ABSTRACT

EFFICIENT HARDWARE IMPLEMENTATIONS OF BIO-INSPIRED NETWORKS

by
Anakha Vasanthakumaribabu

The human brain, with its massive computational capability and power efficiency in small form factor, continues to inspire the ultimate goal of building machines that can perform tasks without being explicitly programmed. In an effort to mimic the natural information processing paradigms observed in the brain, several neural network generations have been proposed over the years. Among the neural networks inspired by biology, second-generation Artificial or Deep Neural Networks (ANNs/DNNs) use memoryless neuron models and have shown unprecedented success surpassing humans in a wide variety of tasks. Unlike ANNs, third-generation Spiking Neural Networks (SNNs) closely mimic biological neurons by operating on discrete and sparse events in time called spikes, which are obtained by the time integration of previous inputs.

Implementation of data-intensive neural network models on computers based on the von Neumann architecture is mainly limited by the continuous data transfer between the physically separated memory and processing units. Hence, non-von Neumann architectural solutions are essential for processing these memory-intensive bio-inspired neural networks in an energy-efficient manner. Among the non-von Neumann architectures, implementations employing non-volatile memory (NVM) devices are most promising due to their compact size and low operating power. However, it is non-trivial to integrate these nanoscale devices on conventional computational substrates due to their non-idealities, such as limited dynamic range, finite bit resolution, programming variability, etc. This dissertation demonstrates the architectural and algorithmic optimizations of implementing bio-inspired neural networks using emerging nanoscale devices.

The first half of the dissertation focuses on the hardware acceleration of DNN implementations. A 4-layer stochastic DNN in a crossbar architecture with memristive devices at the cross point is analyzed for accelerating DNN training. This network is then used as a baseline to explore the impact of experimental memristive device behavior on network performance. Programming variability is found to have a critical role in determining network performance compared to other non-ideal characteristics of the devices. In addition, noise-resilient inference engines are demonstrated using stochastic memristive DNNs with 100 bits for stochastic encoding during inference and 10 bits for the expensive training.

The second half of the dissertation focuses on a novel probabilistic framework for SNNs using the Generalized Linear Model (GLM) neurons for capturing neuronal behavior. This work demonstrates that probabilistic SNNs have comparable performance against equivalent ANNs on two popular benchmarks - handwritten-digit classification and human activity recognition. Considering the potential of SNNs in energy-efficient implementations, a hardware accelerator for inference is proposed, termed as **Spin**tronic **A**ccelerator for **P**robabilistic **S**NNs (SpinAPS). The learning algorithm is optimized for a hardware friendly implementation and uses first-to-spike decoding scheme for low latency inference. With binary spintronic synapses and digital CMOS logic neurons for computations, SpinAPS achieves a performance improvement of 4$\times$ in terms of GSOPS/W/mm^2 when compared to a conventional SRAM-based design.

Collectively, this work demonstrates the potential of emerging memory technologies in building energy-efficient hardware architectures for deep and spiking neural networks. The design strategies adopted in this work can be extended to other spike and non-spike based systems for building embedded solutions having power/energy constraints.

EFFICIENT HARDWARE IMPLEMENTATIONS OF BIO-INSPIRED NETWORKS

by
Anakha Vasanthakumaribabu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology – Newark
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

Helen and John C. Hartmann Department of Electrical and Computer Engineering

May 2020

APPROVAL PAGE

EFFICIENT HARDWARE IMPLEMENTATIONS OF BIO-INSPIRED NETWORKS

Anakha Vasanthakumaribabu

Dr. Durgamadhab Misra, Dissertation Advisor Date
Professor, Department of Electrical and Computer Engineering, NJIT

Dr. Bipin Rajendran, Dissertation Co-Advisor Date
Reader, Department of Engineering, King's College, London

Dr. Osvaldo Simeone, Dissertation Co-Advisor Date
Professor, Department of Engineering, King's College, London

Dr. Catherine Dubourdieu, Committee Member Date
Professor, Institute of Chemistry and Biochemistry, Freie Universität Berlin

---

Dr. Dong-Kyun Ko, Committee Member Date
Assistant Professor, Department of Electrical and Computer Engineering, NJIT

Dr. Hieu Pham Trung Nguyen, Committee Member Date
Assistant Professor, Department of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Anakha Vasanthakumaribabu

Degree: Doctor of Philosophy

Date: May 2020

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, USA, 2020
- Master of Technology in Microelectronics & VLSI Design,
  Indian Institute of Technology Bombay, Mumbai, India, 2016
- Bachelor of Engineering in Electronics and Communication Engineering,
  Govt. Engineering College Bartonhill, Thiruvananthapuram, India, 2010

Major: Electrical Engineering

Presentations and Publications:

A. V. Babu, S. Lashkare, U. Ganguly, B. Rajendran, "Stochastic Learning in Deep Neural Networks Based on Nanoscale PCMO Device Characteristics," *Neurocomputing*, vol. 321, pp. 227-236, Dec 2018.

A. V. Babu, O. Simeone, B. Rajendran, "SpinAPS: A High-Performance Accelerator for Probabilistic Spiking Networks," jorunal manuscript under preparation.

A. V. Babu, B. Rajendran, "Stochastic Deep Learning in Memristive Networks," *Proceedings of 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 214-217, Batumi, Georgia, Dec 2018.

S. R. Kulkarni, A. V. Babu, B. Rajendran, "Spiking Neural Networks – Algorithms, Hardware Implementations and Applications," *Proceedings of 60th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Invited paper, pp. 426-431, Boston, MA, Aug 2017.

S. R. Nandakumar, S. R. Kulkarni, A. V. Babu, B. Rajendran, "Building Brain-Inspired Computing Systems: Examining the Role of Nanoscale Devices," *IEEE Nanotechnology Magazine*, vol. 12, no. 3, pp. 19-35, Sept 2018.

S. R. Kulkarni, A. V. Babu, B. Rajendran, “Acceleration of Convolutional Networks Using Nanoscale Memristive Devices,” *Proceedings of 19th International Conference on Engineering Applications of Neural Networks (EANN)*, pp. 240–251, Springer, Cham, Bristol, U.K., Sept 2018.

Dedicated to my family

ACKNOWLEDGEMENT

I am deeply indebted to the people around me for their support and help, without them this doctoral dissertation would not have been possible. I would like to express my deepest gratitude to Dr. Bipin Rajendran, who has given constant guidance and support throughout this work. I am extremely grateful to Dr. Bipin, for being patient and spending countless hours of his effort in helping me shape this research work. He always motivated me in all the technical discussions we had and taught me to think in different perspectives.

I like to extend my gratitude to Dr. Durgamadhab Misra, for his kind willingness to become my Dissertation Advisor and supporting me in the final phase of the research. I would also like to sincerely thank Dr. Osvaldo Simeone, Dr. Catherine Dubourdieu, Dr. Dong Kyun Ko and Dr. Hieu. P. Nguyen for being part of the dissertation committee and providing valuable feedback for improving this dissertation.

I also acknowledge National Science Foundation (NSF), grant # 1710009 for funding the research work on hardware implementations of probabilistic Spiking Neural Networks. I am extremely grateful to the department of Electrical and Computer Engineering, NJIT for also funding me through Teaching Assistantship in the first year of my research.

I am grateful to my colleagues at the Intelligent Computing Lab at NJIT, for the valuable discussions with Shruti R. Kulkarni, Alireza Bagheri, Vinay Joshi, Bleema Rosenfeld and Nandakumar S. R. I also acknowledge Hyueryung Jang, post doctoral researcher at King's College, London for her valuable input throughout the work on probabilistic SNNs. Finally, my thanks to all the staff and faculty of the ECE department at NJIT who have been very supportive and helpful throughout the Ph.D. program.

Last, but not the least, I would like to acknowledge with gratitude, the support and love of my parents, Babu. P, and Vasanthakumari. S, my sister, Akhila Vasanthakumaribabu, and my husband, Krishnamoorthy Govindaraj. They all kept me going and this dissertation would not have been possible without their constant support.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

The human brain, in addition to controlling all bodily activities, also integrates and processes information from the sense organs to perform complex cognitive tasks, all within a power budget of approximately 20 Watts. The information is processed and encoded through discrete and sparse events in time called action potentials or spikes. The energy efficiency of the brain is attributed to the sparsity of the spikes and event-driven communication between the neurons, with the information possibly encoded in time, frequency, and phase of the spikes. Several worldwide efforts like BRAIN Initiative [1], Human Brain Project [2], China Brain Project [3], etc., are targeted to understand the unique architecture of the human brain and to unveil the underlying cognitive mechanisms.

The human brain with 10^{11} neurons and 10^{15} synapses is estimated to have almost 100,000 kilometers of complex interconnections and a storage capacity of 1 terabyte [4–6]. The fastest supercomputer, today is inferior to nature's design by at least five orders of magnitude in terms of power and the number of operations that can be performed per Joule [7]. The human brain has been a constant inspiration for building energy-efficient computing architectures since the beginning of the computational era. Several neuroscience studies show that synapses, which are the junctions between two nerve cells, form the memory elements, and neurons (nerve cells) form the primitive computational nodes. Learning is typically achieved by modulating the synaptic strength between the neurons, which is termed as the synaptic plasticity. Memory (synapses) and computational elements (neurons) are co-located in the brain when compared to modern digital computers where they are physically separated. Thus, the performance of conventional computing architectures when handling data-intensive workloads gets limited by the energy/power required for transferring data back and forth between the storage and processing units. Machine learning workloads are typically data-intensive, and hence, von Neumann machines are not suitable for providing energy-efficient implementations. While efforts are still on-going to demonstrate brain-like energy efficiency in silicon-based computing systems, challenges associated with co-locating memory and processing units, and implementing large scale neural networks in silicon are yet to be solved [8, 9]. In this context, emerging nanoscale devices are considered as potential candidates to emulate the brain's connectivity in hardware owing to its small size, and low programming power requirements.

In this dissertation, we study some architectural and algorithmic optimizations for implementing deep and spiking neural networks using emerging nanoscale devices. The results obtained in this work could help in building energy-efficient architectures for embedded applications having power/energy constraints.

1.1 Bio-inspired Neural Networks

Inspired by biological neurons, several artificial neural networks have been explored since 1940s for efficient information processing. The evolution of neuromorphic computing can be traced back to the work by McCulloch and Pitts (1943) [10], that demonstrated neurons can perform any arithmetic and logical operations. A perceptron based network was demonstrated for solving non-linear separable problems in 1958 [11]. With further development of the backpropagation algorithm in 1974 [12, 13], the capability of neural networks was extended for solving more complex tasks. The past decade has seen enormous growth in the field of machine learning algorithms due to the ample availability of digital data for analysis and powerful computing platforms. Neural networks can be classified into perceptron based first-generation networks, second-generation artificial or deep neural networks

(ANNs), and third-generation spiking neural networks (SNNs) [14]. We will discuss the different generations of neural networks briefly in the next subsections.

1.1.1 Perceptron Neural Networks

Perceptron networks are first-generation neural networks, formulated by Rosenblatt in 1951. These networks operate on binary inputs and outputs (0,1), similar to the conventional computing systems. The output of a perceptron network is obtained by simple thresholding of the weighted synaptic input. The concepts of the threshold and synaptic weights are inspired by biology. Biological neurons propagate the electric potential to the postsynaptic neurons only if it receives a strong presynaptic input, exhibiting a thresholding behavior. While synaptic interaction in biology can be either spatial or temporal, perceptron networks use spatial summation, which multiplies the input with excitatory (positive) or inhibitory (negative) synaptic weights. Single and multilayer (stacked) perceptrons have been demonstrated to solve several functions like flip-flops, logical functions, etc. However, perceptrons are limited by their inability to solve non-linear functions and did not attract much research attention until the seventies. The development of non-linear models of neural networks, the discovery of training methodologies for multilayered neural networks, and the availability of computational resources in the late seventies led to the development of second-generation neural networks. These second-generation neural networks have demonstrated their ability to solve complex cognitive tasks, and the underlying architecture will be discussed in the subsequent subsection.

1.1.2 Artificial Neural Networks (ANNs)

Artificial or Deep Neural Networks (ANNs/DNNs) are second-generation neural networks in which the output of any neuron is a non-linear function of the weighted sum of the inputs from the previous layer. A simple model of an ANN with N inputs and one output ($N \times 1$ network) is shown in Figure 1.1 (a). The strength of the

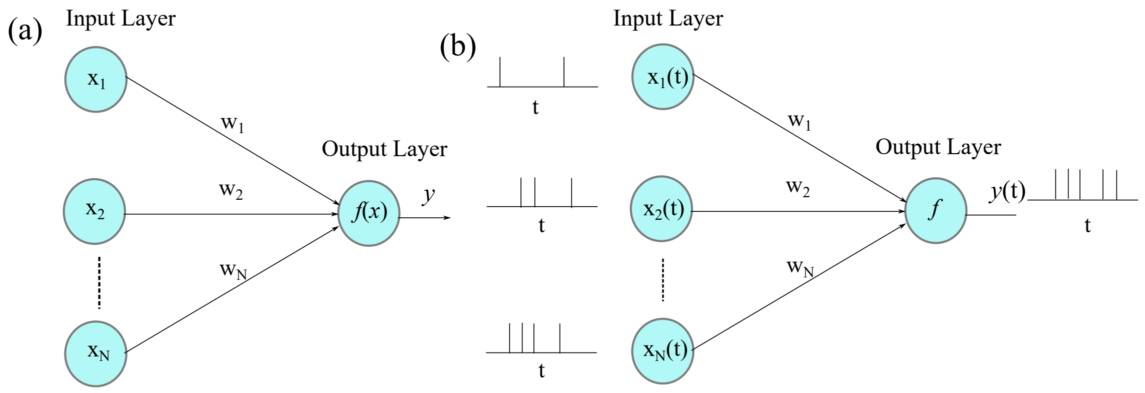


Figure 1.1 Biologically inspired networks illustrating a $N \times 1$ network, with N input neurons and 1 output neuron (a) Artificial Neural Network (ANN) (b) Spiking Neural Network (SNN) model.

connection between any neuron in the input and output layer is referred to as the synaptic weight. The real-valued synaptic weights in the $N \times 1$ network shown in Figure 1.1 (a) can be represented as $w = [w_1, w_2, w_3, \ldots w_N]$. Here w_1 corresponds to the synaptic weight between the first input neuron and the output neuron. Similarly, w_2 is the synaptic strength between the second input neuron and the output neuron, and so on. On asserting real-valued input x $[x = x_1, x_2, x_3, \ldots x_N]$, the effective input to the output neuron becomes $w^T x$ due to the fully connected network architecture. The output y can be determined as $y = f(w^T x)$, where f is a non-linear activation function and can be $\sigma(x)$, $\sinh(x)$, $\tanh(x)$ or $\mathrm{ReLU}(x) = x\mathrm{H}(x)$, with $\mathrm{H}(x)$ denoting the Heaviside step function. The fundamental computation in ANNs is the weighted sum calculation, usually quantified in terms of multiply-accumulate (MAC) operations.

The continuous and differential nature of the neuron's output or activation in ANNs has helped in applying the gradient descent rule for training multilayered (deeper) networks. These deep network models have shown superior performance surpassing humans and have become the state-of-the-art solution for problems like object recognition, gaming, text classification, speech recognition, etc.

1.1.3 Spiking Neural Networks (SNNs)

Spiking Neural Networks (SNNs), widely accepted as the third-generation neural networks, uses biologically plausible neuron models for computations. Unlike the real-valued inputs observed in ANNs, SNNs operate on discrete and sparse events in time called spikes. The spikes are binary with the information possibly encoded in the time, frequency, or phase of these events. In the case of SNNs, the input $x_1, x_2, x_3, \ldots x_N$ corresponding to each of the N neurons are time-dependent as shown in Figure 1.1. Unlike real-valued synaptic weights in ANNs, synapses in SNNs are modeled as a filter ($\alpha(t)$), which converts the incoming spikes to postsynaptic current waveforms. A real-valued synaptic strength (w) then weights this current. Here w_1 is the synaptic weight between the first input neuron and the output neuron, and so on. The final postsynaptic current, $I_{syn}(t)$ can be expressed as $I_{syn}(t) = c(t) \times w$, where $c(t)$ is the filter output corresponding to the input spikes. $c(t)$ can be obtained as,

$$c(t) = \sum_i \delta(t - t^i) * \alpha(t) \tag{1.1}$$

where t^i is the time corresponding to the i^{th} incoming spike and * is the convolution operator. Typically, $\alpha(t)$ can be a double-decaying exponential function or a low pass filter response [15]. The effective postsynaptic current from all the input neurons is then used by the output neuron to generate the spike. The SNN neuron model (indicated as f in Figure 1.1) can be the biologically plausible model developed by Hodgkin and Huxley [16], the computationally simple leaky integrate-and-fire (LIF) model [17], Izhikevich neuron model [18], adaptive exponential integrate-and-fire neuron model (aEIF) [19], etc. Neuronal dynamics in these SNN models are captured by n-order (n=1, 2, or 4) coupled differential equations with the additional constraint of generating an output spike ($y(t)$) whenever the neuron's membrane potential exceeds the threshold value. For example, the differential equations defining the membrane potential $V(t)$ of the LIF neuron model can be expressed as,

$$C\frac{dV(t)}{dt} = -g_L(V(t) - E_L) + I_{syn}(t) \quad \text{and } V(t) \leftarrow E_L, \text{ if } V(t) \geq V_T \tag{1.2}$$

where C is the membrane capacitance, g_L is the leak conductance, E_L is the resting potential of the neuron, $I_{syn}(t)$ is the effective synaptic input current, and V_T is the threshold voltage.

The event-driven nature of the neurons is expected to provide energy-efficient processing for SNNs. However, the computational capability of SNNs is not well established when compared with DNNs due to the lack of powerful learning algorithms. The conventional gradient-descent backpropagation algorithms cannot be applied for SNNs as the cost-functions are discontinuous and hence, non-differentiable. Another impeding factor for the development of SNNs is the training times involved when they are implemented in conventional von Neumann and graphics processing unit (GPU) architectures due to its temporal computing nature. We will review the underlying architecture of traditional von Neumann machines and GPUs in the following section.

1.2 Conventional Computing Architectures

Conventional computing systems are based on the von Neumann architecture, where the memory and processing units are physically separated, as shown in Figure 1.2. The processing unit typically constitutes of an arithmetic and logic unit (ALU), registers for storing the operands needed for ALU computations, cache (static random access memory (SRAM)), and a control unit that orchestrates and co-ordinates the sequential execution of a program. Central Processing Unit (CPU) performs at its best when the data is readily available in the system cache. However,

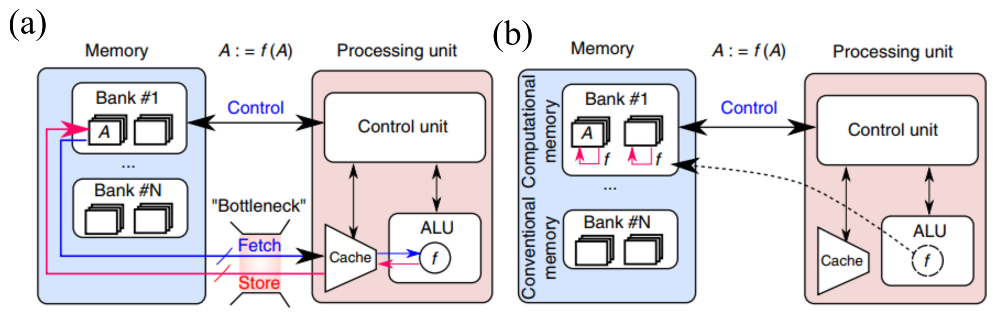


Figure 1.2 (a) Physically separated execution and memory units in a conventional computing machine. Data is fetched from memory for processing and the result is again stored back into the memory after execution. (b) An alternate architecture where the computation is performed within the memory.
Source: [20].

for data-intensive applications, the processing unit has to depend on the off-chip (external) memory as the on-chip cache is expensive and hence, limited in capacity. Thus data is continuously transferred from the external dynamic random access memory (DRAM) for execution in the CPU, and then the result is stored back into the memory. For machine learning applications that are typically data-intensive, the performance of these architectures gets limited by the memory access power, latency, and bandwidth due to the continuous data transfer to and from memory. This limitation imposed by memory is referred to as the "von Neumann bottleneck". CPU performance has been continuously improved over the years through aggressive technology scaling, pipelining, parallelism, memory hierarchies, etc. However, external off-chip memory does not scale as the CPU due to the difficulty in realizing reliable DRAM cells at lower nodes with acceptable access energy, latency, and cost [21]. While efforts are still on-going to improve the DRAM design, it must be noted that every DRAM access requires approximately 200$\times$ energy when compared to on-chip cache access [22]. There has been a growing interest in considering emerging memory technologies for future memory designs due to their low programming energy requirements. These emerging nanoscale devices like Phase Change Memory (PCM), Spin Transfer Torque Random Access Memory (STT-RAM), etc., are scalable, have acceptable latency and bandwidth compared to DRAM with little to no idle power consumption [23].

It is evident that energy-efficient computations are possible if the calculation is done either close or within the memory, and one such architecture is shown in Figure 1.2 (b) [20]. In addition to the conventional memory for storage, computational memory is proposed in Figure 1.2, which helps in co-locating memory and processing units similar to the human brain. Emerging memory devices have been extensively studied for energy-efficient in-memory computations of bio-inspired neural networks in several previous works [20, 24], and we use this idea for implementing DNNs later in the dissertation.

Considering the large number of parallel computations and high memory bandwidth required for machine learning workloads, graphics processing units (GPUs) are highly favored for implementing them. We will give a brief overview on the application-specific GPUs in the next subsection.

1.2.1 Graphics Processing Units (GPUs)

Graphics processing units (GPUs) have the potential to handle applications requiring enormous computational needs, parallelism, and higher throughput [25]. Historically GPUs were used for realistic video graphic displays as they can process thousands of pixels independently at the same time. GPUs act as slave devices to the CPU, and programs can be offloaded to GPUs for execution from the host CPU. GPUs offer a highly parallel architecture hosting multiple programming clusters, each containing several streaming multiprocessors (SM). SM constitutes thousands of registers, shared memory, warp schedulers for maximizing the resource utilization, and execution cores that support integer and floating-point operations. GPUs are optimized to do a large number of floating-point operations per Watt with ample resources providing higher memory bandwidth. Figure 1.3 gives a glimpse of the computational capability of

GPUs in terms of the number of floating-point operations per second per unit Watt (FLOPS per Watt) over the years, and their corresponding die sizes [26]. The best GPU performance indicated in Figure 1.3 is still behind the human brain by 7 orders of magnitude in terms of FLOPS per unit Watt [27]. Though GPUs can provide

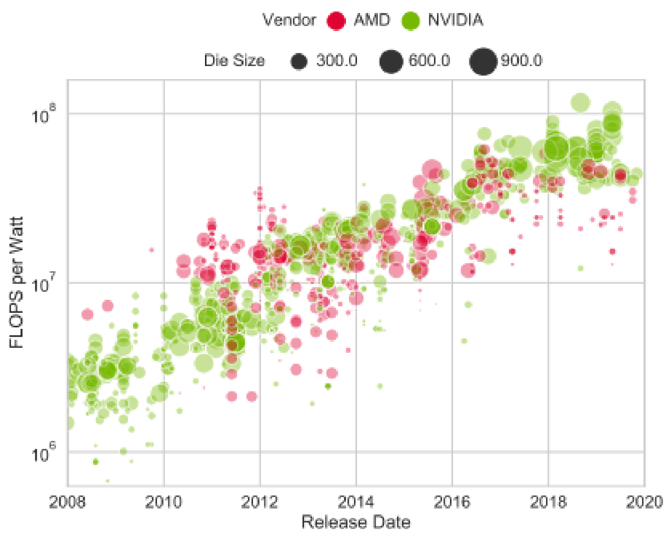


Figure 1.3 Trend of GPU compute power expressed in terms of floating point operations performed per Watt over the years. Die size of the GPU is also indicated. *Source:* [26].

high throughput for compute-intensive applications, they are far away from offering energy-efficient architectures like the human brain.

