4µm2 cross point. Scale bar: 10µm. (c) Fabrication process for CuOx -based single devices and 16×16 crossbar. (d) Device cross-section (callout window) highlighting the 70nm CuOx resistive switching layer sandwiched between 100nm Au electrodes. A 300nm SiO2 passivation layer is deposited on top of the stack.

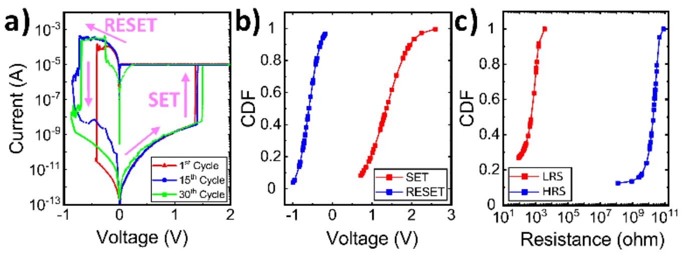


Figure 5. 3 CuOx Device DC Switching Characteristics, Voltage and Resistance Variation.

(a)DC switching characteristics of single devices for 30 cycles. (b) Cumulative distribution function (CDF) of SET (1V to 2.5V) and RESET (-1V to -0.2V) voltages. (c) CDF of high resistance state (100MΩ to 100GΩ) and low resistance state (100Ω - 1kΩ) resistances.

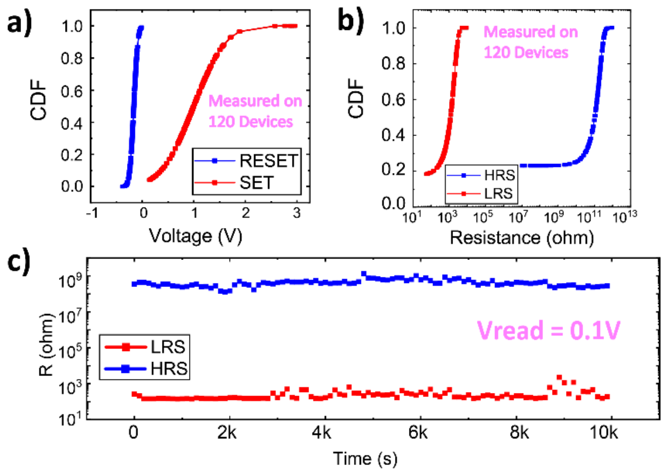


Figure 5. 4 CuOx Device Device-to-Device Variation and Retention.

CDF of the (a) switching voltages and (b) HRS/LRS resistances measured across 120 devices randomly selected on the wafer. (c) Retention characteristics. Device resistance was monitored intermittently using 0.1V read pulses.

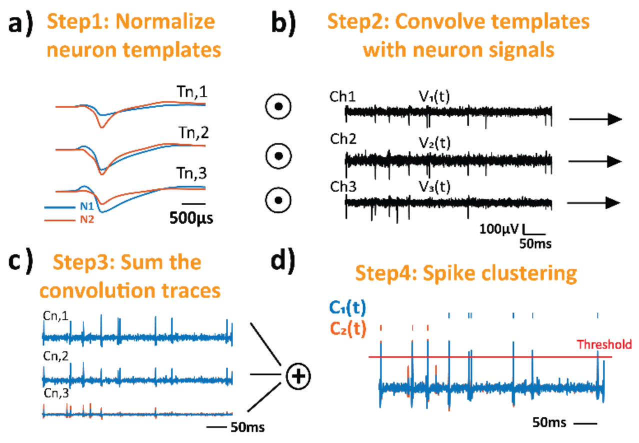

a) Step1: Normalize neuron templates
Tn,1
Tn,2
Tn,3
N1
N2
500μs
b) Step2: Convolve templates with neuron signals
Ch1 V₁(t)
Ch2 V₂(t)
Ch3 V₃(t)
100μV
50ms
c) Step3: Sum the convolution traces
Cn,1
Cn,2
Cn,3
50ms
d) Step4: Spike clustering
C₁(t)
C₂(t)
Threshold
50ms

Figure 5. 5 Spiking Sorting Algorithm.

(a) Normalized templates of N1 and N2. (b) Neural recordings of three channels. (c) Computing the overall activation of neuron n neural recordings i.e., voltage traces with normalized templates N1 and N2. Summing the convolution traces (Cn,m(t)) corresponding to each neuron. d) Thresholding and assigning spikes to neurons N1 or N2 based on whether C1(t) > C2(t) (assign to N1) or C1(t) < C2(t) (assign to N2).

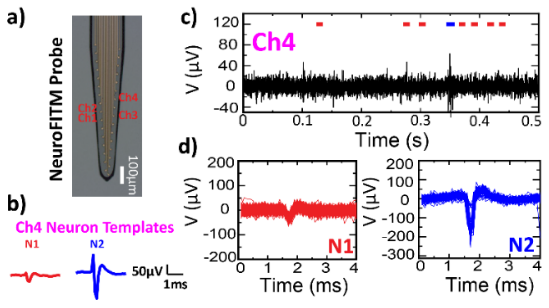


Figure 5. 6 Spiking Sorting Results for NeuroFITM Recording Probe.

(a) Image of a 32-channel NeuroFITM probe with four representative channels highlighted as red. (b) Representative templates for the two neurons in Ch4. (c) Example 500ms-recordings from Ch4 with predicted spike train marked in colored squares. (d) Clustered spikes for N1 and N2 for Ch4.

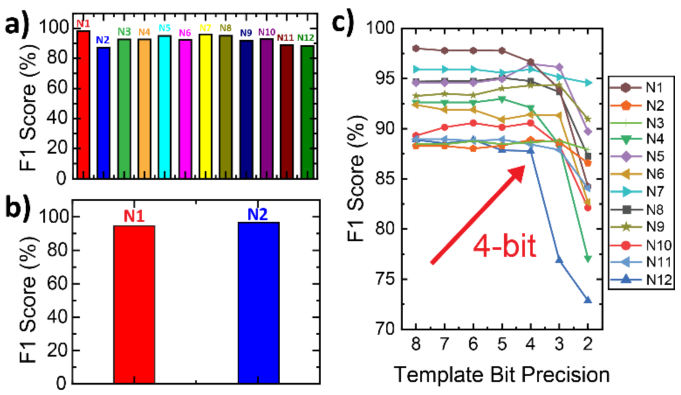


Figure 5. 7 Spiking Sorting F1 Score vs. Template Bit Precision in NeuroNexus-32 and NeuroFITM datasets.

F1 scores (%) for (a) NeuroNexus-32 and (b) NeuroFITM dataset. (c) F1 score (%) as a function of template precision for 12 neurons in NeuroNexus-32 dataset. 4-bit quantized templates are used in hardware experiments.

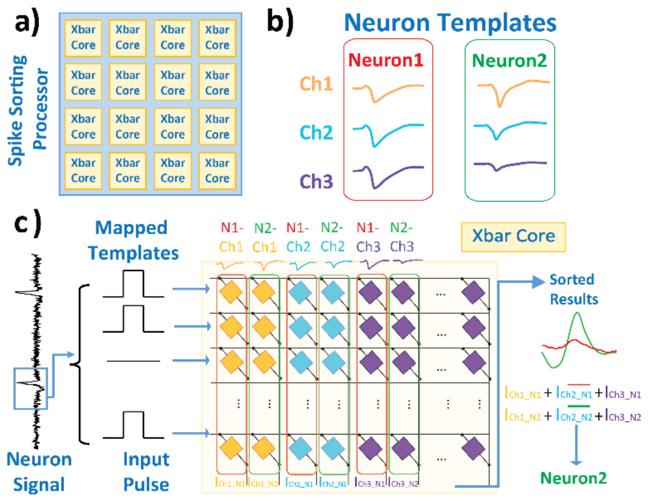


Figure 5. 8 Spiking Sorting Algorithm Mapping to Crossbar Array.

(a) Real-time spike sorting processor with multiple crossbar cores. (b) Representative templates of two neurons with three channels. (c) Crossbar spike sorting: each crossbar column stores a neuron template. 8-bit digitized neural signals are provided as voltage inputs and weighted-sum currents from convolutions are obtained on the BLs. Neuron-wise aggregation of channel currents determines the sorting result.