GPUs are favored for implementing machine learning workloads due to its ability in handling massively parallel computations and the large memory bandwidth offered. For example, Tesla GPU architecture from NVIDIA has dedicated Tensor Cores that are optimized to accelerate large matrix computations and supports mixed-precision mode for matrix multiply-accumulate operations [28]. One of the latest Tesla GPUs, Tesla V100 GPU, can provide a peak high bandwidth memory (HBM) of 900 GB/sec with the Tensor Cores delivering a maximum performance of 125 TFLOPS (Tera Floating-Point Operations Per Second) in mixed-precision mode [28]. However, the high throughput of the GPU based executions has the downside of implementation costs, especially power and area. For instance, the Tesla V100 GPU consumes a maximum power of 300 W in a die area of 815 mm^2 [28]. Thus GPUs are extremely power-hungry and are not feasible for machine learning applications having energy/area constraints.

Having discussed the conventional CPU and GPU architectures, we will briefly give an overview of dedicated hardware architectures that are optimized to handle the workloads of bio-inspired networks efficiently in the next section.

1.3 Overview of Dedicated Hardware for Bio-inspired Networks

1.3.1 Hardware Accelerators for DNNs

Deep Neural Networks, has been successful in surpassing humans in a wide range of tasks such as big data analytics, image recognition, natural language processing, and many others [29]. However, huge computational and memory resources are required for optimizing DNNs to achieve high accuracies. With ever-growing data and further processing needs, handling these workloads in conventional CPUs has performance impacts on throughput and power. Hence, hardware accelerators based on GPUs, ASICs (Application Specific Integrated Circuits), and FPGAs (Field Programmable Gate Arrays) have been considered for implementing DNNs.

Deep Learning typically has two phases - training and inference. Training is an iterative procedure involving large amounts of training data and tuning of millions of trainable parameters, which requires enormous memory and computational resources. Hence, training workloads are typically managed in the cloud using robust GPU architectures, Tensor Processing Units (TPU) [30], etc. Similar to GPUs, TPUs act as slave devices and are interfaced to the host CPU using the PCI (Peripheral Component Interconnect). TPUs are ASIC-based solutions originally introduced in 2015 by Google for accelerating neural network inference in data centers. The TPU architectures can provide a speedup of up to 30× and a power efficiency of up to 80×

when compared to GPUs. The heart of the TPU is a systolic array of 8-bit matrix multiply and add units [30]. TPU architectures have improved over the years with the latest Cloud TPUv3 capable of providing over 100 peta FLOPS computational capability, and around 32 TB HBM for accelerating neural network training [31].

The inference is the repeated execution of the trained network on unseen data. DNNs are nowadays ubiquitously adopted in end devices, from smartphones to IoT (Internet of Things) sensors. These devices generate vast amounts of data at the edge and expect real-time analysis or inference at low power and latency. Therefore, dedicated hardware accelerators for DNNs are necessary for edge devices that can meet the low power and low latency demands. Google's Edge TPU, Intel's Neural Compute Stick 2 (NCS2), NVIDIA's Jetson Nano, and NVIDIA's AGX Xavier are some of the popular ASIC-based hardware accelerators for inference which uses SRAM synapses and conventional digital CMOS logic for accelerating the computations [32–35].

Emerging nanoscale devices have also been considered for accelerating the DNN training as well as inference in a crossbar architectural framework. Several emerging nanoscale devices such as Phase Change Memory (PCM) [36–38], Resistive Random Access Memory (RRAM) [24, 39–42], Spin Transfer Torque Random Access Memory (STT-RAM) [43–47], etc., have been explored for implementing synapses in bio-inspired neural networks. Crossbar implementations with these memristive devices at the cross point inherently support weighted sum calculations and are shown to achieve $\mathcal{O}(1)$ time complexity for DNNs against the $\mathcal{O}(N^2)$ time complexity required by the generic computing machines, where N is the number of neurons in each layer [48, 49]. Implementations using PCM devices have been shown to achieve an acceleration factor of up to 25$\times$ and 3000$\times$ improvement in power compared to GPUs [50,51]. Also, there are several DNN hardware accelerator designs incorporating memristive synapses such as PUMA, PRIME, ISSAC, etc., proposed by the academic research community demonstrating improved latency and energy efficiency compared to GPUs [52–57]. These architectures usually have well defined specific instruction sets, dedicated compiler, and supports most of the machine learning workloads.

1.3.2 Neuromorphic Hardware for SNNs

As discussed in Section 1.2, the computational efficiency of the human brain in a power budget of 20 W inspires the need for brain-like architectures with efficient information processing capabilities. The term Neuromorphic computing, coined by Carver Mead in 1990 [58], mainly focuses on building brain-inspired architectures that offer high parallelism with low power/low latency execution, as well as co-locating memory and computational elements in small footprints [59]. With machine learning ubiquitously adopted for several day-to-day tasks, there is significant focus in building algorithms for SNNs that closely mimic the biological neurons due to its potential to demonstrate brain-like energy efficiency in hardware. The future implementations of such algorithms are dependent on these neuromorphic platforms for faster execution and further optimizations. Also, the temporal nature of the SNNs adversely slows-down the algorithmic optimizations in conventional computing architectures. Hence, it is necessary to have neuromorphic platforms that can accelerate large-scale SNN implementations at low power.

In an effort to build large-scale and energy-efficient neuromorphic computing systems similar to the human brain, several computing platforms have implemented SNNs. Intel's Loihi [60], fabricated in 14 nm process node, supports on-chip learning with 128 neuromorphic cores, and it integrates 130,000 neurons with 130 million synapses. TrueNorth is an inference engine from IBM built in 28 nm CMOS process node, which supports 1 million neurons and 256 million configurable synapses [61]. SpiNNaker [62] from the University of Manchester, BrainScaleS [63] from Heidelberg University, Neurogrid from Stanford [64], DYNAP from INI Zurich

[65], and Braindrop [66], are few other attempts to accelerate SNNs in hardware. The synaptic implementation of these architectures uses conventional SRAM-based designs. However, SRAMs are not the ideal choice for large scale implementation of neural networks in hardware. The six transistor (6T) SRAM designs typically take a minimum area of $120F^2$, where F is the smallest feature size possible in a technology node [67]. In this context, emerging nanoscale devices are considered as potential candidates for emulating the brain's connectivity in hardware owing to its small size ($4F^2$), ability to be packed in dense crossbar arrays, and low power programming requirements [68, 69]. For instance, an all-spin neuromorphic processor for SNNs, comprising of spintronic synapses and neurons with in-memory computing architecture, has shown 1000× energy efficiency and 200× speed up compared to CMOS [70].

Considering the potential of two-terminal nanoscale devices in offering energy-efficient solutions, this dissertation focuses on architectural and algorithmic optimizations for implementing the bio-inspired neural networks using these devices. In this dissertation, we initially focus on non-volatile memory (NVM) based implementations for accelerating DNN training in a crossbar architectural framework with memristive devices at the cross point [71,72]. Having discussed the DNN training, we then present a novel spiking neural network architecture for learning based on a probabilistic neuron model. We propose a hardware accelerator for performing the inference of probabilistic SNNs, which integrates binary spintronic synapses and digital CMOS logic for accelerating the computations. The performance evaluation of the proposed hardware for spike-based systems is estimated using standard simulation tools for digital CMOS logic as well as memory.

1.4 Organization of the Dissertation

This dissertation aims to cover the hardware implementation strategies for spike and non-spike based neural network systems using emerging nanoscale devices. The dissertation is organized as follows.

In Chapter 2, we review the spiking neural networks discussing the basic neuron and synapse models. We also discusses the motivation in building neuromorphic platforms and review the existing hardware platforms for SNNs.

Following the discussion on SNNs, the fundamentals of DNNs and the computational complexity involved in training DNNs are reviewed in Chapter 3. Chapter 3 also discusses stochastic DNNs and how it can be used to accelerate training in a crossbar implementation using memristive devices at the cross point. Chapter 4 details the implementation of a 4-layer stochastic DNN using PCMO devices in a crossbar architecture. The dependence of network performance on programming variability, the on-off ratio of the devices, and network resilience to noisy inputs is also explored in Chapter 4.

The probabilistic framework for SNNs based on the Generalized Linear Neuron Model (GLM) is discussed in Chapter 5. Probabilistic SNNs are then shown to achieve comparable performance with an equivalent ANN having the same architecture. The datasets used for benchmarking the SNN algorithm and the hardware-software co-optimization used for hardware implementation is discussed here. Finally, we discuss a hardware accelerator called SpinAPS (**Spin**tronic **A**ccelerator for **P**robablistic **S**piking Neural Networks) which uses binary spintronic synapses and digital CMOS for computations. We show that the inference accelerator, SpinAPS can achieve 4$\times$ improvement in terms of GSOPS/W/mm^2 compared to conventional SRAM-based synapses. Chapter 6 concludes the dissertation by presenting the future outlook of this work.

CHAPTER 2

INTRODUCTION TO SPIKING NEURAL NETWORKS

Spiking Neural Networks (SNNs), are third-generation neural networks, that closely mimic the information processing observed in the human brain [73]. SNN models can be efficiently used to solve complex cognitive problems and have higher computational capabilities than the conventional artificial neural networks [73, 74]. The artificial SNN models interact through sparse, asynchronous binary signals, and are considered as potential candidates for energy-efficient implementations of neural networks in hardware. This chapter discusses some of the artificial spiking neuron models, synapses, and gives an overview on the dedicated hardware architectures used for realizing SNNs.

This chapter is organized as follows. Section 2.1 describes the behavior and structure of biological neurons. Some of the commonly used neuron and synapse models are detailed in Sections 2.2 and 2.3, respectively. The basic idea of synaptic plasticity is described in Section 2.4. Section 2.5 covers the advantages and challenges of SNNs. We review some of the state-of-the-art neuromorphic hardware platforms in Section 2.6. Finally, the chapter is summarized in Section 2.7.

2.1 Biological Neuron

Neurons are the primary computational elements in the human brain and are responsible for generating action potential or discrete-time events called spikes. The fundamental structure of the neuron, as shown in Figure 2.1 consists of a cell body, dendrites, and an axon. Dendrites are nerve endings which carry input from other connected neurons to the cell body. At equilibrium, the neuron maintains a resting membrane potential usually between -55 mV, and -75 mV. This resting potential is maintained by the concentration of K^+ (Potassium) and Na^+ (Sodium) ions in the

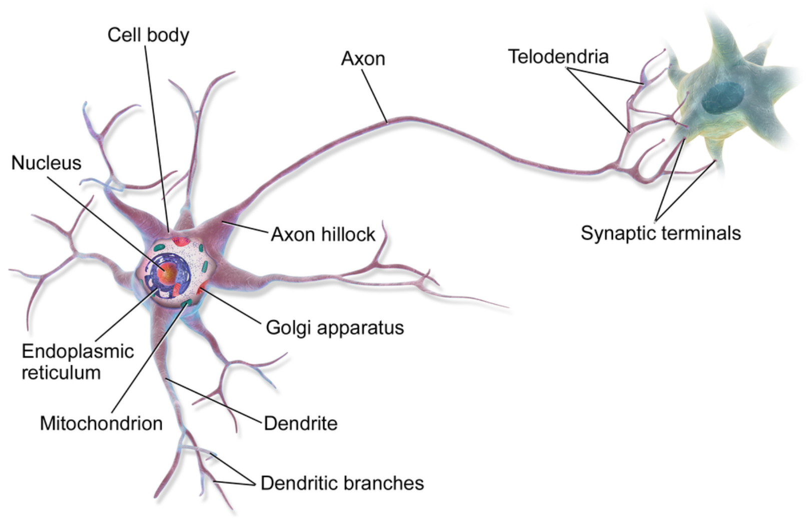


Figure 2.1 Basic structure of a neuron showing the cell body, axon and dendrites. *Source:* [75].

intracellular and extracellular fluids. At the resting state, the neuron's membrane is more permeable to K^+ than Na^+ ions, and the membrane is said to be polarized. On receiving sufficient external stimuli/inputs through dendrites, the membrane potential rapidly rises (depolarization) due to the influx of Na^+ ions [76]. The peak of depolarization is then followed by the inactivation of Na^+ channels and activation of K^+ channels, resulting in a significant efflux of K^+ ions. The outward movement of positive charge reduces the membrane potential, and this process is referred to as repolarization. Repolarization takes the membrane potential to levels more negative than the resting potential (hyperpolarization) for a short amount of time, and the cell finally returns to the resting state. This evolution of neuron's membrane potential to a sufficiently strong input stimulus is called an action potential. An action potential involves multiple phases, such as depolarization, repolarization, and hyperpolarization, as discussed above. It must be noted that the transitory changes in the activation and deactivation of Na^+ and K^+ channels during the depolarization and repolarization phases of the action potential make it harder for the neuron to produce subsequent action potentials irrespective of the input stimulus for a brief interval of time. This time is called the refractory period and limits the number of action potentials that can be produced by the neuron per unit time. The action potentials or impulses are carried along the axon to the synaptic terminals. The information encoded in the action potential of the presynaptic terminals can be transmitted to the postsynaptic neuron through a chemical or an electrical process. Chemical synapses are the most abundant synaptic transmission observed in the human nervous system. In chemical synaptic transmission, the presynaptic neuron releases chemical agents called neurotransmitters that produce current flow in postsynaptic neurons by activating specific receptor molecules [77]. There have been several attempts to model the neuronal and synaptic dynamics to understand the underlying complex mechanisms in the brain. A few of the most commonly used models for artificial spiking neurons and synapses are discussed in the subsequent sections.

2.2 Spiking Neuron Models

A complete and bio-physically accurate neuron model was developed by Hodgkin and Huxley in 1952 which described the neuronal dynamics using fourth-order coupled differential equations [16]. However, this model is computationally complex and not necessary for most of the engineering applications [14]. The coupled differential equations are complicated to solve analytically, and hence, they are solved through an iterative procedure of the numerical methods like Runga-Kutta [78]. Izhikevich, in 1973 reduced the original Hodgkin-Huxley equations to a two-dimensional system [18, 79]. Izhikevich model combines the biological plausibility of Hodgkin-Huxley neurons and computational efficiency of integrate-and-fire neurons. The second-order differential equations of Izhikevich neuron model shown in Equations (2.1) and (2.2) can simulate a rich set of firing patterns exhibited by real biological neurons, as

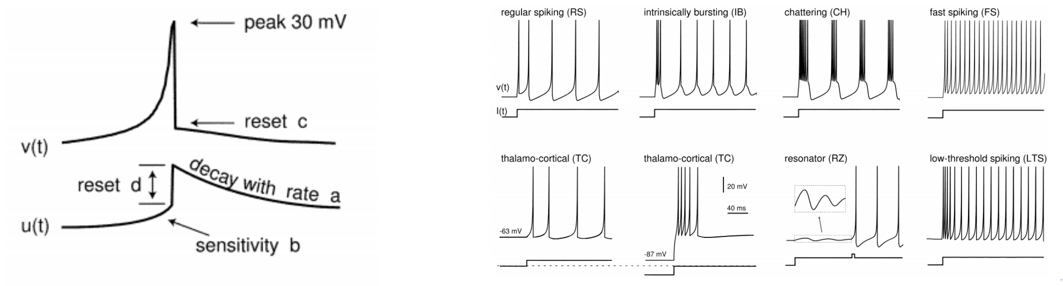


Figure 2.2 Tunable parameters a, b, c, d of the differential equations $v(t)$ and $u(t)$ of Izhikevich neuron model (left); Illustration of neuronal behaviors observed in cortical regions of brain generated using the Izhikevich model (right).
Source: [18].

illustrated by Figure 2.2.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.1}$$

$$u' = a(bv - u) \tag{2.2}$$

Here v is the membrane potential, and u is the recovery variable that accounts for activation of K^+ channels and inactivation of Na^+ ionic channels. The spike condition of the Izhikevich model is

$$\text{if } v \geq 30\text{mV then } v \leftarrow c, u \leftarrow u + d. \tag{2.3}$$

The simplest and computationally efficient model is the basic leaky integrate-and-fire (LIF) model. Though the model does not capture the biological mechanisms involved in the action potential generation, the LIF model is favored due to its low implementation costs [78]. In the LIF model, the neuron behavior is captured in a passive electrical circuit with cell membrane capacitance C and finite leak resistance R in parallel. When an external input current is applied, the membrane capacitance will be charged until it reaches the threshold value and then produces an action potential (spike) followed by a reset. The membrane potential $V(t)$ can be mathematically expressed as,

$$C\frac{dV(t)}{dt} = -g_L\left(V(t) - E_L\right) + I_{syn}(t) \tag{2.4}$$

where C is the membrane capacitance, g_L is the leak conductance, E_L is the resting potential of the neuron, and $I_{syn}(t)$ is the effective synaptic current. As explained in Section 2.1, biological neurons exhibit a refractory time period in which no new spikes are issued after the generation of the first spike. This behavior can be captured in the LIF neuron model by holding the membrane potential $V(t)$ to the resting potential E_L for a time equivalent to the refractory period t_f.

Adaptive exponential integrate-and-fire (aEIF) model, introduced by Brett and Gerstner is a modified form of the basic LIF model [19]. This model uses an additional exponential term in the membrane potential calculation to account for the early activation of voltage-gated channels. aEIF based neuron models have also been able to demonstrate several real neuronal spiking behaviors in the brain, e.g., adapting, bursting, delayed spike initiation, initial bursting, fast spiking, and regular spiking. The aEIF model can be mathematically expressed as,

$$C\frac{dV(t)}{dt} = -g_L(V(t) - E_L) + g_L\Delta_T \exp\left(\frac{V(t) - V_T}{\Delta_T}\right) - w + I_{syn}(t) \tag{2.5}$$

where g_L is the leak conductance, E_L is the resting potential, Δ_T is the slope factor which determines the sharpness of the threshold, V_T is the threshold, w is the adaptation current, and $V(t)$ is the membrane potential. The membrane potential is reset to the resting potential V_r when the potential $V(t)$ exceeds the threshold value V_T. The evolution of the adaptation current is modeled as,

$$\tau_w\frac{dw}{dt} = a(V(t) - E_L) - w. \tag{2.6}$$

Here a corresponds to the level of subthreshold adaptation, and τ_w is the adaptation time constant. The adaptation variable w is updated as $w \leftarrow w + b$, after a spike is issued.

2.2.1 Generalized Linear Models

Generalized Linear Models (GLMs) represent a new class of neuron models that are entirely tractable for computational and statistical analyses [80]. GLMs are closely related to integrate-and-fire models, in addition to the linear dynamics, they incorporate spike-dependent feedback which helps in capturing the non-linear dynamics following a spike generation [81]. GLM neuron models have been successful in mimicking the physiological readings from different regions in the human brain [80, 82, 83]. The GLM model shown in Figure 2.3 is characterized by a set of linear filters, a non-linear activation function, and a conditionally Poisson model is used for generating spikes.

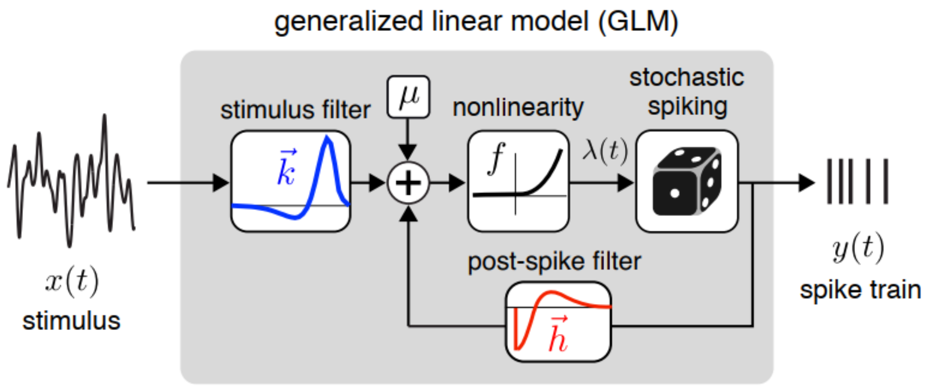


Figure 2.3 Generalized Linear Model (GLM): A generalized linear model in which a neuron's spiking rate, $y(t)$, is nonlinearly determined by a linear function of input stimulus and spike history. A wide variety of neuronal behaviors can be generated by adjusting the shape of the stimulus and feedback filters.
Source: [80].

In GLMs, a linear function of the input (stimulus) and spike history is non-linearly transformed to determine the spike response of the neuron, as shown in Figure 2.3. GLM is characterized by a stimulus filter $\overrightarrow{k}$, to integrate the input stimulus, and has a post spike filter $\overrightarrow{h}$, which captures the dependence of spike history in the probability of spikes and μ, determines the baseline spike rate. The summed output of all these filters is passed to a non-linear activation function f to determine the conditional intensity $\lambda(t)$. The conditional intensity $\lambda(t)$ governs the probability of a spike in the current time bin and can be mathematically expressed as,

$$\lambda(t) = f(\overrightarrow{k}.\overrightarrow{x}(t) + \overrightarrow{h}.\overrightarrow{y}_{hist}(t) + \mu) \tag{2.7}$$

where $\overrightarrow{x}(t)$ is the spatio-temporal input, $\overrightarrow{y}_{hist}(t)$ is the vector corresponding to the spike history at time t, and f is the non-linear exponential activation function which ensures a positive spike rate. The output spikes are then generated based on a conditionally Poisson process. The stochastic nature of GLMs enables it to closely mimic several neuronal behaviors, including single as well as multi spiking characteristics from different regions of the brain. Based on this probabilistic model, an algorithm for training a spiking neural network is derived for the problem of handwritten-digit recognition in [84]. A hardware accelerator for inference is proposed for this GLM-based spiking neural network and the corresponding design strategies are discussed in Chapter 5.

2.3 Synapse Models

Synapses are believed to be the key elements in determining the inter-connectivity of the neurons and hence, responsible for the communication pathways in the brain. Synapses are plastic in nature, which means the strength of the connections between the neurons changes wih respect to the activity patterns of the post and presynaptic neurons. Synapse connects the axon terminal of the presynaptic and dendrites of the postsynaptic neurons. These synapses control the flow of signals at the junction by releasing neurotransmitters. Neurotransmitters bind to the receptors of post synaptic neuron ensuring a current flow to the downstream neurons [14]. This behavior of biological neurons is essentially modeled as a synaptic weight which is a real number in first and second generation neural networks. In the case of spiking networks, synapse is usually modeled as a filter that converts input spike to a current weighted by the real valued synaptic strength. The filter response $\alpha(t)$ is modeled using single or double-decaying exponential functions, or a low pass filter. Some typical choices for the synaptic filter; $\alpha(t) = (e^{\left(\frac{-t}{\tau_1}\right)} - e^{\left(\frac{-t}{\tau_2}\right)})$, $\alpha(t) = (\frac{t}{\tau})e^{\frac{-t}{\tau}}$ and $\alpha(t) = \tau\delta(t)$. The synaptic current $I_{syn}(t)$ flowing to the post-synaptic neuron can be expressed as,

$$c(t) = \sum_i \delta(t - t_i) * \alpha(t) \tag{2.8}$$

$$I_{syn}(t) = c(t) \times w \tag{2.9}$$

where $c(t)$ is the filter output obtained by the convolution operation of the filter $\alpha(t)$ and the input spikes, and w is the synaptic strength. Here t_i is the time corresponding to the i^{th} input spike. The synaptic strength w undergoes modulation based on the spike patterns in the pre- and postsynaptic neurons and the biologically inspired learning rule for synapses is discussed in the next section.