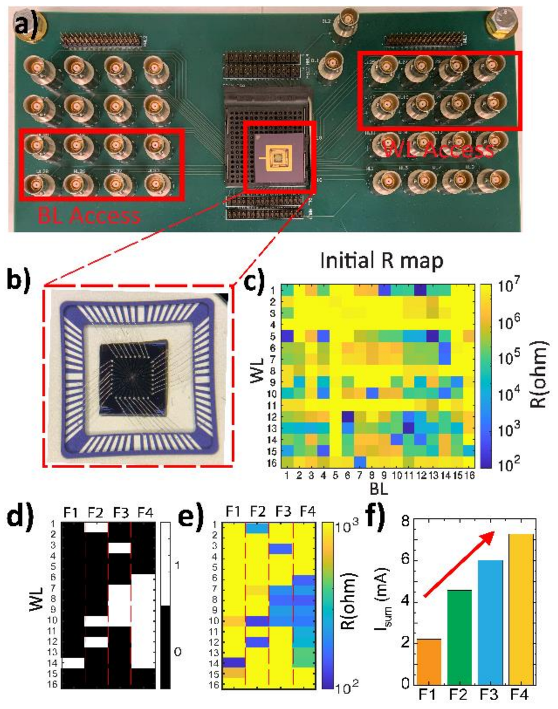


Figure 5. 9 Custom PCB and Weighted Sum Current Verifications from Crossbar.

(a) Custom PCB board to access individual WLs and BLs of the CuOx crossbar for the write and read operations. BLs can be accessed through the connectors shown in the lower left while WLs can be accessed through the connectors in the top right. (b)16×16 crossbar wire-bonded onto a PGA package. (c) Initial resistance map of a 16×16 CuOx crossbar. (d) Four representative binarized (black=0 and white=1) filters (F1-F4) from NeuroFITM. (e) Programmed crossbar columns implementing these filters. (f) I_{sum} measured at V_{WLs}=0.25V for four filters.

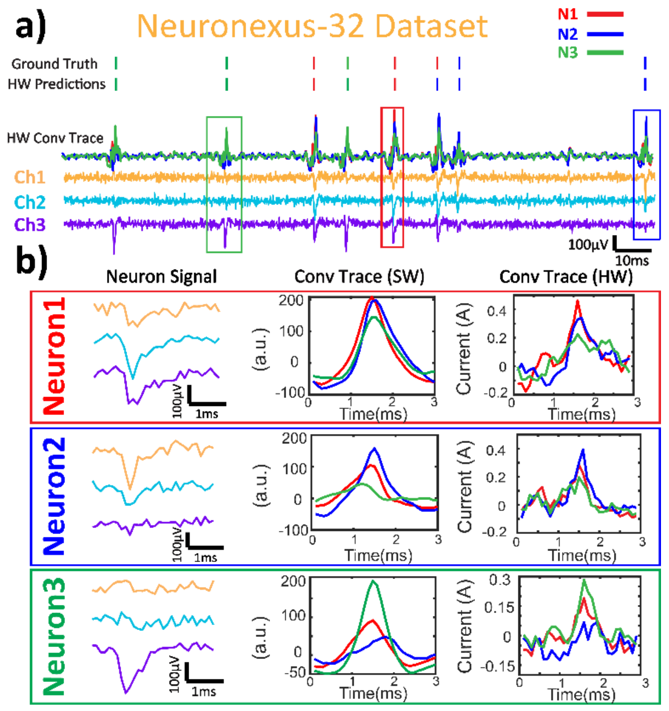


Figure 5. 10 Experimental Results of Spiking Sorting Performed in Hardware (Neuronexus-32 Dataset).

(a) NeuroNexus-32: Ch1,2,3 are used to classify neurons N1, N2, N3. A segment of recordings from Ch1 to Ch3 and predicted hardware (HW) convolution (Conv) traces for three neurons. (b) Representative spike sorting results for N1-N3 showing convolution implemented in HW agrees with the software (SW) implementation.

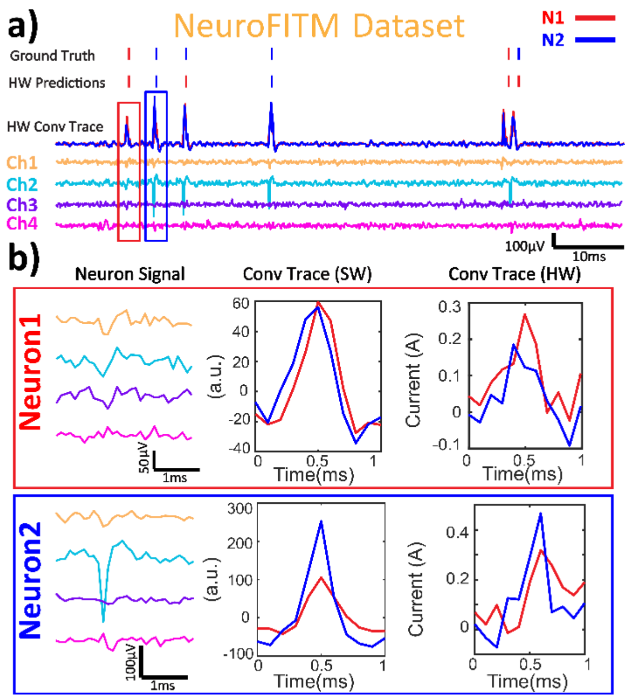


Figure 5. 11 Experimental Results of Spiking Sorting Performed in Hardware (NeuroFITM Dataset).

(a) NeuroFITM: Ch1,2,3,4 are used to classify neurons N1, N2. Segments of recordings from Ch1 to Ch4 and predicted HW conv traces. (b) Representative spike sorting results for N1, N2 implemented in HW agrees with the SW implementation.

Table 5. 1 F1 Score in Software and Hardware Implementations.

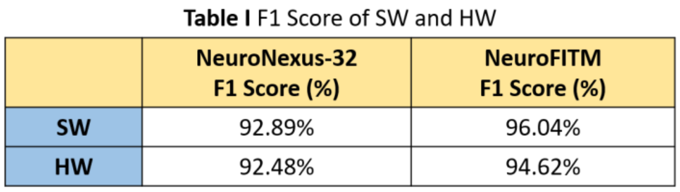


Table I F1 Score of SW and HW

	NeuroNexus-32 F1 Score (%)	NeuroFITM F1 Score (%)
SW	92.89%	96.04%
HW	92.48%	94.62%

Table 5. 2 Benchmarking Results of Spiking Sorting in Hardware

Benchmarking our results against previous works [15, 31] in terms of hardware type, recording data used in the studies, channel count, area/channel, power/channel, sorting latency, and energy/channel. The accuracy obtained on NeuroNexus-32 and NeuroFITM data from software (SW) and hardware (HW) experiments.

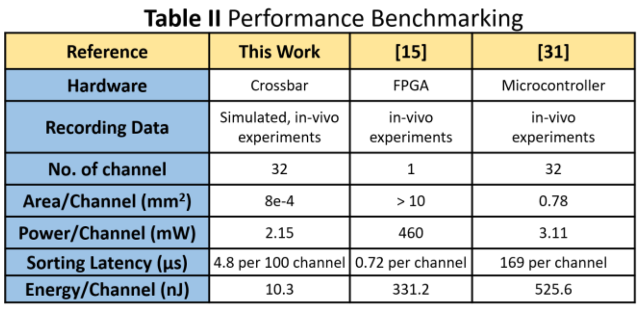


Table II Performance Benchmarking

Reference	This Work	[15]	[31]
Hardware	Crossbar	FPGA	Microcontroller
Recording Data	Simulated, in-vivo experiments	in-vivo experiments	in-vivo experiments
No. of channel	32	1	32
Area/Channel (mm^2)	8e-4	> 10	0.78
Power/Channel (mW)	2.15	460	3.11
Sorting Latency (µs)	4.8 per 100 channel	0.72 per channel	169 per channel
Energy/Channel (nJ)	10.3	331.2	525.6

5.7 References

[1] M. A. Nicolelis, A. A. Ghazanfar, C. R. Stambaugh, L. M. Oliveira, M. Laubach, J. K. Chapin, R. J. Nelson, and J. H. Kaas, "Simultaneous encoding of tactile information by three primate cortical areas," Nature neuroscience, vol. 1, no. 7, pp. 621-630, 1998.

[2] A. P. Georgopoulos, A. B. Schwartz, and R. E. Kettner, "Neuronal population coding of movement direction," Science, vol. 233, no. 4771, pp. 1416-1419, 1986.

[3] E. I. Moser, E. Kropff, and M.-B. Moser, "Place cells, grid cells, and the brain's spatial representation system," Annu. Rev. Neurosci., vol. 31, pp. 69-89, 2008.

[4] R. Q. Quiroga and S. Panzeri, "Extracting information from neuronal populations: information theory and decoding approaches," Nature Reviews Neuroscience, vol. 10, no. 3, pp. 173-185, 2009.

[5] L. Luo, E. M. Callaway, and K. Svoboda, "Genetic dissection of neural circuits: a decade of progress," Neuron, vol. 98, no. 2, pp. 256-281, 2018.

[6] X. Liu, C. Ren, Y. Lu, Y. Liu, J.-H. Kim, S. Leutgeb, T. Komiyama, and D. Kuzum, "Multimodal neural recordings with Neuro-FITM uncover diverse patterns of cortical–hippocampal interactions," Nature Neuroscience, vol. 24, no. 6, pp. 886-896, 2021.

[7] X. Liu, C. Ren, Z. Huang, M. Wilson, J.-H. Kim, Y. Lu, M. Ramezani, T. Komiyama, and D. Kuzum, "Decoding of cortex-wide brain activity from local recordings of neural potentials," Journal of Neural Engineering, 2021.

[8] U. Chaudhary, N. Birbaumer, and A. Ramos-Murguialday, "Brain–computer interfaces for communication and rehabilitation," Nature Reviews Neurology, vol. 12, no. 9, pp. 513-525, 2016.

[9] J. L. Collinger, B. Wodlinger, J. E. Downey, W. Wang, E. C. Tyler-Kabara, D. J. Weber, A. J. McMorland, M. Velliste, M. L. Boninger, and A. B. Schwartz, "High-performance neuroprosthetic control by an individual with tetraplegia," The Lancet, vol. 381, no. 9866, pp. 557-564, 2013.

[10] N. A. Steinmetz, C. Aydin, A. Lebedeva, M. Okun, M. Pachitariu, M. Bauza, M. Beau, J. Bhagat, C. Böhm, and M. Broux, "Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings," Science, vol. 372, no. 6539, 2021.

[11] S. Kim, P. Tathireddy, R. A. Normann, and F. Solzbacher, "Thermal impact of an active 3-D microelectrode array implanted in the brain," IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 15, no. 4, pp. 493-501, 2007.

[12] M. A. Nicolelis and M. A. Lebedev, "Principles of neural ensemble physiology underlying the operation of brain–machine interfaces," Nature reviews neuroscience, vol. 10, no. 7, pp. 530-540, 2009.

[13] S. Gibson, J. W. Judy, and D. Marković, "An FPGA-based platform for accelerated offline spike sorting," Journal of neuroscience methods, vol. 215, no. 1, pp. 1-11, 2013.

[14] M. Pachitariu, N. A. Steinmetz, S. N. Kadir, M. Carandini, and K. D. Harris, "Fast and accurate spike sorting of high-channel count probes with KiloSort," Advances in neural information processing systems, vol. 29, pp. 4448-4456, 2016.

[15] P. K. Wang, S. H. Pun, C. H. Chen, E. A. McCullagh, A. Klug, A. Li, M. I. Vai, P. U. Mak, and T. C. Lei, "Low-latency single channel real-time neural spike sorting system based on template matching," PloS one, vol. 14, no. 11, p. e0225138, 2019.

[16] L. Schäffer, Z. Nagy, Z. Kincses, R. Fiáth, and I. Ulbert, "Spatial information based OSort for real-time spike sorting using FPGA," IEEE Transactions on Biomedical Engineering, vol. 68, no. 1, pp. 99-108, 2020.

[17] S. Yin, Y. Kim, X. Han, H. Barnaby, S. Yu, Y. Luo, W. He, X. Sun, J.-J. Kim, and J.-s. Seo, "Monolithically integrated RRAM-and CMOS-based in-memory computing optimizations for efficient deep learning," IEEE Micro, vol. 39, no. 6, pp. 54-63, 2019.

[18] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," Nature nanotechnology, vol. 15, no. 7, pp. 529-544, 2020.

[19] Y. Shi, A.Ananthakrishnan, S. Oh, X. Liu, G. Hota,G. Cauwenberghs, D.Kuzum, "High Throughput Neuromorphic Brain Interface with CuOx Resistive Crossbars for Real-time Spike Sorting," International Electron Devices Meeting, vol. In press, 2021.

[20] D. A. Adamos, E. K. Kosmidis, and G. Theophilidis, "Performance evaluation of PCA-based spike sorting algorithms," Computer methods and programs in biomedicine, vol. 91, no. 3, pp. 232-244, 2008.

[21] J. Wouters, F. Kloosterman, and A. Bertrand, "Towards online spike sorting for high-density neural probes using discriminative template matching with suppression of interfering spikes," Journal of neural engineering, vol. 15, no. 5, p. 056005, 2018.

[22] A. Bar-Hillel, A. Spiro, and E. Stark, "Spike sorting: Bayesian clustering of non-stationary data," Journal of neuroscience methods, vol. 157, no. 2, pp. 303-316, 2006.

[23] J. A. Herbst, S. Gammeter, D. Ferrero, and R. H. Hahnloser, "Spike sorting with hidden Markov models," Journal of neuroscience methods, vol. 174, no. 1, pp. 126-134, 2008.

[24] H. G. Rey, C. Pedreira, and R. Q. Quiroga, "Past, present and future of spike sorting techniques," Brain research bulletin, vol. 119, pp. 106-117, 2015.

[25] A. P. Buccino and G. T. Einevoll, "Mearec: a fast and customizable testbench simulator for ground-truth extracellular spiking activity," Neuroinformatics, vol. 19, no. 1, pp. 185-204, 2021.

[26] C. Rossant, S. N. Kadir, D. F. Goodman, J. Schulman, M. L. Hunter, A. B. Saleem, A. Grosmark, M. Belluscio, G. H. Denfield, and A. S. Ecker, "Spike sorting for large, dense electrode arrays," Nature neuroscience, vol. 19, no. 4, pp. 634-641, 2016.

[27] X. Hong, D. J. Loy, P. A. Dananjaya, F. Tan, C. Ng, and W. Lew, "Oxide-based RRAM materials for neuromorphic computing," Journal of materials science, vol. 53, no. 12, pp. 8720-8746, 2018.

[28] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in 2016 IEEE International Electron Devices Meeting (IEDM), 2016, pp. 16.2. 1-16.2. 4: IEEE.

[29] P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, and H.-S. P. Wong, "Face classification using electronic synapses," Nature communications, vol. 8, no. 1, pp. 1-8, 2017.

[30] W. Shim, J.-s. Seo, and S. Yu, "Two-step write–verify scheme and impact of the read noise in multilevel RRAM-based inference engine," Semiconductor Science and Technology, vol. 35, no. 11, p. 115026, 2020.

[31] S. Luan, I. Williams, M. Maslik, Y. Liu, F. De Carvalho, A. Jackson, R. Q. Quiroga, and T. G. Constandinou, "Compact standalone platform for neural recording with real-time spike sorting and data logging," Journal of neural engineering, vol. 15, no. 4, p. 046014, 2018.

Chapter 6. Integration of Ag-CBRAM Crossbars and Mott-ReLU Neurons for Efficient Implementation of Deep Neural Networks in Hardware

6.1 Abstract

In-memory computing with emerging non-volatile memory devices (eNVMs) has shown promising results in accelerating matrix-vector multiplications (MVMs). However, activation function calculations are still being implemented with general processors or large and complex neuron peripheral circuits. Here, we present integration of Ag-based conductive bridge random access memory (Ag-CBRAM) crossbar arrays with Mott-ReLU activation neurons for scalable, energy and area efficient hardware implementation of deep neural networks (DNNs). We develop Ag-CBRAM devices that can achieve high ON/OFF ratio and multi-level programmability. Compact and energy efficient Mott-ReLU neuron devices implementing rectified linear unit (ReLU) activation function are directly connected to the columns of Ag-CBRAM crossbars to compute the output from the weighted sum current. We implement convolution filters and activations for VGG-16 using our integrated hardware and demonstrate successful generation of feature maps for CIFAR-10 images in hardware. Our approach paves a new way towards building highly compact and energy efficient eNVMs based in-memory computing system.

6.2 Introduction

Deep neural networks (DNNs) have been widely successful in solving difficult problems in computer vision, speech recognition, machine translation, playing board and video games and medical diagnosis. DNNs have been constantly making breakthroughs in improving the state-of-the-art computational accuracy [1]. Large-scale DNNs require a very large number of matrix vector multiplication (MVM) operations in each layer followed by non-linear neuron activations between the layers (Figure 6.1a). By introducing non-linear transformation to the input, activation function plays an important role in solving vanishing gradient problem and making the network capable to learn and perform more complex tasks [2]. Although in-memory computing with emerging non-volatile memory arrays (eNVMs) considerably accelerate computation of MVMs [3, 4], current approaches still require external processors or complex peripheral circuits to implement neuron activations. Analogue-to-digital converters (ADC) is typically used in computing activation function and propagate data through eNVM layers. However, it has been shown that the power of 9-bit SAR-ADCs is roughly 1W compared to 0.3W dissipated on a 4096 × 4096 eNVM array for MVM operations [5]. The energy and latency overheads associated with separate implementation of the weights and activations significantly increase the energy consumption and constitute a major bottleneck for scalability of the hardware with ever evolving neural network architectures. Recent advances have explored using analogue CMOS circuits [6] or an ADC with reconfigurable function mapping [7] to implement activation functions in hardware. Although these approaches improve processing speed, they are difficult to be directly integrated as part of the eNVM array due to area mismatch compared to the compact array [8]. To overcome this limitation, we have previously developed a Mott activation neuron that implements the rectified linear unit function in the analogue domain [8]. Integration of resistive memory synaptic arrays with Mott-ReLU neurons could enable full hardware implementation of DNNs through direct combination of MVM operations with activation functions.