2.4 Synaptic Plasticity

Electrical impulses or action potentials are transmitted through synapses, which is the junction/gap between two nerve cells. Several physiological studies show that the strength of the synapses is modulated by the activity patterns in the upstream and downstream neurons. Synaptic changes could last for a few seconds called short term plasticity, or it could last longer, resulting in long term plasticity [85]. A well-defined postulate for synaptic modulation was put forward by Hebb, according to which the synapses can be either strengthened or weakened depending on the correlated spiking events of pre- and postsynaptic neurons [86]. However, Hebb's rule doesn't take care of bounding the weights under the persistent firing of pre- and postsynaptic neurons. The Spike-timing-dependent-plasticity (STDP) rule addresses this issue by

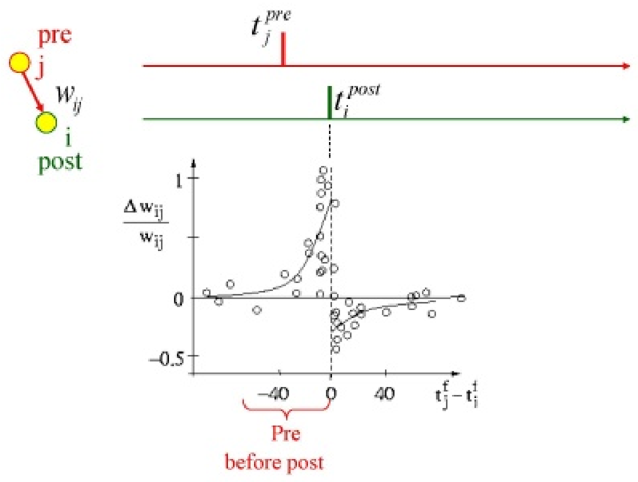


Figure 2.4 Causal firing of pre- and postsynaptic neurons increases the synaptic conductivity and an anti-causal firing would weaken the synaptic strength between the neurons.
Source: [87,88].

the precise timing of the pre- and postsynaptic spikes. An illustration of the basic STDP protocol plotted as a function of synaptic weight change and the relative timing of pre- and postsynaptic action potentials are shown in Figure 2.4. Here, w_{ij} is the synaptic strength between the presynaptic neuron j, and the postsynaptic neuron i, t_j^f corresponds to the presynaptic spike arrival times, and t_i^f represents the postsynaptic spike firing times. The total synaptic change Δw_{ij} can then be expressed as,

$$\Delta w_{ij} = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_i^n - t_j^f) \tag{2.10}$$

where W is the STDP function or the learning window [89, 90]. One of the typical choices of W is

$$W(s) = A_+ \exp(-s/\tau_+) \quad \text{for } s \geq 0 \tag{2.11}$$

$$W(s) = -A_- \exp(s/\tau_-) \quad \text{for } s < 0 \tag{2.12}$$

where $s = t_i^{post} - t_j^{pre}$. Here, parameters (A_+, A_-) depends on the current synaptic weight w_{ij}, and the rate constants (τ_+, τ_-) are typically on the order of 10 ms. The causal firing of the pre- and postsynaptic neurons increases the synaptic conductivity, and the anti causal firing weakens the synaptic strength. The biologically inspired STDP rule has been widely used for unsupervised training of SNNs [91].

In addition to the biologically inspired local learning rules like STDP, there are several supervised training rules which illustrated learning in SNNs [15, 92]. The discontinuous and hence, the non-differentiable nature of spikes makes it non-trivial to use the conventional backpropagation based training algorithms for multi-layered SNNs. Besides the STDP based approaches, methods like variations of error backpropagation [93], and converting the conventionally trained DNNs to SNNs are also used to realize deep SNN models [94]. Other approaches like training the DNNs by incorporating the constraints of spiking neurons [95], and using binary activations or spikes are also explored in several of the previous works [96].

Having discussed the commonly used spiking neural network models, synapses, and synaptic plasticity, we will now outline the advantages and challenges associated with cognitive demonstrations using SNNs.

2.5 Advantages and Challenges with SNNs

One of the biggest motivations behind SNN research is its energy efficiency in performing complex cognitive tasks in the human brain at a power consumption of just 20 W. It is believed that as the artificial neuron models stay closer to biology, computational capability and energy efficiency will improve significantly compared to the current computing architectures. Driven by the fact that the information from the outside world is sparse, biologically inspired SNNs could pave the way for energy-efficient computing in the future. The temporal, sparse and event-driven computing nature of SNNs makes it an ideal candidate for efficient information processing [97–100]. SNNs compute on an event-driven manner, that is, computing occurs if sudden bursts of activity (more spikes/information) are recorded at the input, and not much computations are performed with little or fewer input information. SNNs also have an additional advantage of local processing, which means the computation in any network layer can be initiated when it receives sufficient activity from the previous layer. In contrast, computations in DNNs cannot be completed until all the input sequences are presented. SNNs compute based on the spike activity, and schemes like first-to-spike decoding rule can be used to make output decisions with low latencies, and lower memory accesses [84].

Even though SNNs have the potential of building computationally powerful and power-efficient algorithms, their performance on standard benchmark datasets like MNIST (Modified National Institute of Standards and Technology), ImageNet [101], and CIFAR (Canadian Institute For Advanced Research) [102] are inferior to conventional DNNs [103]. The performance deficit of SNNs against standard DNNs is attributed to the lack of powerful learning algorithms similar to gradient-based backpropagation in DNNs. The discontinuous nature of spikes makes it non-trivial to derive algorithms like backpropagation in SNNs. In addition, the inherent temporal nature of SNNs slows-down the optimization of algorithms in von Neumann architectures. Hence, neuromorphic hardware accelerator platforms are necessary to exploit the potential of SNNs completely. Another critical factor in SNN research is the lack of time-series benchmark datasets. Currently, SNNs are benchmarked against the standard DNN image classification and object detection datasets, which uses static image frames as inputs. The input images from these datasets have to be converted to spike trains before feeding to SNNs, and usually, rate based conversion schemes are used. In rate encoding scheme, the spikes are generated using a Poisson process with the firing rate proportional to the pixel intensity. This method is inefficient and would degrade the performance of SNNs. Therefore, time-series benchmark datasets and efficient signal encoding schemes are necessary for SNNs to validate its performance in solving complex cognitive problems.

There have been significant efforts from the neuromorphic research community to demonstrate the power efficiency and computational ability of SNNs in hardware [104, 105]. We will review some of the hardware platforms and their performance attributes in the next section.

2.6 Neuromorphic Hardware for SNNs

To emulate the brain's power efficiency and to demonstrate the potential of SNNs, several hardware architectures based on state-of-the-art Si CMOS have been illustrated and reviewed in several previous works [104–107]. Here we will give a brief overview of some of the popular hardware architectures used for realizing SNNs.

SpiNNaker from the University of Manchester, a million core microprocessor architecture, fabricated in 130 nm can support up to 1000 neurons per core. The system is designed to simulate large SNNs with brain-like complexity [62]. SpiNNaker2, fabricated in 22 nm a successor of the original chip, is extended to host 10 million microprocessors supporting both spiking and deep neural networks in the same platform [108].

BrainScaleS is a mixed-analog-digital hardware using the wafer-scale integration technology [63]. It uses analog circuits for implementing the neurons and synapses, while communication is done digitally. The basic building unit of the BrainScaleS system is an analog neural network core (ANNC) that combines all the analog circuitry for neurons and synapses. These ANNC cores called High Input Count Analog Neural Network (HiCANN) forms the building block of the wafer-scale system. While BrainScaleS is fabricated in a 180 nm process, the second prototype BrainScaleS-2 is expected to use a 65 nm process with support for dendritic computations and hybrid plasticity [109].

TrueNorth from IBM is a million spiking neuron integrated CMOS inference chip fabricated in 28 nm process technology [61]. TrueNorth has a tiled array of interconnected 4096 neurosynaptic cores, which can map 1 million spiking neurons and 256 million configurable synapses. Each input neuron in the neurosynaptic core inherently supports a fan-out of 256, and they can also be configured to communicate to neurons in other cores. TrueNorth chip can deliver 46 billion synaptic operations per second (SOPS) in real-time with one synaptic operation corresponding to a multiply-accumulate operation upon an input spike arrival. TrueNorth implements an extension of integrate-and-fire neurons, and they are digitally time-multiplexed to amortize the area and power. With just 26 pJ per synaptic event, TrueNorth has been successfully demonstrated its potential in several object detection, gesture recognition, and image classification problems [110].

Loihi is a neuromorphic chip from Intel, fabricated in 14 nm FinFET process with a die size of 60 mm^2 [60]. The Loihi chip has 128 neurosynaptic cores, with each core mapping up to 1024 spiking neurons and 2 Mb of local SRAM. Loihi can perform 30 billion synaptic operations per second (SOPS), with an energy consumption of 15 pJ per synaptic operation [111]. The chip supports several learning rules like pairwise STDP, triplet STDP, and reinforcement learning.

The Tianjic chip is a hybrid architecture, which integrates the support for spiking and non-spiking (artificial neural networks) neural networks in the same platform [112]. Each core of the Tianjic chip is termed as a unified functional core (FCore), which combines the functional elements like axon, synapse, dendrite, and soma. The chip uses a many-core architecture arranged in a two-dimensional mesh with flexible fan-in and fan-out capability. The chip comprises 156 FCores, with approximately $40,000$ neurons and 10 million synapses. Fabricated in 28 nm CMOS process, the chip has a distributed SRAM memory and a total die area of 14.44 mm^2. The chip is reported to provide a peak performance of 1.28 trillion operations per second (TOPS) per Watt for ANNs when running at a clock frequency of 300 MHz. For SNNs, the chip is observed to achieve a peak performance of 650 billion synaptic operations per second (SOPS) per Watt. The chip demonstrated several state-of-the-art DNN and SNN models by programming it with pre-trained models exhibiting a parallel and seamless on-chip communication.

Dynamic neuromorphic asynchronous processors (DYNAP) are a family of neuromorphic processor chips from INI, Zurich [65]. The multicore architecture of these chips uses subthreshold analog circuits for neuronal computations and asynchronous digital logic for communication. DYNAP-SE is one among the chips in the DYNAP family, fabricated in 180 nm CMOS process having four neuromorphic cores. The neuronal dynamics in DYNAP-SE uses adaptive exponential integrate-and-fire (aEIF) and combines 256 such neurons in each core. Subthreshold analog circuits are susceptible to variability; however, this variability is exploited to implement neural sampling and reservoir computing in DYNAP-SE.

Neurogrid, from Stanford University, is a mixed-signal chip intended to simulate large scale neural networks in real-time [64]. Neurogrid has been demonstrated to simulate a system of 1 million neurons, and a billion synapses in real-time with a power consumption of 3 Watts. The neuronal and synaptic dynamics are realized using the subthreshold regime of the MOS devices in the core. Neurogrid integrates 16 NeuroCores with each core hosting 256×256 neurons, and the cores are connected using a tree network. Braindrop [66] is the second neuromorphic chip from Stanford University, which uses a mixed-analog-digital design like Neurogrid. However, braindrop has an added feature of high-level programming abstraction, which does not require hardware expertise up to the neurosynaptic level. Braindrop is estimated to consume just 0.38 pJ per synaptic operation for typical network configurations.

ODIN is a digital ASIC fabricated in 28 nm CMOS process, and supports online learning with a minimum energy per synaptic operation of 12.7 pJ [113]. The neuron implementation in ODIN supports both LIF and Izhikevich behaviors. The online learning implementation is based on spike-driven synaptic plasticity (SDSP).

Darwin Neural Processing Unit (NPU) from Zhejiang University and Hangzhou Dianzi University in China is a system-on-chip (SoC) solution mainly targeted for embedded applications using SNNs [114]. Darwin NPU is a hardware co-processor and uses a digital logic implementation for leaky integrate-and-fire (LIF) neurons. Fabricated in 180 nm CMOS process, NPU supports 8 physical neurons on the chip, and digital time multiplexing is used to implement a maximum of 2048 neurons. It consumes 0.84 mW/MHz at 1.8 V power supply for typical applications like handwritten recognition, electroencephalogram (EEG) decoding, etc.

As discussed, various neuromorphic platforms are hoping to demonstrate the energy efficiency of SNNs for complex cognitive problems in hardware. These hardware platforms are usually benchmarked against the number of synaptic operations performed per second (SOPS) per unit area, and on the energy required per synaptic activity. The main target of these platforms as well as its successor chips is to achieve maximum throughput at lower energy and area costs, definitely a sweet spot for SNN hardware.

2.7 Summary

In this chapter, we reviewed the basic working of a biological neuron and discussed the standard artificial spiking neuron models and synapses. We also described a probabilistic framework for SNNs using the Generalized Linear Model (GLM) neurons and discussed its basic functionality. We briefly presented the biologically inspired learning rule for SNNs and outlined the advantages as well challenges faced by the third generation SNNs in comparison with conventional DNNs. Finally, some of the dedicated hardware platforms that support SNN implementations are reviewed.

In the next chapter, we will give a review on the deep neural network models and discusses the basic steps involved in DNN training. This chapter also presents possible computing schemes of accelerating the DNN computations using nanoscale devices in a crossbar architecture with memristive devices at the cross point.

CHAPTER 3

DEEP NEURAL NETWORKS: TRAINING AND ACCELERATION

3.1 Introduction

Deep Neural Networks (DNNs), also popularly referred to as Deep Learning, has influenced the day-to-day activities of humans through its super-human performance in big data analytics, image recognition, object detection, gaming, and many more [115–117]. DNNs are part of the broad field of Artificial Intelligence (AI), which hopes to provide computers an ability to perform tasks without any explicit programming. The origin of the widely accepted second-generation Artificial or Deep Neural Networks can be traced back to the early 1940s. The perceptron neuron model developed by Rosenblatt in 1958 demonstrated neuron's ability to solve linearly separable problems [11]. With the development of the backpropagation algorithm in 1974 for training multi-layered networks, DNNs became successful in solving complex cognitive tasks [12]. Deeper network models have the advantage of generating high-level features with complex abstractions than what is possible with shallow networks. The massive success of DNNs is mainly attributed to the availability of vast amounts of training data, high-performance computing resources, and new algorithmic developments [117].

Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are the two most popular classes of DNNs. CNNs are widely used for computer vision applications, while RNNs are used for processing time-series data as in speech recognition, text classification, etc [118]. CNNs, inspired by the functional architecture of visual processing observed in the neocortex of animals, is one of the widely used networks for object classification today [119]. Deep CNNs showed promising results in the late 1980s; however, it maintained a dormant state till the 2000s due to the lack of enough labeled data, better algorithms, and powerful computational resources [120, 121]. Deep CNNs saw significant advances in 2012, whose work won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with remarkable network performance [122]. Since then, deep CNNs have become the underlying architecture for all the visual recognition problems. Several deep CNN models have shown exceptional performance over the years on the ImageNet benchmark (shown in Figure 3.1). The ImageNet dataset is a standard benchmark for visual object recognition problems and has around 1 million images in 1000 object categories [101].

The performance of some of the popular deep CNN models is summarized in Figure 3.1 in terms of the ratio of the top-5 percentage accuracy on the ImageNet classes and the number of trainable parameters in millions. The deep CNN models in Figure 3.1 are selected in such a way that the top-5 percentage accuracy has an increasing trend over the years, but the number of trainable network parameters would differ substantially. Over the years, CNNs have undergone significant architectural and algorithmic improvements for achieving a remarkable performance on the ImageNet dataset. With performance improvement, deep CNNs also have to pay for excessive computations in terms of multiply-accumulate (MAC) operations and an increase in the number of trainable parameters. Deep CNNs also presented yet another difficult problem of vanishing gradients, which affected the convergence and accuracy compared to relatively shallower networks. The introduction of residual connections with ResNet-50 from Microsoft resolved the issue of vanishing gradients and became the winner of ILSVRC in 2015 with an error rate beating the humans [123]. Inception series of network models, initially introduced by Google in 2015, incorporated techniques like factorized convolutions to reduce the number of parameters without impacting the network performance [124,125]. Xception combines the advantages of residual connections and depth-wise convolutions to achieve an

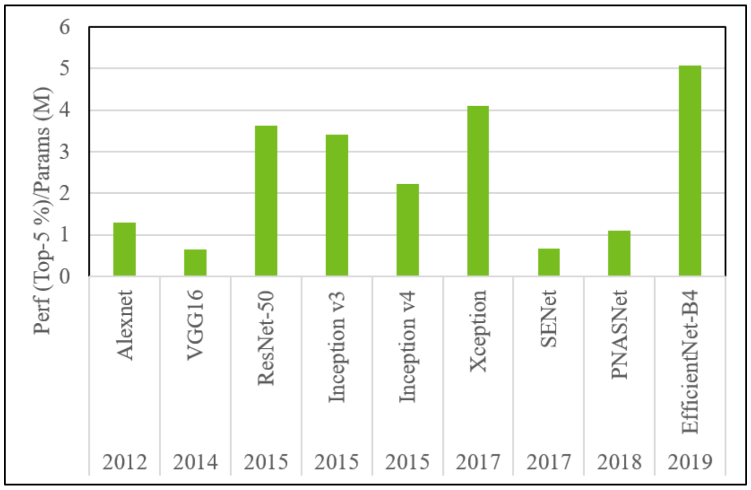


Figure 3.1 Performance of some of the popular deep CNNs expressed in terms of ratio of the top-5% accuracy to the number of trainable parameters in millions. EfficientNet is reported to have the best error rate of 2.9% with minimum number of parameters on the ImageNet database at the time of writing this thesis.

error rate of 5.7% on the ImageNet database [126]. Squeeze-and-Excitation Networks (SENet), attempts to recalibrate channel-wise features by explicitly modeling the dependencies between the channels and achieves a top-5 error rate of 3.8% on ImageNet [127]. Progressive Neural Architecture Search Networks (PNASNet) rely on automatically figuring out the CNN architecture, which achieves the best performance on the selected dataset through novel network search algorithms [128]. EfficientNet models are the latest deep CNNs that achieved the best error rate of 2.9% on ImageNet with fewer parameters compared to the other state-of-the-art models. The base model EfficientNet-B0 has a similar performance to ResNet-50 on ImageNet database, and it requires just 5× fewer parameters and 11× fewer computations than the later [115]. EfficientNet details a principled method of scaling the convnet parameters like depth, width, and resolution for obtaining better performance [115]. The different scaled versions of EfficientNet are available ranging from EfficientNet-B0 to EfficientNet-B7.

At present, deep CNNs are highly successful and form the primary architectural component in computer vision applications like object tracking, pose estimation, text detection, scene labeling, etc [129–131]. The high-performance of these networks comes with the downside of enormous memory and computational requirements. Among the two phases of DNN implementation, training involves an iterative procedure of updating millions of trainable network parameters using massive datasets. Hence, training is typically managed in powerful cloud GPU servers, cloud TPUs, etc., as they have ample computational resources and memory bandwidth. For example, Figure 3.2 shows the training times of ResNet-50 model using GPUs back in 2015 and the acceleration in training observed with GPU architectures over the years. The latest DGX SuperPOD, an AI platform from NVIDIA, can execute the training for ResNet-50 in just 80 secs, however, with a whopping power consumption of 1 MW [132,133]. DGX SuperPOD is a supercomputer optimized for high-performance computing and hosts around 1500 VOLTA GPUs for deep learning acceleration [133].

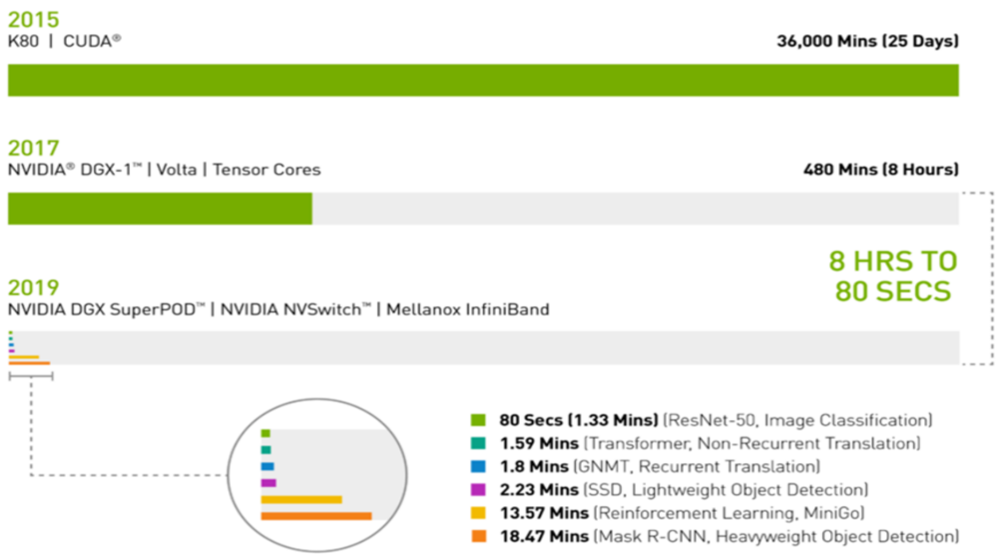


Figure 3.2 NVIDIA's AI platform accelerated the training time from 8 hours to 8 secs for ResNet-50 in the image classification task.
Source: [132].

GPUs are evidently power-hungry, and hence, novel architectural solutions are necessary for ubiquitous adoption of DNNs in day-to-day tasks. Memristive implementation of DNNs in a crossbar architectural framework has been estimated to offer energy/power-efficient designs for DNNs as well as CNNs [24, 51, 134]. These implementations have been estimated to provide 25× speedup and 3000× improvement in power compared to GPU based implementations [51, 135]. Considering the energy efficiency and acceleration offered by the crossbar architectures, we focus on implementing DNN training on these architectures using experimental memristive devices at the cross point. Recently, IBM proposed a scheme for accelerating DNN training with a time complexity of $\mathcal{O}(1)$ using stochastic computing with memristive devices like RPU (Resistive Processing Units) at the cross point [48, 49]. We use the principle of stochastic computing to realize a 4-layer DNN and the study described in this chapter is published in the Neurocomputing journal [71].

This chapter is organized as follows. Section 3.2 reviews the basic operations involved in training DNNs and their computational time complexity. Hardware acceleration of DNN training in a crossbar architecture using RPUs at the crosspoint is introduced in Section 3.3. The concept of stochastic computing is discussed in detail in Section 3.4 and its realization in a 4-layer DNN is considered in Section 3.5. Finally, the chapter is concluded in Section 3.6.

3.2 DNN Training

DNN training is done in two steps - forward pass and a backward pass. In forward pass, the output activations of all the neurons are calculated to determine the cost function and in the backward pass, network weights get updated with the goal of minimizing the cost function. The output of any neuron in layer $(l+1)$ is a non-linear function of the weighted sum of input from layer l and the synaptic weights between these two layers, given by

$$y_j^{(l+1)} = f\left(\sum_{i=1}^{N} w_{ij}^{(l)} x_i^{(l)}\right) \tag{3.1}$$

where f is a non-linear activation function and can be $\sigma(x)$, $\sinh(x)$, $\tanh(x)$ or $\text{ReLU}(x) = xH(x)$, with $H(x)$ denoting the Heaviside step function. The forward pass shown in Equation (3.1) has a time complexity of $\mathcal{O}(N^2)$, where N is the number of neurons in each layer. In backward pass, the weight update required for any layer (l) depends on the error in the succeeding layer $(l+1)$, given by

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} \pm \eta x_i^{(l)} \delta_j^{(l+1)} \tag{3.2}$$

where $\delta_j^{(l+1)}$ is the error in the succeeding layer $(l+1)$ and in turn depends on the synaptic weights in the layer $(l+1)$ and the error in layer $(l+2)$ as shown by Equation (3.3).

$$\delta_j^{(l+1)} \propto \sum_{k=1}^{N} w_{jk}^{(l+1)} \delta_k^{(l+2)} \tag{3.3}$$

The time complexity involved in backward pass with reference to Equations (3.3), (3.2) is $\mathcal{O}(N^2)$. Therefore, network training for data intensive applications such as audio and video analysis could take several days or months even when implemented in parallel architectures such as GPUs [136]. In order to accelerate DNN training, the forward pass, backward pass and the weight update have to be executed in parallel. Many of the previous works demonstrated the forward pass and the first step of backward pass shown in Equations (3.1) and (3.2) using crossbar arrays with memristive devices at the cross point [137–140]. Hence, the challenge is to achieve parallel weight update using memristive devices for accelerating DNN training.

3.3 Parallel Weight Update Using RPU Devices

Recently a scheme has been processed for implementing parallel weight update using resistive processing units (RPUs), with the idea of stochastic weight updates [48]. In the stochastic weight update scheme, stochastic pulses of constant amplitude and opposite polarity representing x_i and δ_j is presented across the rows and columns respectively. The pulsing scheme is designed in such a way that when there is no overlap between the two pulses, the device conductance remains unperturbed. On the overlap of these two pulses, the device conductance changes by an integer multiple of minimum allowed conductance change ($\Delta \mathrm{w}_{min}$) per coincident pulse pair. Therefore, DNN training can be implemented with $\mathcal{O}(1)$ time complexity using RPU devices at the crossbar as the stochastic weight update can happen in parallel for all the synapses. Based on simulations, it has been projected that the proposed RPU device must satisfy the stringent requirement of 1000 levels and at least a dynamic range of 10.