To that end, in this work, we experimentally investigate integration of Ag-CBRAM synaptic crossbars for MVMs and compact Mott neuron devices for activation functions to implement a VGG-16 inference task. Our hardware demonstration concentrates on convolutional and activation layers, which are main building blocks of VGG-16. Our Ag-CBRAM device exhibits high ON/OFF ratio (~10^{10}) and 4-bit multi-level switching, which are suitable for performing large scale MVMs in DNNs. Mott activation neurons integrated with CBRAM arrays emulate characteristics of ReLU activation function, which is the most frequently used activation functions in DNNs [9]. As shown in Figure 6.1b, a crossbar comprises Ag-CBRAM devices implementing synaptic layer accept inputs in the wordlines (WLs) and generates weighted sum current in the bitlines (BLs). Each column of Ag-CBRAM crossbars is connected to a nano-scale Mott neuron device for direct computation of ReLU activation using the weighted sum. The outputs of Mott ReLU devices can be directly fed to the WLs of the following Ag-CBRAM layers (Figure 6.1c). The rest of the paper is organized as follows. First, we present characterization of Ag-CBRAM devices including DC switching behavior, variation, retention, endurance and multi-level switching. Then, we share our results on the volatile four-terminal Mott activation neuron device based on vanadium dioxide (VO_2) and experimentally measured input-output characteristics for implementation of ReLU activation function. Transient response of the device is also measured to validate its low energy consumption. Lastly, we demonstrate hardware implementation of a CIFAR-10 image classification task using VGG-16 by integrating Ag-CBRAM crossbars and Mott-ReLU neurons using a custom PCB board. Our results based on integration of Ag-CBRAM crossbars and Mott-ReLU neurons suggest that the small size and energy efficiency of the Mott activation neuron can replace power-hungry CMOS circuits for ReLU activation and allow direct stacking of multiple synaptic layers.

6.3 Results

6.3.1 Ag-based CBRAM

The simplicity of fabrication makes lateral eNVM devices desirable for direct integration on the back end of line (BEOL) CMOS circuitry. For implementing synaptic layers, we first developed a lateral Ag-based CBRAM crossbar that can be fabricated at the wafer scale as shown in Figure 6.2a. The 4-inch wafer contains 16×16 and 32×32 crossbar arrays as well as single devices for electrical characterization (Figure 6.2b). For crossbar fabrication, we started with 300nm SiO_2/Si wafer and deposited 50nm-thick Ag layer via DC sputtering. Then a 5 µm × 20 µm Ag channels along with its BL were patterned via photo-lithography and wet-etching. Next, 250 nm SiO_2 was deposited as an insulating layer with PECVD method. After patterning SiO_2 to open via holes, 200nm Cr/Au was deposited and patterned for WLs as well as defining contact pads for BLs and WLs. Single test devices were fabricated in a similar way. The Ag-CBRAM devices (Figure 6.2c) exhibit an initial low resistance as fabricated and need to undergo an oxidation step whereby the device is transformed from its highly conductive “pristine” state (Figure 6.2c) to an oxidized high-resistance state (HRS) to initialize the subsequent switching (Figure 6.2d, e) using a low amplitude voltage sweep. Figure 6.3a demonstrates this forming process. The forming process here is different from the conventional forming process involving formation of a conductive filament in metal oxide-based RRAM devices. Instead, here the forming step transforms the conductive metal layer into an oxideized state which exhibits resisitve switching. As the voltage input was swept from 0V to 1V, the device current increased nearly proportionately up to ~45 mA. However, when the input bias reached ~0.8V, the resistance of the Ag channel suddenly increased from its $R_{initial}$ = 14.4 Ω to R_{formed} = ~10^{12}Ω. By comparing the optical images of the pre-formed (Figure 6.2c) and post-formed (Figure 6.2d) device, we noticed that the left part of the Ag channel became visibly darker shown in Figure 6.2e, likely due to the formation of resistive silver oxide. This observation explains the forming-induced transition to HRS.

Thereafter, the device operates as a resistive switching memory. Figure 6.3b shows the bipolar I-V characteristics of the Ag-CBRAM as measured from a DC double sweep cycle. Here, the applied voltage was increased in 5mV steps for the positive (0V to 2V) and negative (0V to -1V) voltage ramps while enforcing compliance currents of 500μA and 10mA to achieve SET and RESET respectively. To investigate the consistency of switching operations, we characterized the statistical distribution of switching voltages and device resistances by performing 50 DC switching experiments (Figure 6.3c). The average switching voltage for the SET was ~1.77V whereas that for RESET was ~ -0.35V. Figure 6.3d records the average low resistance state (LRS) and HRS resistances of ~340 Ω and ~3×10^{13} Ω with good uniformity. These values translate to an ultra-high ~10^{10} ON/OFF ratio of the device which distinctly favours its flexibility in mapping a wide range of neural network weights [10]. It is noteworthy that the low resistance of the Ag-CBRAM promises low latency operation whereas the high HRS can help lower the static power consumption of the device by suppressing leakage currents.

Device reliability is essential for implementing network training and inference that requires frequent switching and long-term storage of the weights [11]. Figure 6.4 shows that our device can retain the HRS and LRS states for more than 10^4 s at room temperature. In addition to being non-volatile with long retention, our Ag CBRAM devices exhibit high endurance. Figure 6.4b shows that the device can be switched between LRS (~$10^4\Omega$) and HRS (~$10^{12}\Omega$) for at least 10^4 cycles without any observable degradation. The HRS and LRS over 10^4 cycle switching is presented in Figure 6.4c, which indicates our device maintains low variations in both states. These results confirm the capability of Ag-CBRAM for achieving reliable crossbar operation with long-term stability.

While we demonstrated conventional memory application for Ag-CBRAM device, gradual resistance switching is a key to achieve higher storage density by mapping multi-bit weights to a single device. By controlling the current compliance levels from 100pA to 1mA, the device can reliably switch between 16 states spanning 7 orders of magnitude in resistance as shown in Figure 6.5a. Moreover, we characterized the mean (Figure 6.5b) and standard deviation (Figure 6.5c) of each distinct resistance level versus compliance current. Our results validate that the Ag-CBRAM device has multi-level programmability with minimal overlap between different levels.

6.3.2 Mott-ReLU Activation Neuron

We have previously developed array of Mott-ReLU neuron devices to implement activation function layer [8]. Each Mott-ReLU neuron device has four terminals that allow exploiting of a thermal driven Mott transition of VO_2, which emulates ReLU activation function in a single device (Figure 6.6a). Mott-ReLU devices were fabricated by depositing 70nm VO_2 film via reactive sputtering. Then device switching area was defined by two Ti (20 nm)/Au (30 nm) electrodes with 50 nm gap using e-beam lithography and evaporation. Then, 70 nm Al_2O_3 was deposited as the electrical insulation layer. Lastly, a local heater was defined by patterning Ti (20 nm)/Au (30 nm) nanowire on the VO_2 gap using e-beam lithography and evaporation. A SEM image of a fabricated device is shown in Figure 6.6b. Figure 6.6c explains the operation of the device. The resistance of the heater is ~30 Ω and the initial resistance the VO_2 gap is ~10kΩ. ReLU activation function can be emulated by applying current bias to the nanowire that induce thermal gradual resistivity switching on the VO_2 gap. The VO_2 gap formed a voltage divider circuit with a load resistor to generate voltage output (V_{OUT}) that can be directly fed as an input to the synaptic layer. By flowing current through the heater, the temperature of the VO_2 gap is precisely modulated to induce thermal-driven gradual resistive switching. As a result, the resistance change of the VO_2 gap modulates V_{out} through V_{load} and successfully emulates ReLU function as shown in Figure 6.6c. The device shows precision higher than 4-bit. As shown in Figure 6.6d, Mott-ReLU neurons show low latency of ~61.4 ns, while consuming 199.5pJ per operation.

To further understand how to achieve the optimal energy efficiency for our device, we developed an empirical thermal model in SPICE to project energy consumption of the device. This compact thermal model consists of Joule-heating model of the heater, thermal model of VO_2 and coupling model between heater and VO_2 gap (Figure 6.7a). Figure 6.7b lists model parameters and equations (1) and (2) govern the heater current and latency estimation in the model. By varying heater thermal resistance, our model indicates that heater current can be reduced by 3.4× when the thermal resistance of the nanowire heater increased by 10× (Figure 6.7c). Therefore, replacing the heater material with a higher thermal resistance material such as Ti can significantly improve thermal coupling and allow generated heat to be more confined within the VO_2 gap. In addition, the latency can be further reduced to ~3.8ns by minimizing the parasitic capacitance of the Mott ReLU below 10^{-11} F as shown in Figure 6.7d. As a result, our model estimates the energy consumption of Mott-ReLU neurons can be minimized down to ~0.638pJ at single device level by careful engineering the heater material to enhance the thermal coupling and reduce parasitic capacitance.

Table6.1 summarizes the energy, latency, area and leakage performance of Mott ReLU activation devices against other activation devices or circuits at single ReLU level. Our Mott ReLU device can already achieve ~17× energy reduction compared with Analogue CMOS circuits [6]. With optimized thermal coupling, the device is projected to achieve ~30× energy reduction compared with digital ADC implementation [7]. The device can also provide 450-1500×

improvement in area and 1.5-3× improvement in latency. These substantial performance gains of activation layers motivate our integration with Ag-CBRAM array in Section 1.3 that can achieve more efficient DNN implementation in hardware.

6.3.3 Ag-CBRAM and Mott-ReLU Integration for a DNN Application

To demonstrate core operations of DNN inference with our hardware, we focused on VGG-16 for CIFAR10 image classification task in a hardware. First, we designed a custom PCB to integrate the Ag-CBRAM crossbar arrays with Mott-ReLU arrays (Figure 6.8a). The board is capable of monitoring two arrays simultaneously and verifying weighted sum and activation results. We used 16×16 Ag-CBRAM crossbar for this demonstration (Figure 6.8b). The callout window of Figure 6.8b shows a representative device in the crossbar. Figure 6.8c shows an array that contains 44 Mott-ReLU devices that can be individually connected to the BLs of the crossbar via PCB.

Then we investigated how to efficiently map VGG-16 to our hardware. VGG-16 is a convolutional neural network that is 16 layers deep, which was widely used for computer vision applications. Figure 6.9a shows the representative CIFAR-10 images from 10 classes and network architecture. In VGG-16, there are 13 convolutional layers in which each layer is followed by ReLU activation layers, 5 max pooling layers, and 3 fully connected layers in order. For the hardware demonstration, we focused on convolutional layers. Max pooling layers and fully connected layers were implemented in software. Before we map full-precision (64-bit) weights in VGG-16 into hardware, we performed post-training uniform quantization of both weights and activation function with various precision and investigated its impact on inference accuracy. Figure 6.9b shows that 5-bit weights precision and 4-bit activation precision are the minimal bit precision allowed to ensure there is no significant accuracy degradation. Although each Ag-CBRAM cell in our crossbar array has gradual resistive switching capabilities as shown in Figure 6.5a, analogue approach requires custom peripheral neuron circuits to precisely vary current compliance and realize fine control of resistance levels. Therefore, we chose to use digital implementation for this array level demonstration to ensure better controllability of the resistance states. Figure 6.9c explains the mapping of the network to the Ag-CBRAM crossbar arrays using binary weights (HRS~10^{12} Ω, LRS~20 kΩ). In this illustration, N of 3×3 convolutional filters are unrolled to N of 9×1 vectors and mapped to columns of the crossbar. We quantized the filter weights into 5-bit binary representation to minimize memory size while maintaining high accuracy. As a result, five columns are used to represent MSB to LSB of the weights. As the filter slides across the input image, the part of the input (W×W) overlaps with the filter is also unrolled to a 9×1 vector and feed into the WLs of the crossbar. The crossbar performs MVM and the weighted sum current is accumulated at the end of each column. Activation layers are implemented by connecting a Mott-ReLU to each column. Mott-ReLU neurons rectify weighted sum and produce final pixel values in output feature maps (OFM).

Before implementing CIFAR-10 classification task in our hardware, we first tested whether Ag-CBRAM crossbars can drive Mott-ReLU neurons (Figure 6.10a). We varied the input voltage (V_{in}) to a column of the CBRAM array by sweeping it from –250 mV to 250 mV when ~2/3 of devices on a column of the CBRAM array are set to a LRS while the others are set to HRS (Figure 6.10b). Moreover, we varied the number of LRS in the column of Ag-CBRAM array from 0%-100% (Figure 6.10c). For both cases, 1.1 V is applied as V_{DD} to the VO_2 gap of Mott-ReLU with a 3.3-kΩ-load resistor connected in series, and 7 mA of offset current is applied to the heater. As can be seen in Figure 6.10b, c, the Mott-ReLU neuron shows ReLU input-output characteristics.

After verifying Mott-ReLU neuron can be driven by Ag-CBRAM crossbar, we then converted CIFAR10 images into 8-bit pulse trains and fed into the crossbar to generate weighted sum current, I_{sum}, which is a result of application of convolution filters to the images. Mott-ReLUs rectify I_{sum} and generate ReLU output (V_{out}). Figure 6.11 shows representative experimental results from network operations performed for first (layer1) and last (layer13) convolution layer of VGG-16 on a 32×32 input image from the dog class. For each layer, we presented both software

(SW) simulated result and measured result in hardware (HW) side by side for comparisons. Figure 6.11a and g shows 3×3 quantized convolution filters. After mapping these filters using the approach described previously to the Ag-CBRAM array, I_{sum} is measured at the end of each BLs. Figure 6.11b and h show measured I_{sum} in real time for 4 representative patches (each patch contains 5×5 pixels) highlighted as red boxes in Figure 6.11c and i. Slide number represents the position of the filter as it slides across each patch. I_{sum} from each BLs drives individual Mott activation neurons in the PCB and output voltage of the neuron device is shown in Figure 6.11d and j. These results indicate that our hardware implementation of convolution filters and activations can reliably generate OFMs and ReLU output without additional driver circuits and achieve close to ideal software results (Figure 6.11e, f, k, and l). The learned OFMs represent abstract features of the dog class in layer 1 and 13. Based on the measured results in hardware, the estimated classification accuracy for the entire CIFAR-10 dataset using our hardware is 93.04%, approaching ideal software accuracy (~94%). Energy efficiency is estimated as 25.7 TOPS/W.

6.4 Conclusion

In this work, a direct integration of Ag-CBRAM array with Mott-ReLU activation neurons are successfully demonstrated in hardware. Our Ag-CBRAM device shows ultra-high ON/OFF ratio, low variation, reliable endurance and retention. In addition, Ag-CBRAM has multi-level switching capability with 16 states, making it an ideal synaptic device for neural network operation. The simplicity of fabrication for lateral Ag-CBRAM array makes it easy to be integrated with the back end of the line (BEOL) of CMOS chips. The four-terminal Mott-ReLU device embodies ReLU characteristics and can be directly driven by weighted sum currents generated in Ag CBRAM array. The small footprint of the device allows stacking in between synaptic layers for scalable in-memory computing system. The hardware demonstration shows that Ag CBRAM arrays integrated with Mott ReLU devices offer a compact and scalable solution for accelerating DNNs with close to software accuracy. Our approach opens new avenues in implementing deeper and more complex network architecture with higher area and energy efficiency using eNVM based synaptic arrays and Mott-ReLU activation devices.

6.5 Acknowledgments

This work was supported by Office of Naval Research (N000142012405), the National Science Foundation (ECCS-1752241, ECCS-2024776 and ECCS-1734940), and the National Institutes of Health (DP2 EB030992). The experimental aspects of this work were supported as part of the Quantum Materials for Energy Efficient Neuromorphic Computing (Q-MEEN-C) Energy Frontier Research Center (EFRC), funded by the US Department of Energy, Office of Science, Basic Energy Sciences under award #DE-SC0019273. The fabrication of the devices was performed at the San Diego Nanotechnology Infrastructure (SDNI) of the University of California San Diego, supported by the National Science Foundation (ECCS-1542148).

Chapter 6, is a reprint of the material that is under submission and to appear in publication as Yuhan Shi, Sangheon Oh, Javier del Valle, Pavel Salev, Ivan K. Schuller, Duygu Kuzum. “Integration of Ag-CBRAM Crossbars and Mott-ReLU Neurons for Efficient Implementation of Deep Neural Networks in Hardware.” The dissertation author was the first author of this paper. Chapter 6, in part, is also a reprint of Sangheon Oh, Yuhan Shi, Javier Del Valle, Pavel Salev, Yichen Lu, Zhisheng Huang, Yoav Kalcheim, Ivan K. Schuller, and Duygu Kuzum. "Energy-efficient Mott activation neuron for full-hardware implementation of neural networks." Nature nanotechnology 16, no. 6 (2021): 680-687. The dissertation author was the primary author of this paper.