3.4 Stochastic Computing

Stochastic computing was introduced in the late 1960s [141,142]. Using the stochastic framework for computation, a number $X \in$ [0,1] can be represented as a Bernoulli sequence X_1, X_2, $X_3 ... X_N$ such that the random variable X has a probability of $P(X_i = 1) = X$, and N is the length of the Bernoulli sequence [143]. For example in the stochastic framework, $X = \frac{6}{8}$ can be represented as 01011111. Consider two scalar quantities a, b appropriately scaled to lie in the range [0,1]. Let **A** and **B** are two uncorrelated N bit long sequences representing a and b such that $P(A_i = 1) = a$, $P(B_i = 1) = b$. Let C represent the bit wise logical AND operation of sequences **A** and **B**. Therefore,

$$P(C_i = 1) = P(A_i = 1)P(B_i = 1) = ab \tag{3.4}$$

$$P(C_i = 0) = 1 - P(A_i = 1)P(B_i = 1) = 1 - ab \tag{3.5}$$

The expectation and variance of the binary random variable C_i is $\mathrm{E}(C_i) = ab$, $\mathrm{Var}(C_i) = ab(1-ab)$. Let the number represented by the Bernoulli sequence $\mathcal{C} = C_1, C_2, ...C_N$

be obtained by averaging the N independent random variables C_i,

$$C = \frac{1}{N}\sum_{i=1}^{N} C_i \tag{3.6}$$

$$\implies E(C) = ab \ , \ Var(C) = \frac{ab(1-ab)}{N} \tag{3.7}$$

As the length of Bernoulli sequence increases, the estimated average approaches the scalar product of $a \times b$. Therefore, a key advantage of stochastic framework is that multiplication can be implemented using simple coincidence detection [144–146]. Moreover, stochastic arithmetic can be implemented by computational elements which are very small and compatible with VLSI design technology. Stochastic computation is suitable for applications where certain level of inexactness can be tolerated such as in image processing [147], error correcting codes [148] as well as in artificial neural networks [149, 150].

In order to achieve $\mathcal{O}(1)$ time complexity for DNN training, the proposed ideal RPU device requires 1000 levels which is difficult to meet experimentally. However, the experimental memristive devices are highly non-ideal with limited dynamic range, finite conductance resolution and has significant programming variability. Therefore, here we study a 4-layer stochastic DNN using memristive devices, whose synaptic behavior is derived from the experimentally observed electrical behavior of fabricated $Pr_{0.7}Ca_{0.3}MnO_3$ devices. We then analyze the performance of stochastic DNNs to various device parameters such as - on-off ratio, programming variability and tolerance to noisy input for inference.

3.5 Stochastic DNNs

3.5.1 Network Architecture

A basic 4-layer deep network is used as the baseline for studying stochastic DNNs for the handwritten-digit classification benchmark. The network architecture of the 4-layer deep network is shown in Figure 3.3 and is trained with a standard database for handwritten digits - MNIST (Modified National Institute of Standards and Technology) with 50,000 digits for training, 10,000 digits for validation from the training set. The performance of the network is tested on the unseen set of 10,000

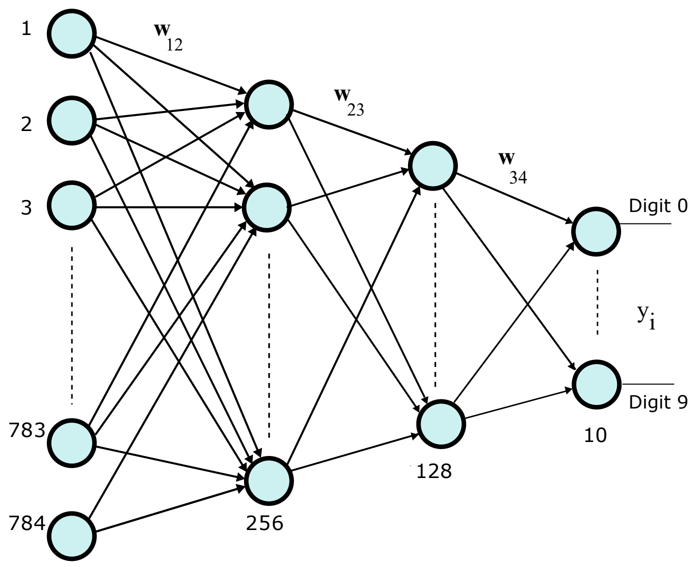


Figure 3.3 A 4-layer deep neural network with 784-256-128-10 neurons in each layer used for hand-written digit classification (Simulated using MATLAB).
Source: [68].

digits in the test set. Each of the input image is of size 28×28 and hence, 784 input neurons are used in the input layer. The network uses sigmoid activation function for the hidden layer neurons and softmax function for the output layer. Network training is done with an aim to minimize the multi-class cross entropy objective function and the weights are updated after every image is presented. The entire representation of 50,000 images constitutes one epoch and the network is trained for over 30 epochs.

3.5.2 Floating Point DNN

The 4-layer DNN which uses floating point precision during training is taken as the baseline network and is referred to as the 'floating DNN'. The corresponding error in the training and test set as a function of epochs is shown in Figure 3.4. The network reaches a maximum test accuracy of 98% after 30 epochs and is used as the baseline to determine the number of bits required for the stochastic DNNs.

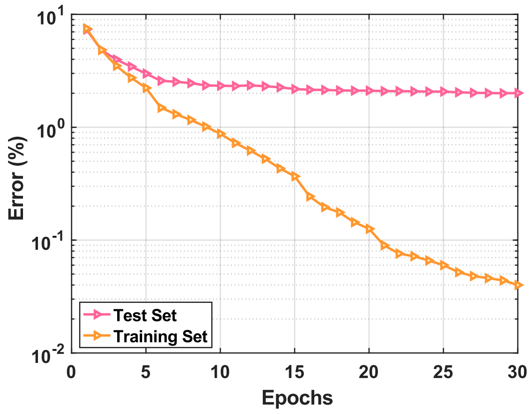


Figure 3.4 Training and test error for floating point DNN with weight update using Equation (3.2). Training and test error decreases with epoch and reaches a maximum test accuracy of 98%. This network is used as the baseline to determine the bit length required for stochastic DNNs.
Source: [71].

3.5.3 Stochastic DNN

The network which uses stochastic pulses for forward pass, backward pass and the weight update is referred to as the 'stochastic DNN'. In forward pass, the real valued input to each layer is converted to a stochastic pulse stream of length BL to compute the neuronal output. In the first step of backward pass, (shown in Equation (3.3)) the error in the layer $(l+1)$ is calculated by sending stochastic pulses corresponding to the real valued error in layer $(l+2)$. In order to perform the weight update for any layer l, the error at layer l and input to the same layer x_i has to be multiplied as shown in Equation (3.2). In the stochastic framework, if x_i, $\delta_j \in$ [0,1] is represented by stochastic pulses of length BL, then the multiplication of x_i and δ_j can be implemented by coincidence detection of the two sequences as explained in section 3.4

$$w_{ij}^{(l)} = w_{ij}^{(l)} \pm B \left(\sum_{n=1}^{BL} x_{i,n}^{(l)} \wedge \delta_{j,n}^{(l+1)} \right) \tag{3.8}$$

Hence, the weight update rule can be modified as shown in Equation (3.8), where BL is the length of the stochastic pulse sequence that can be used to approximately represent the Bernoulli sequences, B is the bin-width or the minimum possible

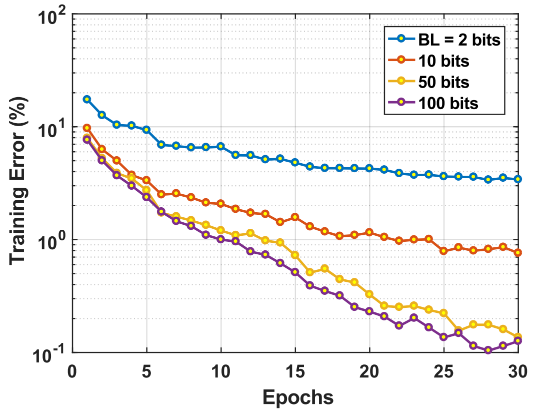


Figure 3.5 Training and test error for floating point DNN with weight update using Equation (3.2). Training and test error decreases with epoch and reaches a maximum test accuracy of 98%. This network is used as the baseline to determine the bit length required for stochastic DNNs.
Source: [71].

conductance change corresponding to one coincidence event of $\delta_{j,n}^{(l+1)}$ and $x_{i,n}^{(l)}$. Therefore, the overall conductance changes by integer multiple of the number of coincidences of the two stochastic pulse streams.

In order to determine the length of the stochastic bit sequence BL, we converted the baseline floating point network to a stochastic network which uses stochastic pulses for forward and backward pass. We used BL of 2, 10, 50 and 100 for training the 4-layer network and the corresponding training error is shown in Figure 3.5. The test accuracies of the trained network after 30 epochs of training is shown in the Table **??**. In order to have a fair comparison of stochastic DNN with floating point DNN, the learning rates for both the networks are kept the same. As the length of the stochastic pulse stream BL increases, the test accuracy increases and becomes closer to the floating point baseline network. The Table **??** shows the average test accuracy over five different MATLAB iterations. As there is only marginal improvement in test accuracy when BL is changed from 10 to 100, $BL = 10$ is used for the rest of the study.

Table 3.1 Maximum Test Accuracy of Floating Point and Stochastic DNNs for Hand-written Digit Classification

Network	BL	Test Accuracy
Stochastic	2	95.95%
Stochastic	10	97.74%
Stochastic	50	98.09%
Stochastic	100	98.14%
Floating Point		98%

3.6 Summary

In this chapter, we reviewed some of the state-of-the-art deep CNNs and the basic steps in training DNNs using the backpropagation algorithm. It can be noticed that the different steps of the backpropagation algorithm for training DNNs has a time complexity of $\mathcal{O}(N^2)$, where N is the number of neurons in any layer. We also show that the principle of stochastic computing offers a way to implement the costly multiplication using coincidence (AND) gates in hardware. We simulated a 4-layer DNN with the backpropagation algorithm implemented using the stochastic computing principles and found that a bit length of 10 bits is sufficient to maintain acceptable accuracy with the floating-point baseline. Stochastic DNNs with BL=10 will be used as the baseline for studying the dependence of experimental memristive synapses in the training phase of DNNs in the next chapter.

CHAPTER 4

IMPLEMENTATION OF STOCHASTIC DNNS USING MEMRISTIVE SYNAPSES

4.1 Introduction

DNNs have shown remarkable success in many of the machine learning problems recently. However, training of deep neural networks is computationally intensive and requires large training times with a time complexity of $\mathcal{O}(N^2)$ (N is the number of neurons in each layer) in von-Neumann machines; hence, several hardware approaches have been proposed for accelerating DNN training [151, 152]. However, none of these approaches have mitigated the limitations with respect to power, area and training time. Hence, there are several proposals to employ non-volatile memory (NVM) based synapses for an efficient acceleration of neural network training and inference [36, 153–155]. We focus on NVM-based implementations for accelerating DNN training in this chapter and the methodologies described are published in the Neurocomputing journal [71].

Two-terminal memristive devices are an ideal choice for implementing electrical synapses due to its small size, enabling them to be densely packed in crossbar arrays. Further, thanks to their low power programming and read characteristics and the ability to store and retain multiple bits in a single device [41, 156], they offer one possible way to emulate the brain's connectivity in hardware. Moreover, it has been numerically estimated that NVM based on-chip learning systems promise upto 25$\times$ speed up and 3000$\times$ improvement in power compared to GPU (Graphics Processing Unit) based implementations [51]. However, most of the memristive devices being explored today also have non-ideal characteristics such as finite on-off ratio, finite conductance resolution and has temporal and spatial conductance variability during programming and read [24, 157]. Therefore, new architectural and device level optimizations are necessary to obtain the projected performance enhancements in hardware.

It has recently been proposed that DNNs can be implemented using tiled arrays of 2D crossbars of resistive processing units (RPU), which are memristive devices that can store multiple analog states and also adjust its conductivity based on identical sequence of voltage pulses [48]. If such crossbar arrays can be designed and all the weights in the array can be updated in parallel, the training time can be accelerated by replacing the vector cross-product operation with AND operation of stochastic bit streams representing neuronal signals. One of the most challenging requirements to be satisfied by an ideal RPU device is that it must be possible to incrementally program it to nearly 1000 reliable conductance states within a dynamic range of 10 by the application of identical sequence of voltage pulses. This is a stringent requirement and has not been demonstrated so far on experimental devices. $Pr_{0.7}Ca_{0.3}MnO_3$ (PCMO) based RRAM devices have been explored for neuromorphic hardware due to its analog conductance response by previous authors. However, most of these schemes use complex programming methods which cannot support parallel synaptic communication or updates, which is crucial for obtaining $\mathcal{O}(\mathit{1})$ time complexity operation and hardware acceleration. In [135], numerical simulations of a 3-layer network with experimental PCMO characteristics having an on-off ratio of 5 and 256 conductance states showed a recognition accuracy of 90.55% for hand-written digit classification, although pulses with variable amplitudes were used for device programming. Linear and symmetric conductance response are shown to improve network performance [135] and several programming strategies have been explored to compensate for the non-linear and asymmetric conductance response at the cost of higher power and chip area [50, 158, 159].

Towards the goal of attaining parallel synaptic communication and weight update, we fabricated and characterized PCMO devices specifically optimized for analog and incremental programming upon the application of identical programming pulses. Using the measured characteristics, we study the performance of DNNs trained in a stochastic fashion for the exemplary hand-written image recognition task. We then conduct several numerical studies to determine the crucial device parameters for improving network accuracy and training times.

This chapter is organized as follows. We will discuss the basic characteristics of PCMO devices used in the crossbar implementation in Section 4.2. The basic 4-layer network discussed in the previous chapter is then extended to a crossbar compatible implementation with PCMO device conductances as the synaptic weights at the crossbar in Section 4.3. Section 4.4 discusses the algorithm level optimizations, and Section 4.5 presents the performance of stochastic DNNs for different optimizations of the PCMO device characteristics. Section 4.7 demonstrates the robustness of stochastic inference engines using low on-off ratio devices to noise corrupted test data by using higher precision for network encoding for inference as compared to training. Finally, the chapter is concluded in Section 4.8.

4.2 PCMO Device as Synapse

PCMO based RRAM devices are non-filamentary in nature and hence, exhibit high endurance and low variability compared to the filamentary switching devices such as those based on HfO_2 [160]. PCMO devices are favored for neuromorphic hardware due to its simple structure, fast switching speed and area scalability [161,162]. Therefore, we choose PCMO device as the memristive synapse for studying the acceleration of stochastic DNNs.

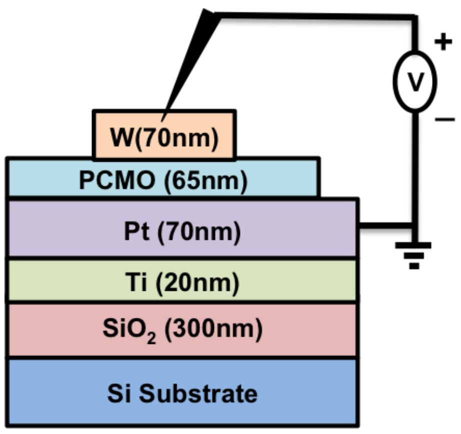


Figure 4.1 $Pr_{0.7}Ca_{0.3}MnO_3$ device structure with Tungsten (W) as the top contact and Platinum (Pt) as the bottom contact.
Source: [71].

4.2.1 Device Fabrication

The $Pr_{0.7}Ca_{0.3}MnO_3$ based RRAM devices were fabricated on 4" Si wafer using a 2 mask lithography process by our collaborators at IIT Bombay. To isolate the device from substrate, 300 nm thick SiO_2 was grown by thermal wet oxidation. Ti (20 nm)/ Pt (70 nm) was then deposited on SiO_2 by DC sputtering. Ti acts as an adhesion layer between SiO_2 and Pt with Pt as the bottom contact for the device. This was followed by deposition of 65 nm thick PCMO alloy using RF sputtering. Different sizes of devices were obtained by defining via-holes of 1 μm in SiO_2 by electron beam lithography (EBL). Finally, Tungsten (W) top contact pads were created using EBL followed by liftoff of W. The device schematic is shown in Figure 4.1. All the electrical measurements were done using Agilent B1500A/B1530A semiconductor analyzer at room temperature at IIT Bombay.

4.2.2 Device Characterization

The fabricated $Pr_{0.7}Ca_{0.3}MnO_3$ device is characterized and the corresponding resistive switching behavior is shown in Figure 4.2 (a). As can be seen from the current-voltage characteristics, significant non-linearity is observed in the conductance of the device as a function of voltage. For instance, the ratio $\eta = g(V)/g(V/2) > 50$ for the operating

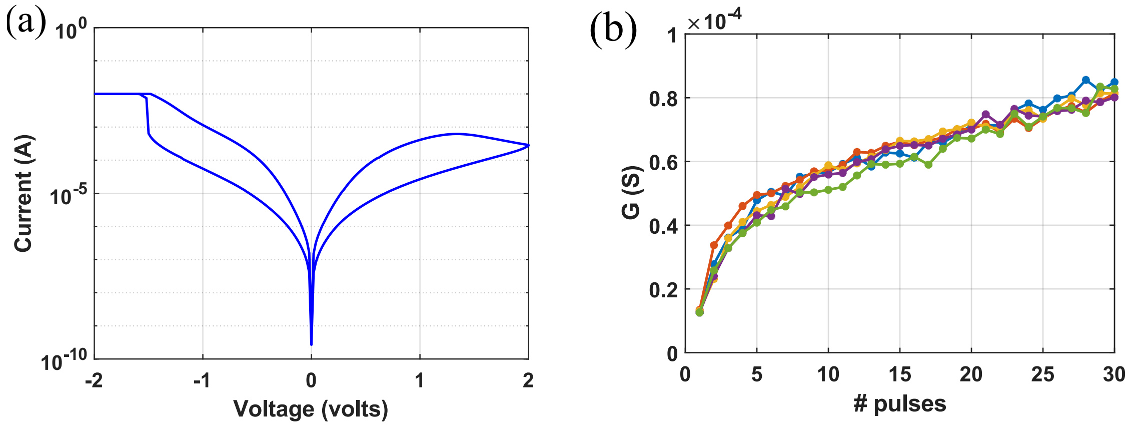


Figure 4.2 (a) Current-Voltage characteristics of the fabricated PCMO device showing non-linearity. (b) The conductance response of the PCMO device showing the variability across five different measurements on the same device. The approximately linear region of the conductance response from pulse #5 to pulse # 30, with an on-off ratio of 1.8 and a resolution of 26 states is used for training stochastic DNNs. *Source:* [71].

region of the device, where $g(V)$ is the conductivity of the device measured at voltage V. This allows PCMO based crossbar arrays to be used for synaptic and memory applications without a current limiting diode or access device at every cross point, based on a $V/2$ programming scheme, as explained in Section 4.3.1 [163, 164]. The selectivity of PCMO devices during programming greatly depends on the non-linearity in the I-V characteristics and is in-turn determined by the PCMO composition [162].

On the application of positive and negative voltage polarity, the PCMO RRAM shows SET (i.e., low to high conductance state) and RESET (i.e., high to low conductance state) switching respectively. The device is initialized to low conductance state by applying a RESET pulse with amplitude 2 V lasting 1 ms [164]. After initializing, WRITE pulses of amplitude -2.2 V and duration 100 ns were applied. A READ pulse (-0.5 V, 5 μs) follows each WRITE pulse to measure conductance change. The conductance response of PCMO device across five different measurements on the same device is shown in the Figure 4.2 (b). The approximate linear region in the conductance characteristics ranging from pulse number 5 to 30 is utilized for training stochastic DNNs. The conductance response clearly shows a variability for repeated measurements and the average programming variability is captured in the simulation of stochastic DNNs.

4.3 Crossbar Compatible Implementation

This section describes the details of the crossbar implementation of the 4-layer network for stochastic DNN training (Figure 3.3).

4.3.1 Two Devices per Synapse

As proposed in [165], two PCMO devices are used per synapse, so that both positive and negative weights can be realized in the network. The synaptic weight is encoded as the difference of the PCMO device conductances, i.e., G_{eff} = G^{+} - G^{-} as shown in Figure 4.3.

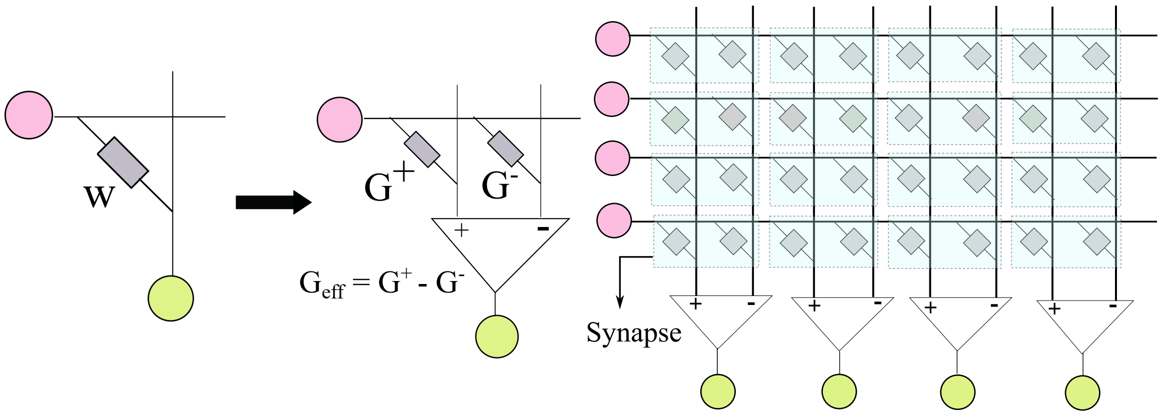


Figure 4.3 Every synapse is represented using two PCMO devices scheme such that the effective synaptic weight, $G_{eff}= G^{+} - G^{-}$ (left); Illustration of a fully connected crossbar network with four input neurons and four output neurons (right).
Source: [71].

If the minimum voltage required for increasing the device conductance is V_p, a $V_p/2$ pulsing scheme is used, where pulses of $+V_p/2$ amplitude are used to encode the 1s in the stochastic streams of the x_i in the rows and pulses of amplitude $-V_p/2$ are used to encode the 1s in the stochastic streams of the δ_j in the columns. Thus, when there is a coincidence of 1s in the two pulse streams, V_p voltage drops across the device, perturbing its conductance. For this pulsing scheme to work accurately, the synaptic device conductance is un-perturbed for pulses of amplitude $\pm V_p/2$, but undergoes conductance transitions for pulses of amplitude V_p. This condition is satisfied in PCMO devices as they exhibit a strong non-linearity as described earlier.

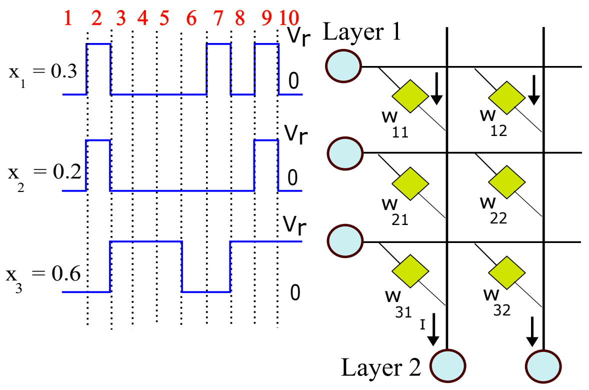


Figure 4.4 Crossbar compatible implementation of forward pass (Read mode). The real valued input to each neuron in any layer is converted to a stochastic pulse stream and depending on the presence or absence of the pulse, current flows downstream to layer 2. This configuration is used for inference as it requires only forward pass. *Source:* [71].

Forward Pass In forward pass, the devices will be operated in read mode and a voltage much lesser than the programming voltage (denoted by V_r) can be used (Figure 4.4). The real valued input that lie in the range [0,1] is converted to a stochastic pulse stream with $BL = 10$ and applied to the cross-bar wires across the rows. The forward pass is illustrated in Figure 4.4 with 3 neurons in layer 1 and 2 neurons in layer 2. The stochastic pulse stream has 10 slots (since $BL = 10$), and depending on the presence or absence of pulse a current flows downstream and gets sensed by the sense amplifier. Thus the forward activation functions can be calculated across all layers in parallel achieving a time complexity of $\mathcal{O}(1)$.

Backward Pass The backward pass involves two steps - illustrated by Equations (3.2), (3.3) and is demonstrated in Figure 4.5 (a), (b). In the first step, matrix

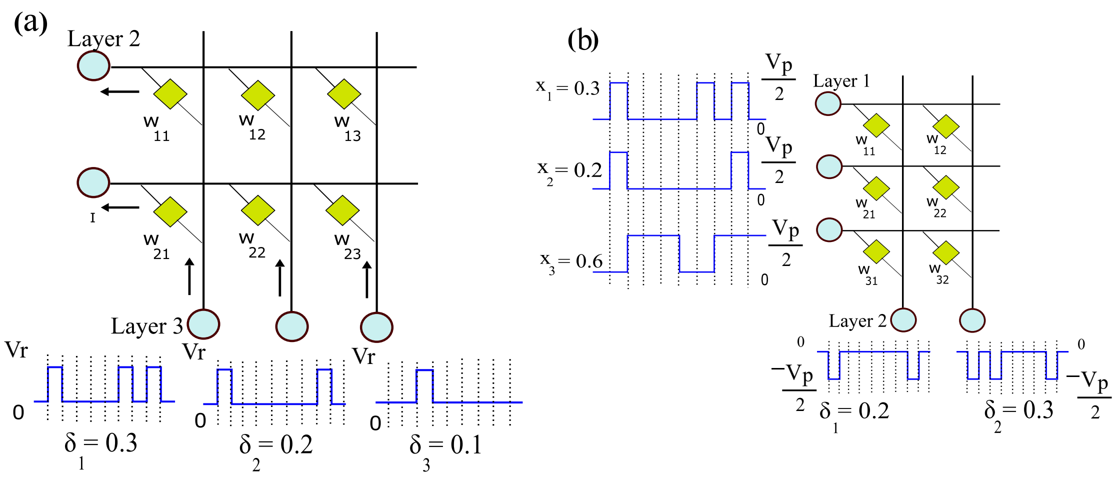


Figure 4.5 Illustration of backward pass. (a) First step of backward pass (Read mode): Matrix multiplication of the error term with the transpose of the weight matrix is done by feeding the error voltage as stochastic pulses to the column such that the total current is summed across the neurons in layer 2. (b) Second step of backward pass (Write mode): For updating the weights in layer 1, the error obtained at layer 2 and input to layer 1 is scaled by the learning rate and applied as stochastic pulses through the crossbar wires. A $V_p/2$ pulsing scheme is used such that the conductance changes only when each device experiences a minimum voltage of V_p. In the case of inference, the configurations shown in (a) and (b) will not be used as no training is involved.
Source: [71].

multiplication of transpose of the weight matrix with the error term is achieved by passing the stochastic error voltage pulses as input to the columns (layer 3) and the currents get summed up across the rows in layer 2. Since the devices are operated in read mode, a voltage of V_r is used to encode the stochastic pulse stream. In the second step of back propagation, voltages corresponding to these current values are then adequately scaled to incorporate the learning rate and presented across the column as stochastic pulses as shown in Figure 4.5 (b). The input to layer 1 will also be scaled and subsequently converted to stochastic pulse streams and applied along the rows. Based on the number of coincidences in the two pulse streams (across the column and row), the device will experience an effective voltage of $V_p/2 - (-V_p/2) = V_p$, thereby changing the conductance by an integral multiple of the minimum resolution B (which in real devices could exhibit significant variability). As a result, by effectively utilizing the locality and parallelism of the back propagation algorithm, weight updates can happen in parallel across all devices in the crossbar achieving a time complexity of $\mathcal{O}(1)$ [138–140].

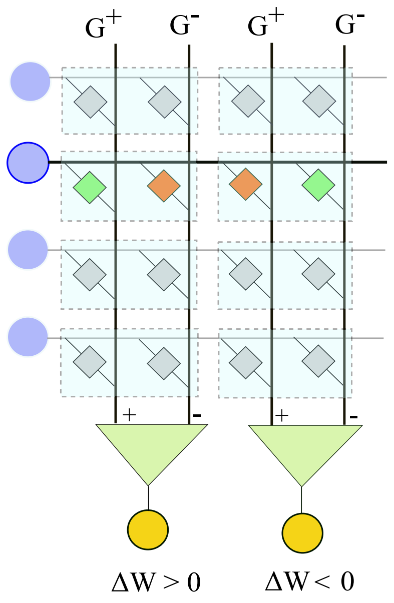


Figure 4.6 Unidirectional weight update scheme in two PCMO devices per synapse in the crossbar. When the required weight update (ΔW) for a synapse > 0, G^+ will get selected and the conductance increases such that $G_{eff} = G^+ - G^-$ is effectively increased. On the other hand, if $\Delta W < 0$ then G^- is selected for weight increment and $G_{eff} = G^+ - G^-$ is reduced. The device marked green is selected for weight increment and the unselected is indicated in red.
Source: [71].

Since most of the memristive devices exhibit gradual conductance change in one direction, we assume a unidirectional weight update scheme [166]. In this scheme, conductance of the device will be always increased and the device representing G^+ (G^-) will be selectively programmed to increase (decrease) the weight. The weight update schematic is illustrated in Figure 4.6. If the calculated incremental weight update (ΔW) using the back propagation algorithm is positive, then the conductance of G^+ (shown in green in Figure 4.6) is increased so that G_{eff} is increased. On the other hand, if the weight update required is negative, then the conductance of G^- is increased to effectively reduce G_{eff}.

4.4 Modifications to the Four-layer Network

All the PCMO devices in the network are initialized randomly between the minimum and maximum conductance values. As the actual device conductance values are not compatible for network learning, they are appropriately scaled during training. The scaling factor is absorbed in the calculation of activation functions. Due to the limited on-off ratio and conductance resolution, the devices could easily saturate and learning could stop. To avoid this, the devices are reset periodically (data refresh) based on the conductance values [165]. During reset, if the effective synaptic weight (G_{eff}) is positive, then G^+ is set to ($G_{min} + G_{eff}$) while G^- is kept to G_{min} such that G_{eff} is unperturbed. Similarly, when G_{eff} is negative, G^+ is set to G_{min} and G^- is set to ($G_{min} + G_{eff}$). Therefore, the effective weight $G_{eff} = G^+ - G^-$, at any synapse will remain unaffected after a reset operation.

We studied two possible reset mechanisms - conditional reset and global reset. In conditional reset, the devices are reset whenever one of the two devices of the synapse reach the maximum conductance value and no more increment in conductance is possible. However, conditional reset requires constant monitoring of conductance at every synapse. A cost effective solution is hence, to use a global reset in which all the devices are reset after a fixed number of training steps [38]. The optimization of reset intervals is discussed in the subsequent Section 4.5.1.

4.5 Network Response with PCMO Device

The baseline stochastic network is extended to a crossbar compatible implementation as mentioned in the previous section with two PCMO devices per synapse. Here we simulate stochastic DNNs using $Pr_{0.7}Ca_{0.3}MnO_3$ devices, whose synaptic behavior is determined from the electrical measurements as explained in section 4.2.2. These PCMO devices has an on-off ratio of 1.8 and approximately only 26 discernible levels including the minimum and maximum conductance states. The conductance of the

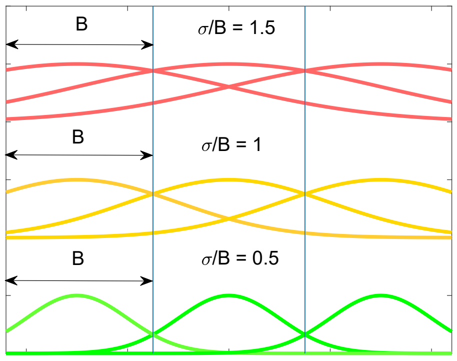


Figure 4.7 Illustration of programming variability as a function of σ/B. Here σ is the standard deviation of the Gaussian noise and B is the minimum change in conductance obtainable under the chosen programming conditions.
Source: [71].

device varies linearly with pulse number and has a programming variability associated for each incremental change. The impact of programming variability on network performance is studied using the parameter σ/B, where σ is the standard deviation of the Gaussian noise and B is the minimum change in conductance obtainable under the chosen programming conditions. The illustration of different programming variability as a function of σ/B is shown in Figure 4.7. It can be noticed from the Figure 4.7 that as the ratio σ/B is increased, the conductance can easily spill over to the neighboring bins. As discussed, the devices have to be periodically reset to facilitate continuous learning and the reset interval in turn depends on the amount of programming variability (σ/B).

4.5.1 Optimization of Reset Interval

During network training, the devices are periodically reset after presenting a fixed number of input images to the network. Here we study the optimization of reset interval for $Pr_{0.7}Ca_{0.3}MnO_3$ devices with different programming variability (captured by the parameter σ/B). The optimization of reset interval for σ/B=0.5, 1, 1.5 is shown in Figure 4.8.

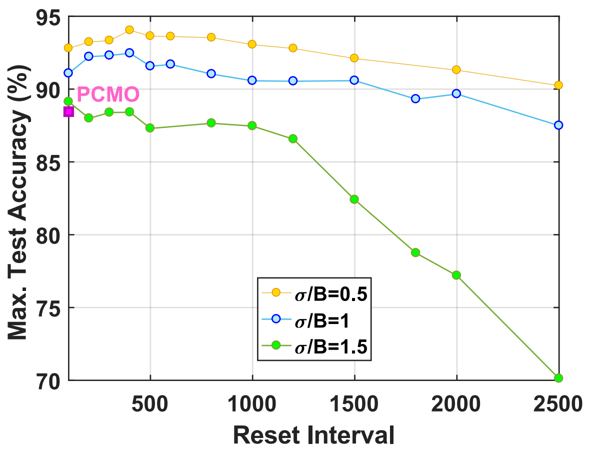


Figure 4.8 Simulated maximum test accuracies for stochastic DNNs with different choices of reset intervals. An optimum reset interval of 400 is chosen for devices having a variability ratio of σ/B=0.5, 1 and 100 is used for devices with variability ratio of σ/B=1.5. The accuracy of the experimental PCMO device based network from Figure 4.9 is also indicated.
Source: [71].

The choice of the reset frequency is based on the network performance. In these simulations, the variability encountered by the devices during the reset process is assumed to be negligible compared to the programming variability associated with the weight update as iterative programming schemes can be used at the time of the infrequent reset operation [167]. Therefore, from Figure 4.8, the reset interval is chosen to be 400, 400, 100 for $\sigma/B = 0.5$, 1 and 1.5 respectively. In the case of devices with low programming variability having $\sigma/B \leq 0.5$, the network can tolerate a higher reset interval with minimum degradation in test accuracy.

The conductance deviation for our experimental PCMO is approximately 1.59× the bin-width (B) of the conductance states ($\sigma = 1.59B$). The training and test accuracy corresponding to stochastic DNN with PCMO device is shown in Figure 4.9. A reset interval of 100 is used for simulating PCMO based memristive synapses to obtain a maximum test accuracy of 88.1% after 10 epochs. The inferior performance of the network from the baseline is due to the increased variability in programming, causing the device conductance levels to spill over into the neighboring levels. In order

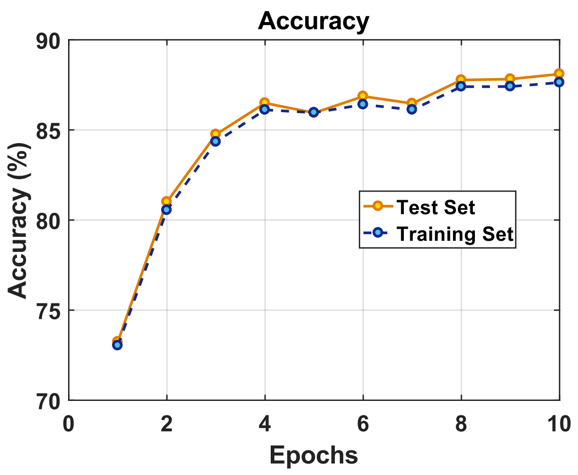


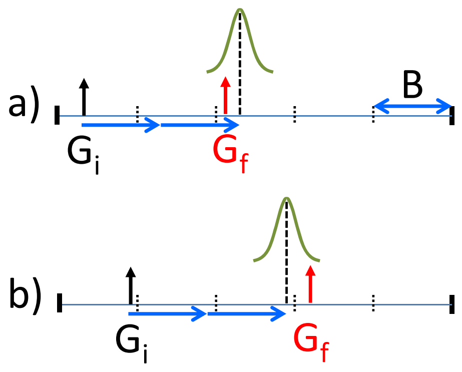
Figure 4.9 Simulated test and training accuracy of stochastic DNNs with experimental PCMO device at the synapse. Shown here is the average response of three different MATLAB iterations. The test accuracy reaches a maximum value of 88.1% after 10 epochs.
Source: [71].

to understand the fundamental reasons for performance degradation with memristive implementations and to optimize device and network performance, we now study the role of specific device characteristics on training and inference accuracy.

4.6 Impact of Device Parameters on Network Performance

4.6.1 Sensitivity to Conductance Variability

Conductance variability is a critical parameter of practical NVM devices and its effect on the learning of stochastic networks for hand-written digit classification is discussed here. Repeated pulse measurements on the fabricated $Pr_{0.7}Ca_{0.3}MnO_3$ device shows significant variations in the conductance values for the same pulse number (see Figure 4.2). The parameter σ/B, which is the ratio of the standard deviation of the PCMO conductance variability to the bin-width of the conductance levels, is used to study the impact of programming variability on network performance. An illustration of memristive programming variability is shown in Figure 4.10 [68]. The conductance variations are introduced in the simulations as a zero mean Gaussian noise with a standard deviation $\sigma/B = 0.1, 0.3$ and so on. The synaptic devices are randomly



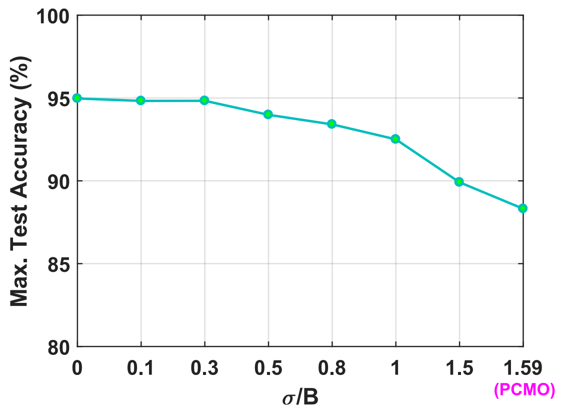


Figure 4.10 Cartoon illustrating the effect of memristive programming variability with an initial conductance G_i, final conductance G_f and σ representing the standard deviation of programming noise. (a) Final Conductance (G_f) obtained by a pulse overlap of 2 from the initial conductance (G_i) exhibits an underestimate, and (b) represents an overestimate of G_f; Maximum generalization (test) accuracy of stochastic DNNs ($BL = 10$) when trained with PCMO devices of different programming variability. Here σ/B is the ratio of standard deviation of the conductance variability to the bin-width of the conductance states (right).
Source: [68, 71].

initialized to any of the 26 conductance states with programming variability. Except for the standard deviation of the conductance variations, all the other parameters such as on-off ratio and number of levels are kept the same as that of experimental PCMO device. The response of stochastic DNNs is observed for $\sigma/B = 0, 0.1, 0.3, 0.8, 1, 1.5$ as shown in Figure 4.10. For $\sigma/B < 0.5$, where the standard deviation of the Gaussian is lesser than the bin-width, the obtained device conductance is close to the desired conductance resulting in close-to baseline accuracies. With standard deviations close to zero, the network achieves a test accuracy close to 95% even with a low on-off ratio of 1.8. This shows that programming variability is critical for training of stochastic DNNs when compared to on-off ratio of the memristive devices. When σ/B ratio is gradually increased from 0 to 0.3, the test accuracy remains almost constant and degrades by 2% at $\sigma/B = 1$. For $\sigma/B > 1$, where the standard deviation is more than the bin-width, the device conductance could easily jump to the neighboring conductance states thus showing inferior network performance.

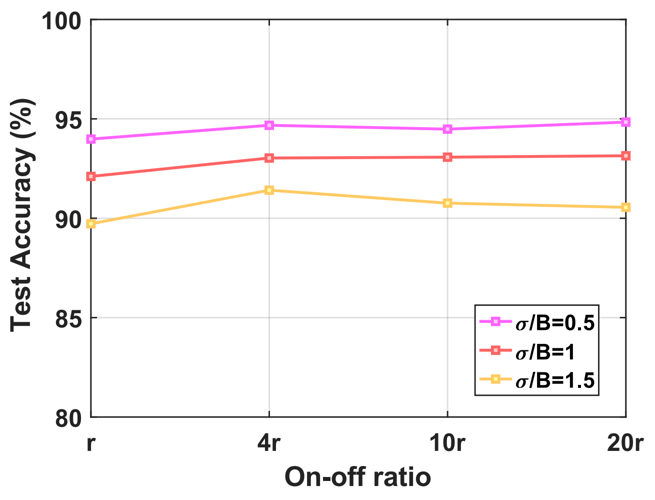


Figure 4.11 Comparison of test accuracy of stochastic DNNs for different on-off ratios of PCMO device at the crossbar. Here, r denotes the experimental on-off ratio of 1.8. The average test accuracies of three different MATLAB iterations is shown here. The accuracy is almost independent of the on-off ratio of the device for a fixed σ/B ratio.
Source: [71].

4.6.2 Impact of On-off Ratio

The performance of stochastic DNNs is analyzed for different on-off ratios of the PCMO devices while keeping the number of levels and programming variability (σ/B) invariant. The device is assumed to have a resolution of 26 states and different scaling for the device on-off ratio compared to the experimental value. The corresponding test accuracy for different on-off ratios ($4r$, $10r$ and $20r$ compared to the baseline experimental on-off ratio of $r = 1.8$) is shown in the Figure 4.11. It can be noticed that the test accuracy is largely independent of the on-off ratio of the PCMO devices when the programming noise is kept fixed. This study illustrates the need for synaptic devices with minimum programming variability rather than high on-off ratio. Since programming variability is inherent to most nanoscale memristive devices, algorithm level optimizations are necessary to obtain better performance.

4.7 Inference in the Presence of Noise

One of the advantages of systems based on stochastic encoding is its inherent tolerance to noise and lesser vulnerability to variability than their deterministic counterparts [168, 169]. Here, we study the stability of stochastic inference engines by observing the network response to noise-corrupted test data. The stochastic DNNs are trained with $BL = 10$ using synaptic devices and these trained weights corresponding to the best generalization accuracy is then used for inference analysis [68]. Zero mean Gaussian noise of variances $\sigma_i^2 = 0, 0.01, 0.1$ is added to the normalized input images in the MNIST test set. The noise added input is then limited in the range [0,1] for evaluating the robustness of stochastic inference engines. A noise corrupted sample image with its corresponding signal to noise ratio (SNR in dB) is shown in the Figure 4.12. Noise resilience for inference is studied for PCMO devices with programming

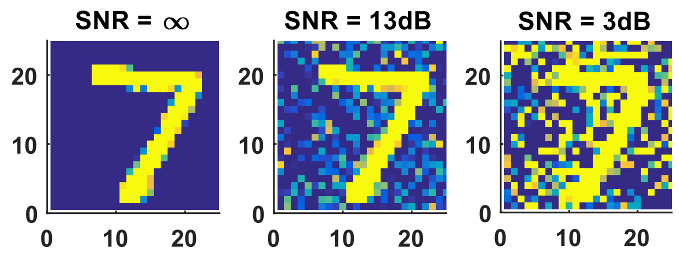


Figure 4.12 A sample test image '7' showing the signal to noise ratio (SNR) after introducing Gaussian noise of different variances. The noise added images are then fed as input to stochastic DNNs to evaluate the inference response.
Source: [71].

variability ratios of $\sigma/B = 0.5$, 1, 1.5. Two variants of stochastic inference engines are used in our study, one with $BL = 10$ and the other $BL = 100$ even though training is done using $BL = 10$.

The inference response of PCMO based synapses with different programming variability to noise added test set is shown in Figure 4.13. The percentage improvement in test accuracy for stochastic inference engines with $BL = 100$ over $BL = 10$ is shown in Figure 4.13. It is observed that as the programming variability

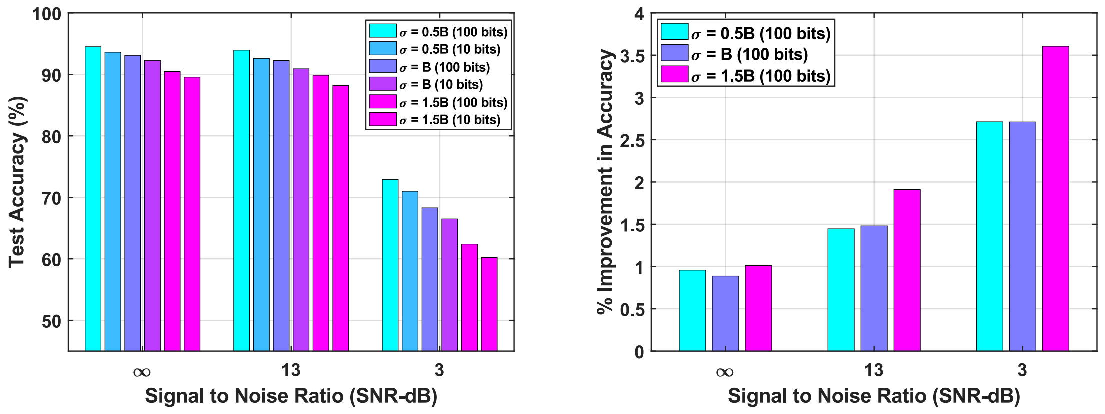


Figure 4.13 Study of stochastic inference engines to noise corrupted test data using trained weights as a function of the programming variability of the device ($\sigma/B = 0.5, 1$ and 1.5) (left); Percentage improvement in test accuracy of stochastic inference engines with $BL = 100$ when compared to $BL = 10$ using devices with different conductance variability. Using inference engines with $BL = 100$, a remarkable improvement in test accuracy can be noticed for devices with higher programming variability (right).
Source: [71].

increases, stochastic inference engines with $BL = 100$ has a superior performance compared to $BL = 10$. This study shows that even though the expensive network training is performed using as few as 10 bits, the inference accuracy of the network can be improved, especially when dealing with noisy inputs, just by increasing the bit resolution used during the relatively in-expensive forward pass operations necessary for inference.

4.8 Summary

In this chapter, we discussed the need for non-von Neumann architecture and a hardware based approach for implementing neuromorphic systems with memristive synapses at the crossbar. DNNs can be trained with a time complexity of $\mathcal{O}(1)$ using RPU devices at the crossbar by representing the real-valued neuronal output and feedback terms as stochastic bit-streams. We used a non-filamentary $Pr_{0.7}Ca_{0.3}MnO_3$ device to understand the device parameters that are critical to the performance of stochastic DNNs. These devices were fabricated using a standard lithography process and electrically characterized to determine the on-off ratio, bit-resolution and programming variability. Using these measured characteristics, we studied the network performance as a function of the device parameters such as on-off ratio, frequency of conductance reset to account for the limited dynamic range and programming variability. We showed that programming variability is paramount to the network performance and demonstrated that if the conductance variability is kept minimum, network test accuracies close to 95% is obtainable using PCMO based devices. Since programming variability is inherent to most nanoscale devices, algorithms that could mitigate these non-ideal characteristics have to be developed for enhanced network performance. We also demonstrated approaches to improve network inference accuracy without incurring significant costs for realizing on-chip learning, especially for noisy real-world inputs.

Having discussed the memristive implementation of DNNs for accelerating the DNN training, we will present the design strategies for spiking networks based on a probabilistic framework using the GLM neuron model in the next chapter. The proposed accelerator SpinAPS - **Spin**tronic **A**ccelerator for **S**piking Networks is an ASIC that combines the advantages of emerging spintronic memory for synaptic storage and digital CMOS logic for neuronal computations.

CHAPTER 5

SPINAPS: A HIGH-PERFORMANCE ACCELERATOR FOR PROBABILISTIC SNNS

5.1 Introduction

The growth of data and associated processing requirements has resulted in intensive research efforts for energy-efficient computing architectures that are capable of approaching the performance of the human brain. Unlike the dominant Deep Neural Networks (DNNs) which rely on real-valued activations, Spiking Neural Networks (SNNs) communicate through discrete and sparse tokens in time called spikes, mimicking the operation of the brain. SNNs are ideal for real-time applications as they take advantage of the temporal dimension for data encoding and processing. However, SNNs lag behind DNNs in terms of computational capability demonstrations due to the current lack of powerful learning algorithms [14].