6.6 Figures

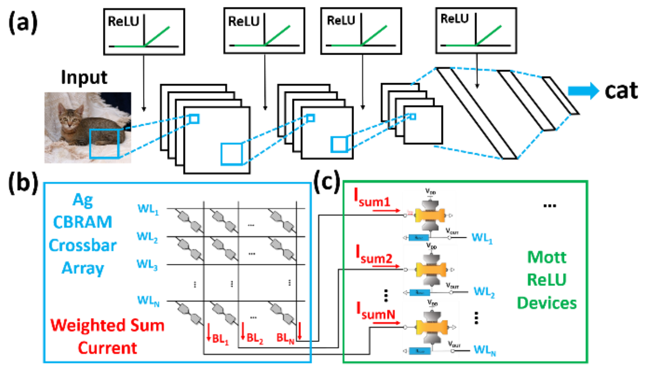


Figure 6. 1 DNN Architecture and Integration of Ag-CBRAM Crossbar and Mott ReLU Neuron.

(a) Representative DNN consisting of convolutional layers and ReLU activations. ReLU activations are applied after each convolutional layer. (b) Convolutional layers are implemented with Ag-CBRAM crossbar arrays. Ag-CBRAM devices are arranged in a crossbar fashion with inputs feed into the WLs and weighted sum current accumulated in BLs. (c) ReLU activation layers are implemented with Mott-ReLU devices. The weighted sum current in BLs drive inputs of Mott devices and outputs of the devices are fed to the subsequent Ag-CBRAM layers.

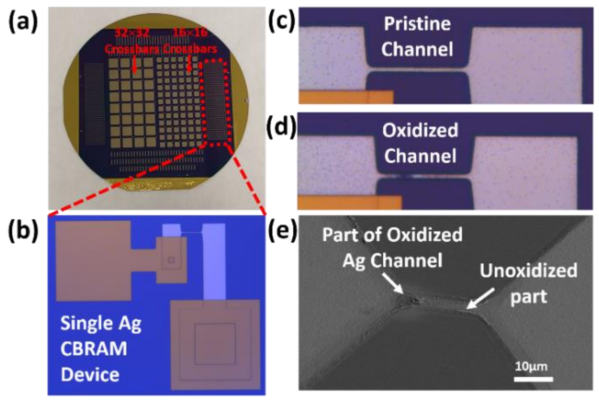


Figure 6. 2 Ag-CBRAM Crossbar Device and SEM Image.

(a) Wafer-scale image of Ag-CBRAM crossbar arrays (16×16 and 32×32) and (b) single devices with 5μm×20μm channel. Microscope images of (c) Pristine and (d) Oxidized Ag channel. The lateral Ag-CBRAM devices have BL and WL pads defined with the narrow channel. (e) SEM for oxidized channel which shows darker colour compared with unoxidized part.

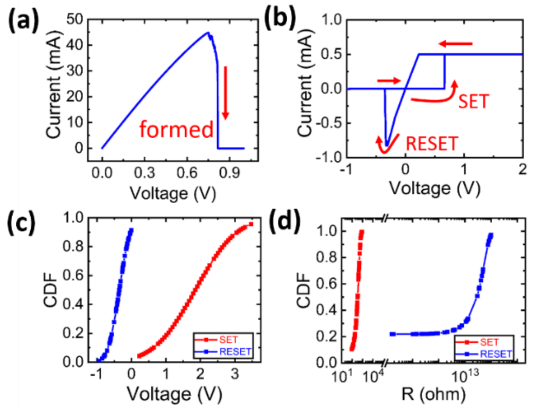


Figure 6. 3 Ag-CBRAM Device DC Characteristics, Voltage and Resistance Variation.

(a) Device initialization by applying a voltage sweep to oxidized pristine Ag-channel. (b) Bipolar I-V switching characteristic measured by a typical DC double sweep. Binary states are obatined. The cycle to cycle (C2C) variation of (c) switching voltages and (d) resistance represented using cumulative distribution function (CDF). C2C variations are obtained by applying 50 DC double sweeps to a Ag-CBRAM device.

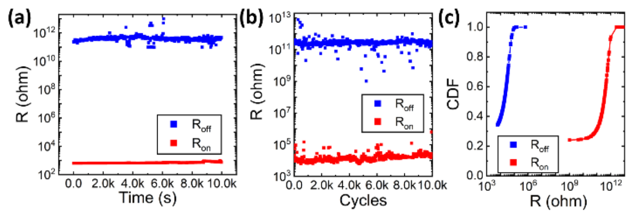


Figure 6. 4 Ag-CBRAM Device Retention and Endurance.

(a) Retention behaviour of Ag-CBRAMs. The device is first SET to LRS. A sampling measurement that lasts 10^4s is performed to constantly monitor the device resistance. The device is then switched to HRS and same measurement is performed. (b) Endurance of Ag-CBRAMs. 10^4 cycles are achieved by alternatively applying SET and RESET pulses to the device while monitoring the device resistance. During the test, the RESET and SET transitions were achieved using -4V/100µs and 3V/5ms voltage pulses, respectively. (c) CDF of resistance in pulse programming, which is extracted from the endurance measurements in (b).

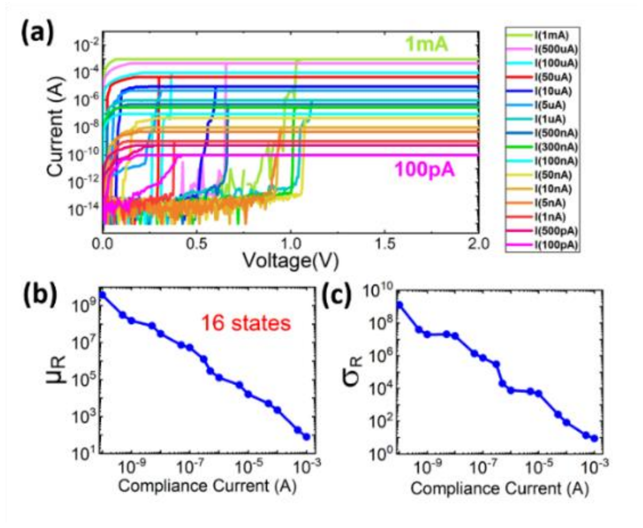


Figure 6. 5 Ag-CBRAM Device Multi-level Switching Characteristics.

(a) Multi-level switching characteristics using different SET current compliance from 100pA to 1mA. The SET sweep is applied from 0V to 2V with 100mV steps for all compliance current conditions. (b) mean and (c) standard deviation of 16 (4-bit) resistance states as a function of compliance current.

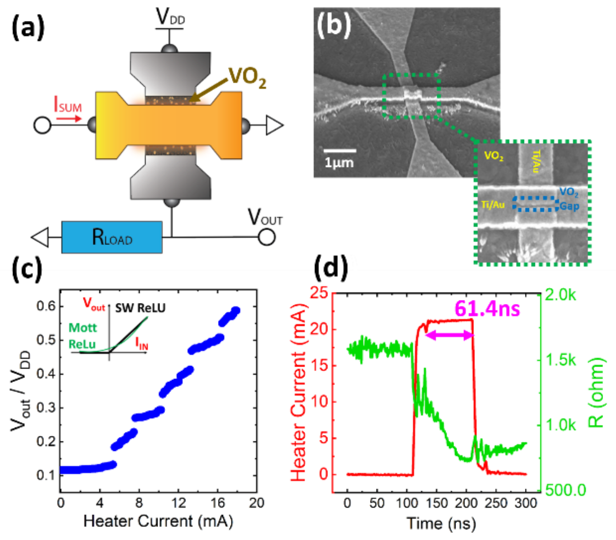
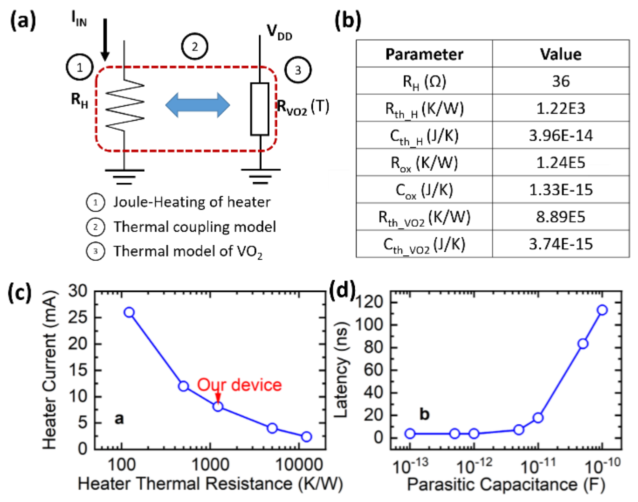


Figure 6. 6 Mott-ReLU Neuron Switching Characteristics and SEM Image.

(a) Schematic and (b) SEM of Mott-ReLU device. The heater driven by weighted sum current (I_{sum}) enables the change of VO_2 resistance to emulate ReLU characteristics. (c) Measured Mott-ReLU activation output characteristic. The inset illustrates software ReLU in black in compare with Mott ReLU in green. (d) Pulse measurement for latency. Output of the Mott device becomes stable after input applied to heater for ~61.4 ns.



Parameter	Value
R_H (Ω)	36
R_{th_H} (K/W)	1.22E3
C_{th_H} (J/K)	3.96E-14
R_{ox} (K/W)	1.24E5
C_{ox} (J/K)	1.33E-15
R_{th_VO2} (K/W)	8.89E5
C_{th_VO2} (J/K)	3.74E-15

Figure 6. 7 Compact Thermal Model of Mott-ReLU Neuron Device.

(a) A schematic shows the compact thermal model for the Mott ReLU device built in SPICE. The model has three parts: Joule-heating of heater, thermal coupling model for heater and VO_2 and thermal model of VO_2. (b) SPICE model parameters. (c) Heater current as heater thermal resistance of nanowire heater increases while keeping the resistance of the VO2 gap to 1 kΩ. (d) Latency of the Mott ReLU device as parasitic capacitance increases. ~3.8ns latency can be achieved by reducing parasitic capacitance < 10^{-11}F.

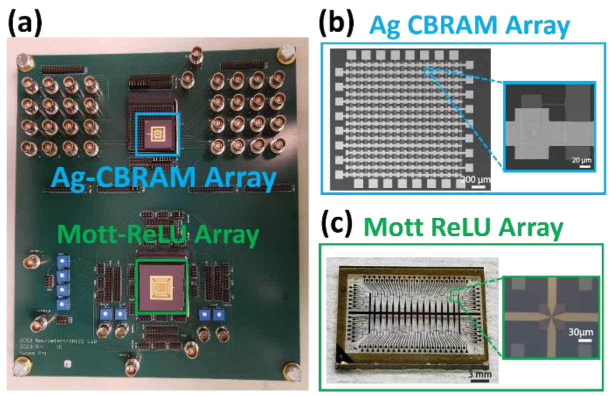


Figure 6. 8 Integration of Ag-CBRAM Crossbar Array and Mott ReLU Neuron Array in Custom PCB.

(a) Custom PCB integrating Ag-CBRAM crossbar (top) and Mott-ReLU array (bottom). (b) SEM image of 16x16 Ag-CBRAM crossbar array. Call out window shows single Ag-CBRAM device. (c) Mott-ReLU array including 44 Mott-ReLU neurons. Call out window shows single Mott-ReLU device.

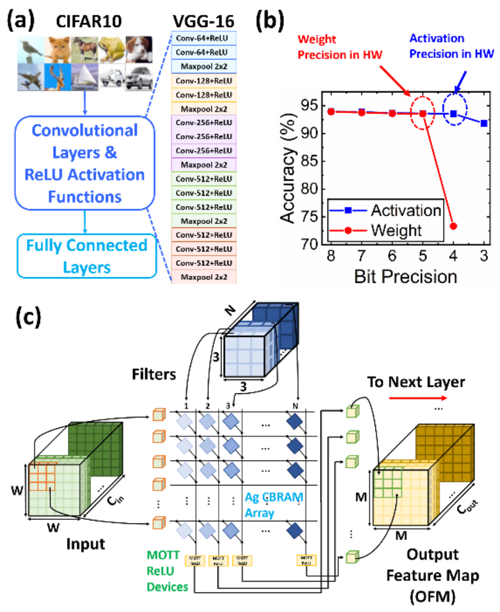


Figure 6. 9 VGG-16 Network Architecture and Mapping to Hardware.

(a) VGG-16 network architecture for CIFAR10 used for hardware implementation. (b) Post-training quantization using trained VGG-16 weights. 5-bit and 4-bit for weights and activation are use in hardware. (c) The network is mapped to the Ag-CBRAM crossbar by unrolling the filters. The weighted sum current in each BLs is fed into the Mott-ReLU neurons at the end of each column.

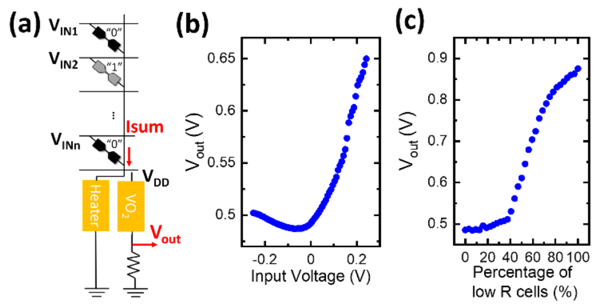


Figure 6. 10 Verification of Using Columns of Ag-CBRAM Array Drives Mott-ReLU Neuron.

(a) A column of Ag-CBRAM array with Mott-ReLU neuron connected at the end. Ag-CBRAM coloured in black represented LRS while coloured in grey represented HRS. ReLU input-output characteristics measured using PCB by (b) varying the input to the crossbar from –0.25V to 0.25V or (c) changing the number of LRS cells while fixing input voltage to 130mV.

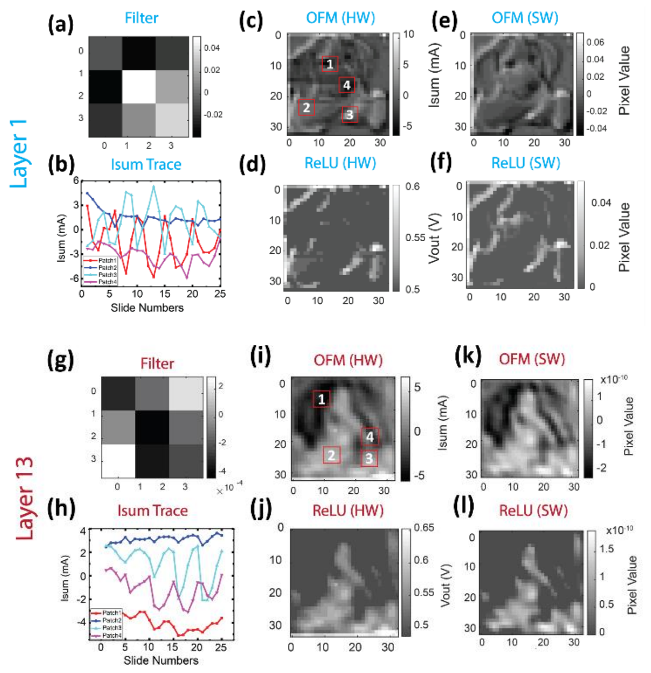


Figure 6. 11 Hardware Implementation of Convolution and ReLU Activation Layers of VGG-16.

Hardware implementation of convolution and ReLU activation layers of VGG-16 (Figure 6.9a) using Ag-CBRAM array and Mott-ReLU neurons on a CIFAR10 image. Representative results for layer 1 (a-f) and layer 13 (g-l) are shown. (a) and (g) are representative 3×3 convolutional filters in layer1 and layer13 respectively. The filter weights are quantized to 5-bit. (b) and (h) are measured weighted sum current trace from 4 patches shown in (c) and (i). (c) and (i) are OFMs obtained using Ag CBRAM synaptic array and in compared with OFMs generated in software ((d) and (j)). (e) and (k) are final OFMs after passing Mott-ReLU activation layers and in compared with ReLU results in software((f) and (l)).

Table 6. 1 Energy, Latency, Area and Leakage of Mott ReLU, Analogue CMOS and Digital ADC

*Shows projected optimal energy and latency when the thermal resistance of the heater is increased by 10× and the parasitic capacitance of a Mott ReLU is < 10^{-11}F. **The area is only the area per neuron circuit.

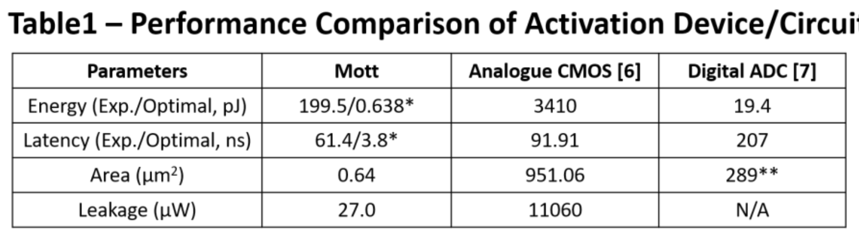


Table1 – Performance Comparison of Activation Device/Circuit

Parameters	Mott	Analogue CMOS [6]	Digital ADC [7]
Energy (Exp./Optimal, pJ)	199.5/0.638*	3410	19.4
Latency (Exp./Optimal, ns)	61.4/3.8*	91.91	207
Area (μm^2)	0.64	951.06	289**
Leakage (μW)	27.0	11060	N/A

6.7 Reference

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *nature,* vol. 521, no. 7553, pp. 436-444, 2015.