Conventional neural networks have millions of trainable parameters and are trained using von Neumann machines, where the memory and computation units are physically separated. The performance of these implementations is typically limited by the “von Neumann bottleneck” caused by the constant transfer of data between the processor and the memory. SNN implementations on these platforms becomes inherently slow due to the need to access data over time in order to carry out temporal processing. Hence, hardware accelerators are necessary to exploit the full potential of SNNs and also to develop efficient learning algorithms. In an effort to build large-scale neuromorphic computing systems that can emulate the energy efficiency of the human brain, several computing platforms have implemented SNNs. While Tijanic chip [112], Intel’s Loihi [60] and IBM’s TrueNorth [61] realize spiking neurons and make use of static random access memory (SRAM) to store the state of synapses and neurons, recent research efforts have proposed the use of tiled crossbar arrays of two terminal nanoscale devices for implementing large-scale DNN systems [39,48,56,170]. Spintronic devices have also been explored for implementing synaptic weights owing to their fast read/write characteristics, high endurance, and scalability [43,44]. The stochastic nature of the spintronic devices has been leveraged to model neural transfer functions in a crossbar architecture [171]. These designs for SNNs have shown over 20× energy improvements over conventional CMOS designs. Notably an all-spin neuromorphic processor for SNNs, comprising of spintronic synapses and neurons with in-memory computing architecture, has shown 1000× energy efficiency and 200× speed up compared to CMOS [70].

5.1.1 Our Contributions

The main contributions of this work are listed as follows.

- We propose a hardware accelerator for inference, **Spin**tronic **A**ccelerator for **P**robabilistic **S**NNs (SpinAPS) that integrates binary spintronic devices to store the synaptic states and digital CMOS neurons for computations. The potential of hardware implementations of Generalized Linear Model (GLM)-based probabilistic SNNs trained using the energy efficient first-to-spike rule are evaluated for the first time in this work.
- We evaluate the performance of probabilistic SNNs on two benchmarks - handwritten digit recognition and human activity recognition datasets, and show that SNN performance is comparable to that of equivalent ANNs.
- SpinAPS achieves 4× performance improvement in terms of GSOPS/W/mm^2 when compared to an equivalent design that uses SRAM to store the synaptic states.

The rest of the chapter is organized as follows. In Section 5.2, we review the architecture of GLM-based probabilistic SNNs and also explain the first-to-spike rule for classification. The algorithm optimization strategies for an efficient hardware implementation for the first-to-spike rule is considered in Section 5.3. The architecture details of SpinAPS core and the mapping of the GLM kernels into the memory is discussed in Section 5.4. Section 5.5 details the relevant digital CMOS neuron logic blocks and memory design. We then evaluate the performance of our hardware accelerator for different bit precision choices in Section 5.6. Finally, we conclude the chapter in Section 5.7.

5.2 Review of FtS Classification Using GLM-based SNN

Generalized Linear Models (GLM) yield a probabilistic framework for SNNs that is flexible and computationally tractable [80, 172]. As shown in Figure 5.1, in the GLM neuron model the inputs (stimuli) are filtered by the stimulus kernels (α), while the output spike history is filtered by the feedback kernel (β) to define the neuron's membrane potential. The spiking probability at each time instant is given by a non-linear function of the membrane potential. GLM neurons have reproduced a wide range of spiking neuronal behaviors observed in human brain by appropriately tuning the stimulus and feedback kernels [80]. Learning rules for a GLM SNN based on rate and first-to-spike decoding rules have been derived in a number of works reviewed in [172–174].

5.2.1 GLM Architecture

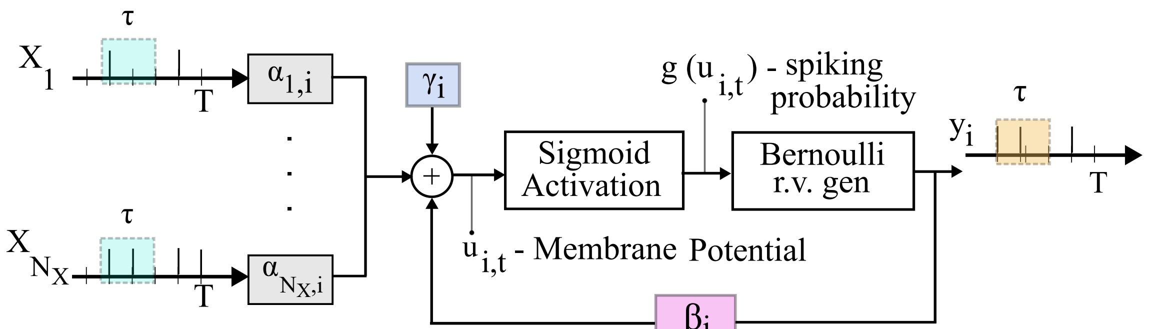


Figure 5.1 The Generalized Linear Model (GLM) model used for SNN learning in this work. N_X input neurons and one output neuron is shown for simplicity.

The basic architecture of GLM-based probabilistic SNNs used for SNN training is shown in Figure 5.1. We focus on a 2-layer SNN, which has N_X presynaptic neurons encoding the input and N_Y output neurons corresponding to the output classes. Each of the input neurons receives a spike train having T samples through rate encoding. The input is normalized in the range [0,1] for image classification and in the range [-1,1] for activity recognition. The spikes are then issued through a Bernoulli random process. For cases where the sign of the input is vital in achieving learning, the negative sign is absorbed in the sign bit of the corresponding weights. The membrane potential of i^{th} output neuron at any instant t can be expressed as,

$$u_{i,t} = \sum_{j=1}^{N_X} \alpha_{j,i}^T x_{j,t-\tau}^{t-1} + \beta_i^T y_{i,t-\tau}^{t-1} + \gamma_i \tag{5.1}$$

where $\alpha_{j,i}$ denotes the stimulus kernel; $x_{j,t-\tau}^{t-1}$ represents the input spike window having τ spike samples, β_i denotes the feedback kernel; $y_{i,t-\tau}^{t-1}$ represents the output spike window with τ samples; and γ_i is the bias parameter. Here j refers to the index of the pre-synaptic neuron and i refers to the index of the post-synaptic neuron. The stimulus and feedback kernels in GLM can be defined as the weighted sum of fixed basis functions with the learnable weights, and they are expressed as shown below.

$$\alpha_{j,i} = \mathbf{A}\mathbf{w}_{j,i} \ , \beta i = \mathbf{B}\mathbf{v}_i \tag{5.2}$$

The matrices $\mathbf{A}$, $\mathbf{B}$ are the basis vectors, defined as $\mathbf{A}$=$[a_1, \ldots a_{K_\alpha}]$ and $\mathbf{B}$=$[b_1, \ldots b_{K_\beta}]$. The prior work in GLM [84] uses real-valued raised cosine basis vectors for the stimulus and the feedback kernels. The vectors $\mathbf{w}_{j,i}$=$[w_{j,i,1}, \ldots w_{j,i,K_\alpha}]^T$ and $v_i = [v_{i,1}, \ldots v_{i,K_\beta}]^T$ are the learnable weights in the network with K_α, K_β denoting the number of basis functions. The spiking probability of the output neuron is then decided based on the sigmoid non-linearity applied to the membrane potential.

Two main decoding strategies have been considered for the given SNN architecture - rate decoding and first-to-spike decoding, discussed in detail below.

When using the rate decoding scheme for inference, the network decision is based on the neuron with the maximum spike count as illustrated in Figure 5.2. During the training phase, the labeled neuron is assigned with a desired spike train pattern and the unlabeled neurons are assigned with a pattern of all zeros. Using the maximum likelihood definition, the sum of log probabilities of the output spikes of the desired neuron is maximized i.e.,

$$L(\theta) = \sum_{i=1}^{N_Y} \log p_{\theta_i}(\mathbf{y}_i(c)|\mathbf{x}) \tag{5.3}$$

where θ_i denotes the learnable parameters of the network $\theta = \mathbf{W}, \mathbf{V}, \gamma$ with $\mathbf{W} = \mathbf{w}_i$, $\mathbf{V} = \mathbf{v}_i$ and $\gamma = \gamma_i$, $i = 1 \ldots N_Y$. The negative log-likelihood - $L(\theta)$ is convex with respect to θ and can be minimized by stochastic gradient descent. For

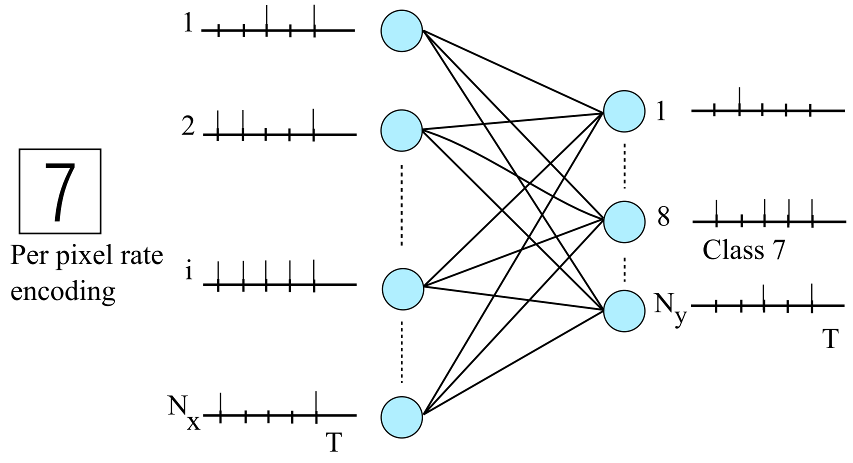


Figure 5.2 Illustration of rate decoding for an input image of '7'. The eighth output neuron corresponding to the class '7' will issue more spikes compared to others once the network is fully trained.

the first-to-spike scheme, a decision is made when one of the output neuron spikes. It has been shown in [84] that the first-to-spike rule exhibits a low inference complexity compared to rate decoding due to its ability to make decisions early. Hence, we choose the first-to-spike scheme for our hardware optimization studies.

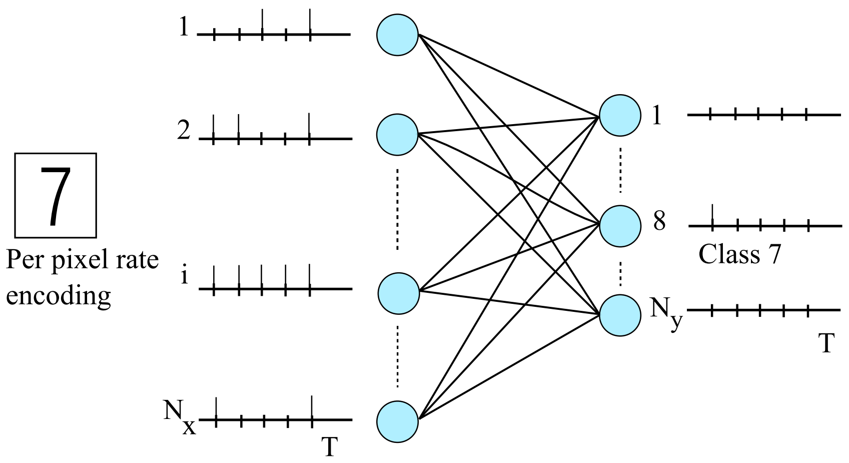


Figure 5.3 Illustration of first-to-spike (FtS) rule decoding scheme for the GLM-based SNN on the handwritten digit benchmark.

5.2.2 First-to-Spike (FtS) Decoding

The fundamental idea of first-to-spike scheme is illustrated in Figure 5.3. The kernels for the GLM-based SNN are trained using the maximum likelihood criterion, which maximizes the probability of obtaining the first spike at the labeled neuron and no spikes for all other output neurons up to that time instant. This probability can be mathematically expressed as

$$p_t(\theta) = \prod_{i=1, i\neq c}^{N_Y} \prod_{t'=1}^{t} \overline{g}(u_{i,t'}) g(u_c, t) \prod_{t'=1}^{t-1} \overline{g}(u_{c,t'}) \tag{5.4}$$

where c corresponds to the labeled neuron, u denotes the membrane potential, and $g(u)$ denotes the sigmoid activation function applied to the membrane potential u. Also $\overline{g}(u) = 1 - g(u)$. The weight update rules are derived by maximizing the log probability in Equation (5.4) and are summarized below [84].

$$\nabla_{\mathbf{w}_{j,i}} L(\theta) = \begin{cases} -\sum\limits_{t=1}^{T} h_t g(u_{i,t}) \mathbf{A}^T \mathbf{x}_{j,t-\tau_y}^{t-1} & i \neq c \\ -\sum\limits_{t=1}^{T} [h_t g(u_{c,t}) - q_t] \mathbf{A}^T \mathbf{x}_{j,t-\tau_y}^{t-1} & i = c \end{cases}$$

$$\nabla_{\gamma_i} L(\theta) = \begin{cases} -\sum\limits_{t=1}^{T} h_t g(u_{i,t}) & i \neq c \\ -\sum\limits_{t=1}^{T} [h_t g(u_{c,t}) - q_t] & i = c \end{cases}$$

where

$$q_t = \frac{p_t(\theta)}{\sum_{t=1}^{T} p_t(\theta)} \qquad\qquad h_t = \sum_{t'=t}^{T} q_{t'}$$

Note that the feedback kernels are not necessary for the first-to-spike rule as the network dynamics need not be calculated after the first spike is observed.

5.2.3 Datasets Used in This Study

Throughout this work, we evaluate the performance of Probabilistic SNNs on handwritten digit recognition and human activity recognition (HAR) datasets [175]. With $60,000$ training and $10,000$ test images, each of the input image in the handwritten digit database has 784 (28×28) pixels corresponding to the 10 $(0, 1, \ldots, 9)$ digits. The HAR dataset has a collection of physical activities feature extracted from the time series inputs of the embedded sensors in a smart phone. The database has roughly $7,000$ training samples and $3,000$ test samples corresponding to six types of physical activities. Each record in the HAR dataset has a 561 feature datset vector with time, frequency domain variables and an activity label corresponding to the six physical activites such as walking, walking upstairs, walking downstairs, sitting, standing and laying.

5.3 Hardware-Software Co-optimization

In this section, we discuss the design choices that facilitate our hardware implementation. We start by observing the floating-point baseline accuracy of FtS in Figure 5.4, where the kernels are trained using the techniques demonstrated in [84]. For the purpose of designing the inference engine, we assume that the kernels are fixed.

Choice of Basis Vectors Most prior works using GLM neuron models use real-valued raised cosine basis functions for implementing the stimulus and the feedback kernels [80, 82, 84]. The stimulus kernel (α) and feedback kernel (β) can be obtained as the weighted combination of the basis vectors and the learnable parameters in the network as shown in Equation (5.2). Hardware implementation of kernels employing real-valued basis vectors would require K multipliers and $(K-1)$ adders at every synapse, where K is the number of basis functions used per kernel. Hence, we propose the hardware friendly binary basis vectors for realizing the kernels. This would simplify the kernels as, $\alpha_{i,j} = w_{i,j}$ and $\beta_i = v_i$ eliminating the need for multipliers and adders at the synapses. It can be seen that the network performance

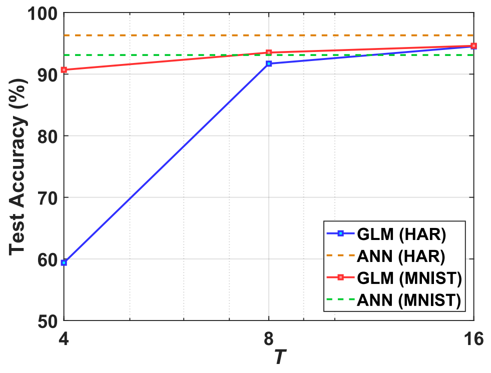


Figure 5.4 The test accuracy of GLM on human activity recognition (HAR) and handwritten digit recognition. A comparable performance is achieved with respect to an ANN having the same architecture. Here presentation time T is kept the same as the spike integration window τ.

improves with higher presentation times T and spike integration windows τ, reaching a maximum test accuracy for $T = 16$ and $\tau = 16$. Note that for the handwritten digit benchmark, the network accuracy of the GLM SNN trained using the FtS rule with ($T = 8$, $\tau = 8$) is 93.5%, which is at par with the 93.1% accuracy of a 2-layer artificial neural network (ANN) with the same architecture. In the case of HAR dataset, GLM achieves a maximum test accuracy of 94.5% with $T = 16$, $\tau = 16$ which is comparable with ANN test accuracy of 96.3%. Hence, we chose $T = 8$, $\tau = 8$ for handwritten digit recognition and $T = 16$, $\tau = 16$ for HAR benchmarks as the baseline. We now discuss software optimization strategies for implementing the first-to-spike scheme in an energy efficient manner in hardware.

Sigmoid Activation Function As shown in Figure 5.1, the basic GLM neuron architecture uses a sigmoid activation function to determine the spike probability of the output neurons. The implementation of sigmoid functions in hardware is relatively complex as it involves division and exponentiation, incurring significant area and power [176]. Digital implementations for sigmoid mainly fall into three categories- piecewise linear (PWL) approximations, piecewise second order approximations and combinational approximations. Here we follow the piecewise linear approximation (PWL) demonstrated in [177] for implementing the sigmoid activation function. In the PWL approximation, the sigmoid is broken up into integer set of break points such that the resulting function can be expressed as powers of two. Considering the negative axis alone, the PWL approximation y for input x can be expressed as

$$y_{x<0} = \frac{\frac{1}{2} + \frac{\hat{x}}{4}}{2^{|(x)|}} \tag{5.5}$$

where (x) is the integer part of x and $\hat{x}$ is the fractional part of x. Using the symmetry of sigmoid function, the values in the positive x-axis can be obtained as $y_{x>0} = 1 - y_{x<0}$. The output of the PWL approximation is then compared with a pseudo-random number generated from the linear feedback shift register (LFSR) and a spike is generated if the PWL value is greater than the LFSR output. A 16-bit LFSR is used and is assumed to be shared among all the output neurons.

Quantization We now aim at finding the minimum bit precision required for the learnt weights $w_{j,i}$ and biases γ_i in order to maintain close to baseline floating-point accuracy for the network. Starting with the baseline networks for the two datasets obtained using floating point parameters, we quantize them into discrete levels and study the inference accuracy with the PWL approximation for the sigmoid. With b-bit quantization, one bit is reserved for the sign to represent both positive and negative parameters. We use a uniform quantizer with quantization step given by the relation

$$w_q = \frac{w_{max} - w_{min}}{2^{\mathrm{b}-1}}, \;\; \gamma_q = \frac{\gamma_{max} - \gamma_{min}}{2^{\mathrm{b}-1}}, \tag{5.6}$$

respectively for weights and biases. We also quantize the membrane potential and the output of the PWL activation to b bits. We summarize the inference performance of the GLM SNN as a function of bit precision in Figure 5.5 (a). Network performance degrades with lower choices of b, but we note that 5 bit resolution is sufficient for maintaining close to baseline floating-point inference accuracy for the two benchmarks. Even though the input spike pattern lasts for T algorithmic time

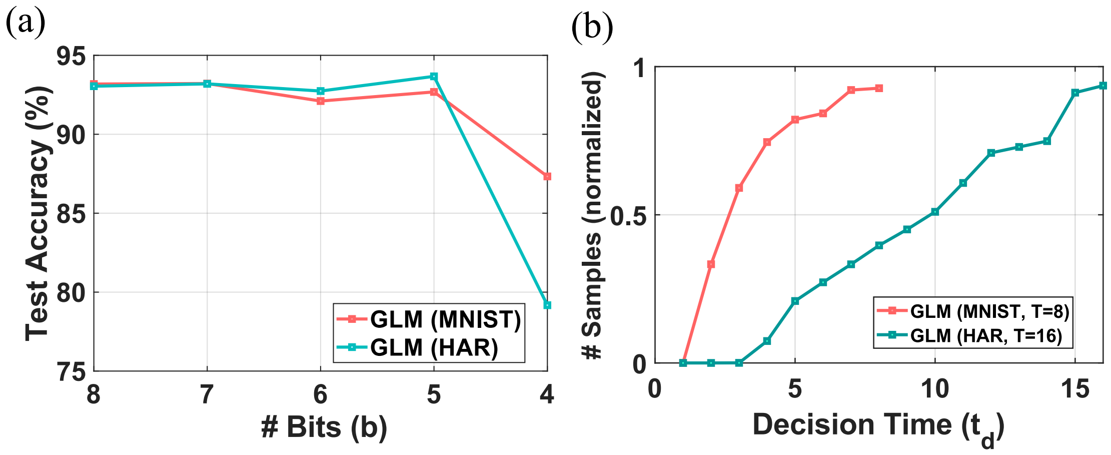


Figure 5.5 (a) Test performance of the GLM SNN after quantization of weights with PWL approximation and by using a 16-bit LFSR. (b) Cumulative distribution of the number of input samples classified as a function of decision time t_d. An early decision can be made in first 4 time steps for classifying 75% of the images in the handwritten digit benchmark.

steps, the first-to-spike learning rule allows a decision to be made even before all the spikes are presented to the network. The cumulative distribution of the number of samples classified as a function of decision time t_d is shown in Figure 5.5 (b). It can be observed for the handwritten digit benchmark, around 75% of the samples are classified in the first 4 algorithmic time steps. Furthermore this network achieves the same performance as that of a 2-layer ANN in 8 time steps. Thus GLM SNN can leverage the ability of first-to-spike rule in making decisions with reduced latency and memory accesses.

5.4 Overview of SpinAPS Architecture

As illustrated in Figure 5.6, the core architecture of SpinAPS consists of binary spintronic devices to store the synaptic states and digital CMOS neurons to perform the neuronal functionality. The SpinAPS core accepts input spike at every processor time step t, reads the synapses corresponding to the spike integration window τ from the memory to compute the membrane potential and applies the non-linear PWL activation function to determine the spike probability of the output neurons. A pseudo-random number generated from the LFSR is then used to actually determine whether a spike is issued or not as explained in Section 5.3. The word width, and hence, the memory capacity of the banked STT-RAM memory used in the SpinAPS core can be designed based on the bit precision b required for the synapses. The generic "neuro-synaptic core" of SpinAPS consists of a banked memory with spintronic synapses and peripheral driver circuits to perform both read or write operations into the memory. The word width of the memory, and hence, the memory capacity, can be designed based on the bit precision b required for the synapses. The design of the synaptic memory for different bit precision values of the synapses is discussed in Section 5.5.2. SpinAPS co-locates the dense STT-RAM memory array and digital CMOS computations in the same core, thereby avoiding the traditional von Neumann bottleneck. Following the principles demonstrated in IBM's TrueNorth chip [61], a tiled architecture with 4096 SpinAPS cores, with each core having 256 neurons, can be used to realize a system with 1 million neurons as illustrated in Figure 5.7. If the network has more number of neurons than that is possible from a core, then the feature can be split to a different core while maintaining the same performance [61, 178]. We next discuss the basic characteristics of the spintronic

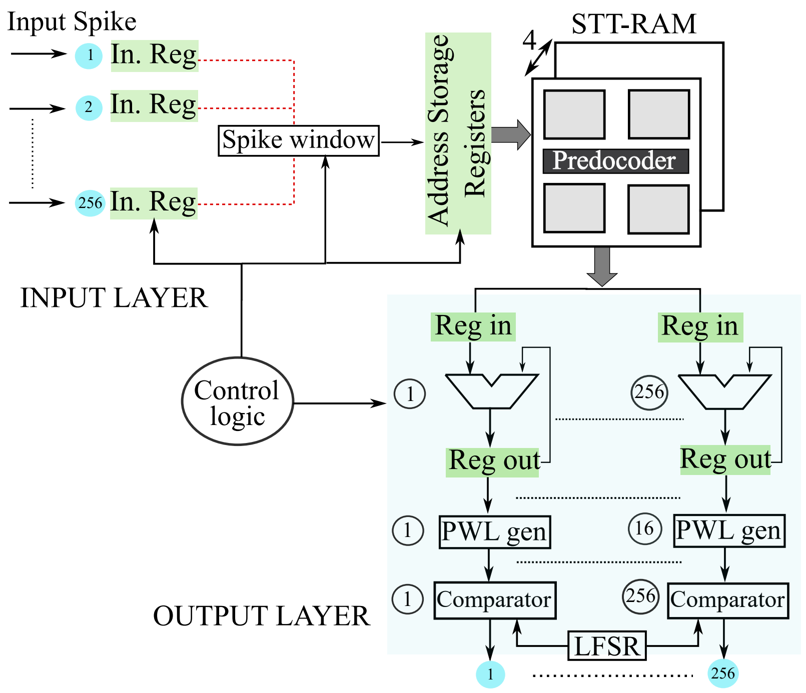


Figure 5.6 The core architecture of SpinAPS having binary STT-RAM devices to store the synaptic states and digital CMOS neurons. Assuming a spike integration window of $\tau = 7$, the core can map 256 input and 256 output neurons.

synapses and how we can map the parameters of probabilistic SNNs into the memory.

5.4.1 STT-RAM as Synapse

Synapses in neural networks scale as N^2, where N is the number of neurons in each layer. SRAM circuits have been used to realize the synapses of the neuromorphic chips demonstrated so far [60, 61]. However this approach does not scale well for a large-scale neural network system. Hence, there have been extensive efforts to use two terminal nanoscale devices as synapse in the neuromorphic hardware owing to its small size and its ability to be packed in dense crossbar arrays [179,180]. These devices have the potential to emulate brain's connectivity and power efficiency in hardware due to their dimensional scalability, nonvolatile nature, low-power programming and read characteristics. Recently, spintronic devices have been explored for its use as nanoscale synapses [43,44,171] mainly due to its high read and write bandwidths, low

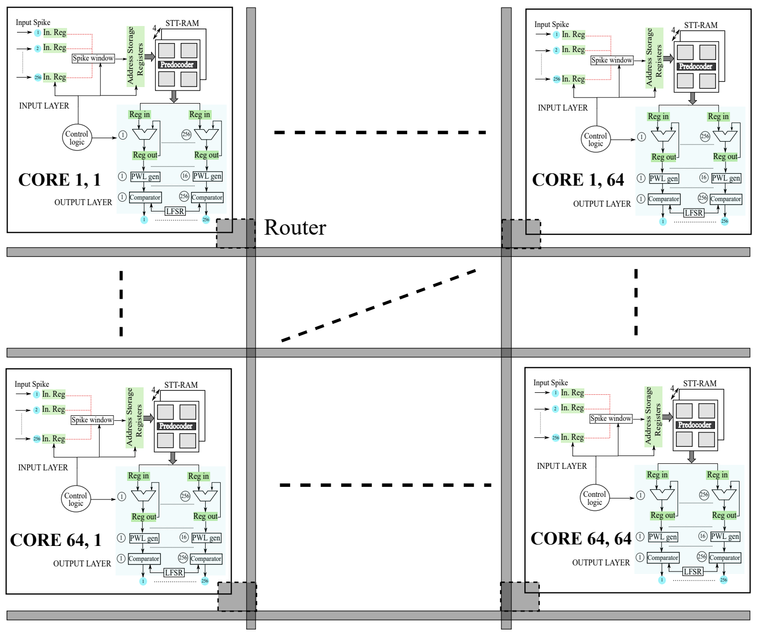


Figure 5.7 The tiled architecture of SpinAPS obtained by arranging the neuro-synaptic cores that communicate to each other using a packet routing digital mesh network.

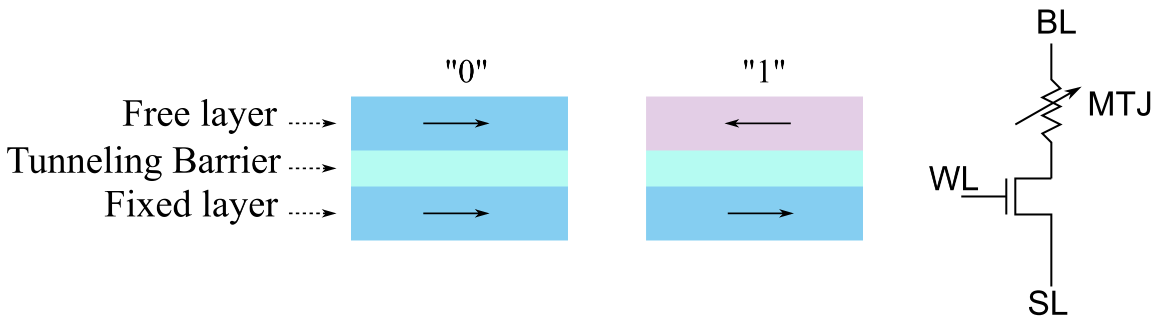


Figure 5.8 Illustration of basic cell structure of 1T/1MTJ STT-RAM with low ("0") and high resistance ("1") states.

power consumption and excellent reliability [181–183]. These spintronic devices are compatible with CMOS technology and fast read/write has been demonstrated in sub 10 ns regime [184, 185]. Here we propose binary STT-RAM devices as the nanoscale electrical synapses for the SpinAPS core illustrated in Figure 5.6. Structurally, STT-RAM uses a Magnetic Tunnel Junction (MTJ) with a pair of ferromagnets separated by a thin insulating layer. These devices have the ability to store either '0' or '1' depending on the relative magnetic orientation of the two ferromagnetic layers as shown in Figure 5.8. The memory array is typically configured based on the crossbar architecture with an access transistor connected in series with the memory device to selectively read and program it.

5.4.2 Synaptic Memory Architecture

Crossbar memory architectures with memristive devices at the cross point have been explored for accelerating the hardware implementation of neural networks [48, 170]. Here we discuss the mapping of the learnable parameters, stimulus kernel (α) and the bias parameters (γ) of GLM-based SNN into the spintronic memory to achieve accelerated hardware performance.

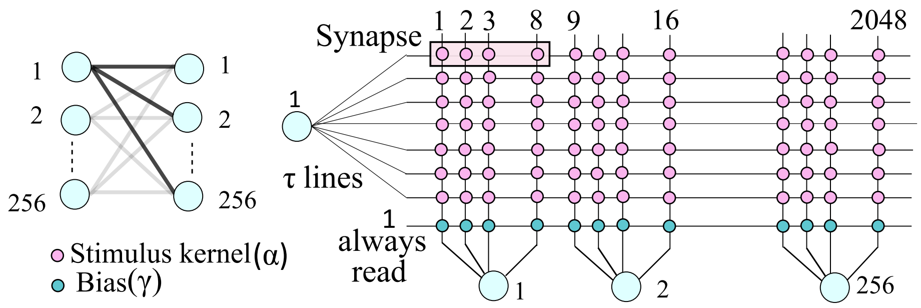


Figure 5.9 Kernel mapping of GLM SNN to the spintronic memory of SpinAPS core, showing that devices on τ word-lines are used to represent the kernel parameters of every input neuron.

Kernel Mapping Here we choose ($T = 8$, $\tau = 7$) to describe the mapping of 8-bit synapses, hence, the word width (vertical lines in Figure 5.9) required for the memory is $256 \times 8 = 2048$ bits. Each of the input neuron generates a bit pattern of length τ, hence, unique kernel weights are provided for every neuron, requiring $256 \times 7 = 1792$ word lines in the array. Thus, a network with 256 input and 256 output neurons can be mapped to a memory array with 2048×2048 memory devices. At the input side, each of the input neuron generates a bit pattern of length τ using the spike window generator shown in Figure 5.6. For example if the bit pattern generated by the first input neuron is "1010010", then the synaptic weights corresponding to 1^{st},

3^{rd}, and 6^{th} word line will be read sequentially. The address corresponding to these word lines are stored in registers, indicated as address storage registers in Figure 5.6. These synaptic weights will be added at the output neuron to compute the membrane potential. One word line will be sufficient for mapping γ as (256×8) bits is equivalent to the word width of the core. As γ determines the baseline firing rate of the output neurons, the word line voltage corresponding to the γ line is kept high indicating an "always read" condition. Once all the active word lines are read sequentially, the accumulated membrane potential will be fed into the sigmoid generator to determine the firing probability.

In this memory architecture, each neuron inherently supports a fan-out of 256. Assuming every output neuron in any core supports a fan-out connectivity of 1000 spanning across multiple cores, each of them will have to store 4 destination addresses $(256 \times 4 = 1024)$. In a memristive implementation, these addresses can be stored using binary STT-RAM cells within the array. This could save the area required for LUTs (look up tables) for storing the destination addresses [69]. For example, if there are 4096 cores, 12 bits will be required to address these cores and an additional 11 bits to uniquely identify the word line within a core. This would result in roughly storing $24K$ bits per core; which is equivalent to 12 lines in the array for storing the required fan-out.

5.5 High-level Design

5.5.1 Neuron Implementation

We now describe the digital CMOS implementation of the relevant blocks. The baseline design assumes 8-bit precision and is synthesized using TSMC 45 nm logic process running at a clock frequency of 500 MHz. We assume 8-bit precision for the computational blocks for our baseline design, and will discuss how this architecture can be modified to implement reduced bit precision architectures.

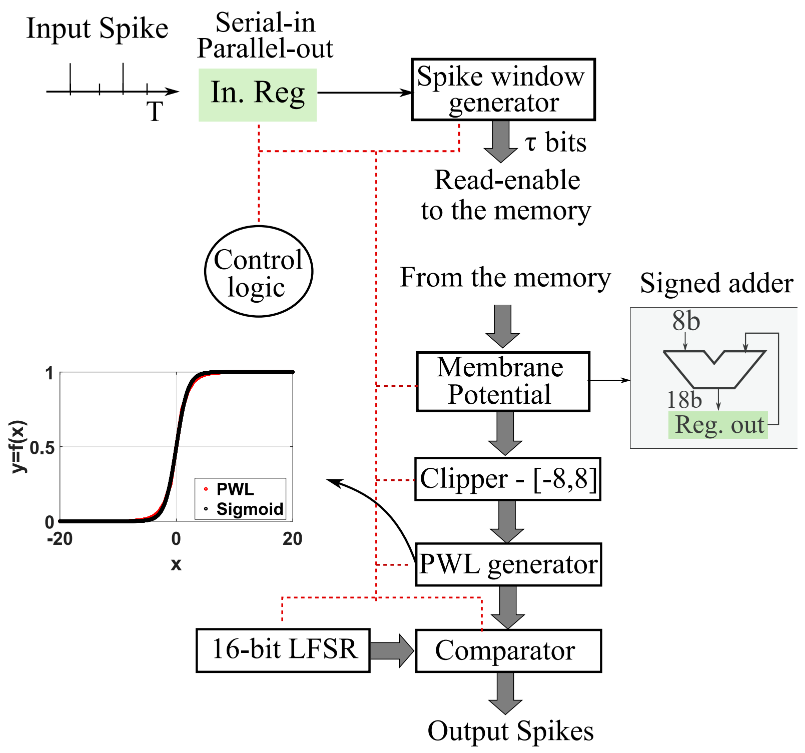


Figure 5.10 The digital CMOS logic blocks of SpinAPS core which is used to perform the probabilistic computations. Here we show the computations associated with one of the neurons in the core.

Generation of Spike Window As discussed in Section 5.2, normalized input spikes are transformed to generate spikes of length T using a Bernoulli random process. At each time instant, $t = 1 \dots T$, incoming spikes are latched at the input neuron (marked as 'In. Reg', a serial-in, parallel out shift register in Figure 5.6) which is then translated into an activation pattern of length τ, and applied as read enable signals to the word lines associated with each kernel weights. The spike window generator circuit uses a multiplexer for selecting the bit pattern of length τ from the input register and whose select lines are configured by the controller for every t. For example, when $t = 1$, the select line will be "000" and as no input spikes have arrived before, the output of the multiplexer would be "0000000". When $t = 3$, the select line becomes "010", and the multiplexer output will be "$s_2s_1$000000", where $s_i = 1/0$ represents the presence/absence of a spike at $t = i$ and so on. As discussed in Section 5.4, the synapses associated with the input neurons are read sequentially and hence, the logic circuit for the spike window generator can be shared among all the input neurons. The synaptic weights are stored in the registers and the sign bit of the weights can be optionally flipped depending on the sign of the input before computing the membrane potential (u). Membrane potential (u) is obtained by adding the synaptic weights, whose word lines are active depending on the bits in the spike integration window τ. Here we assume a fixed point representation for the synaptic weights. u is calculated at every time instant, $t \in [1 \ldots T]$ using an 18-bit signed adder. The addition is pipelined with the memory read providing dedicated adders for each of the output neurons and the result is stored in an 18-bit register. The accumulation continues until all the active word lines are read one by one.

Piecewise Linear Approximation (PWL) for Sigmoid Calculation Our GLM neuron model uses a non-linear sigmoid activation function for generating the spike probabilities for the output neuron based on the value of u. Here we adopt the traditional piecewise linear approximation (shown in Equation 5.5) for the sigmoid using adders and shift registers as described in [177]. The 18-bit membrane potential u is clipped to an 8-bit fixed point number in the range [-8,8] before applying to the Piecewise Linear Approximation (PWL) sigmoid generator which is shared among 16 output neurons. The clipped fixed point representation assumes 1 bit for the sign, 4 bits for the integer and 3 bits for the fractional part. The output of the sigmoid generator is an unsigned 8-bit number whose values lie in the range $[0, 255]$ and is compared with the 8-bit pattern from the 16-bit LFSR (Linear Feedback Shift Register) to generate output spikes.

5.5.2 Design of the Synaptic Memory

We describe the memory organization for the baseline design with 8-bit precision and ($T = 8$, $\tau = 7$) as discussed in Section 5.3. The required memory capacity for any b-bit precision would become $2048 \times 256 \times$ b bits. We design and analyze the STT-RAM

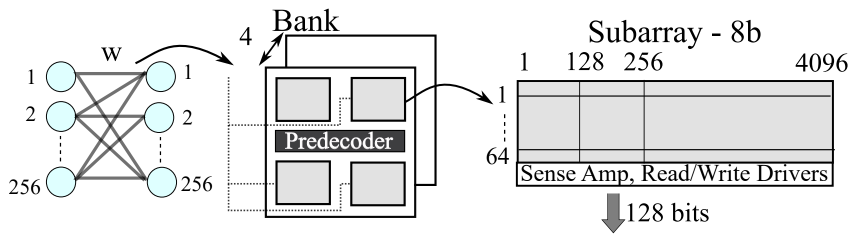


Figure 5.11 Banked STT-RAM organization for mapping the synapses of GLM SNN having 256 input and 256 output neurons with 8-b precision weights.

based synaptic memory using DESTINY, a comprehensive tool for modeling emerging memory technologies [186]. STT-RAM cell architecture with 1T/1MTJ configuration is simulated using the parameters mentioned in Table **??** with the feature size $F = 70\,\text{nm}$ [187]. In Table **??**, $\mathrm{V_r}$ is the read voltage, $\mathrm{t_r}$ is the read pulse width, $\mathrm{I_p}$ is the programming current and $\mathrm{t_w}$ denotes the write pulse width. Representative values for read and write pulse width is assumed based on experimentally reported chip-scale demonstrations of STT-RAM [184,187]. For the baseline design, a 512 KB

Table 5.1 Simulation Parameters for STT-RAM Array using DESTINY with the Feature Size $F = 70\,\text{nm}$

Parameter	Cell Area	$\mathrm{R_{on}}$	$\mathrm{R_{off}}$	$\mathrm{V_r}$	$\mathrm{t_r}$	$\mathrm{I_p}$	$\mathrm{t_w}$
	$\mathrm{F^2}$	Ω	Ω	mV	ns	μA	ns
Value	24	2500	5000	80	5	150	10

array is simulated with an optimization target of minimizing the read-energy-delay product. The optimized solution of the memory mapping for 8-bit precision is shown in Figure 5.11. The read energy per word line for 8b is around 535 pJ with a latency of 7.34 ns. The simulated STT-RAM array running at 100 MHz has an area efficiency of 43% with a total synaptic area of $1.14\,\text{mm}^2$ and power of 53.5 mW. In addition to the base-line design, we also studied memory organization for reduced precision (b$= 5-7$ bits), translating to a total memory capacity of $2048 \times 256 \times \text{b}$ bits respectively. The corresponding subarray organization for each of the bit precision using DESTINY simulations comes to $64 \times (2 \times 256 \times \text{b})$.

5.6 Performance Evaluation

We use a commonly used performance metric to benchmark our accelerator design - the number of billions of synaptic operations that can be performed per second per watt (GSOPS/W) and per unit area (GSOPS/W/mm^2). A synaptic operation refers to reading one active word line (256 b-bit synapses), computing the spike probability and then issuing the output spike. We compare these performance metrics of SpinAPS with an equivalent SRAM-based design for different bit precisions (shown in Table 5.2), with neuron logic implementation kept the same. The SRAM memory is simulated using DESTINY [186] and clocked at 250 MHz. Independent of bit precision, SpinAPS can perform 25.6 GSOPS while the SRAM-based design achieves 64 GSOPS due to its higher clock rate. For 8-bit precision, SpinAPS core is approximately 6× better in synaptic power and 3× in synaptic area when compared to SRAM. For the values reported in Table 5.2, a 10% overhead is considered for the spike routing infrastructure between the cores and 20% overhead is added to consider the area and power of the core's controller [61, 65, 188]. It can be noticed

Table 5.2 Performance Comparison of STT-RAM and SRAM-based Designs

Precision	GSOPS/W		GSOPS/W/mm^2	
	SRAM	STT-RAM	SRAM	STT-RAM
5	353	474	177	559
6	283	412	119	415
7	230	366	83	322
8	193	311	61	239

that the SpinAPS core can achieve a maximum performance improvement of 4× and a minimum of 3× in terms of GSOPS/W/mm^2 when compared to an equivalent SRAM-based design for 8b and 5b precision choices respectively. The average energy per algorithmic time step t of the SpinAPS core and the total area for the design as a function of the bit precision is shown in Fig. 5.12.

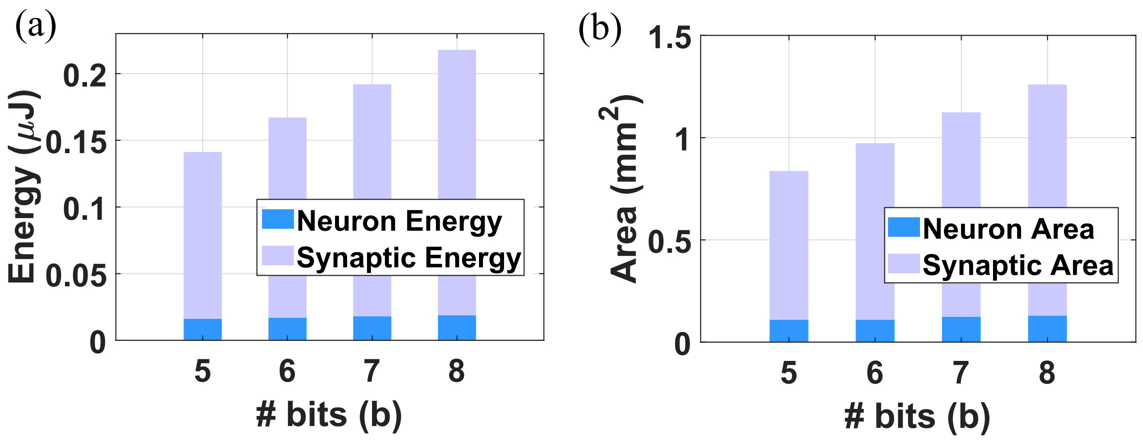


Figure 5.12 (a) Average energy per processor time step (t) of SpinAPS for each of the bit choices for the handwritten digit benchmark. One processor time step corresponds to reading all the active word lines (on an average 365), compute the membrane potential and generating an output spike. (b) Area of the SpinAPS core as a function of bit precision.

Here we compare the performance of SpinAPS with an STT-RAM based inference accelerator for SNNs, memristor-based inference engines for DNNs, and GPUs. For instance, the 4× performance improvement in terms of GSOPS/W/mm^2 obtained for SpinAPS in comparison with the SRAM-based synaptic implementation is comparable with the 6× performance improvement reported in [189], which benchmarks custom design of STT-RAM and SRAM synapses. In addition, SpinAPS design, when extrapolated to 32 nm technology node achieves 850 GSOPS/W, which is competitive with recent memristor-based DNN inference engines that reported 800 GSOPS/W [57] at 32 nm and 1060 GSOPS/W [56] at 32 nm. Though not a fair comparison, SpinAPS can achieve approximately 20×, and 4× improvement in terms of GSOPS/W, and GSOPS/mm^2 respectively against state-of-the-art GPUs like Tesla V100 [28]. While implementations employing analog phase change memory (PCM) devices in crossbar arrays have been projected to provide two orders of magnitude improvement in energy efficiency when compared to GPUs using all-parallel array operations and analog neuron excitations [170], SpinAPS achieves moderate improvement over GPUs with sequential synaptic accesses and representative design choices.

5.7 Summary

In this chapter, we discussed a hardware accelerator for inference of probabilistic spiking neural networks, SpinAPS using binary STT-RAM devices and digital CMOS neurons. The probabilistic SNNs based on the GLM neuron model are trained using the novel first-to-spike rule and its performance is benchmarked on two standard datasets, exhibiting comparable performance with an equivalent ANN. We discussed the design of the basic elements in the SpinAPS core, considering different bit precision choices and the trade-off associated with the performance and hardware constraints. SpinAPS leverages the ability of first-to-spike rule in making decisions with low latency achieving approximately $4\times$ performance improvement in terms of GSOPS/W/mm^2 compared to SRAM-based design.

CHAPTER 6

CONCLUSIONS AND FUTURE OUTLOOK

The current digital era is witnessing an exponential growth of data due to the plethora of devices generating data in the form of videos and images, machine to machine communication, Internet of Things (IoT) sensors, metadata created by the edge devices, etc. Machine learning is going to be ubiquitously applied to process these data for analytics and for improving end-to-end performance. Hence, it is necessary to deploy the neural networks in a hardware friendly way for power-constrained applications without compromising the throughput.

Two terminal nanoscale devices are capable of implementing large scale neural networks in hardware owing to its compact size and lower programming energy requirements. However, these nanoscale devices are non-ideal and pose a significant challenge to integrate with computational substrates. We captured the non-ideal characteristics of the memristive devices in training the deep neural networks through several simulation studies discussed in this work. Among the non-ideal characteristics of the two-terminal nanoscale devices, programming variability is found to be having a critical role in determining the network performance.

We also showed the potential of the probabilistic framework for spiking neural networks (SNNs) in achieving comparable performance with equivalent second-generation artificial neural networks on two popular datasets. The inference engine SpinAPS, a spintronic accelerator for probabilistic spiking networks, combined one of the emerging memory technology STT-RAM for implementing the synapses and digital CMOS logic for neuronal computations. SpinAPS was shown to achieve $4\times$ improvement in terms of GSOPS/W/mm^2 when compared to a conventional

SRAM-based design. During this research, we could identify several aspects of future work and are listed below.

- Implementation of on-chip learning for probabilistic SNNs: Dedicated hardware could accelerate the training time for probabilistic SNNs, just like ASIC and FPGA-based solutions became handy for accelerating the DNN training. Initial hardware-software co-optimization studies for GLM-based probabilistic SNNs revealed that a minimum of 13-bit precision multipliers are required for computing the probabilities at every time instant. It might be interesting to optimize the algorithm so that the costly multipliers can be replaced with hardware friendly operations like stochastic computing.

- Encoding schemes for time-series datasets: SNNs can process time-series inputs due to its inherent temporal nature. The time-series input to the SNNs could be either a real number or discrete output events from a neuromorphic sensor [190–192]. For example, the HAR dataset used in our work also has additional raw time-series inputs that are real-valued, and the feature extracted time-invariant data values were used in Chapter 5. Send-on-delta and Sigma-delta schemes can be potentially used for encoding such time-series inputs [193–195]. Initial experiments using the send-on-delta scheme for GLM-based probabilistic SNNs on the HAR dataset did not yield high accuracies. It could be due to the limitation of having a single layer of GLM neurons for training. However, the effectiveness of send-on-delta and other possible encoding schemes [196] inspired by biology can be pursued for multilayered SNN architectures for better validation [197].