[2] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *2019 7th international conference on smart computing & communications (ICSCC)*, 2019: IEEE, pp. 1-4.

[3] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature electronics,* vol. 1, no. 6, pp. 333-343, 2018.

[4] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang and H. Li, "Emerging non-volatile memories: Opportunities and challenges," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2011, pp. 325-334.

[5] T.-J. Yang and V. Sze, "Design considerations for efficient deep neural networks on processing-in-memory accelerators," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019: IEEE, pp. 22.1. 1-22.1. 4.

[6] O. Krestinskaya, K. N. Salama and A. P. James, "Learning in memristive neural network architectures using analog backpropagation circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 66, no. 2, pp. 719-732, 2018.

[7] M. Giordano, G. Cristiano, K. Ishibashi, S. Ambrogio, H. Tsai, G. W. Burr and P. Narayanan, "Analog-to-digital conversion with reconfigurable function mapping for neural networks activation function acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 9, no. 2, pp. 367-376, 2019.

[8] S. Oh, Y. Shi, J. Del Valle, P. Salev, Y. Lu, Z. Huang, Y. Kalcheim, I. K. Schuller and D. Kuzum, "Energy-efficient Mott activation neuron for full-hardware implementation of neural networks," *Nature nanotechnology,* vol. 16, no. 6, pp. 680-687, 2021.

[9] K. Eckle and J. Schmidt-Hieber, "A comparison of deep networks with ReLU activation function and linear spline-type methods," *Neural Networks,* vol. 110, pp. 232-242, 2019.

[10] S. Yin, Y. Kim, X. Han, H. Barnaby, S. Yu, Y. Luo, W. He, X. Sun, J.-J. Kim and J.-s. Seo, "Monolithically integrated RRAM-and CMOS-based in-memory computing optimizations for efficient deep learning," *IEEE Micro,* vol. 39, no. 6, pp. 54-63, 2019.

[11] Y. Zhang, P. Huang, B. Gao, J. Kang and H. Wu, "Oxide-based filamentary RRAM for deep learning," *Journal of Physics D: Applied Physics,* vol. 54, no. 8, p. 083002, 2020.

Chapter 7. Conclusion and Future Directions

7.1 Conclusion

In the past decade, artificial intelligence (AI) undergoes unprecedented breakthrough and achieves performances exceeding humans in many challenging tasks involving image recognition, language processing, and video analysis. However, further advances in AI require training and interface larger deep neural networks (DNNs), which takes days to weeks and consumes many kilowatts of energy in traditional digital hardware, preventing their widespread use for power-limited applications or edge devices. In traditional digital hardware such as CPU or GPU, massive amounts of data have to shuttle between off-chip memory and processing units, which is known as the bottleneck of von Neumann architecture. On the other hand, neuro-inspired in-memory computing is faster and more energy efficient by performing massive matrix vector multiplications directly and parallelly in memory. eNVM based in-memory computing platforms have been suggested as a promising path to map DNNs onto hardware. However, device-level and network-level energy consumption needs to be substantially decreased for realization of low power in-memory computing systems, which can be trained with large-scale and realistic data sets. The algorithm and hardware co-design methodology plays a critical role in order to address this challenge.

In this dissertation, we present several algorithm and hardware co-design methodologies that applies to wide range of eNVM devices such as CBRAM, RRAM, PCM and STT-MRAM in Chapter 1 to Chapter 4. Guided by the benchmarking results from co-design methodologies, large scale spike sorting implemented using CuOx based RRAM crossbar and DNNs deployed using Ag-CBRAM crossbar array integrated with Mott ReLU neurons are presented in Chapter 5 and Chapter 6 respectively. These two hardware implementations of in-memory computing on real world computation tasks show significant energy efficiency improvement, latency reduction and small footprint compared to their digital counterparts, paving a way towards building next generation in-memory AI hardware.

More specifically, Chapter 1 and Chapter 2 of this dissertation demonstrates unsupervised learning with pruning using large-scale (512 kb) subquantum CBRAM array for the first time. Our subquantum CBRAM devices are directly programmable to arbitrary conductance states and consume 10× less power than filamentary RRAM. Direct programing enables implementation of any weight update rule with high complexity. In addition, we develop a pruning algorithm that can be implemented during the training. All other algorithmic pruning methods in literature prune the network after the training and do not solve the energy consumption and training time problems. Combining device level energy savings provided by subquantum CBRAM with network level energy savings by pruning can lead to a total energy reduction of 100× for hardware implementation of learning. In Chapter 3, a C++ based benchmarking platform is demonstrated to estimate energy, latency, and area accurately for different STT-MRAM based in-memory computing architectures for convolutional neural network (LeNet-5). This tool provides great insights and benefits for the researchers who want to use STT-MRAM for in-memory computing applications at deeper technology nodes. In Chapter 4, we develop an adaptive quantization technique to improve the classification accuracy for hardware learning using low-bit precision synaptic devices. This device-algorithm co-design approach allocates more levels to important weights based on the evolution of the weight distribution during training. As a proof of concept, we apply our adaptive quantization to a spiking neural network implemented with phase change memory (PCM) based synapses for unsupervised learning. Our result shows that the adaptive quantization improves accuracy by 75% compared with the prevailing uniform quantization method for 4-bit weight representation, making online learning with low bit precision possible. The co-design methodologies presented in this dissertation is applicable to other state-of-the-art DNNs to reduce the energy and time cost in implementing network training or interface in low-power applications.

Besides aforementioned algorithm and hardware co-design efforts, realizing large-scale real-world algorithms in eNVM array is also essential to understand system level performance and hardware limitations. In Chapter 5, we demonstrated a high throughput neuromorphic brain interface for real-time spike sorting based on CuOx resistive crossbars. These crossbars were fabricated using a low-temperature reactive sputtering process, enabling BEOL-integration with CMOS-based spike detection circuits. Hardware implementation of template matching using CuOx crossbars accurately classified spikes from individual neurons recorded in vivo, offering substantial performance gains in area, power, latency, and energy for neural probes with high channel counts. In Chapter 6, we successfully integrated Ag-CBRAM and Mott-ReLU devices in a custom PCB and demonstrate CIFAR10 image classification using VGG-16 in hardware. Convolution filters in VGG-16 are implemented using Ag-CBRAM array and activation functions are performed using Mott-ReLU device array. Our integration achieves close to software classification accuracy.

7.2 Future Directions

The algorithm-hardware co-design approach applied on diverse eNVM devices along with hardware demonstrations presented in this dissertation opens up new opportunities towards wide adoption of eNVM based in-memory computing system. The techniques and real hardware demonstrations shown in this dissertation also signify the importance of co-optimization between device, architecture and algorithms for improving the trade-offs between energy, latency and area in designing next-generation eNVM hardware. To realize eNVM based in-memory computing system with outstanding performance as well as versatility to adopt complex AI tasks, there still are challenges yet to be addressed.

From device perspective, eNVM devices should be specifically optimized targeting in-memory computing applications. In-memory computing requires different device characteristics than conventional digital memory applications. For example, the resistances of the device should ideally be engineered in Kohm to hundreds Mohm range to limit the current flow and avoid additional sneak path current when devices are arranged in a crossbar architecture. Wire resistance is also a critical parameter in preventing significant IR drop in BLs and WLs of the crossbar. As a result, crossbar array needs thicker metal wires while thin metal wires are used to maximum area density in conventional application. Moreover, eNVM with multi-level conductance switching capability is key to realize analog computation. Filamentary based eNVM devices are prevailing in state-of-the-art literatures. However, they suffer from limited conductance levels, high device to device variations and sensitive to conductance relaxation. It is worth to further look into other switching mechanism such as bulk switching or electrochemical based switching that can potentially provide better analog conductance tunability.

From circuit perspective, designing and demonstrating eNVM devices that can be fully integrated with BEOL of CMOS circuits is critical. Although key matrix-vector operations of eNVM crossbar arrays have been demonstrated several times in this dissertation, they heavily relied on external PCB boards and standard semiconductor parameter analyzer to provide the required interface and control circuitry. It dramatically decreases the throughput of the implementation and diminishes the potential energy and latency benefits offered by eNVM array in hardware. A fully functional system that integrates crossbars with CMOS neuron circuitry and equips with necessary ADC and DAC buses are desirable in the future to allow the prototypes to be scaled to larger systems.

From algorithm and architecture perspective, network models and architectures should be co-designed with careful considerations including device characteristics and hardware limitations. Network architectures should be tightly correlated with input-output relationship of the device, operating range of the device, variations in device and fan-in/out of the device. Efficient pipelining and program scheduling as well as multi-core architecture that can map multiple layers of networks in a single chip are crucial to further advance eNVM based in-memory computing system.