- Random number generation in hardware: Random number generators are an integral block in brain-inspired systems. Our studies on stochastic DNNs using memristive networks showed that programming variability is inherent to the nanoscale devices, and there are several prior works on utilizing this variability for generating true random numbers in hardware [198–201]. Conventionally, digital CMOS-based pseudo-random linear feedback shift registers (LFSRs) are used in hardware and faces challenges concerning the area as well as power when implementing large scale neural network systems. It is interesting to explore and model STT-RAM based true random number generators for our proposed inference engine for probabilistic spiking networks - SpinAPS.

BIBLIOGRAPHY

[1] A. P. Alivisatos, M. Chun, G. M. Church, R. J. Greenspan, M. L. Roukes, and R. Yuste, “The Brain Activity Map Project and the Challenge of Functional Connectomics,” *Neuron*, vol. 74, no. 6, pp. 970–974, Jun 2012.

[2] K. Amunts, C. Ebell, J. Muller, M. Telefont, A. Knoll, and T. Lippert, “The Human Brain Project: Creating a European Research Infrastructure to Decode the Human Brain,” *Neuron*, vol. 92, no. 3, pp. 574 – 581, 2016.

[3] M.-m. Poo, J.-l. Du, N. Ip, Z.-Q. Xiong, B. Xu, and T. Tan, “China Brain Project: Basic Neuroscience, Brain Diseases, and Brain-Inspired Computing,” *Neuron*, vol. 92, no. 3, pp. 591–596, Nov 2016.

[4] M. A. Hofman, “Evolution of the human brain: when bigger is better,” *Frontiers in neuroanatomy*, vol. 8, pp. 15–15, Mar 2014.

[5] C. Cherniak, “The bounded brain: Toward quantitative neuroanatomy,” *Journal of Cognitive Neuroscience*, vol. 2, no. 1, pp. 58–68, 1990.

[6] M. A. Hofman, “Chapter 18 - design principles of the human brain: An evolutionary perspective,” in *Evolution of the Primate Brain*, ser. Progress in Brain Research, M. A. Hofman and D. Falk, Eds. Elsevier, 2012, vol. 195, pp. 373 – 390.

[7] “The most powerful computers on the planet,” https://www.ibm.com/thought-leadership/summit-supercomputer/, Online; Accessed: 15-January-2020.

[8] “The Brain as Computer: Bad at Math, Good at Everything Else,” https://spectrum.ieee.org/computing/hardware/the-brain-as-computer-bad-at-math-good-at-everything-else, Online; Accessed: 15-January-2020.

[9] K. Roy, A. Jaiswal, and P. Panda, “Towards spike-based machine intelligence with neuromorphic computing,” *Nature*, vol. 575, no. 7784, pp. 607–617, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1677-2

[10] W. S. McCulloch and W. Pitts, “Neurocomputing: Foundations of research,” J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27.

[11] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533 EP –, Oct 1986.

[13] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Effiicient BackProp,” in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50.

[14] S. R. Nandakumar *et al.*, “Building brain-inspired computing systems: Examining the role of nanoscale devices,” *IEEE Nanotechnology Magazine*, vol. 12, no. 3, pp. 19–35, Sep. 2018.

[15] F. Ponulak and A. Kasiński, “Supervised learning in spiking neural networks with resume: Sequence learning, classification, and spike shifting,” *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010, pMID: 19842989.

[16] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, Aug 1952, pMC1392413[pmcid].

[17] L. Abbott, “Lapicque’s introduction of the integrate-and-fire model neuron (1907),” *Brain Research Bulletin*, vol. 50, no. 5, pp. 303 – 304, 1999.

[18] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov 2003.

[19] R. Brette and W. Gerstner, “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity,” *Journal of Neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005, pMID: 16014787.

[20] A. Sebastian, T. Tuma, N. Papandreou, M. Le Gallo, L. Kull, T. Parnell, and E. Eleftheriou, “Temporal correlation detection using computational phase-change memory,” *Nature Communications*, vol. 8, no. 1, p. 1115, 2017.

[21] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, “Tiered-latency dram: A low latency and low cost dram architecture,” in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 615–626.

[22] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, “Dark Memory and Accelerator-Rich System Optimization in the Dark Silicon Era,” *IEEE Design Test*, vol. 34, no. 2, pp. 39–50, April 2017.

[23] O. Mutlu, “Memory scaling: A systems architecture perspective,” in *2013 5th IEEE International Memory Workshop*, May 2013, pp. 21–25.

[24] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, L. L. Sanches, I. Boybat, M. L. Gallo, K. Moon, J. Woo, H. Hwang, and Y. Leblebici, “Neuromorphic computing using non-volatile memory,” *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017.

[25] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[26] Y. Sun, N. B. Agostini, S. Dong, and D. Kaeli, "Summarizing cpu and gpu design trends with product data," 2019.

[27] "Computers versus Brains," https://www.scientificamerican.com/article/computers-vs-brains/, Online; Accessed: 9-February-2020.

[28] "NVIDIA V100 Tensor Core GPU," https://www.nvidia.com/en-us/data-center/v100/, Online; Accessed: 9-February-2020.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[30] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, and et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12.

[31] "Cloud TPU ," https://cloud.google.com/tpu/, Online; Accessed: 28-February-2020.

[32] "An ecosystem for local AI," https://coral.ai/about-coral/, Online; Accessed: 15-January-2020.

[33] "Intel Neural Compute Stick 2 (NCS2)," https://software.intel.com/en-us/neural-compute-stick, Online; Accessed: 15-January-2020.

[34] "Jetson Nano Developer Kit," https://developer.nvidia.com/embedded/jetson-nano-developer-kit, Online; Accessed: 15-January-2020.

[35] "Jetson AGX Xavier Developer Kit," https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit, Online; Accessed: 15-January-2020.

[36] B. L. Jackson, B. Rajendran, G. S. Corrado, M. Breitwisch, G. W. Burr, R. Cheek, K. Gopalakrishnan, S. Raoux, C. T. Rettner, A. Padilla, A. G. Schrott, R. S. Shenoy, B. N. Kurdi, C. H. Lam, and D. S. Modha, "Nanoscale Electronic Synapses Using Phase Change Devices," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 12:1–12:20, May 2013.

[37] D. Kuzum, R. G. D. 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, 2012, pMID: 21668029.

[38] G. Burr, R. Shelby, C. di Nolfo, J. Jang, R. Shenoy, P. Narayanan, K. Virwani, E. Giacometti, B. 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," in *Electron Devices Meeting (IEDM), 2014 IEEE International*, Dec 2014, pp. 29.5.1–29.5.4.

[39] D. Kuzum, S. Yu, and H.-S. P. Wong, "Synaptic electronics: materials, devices and applications," *Nanotechnology*, vol. 24, no. 38, Sep 2013.

[40] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, "A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation," *Advanced Materials*, vol. 25, no. 12, pp. 1774–1779, 2013.

[41] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis, "Integration of nanoscale memristor synapses in neuromorphic computing architectures," *Nanotechnology*, vol. 24, no. 38, p. 384010, 2013.

[42] 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, Apr 2010. [Online]. Available: https://doi.org/10.1021/nl904092h

[43] N. Locatelli *et al.*, "Spintronic devices as key elements for energy-efficient neuroinspired architectures," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 994–999.

[44] Y. Wang *et al.*, "A case of precision-tunable STT-RAM memory design for approximate neural network," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015.

[45] M. Sharad, C. Augustine, G. Panagopoulos, and K. Roy, "Spin based neuron-synapse module for ultra low power programmable computational networks," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, June 2012, pp. 1–7.

[46] A. Sengupta, Y. Shim, and K. Roy, "Proposal for an All-Spin Artificial Neural Network: Emulating Neural and Synaptic Functionalities Through Domain Wall Motion in Ferromagnets," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, no. 6, pp. 1152–1160, Dec 2016.

[47] S. Fukami and H. Ohno, "Perspective: Spintronic synapse for artificial neural network," *Journal of Applied Physics*, vol. 124, no. 15, p. 151904, 2018.

[48] G. Tayfun and Y. Vlasov, "Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices," *CoRR*, vol. abs/1603.07341, 2016.

[49] T. Gokmen, M. Onen, and W. Haensch, "Training deep convolutional neural networks with resistive cross-point devices," *Frontiers in Neuroscience*, vol. 11, p. 538, 2017.

[50] S. Sidler, I. Boybat, R. M. Shelby, P. Narayanan, J. Jang, A. Fumarola, K. Moon, Y. Leblebici, H. Hwang, and G. W. Burr, "Large-scale neural networks implemented with Non-Volatile Memory as the synaptic weight element: Impact of conductance response," in *2016 46th European Solid-State Device Research Conference (ESSDERC)*, Sept 2016, pp. 440–443.

[51] G. W. Burr, P. Narayanan, R. M. Shelby, S. Sidler, I. Boybat, C. di Nolfo, and Y. Leblebici, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 4.4.1–4.4.4.

[52] Ming Cheng, Lixue Xia, Zhenhua Zhu, Yi Cai, Yuan Xie, Yu Wang, and Huazhong Yang, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[53] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 27–39.

[54] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Yu Wang, Hao Jiang, M. Barnell, Qing Wu, and Jianhua Yang, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[55] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 541–552.

[56] A. Ankit *et al.*, "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference," *CoRR*, 2019.

[57] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 14–26.

[58] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct 1990.

[59] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *CoRR*, vol. abs/1705.06963, 2017. [Online]. Available: http://arxiv.org/abs/1705.06963

[60] M. Davies *et al.*, “Loihi: A neuromorphic Manycore Processor with On-Chip Learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

[61] P. A. Merolla *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, 2014.

[62] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The SpiNNaker Project,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[63] K. Meier, “A mixed-signal universal neuromorphic computing system,” in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 4.6.1–4.6.4.

[64] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary *et al.*, “Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[65] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, “A Scalable Multicore Architecture with Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs),” *IEEE Transactions on Biomedical Circuits and Systems*, Feb 2018.

[66] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart *et al.*, “Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model,” *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, Jan 2019.

[67] G. W. Burr, M. J. Breitwisch *et al.*, “Phase change memory technology,” *Journal of Vacuum Science & Technology B*, vol. 28, no. 2, pp. 223–262, 2010.

[68] A. V. Babu and B. Rajendran, “Stochastic deep learning in memristive networks,” in *2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2017, pp. 214–217.

[69] B. Rajendran, Y. Liu, J. Seo, K. Gopalakrishnan, L. Chang, D. J. Friedman, and M. B. Ritter, “Specifications of Nanoscale Devices and Circuits for Neuromorphic Computational Systems,” *IEEE Transactions on Electron Devices*, vol. 60, no. 1, pp. 246–253, Jan 2013.

[70] A. Sengupta, A. Ankit, and K. Roy, “Performance analysis and benchmarking of all-spin spiking neural networks (special session paper),” in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 4557–4563.

[71] A. V. Babu, S. Lashkare, U. Ganguly, and B. Rajendran, “Stochastic learning in deep neural networks based on nanoscale pcmo device characteristics,” *Neurocomputing*, vol. 321, pp. 227 – 236, 2018.

[72] S. R. Kulkarni, A. V. Babu, and B. Rajendran, “Acceleration of convolutional networks using nanoscale memristive devices,” in *Engineering Applications of Neural Networks.* Cham: Springer International Publishing, 2018, pp. 240–251.

[73] W. Maas, "Networks of spiking neurons: The third generation of neural network models," *Trans. Soc. Comput. Simul. Int.*, vol. 14, no. 4, p. 1659–1671, Dec. 1997.

[74] W. Maass, "Lower bounds for the computational power of networks of spiking neurons," *Neural Computation*, vol. 8, no. 1, pp. 1–40, Jan 1996.

[75] Wikipedia, "Neuron — Wikipedia, the free encyclopedia," 2004, [Online; accessed 10-January-2020].

[76] E. Kandel, J. Schwartz, T. Jessell, S. Siegelbaum, and A. Hudspeth, *Principles of Neural Science, Fifth Edition.* McGraw-Hill Education, 2012.

[77] D. Purves, *Neuroscience.* Sunderland, Mass. : Sinauer Associates, 2012.

[78] S. Valadez-Godínez, H. Sossa, and R. Santiago-Montero, "The step size impact on the computational cost of spiking neuron simulation," in *2017 Computing Conference*, July 2017, pp. 722–728.

[79] E. Izhikevich, "Dynamical systems in neuroscience," *MIT Press*, p. 111, Jul. 2007.

[80] A. I. Weber and J. W. Pillow, "Capturing the Dynamical Repertoire of Single Neurons with Generalized Linear Models," *Neural Computation*, 2017.

[81] W. Gerstner, "Chapter 12 a framework for spiking neuron models: The spike response model," in *Neuro-Informatics and Neural Modelling*, ser. Handbook of Biological Physics, F. Moss and S. Gielen, Eds. North-Holland, 2001, vol. 4, pp. 469 – 516.

[82] J. W. Pillow, J. Shlens, L. Paninski *et al.*, "Spatio-temporal correlations and visual signalling in a complete neuronal population," *Nature*, vol. 454, pp. 995 EP –, Jul 2008.

[83] J. W. Pillow, L. Paninski, V. J. Uzzell, E. P. Simoncelli, and E. J. Chichilnisky, "Prediction and Decoding of Retinal Ganglion Cell Responses with a Probabilistic Spiking Model," *Journal of Neuroscience*, vol. 25, no. 47, pp. 11 003–11 013, 2005.

[84] A. Bagheri *et al.*, "Training Probabilistic Spiking Neural Networks with First- To-Spike Decoding," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018.

[85] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* The MIT Press, 2005.

[86] D. O. Hebb, *The organization of behavior: A neuropsychological theory.* New York: Wiley, 1949.

[87] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[88] "Spike-timing dependent plasticity ," http://www.scholarpedia.org/article/Spike-timing˙dependent˙plasticity, Online; Accessed: 25-February-2020.

[89] W. Gerstner, R. Kempter, J. L. van Hemmen, and H. Wagner, "A neuronal learning rule for sub-millisecond temporal coding," *Nature*, vol. 383, no. 6595, pp. 76–78, 1996.

[90] R. Kempter, W. Gerstner, and J. L. van Hemmen, "Hebbian learning and spiking neurons," *Phys. Rev. E*, vol. 59, pp. 4498–4514, Apr 1999.

[91] 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.

[92] N. Anwani and B. Rajendran, "NormAD - Normalized Approximate Descent based supervised learning rule for spiking neurons," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[93] 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.

[94] S. M. Bohte, J. N. Kok, and H. L. Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17 – 37, 2002.

[95] E. Hunsberger and C. Eliasmith, "Spiking Deep Networks with LIF Neurons," *CoRR*, vol. abs/1510.08829, 2015.

[96] M. Pfeiffer and T. Pfeil, "Deep Learning With Spiking Neurons: Opportunities and Challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.

[97] H. Mostafa, "Supervised Learning Based on Temporal Coding in Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, July 2018.

[98] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.

[99] S. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbruck, "Event-driven sensing for efficient perception: Vision and audition algorithms," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 29–37, Nov 2019.

[100] M. A. Mahowald and C. Mead, "The silicon retina," *Scientific American*, vol. 264, no. 5, pp. 76–82, 1991.

[101] "IMAGENET," http://www.image-net.org/, Online; Accessed: 16-January-2020.

[102] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[103] B. Rajendran, A. Sebastian, and E. Eleftheriou, "Building next-generation ai systems: Co-optimization of algorithms, architectures, and nanoscale memristive devices," in *2019 IEEE 11th International Memory Workshop (IMW)*, May 2019, pp. 1–4.

[104] B. Rajendran, A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou, "Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 97–110, Nov 2019.

[105] C. S. Thakur, J. L. Molin *et al.*, "Large-Scale Neuromorphic Spiking Array Processors: A Quest to Mimic the Brain," *Frontiers in Neuroscience*, vol. 12, p. 891, 2018.

[106] A. R. Young, M. E. Dean, J. S. Plank, and G. S. Rose, "A Review of Spiking Neuromorphic Hardware Communication Systems," *IEEE Access*, vol. 7, pp. 135 606–135 620, 2019.

[107] S. R. Kulkarni, A. V. Babu, and B. Rajendran, "Spiking neural networks — Algorithms, hardware implementations and applications," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2017, pp. 426–431.

[108] C. Mayr, S. Hoeppner, and S. Furber, "SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning," 2019.

[109] J. Schemmel, L. Kriener, P. Müller, and K. Meier, "An accelerated analog neuromorphic hardware system emulating NMDA- and calcium-based non-linear dendrites," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 2217–2226.

[110] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 7388–7397.

[111] A. Lines, P. Joshi, R. Liu, S. McCoy, J. Tse, Y. Weng, and M. Davies, "Loihi asynchronous neuromorphic research chip," in *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2018, pp. 32–33.

[112] J. Pei, L. Deng, S. Song *et al.*, "Towards artificial general intelligence with hybrid tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.

[113] C. Frenkel, M. Lefebvre, J. Legat, and D. Bol, "A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 1, pp. 145–158, Feb 2019.

[114] J. Shen, D. Ma, Z. Gu *et al.*, "Darwin: a neuromorphic hardware co-processor based on Spiking Neural Networks," *Science China Information Sciences*, vol. 59, no. 2, pp. 1–5, 2016.

[115] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019.

[116] D. Silver, A. Huang, C. J. Maddison *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[117] A. Shrestha and A. Mahmood, "Review of Deep Learning Algorithms and Architectures," *IEEE Access*, vol. 7, pp. 53 040–53 065, April 2019.

[118] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. K. Cheung, and G. A. Constantinides, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Comput. Surv.*, vol. 52, no. 2, May 2019.

[119] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[120] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, *Handwritten Digit Recognition with a Back-Propagation Network*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, p. 396–404.

[121] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017, pMID: 28599112.

[122] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[123] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.03385, 2015.

[124] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.

[125] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 4278–4284.

[126] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, 2016.

[127] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 7132–7141.

[128] C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy, “Progressive neural architecture search,” *CoRR*, vol. abs/1712.00559, 2017.

[129] J. Fan, W. Xu, Y. Wu, and Y. Gong, “Human tracking using convolutional neural networks,” *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1610–1623, Oct 2010.

[130] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning hierarchical features for scene labeling,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, Aug 2013.

[131] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1653–1660.

[132] “NVIDIA Sets Eight Records in AI Performance,” https://blogs.nvidia.com/blog/2019/07/10/mlperf-ai-performance-records/, Online; Accessed: 31-January-2020.

[133] “NVIDIA DGX SuperPOD Delivers World Record Supercomputing to Any Enterprise,” https://devblogs.nvidia.com/dgx-superpod-world-record-supercomputing-enterprise/, Online; Accessed: 31-January-2020.

[134] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, “Fully hardware-implemented memristor convolutional neural network,” *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[135] J. W. Jang, S. Park, G. W. Burr, H. Hwang, and Y. H. Jeong, “Optimization of conductance change in $Pr_{1-x}Ca_xMnO_3$-based synaptic devices for neuromorphic systems,” *IEEE Electron Device Letters*, vol. 36, no. 5, pp. 457–459, May 2015.

[136] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale Video Classification with Convolutional Neural Networks,” in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR ’14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1725–1732.

[137] K. Steinbuch, “Die lernmatrix,” *Kybernetik*, vol. 1, no. 1, pp. 36–45, Jan 1961.

[138] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Design Trade-offs for High Density Cross-point Resistive Memory," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 209–214.

[139] Y. Wang, B. Li, R. Luo, Y. Chen, N. Xu, and H. Yang, "Energy Efficient Neural Networks for Big Data Analytics," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 345:1–345:2.

[140] J. J. Yang, D. B. Strukov, and D. Stewart R, "Memristive devices for computing," *Nature Nanotechnology*, pp. 13–24, 2013.

[141] W. J. Poppelbaum, C. Afuso, and J. W. Esch, "Stochastic computing elements and systems," in *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, ser. AFIPS '67 (Fall). New York, NY, USA: ACM, 1967, pp. 635–644.

[142] B. R. Gaines, *Stochastic Computing Systems.* Boston, MA: Springer US, 1969, pp. 37–172.

[143] S. Gupta, V. Sindhwani, and K. Gopalakrishnan, "Learning Machines Implemented on Non-Deterministic Hardware," *CoRR*, vol. abs/1409.2620, 2014.

[144] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–4.

[145] A. Armin and J. P. Hayes, "On the Functions Realized by Stochastic Computing Circuits," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. New York, NY, USA: ACM, 2015, pp. 331–336.

[146] S. Gupta and K. Gopalakrishnan, "Revisiting Stochastic Computation: Approximate Estimation of Machine Learning Kernels," 2014.

[147] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic Circuits for Real-time Image-processing Applications," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 136:1–136:6.

[148] Q. T. Dong, M. Arzel, C. Jego, and W. J. Gross, "Stochastic Decoding of Turbo Codes," *IEEE Transactions on Signal Processing*, vol. 58, no. 12, pp. 6421–6425, Dec 2010.

[149] M. L. Alomar, V. Canals, V. Martínez-Moll, and J. L. Rosselló, *Stochastic-Based Implementation of Reservoir Computers.* Cham: Springer International Publishing, 2015, pp. 185–196.

[150] J. Li, A. Ren, Z. Li, C. Ding, B. Yuan, Q. Qiu, and Y. Wang, "Towards acceleration of deep convolutional neural networks using stochastic computing," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 115–120.

[151] A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, and B. Catanzaro, "Deep Learning with COTS HPC Systems," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13. JMLR.org, 2013, pp. III–1337–III–1345.

[152] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 609–622.

[153] B. Rajendran, Y. Liu, J. Seo, K. Gopalakrishnan, L. Chang, D. Friedman, and M. Ritter, "Specifications of nanoscale devices and circuits for neuromorphic computational systems," *IEEE Transactions on Electron Devices*, vol. 60, no. 1, pp. 246–253, 2013.

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

[155] 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.

[156] G. Snider, "Instar and outstar learning with memristive nanodevices," *Nanotechnology*, vol. 22, no. 1, p. 015201, 2011.

[157] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, "A Low Energy Oxide-Based Electronic Synaptic Device for Neuromorphic Visual Systems with Tolerance to Device Variation," *Advanced Materials*, vol. 25, no. 12, pp. 1774–1779, 2013.

[158] P. Y. Chen, B. Lin, I. T. 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 *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 194–199.

[159] A. Fumarola, P. Narayanan, L. L. Sanches, S. Sidler, J. Jang, K. Moon, R. M. Shelby, H. Hwang, and G. W. Burr, "Accelerating machine learning with Non-Volatile Memory: Exploring device and circuit tradeoffs," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, Oct 2016, pp. 1–8.

[160] N. Panwar and U. Ganguly, "Variability Assessment and Mitigation by Predictive Programming in $Pr_{0.7}Ca_{0.3}MnO_3$ based RRAM," in *2015 73rd Annual Device Research Conference (DRC)*, June 2015, pp. 141–142.

[161] D. J. Seong, M. Hassan, H. Choi, J. Lee, J. Yoon, J. B. Park, W. Lee, M. S. Oh, and H. Hwang, “Resistive-Switching Characteristics of Al/ $Pr_{0.7}Ca_{0.3}MnO_3$ for Nonvolatile Memory Applications,” *IEEE Electron Device Letters*, vol. 30, no. 9, pp. 919–921, Sept 2009.

[162] P. Kumbhare and U. Ganguly, “Ionic Transport Barrier Tuning by Composition in $Pr_{0.7}Ca_{0.3}MnO_3$-Based Selector-less RRAM and Its Effect on Memory Performance,” *IEEE Transactions on Electron Devices*, vol. 65, no. 6, pp. 2479–2484, June 2018.

[163] N. Panwar, D. Kumar, N. K. Upadhyay, P. Arya, U. Ganguly, and B. Rajendran, “Memristive Synaptic Plasticity in $Pr_{0.7}Ca_{0.3}MnO_3$ RRAM by Bio-mimetic Programming,” in *72nd Device Research Conference*, June 2014, pp. 135–136.

[164] 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 Transactions on Electron Devices*, vol. 64, no. 9, pp. 3967–3970, Sept 2017.

[165] O. Bichler, M. Suri, D. Querlioz, D. Vuillaume, B. DeSalvo, and C. Gamrat, “Visual Pattern Extraction Using Energy-Efficient 2-PCM Synapse Neuromorphic Architecture,” *IEEE Transactions on Electron Devices*, vol. 59, no. 8, pp. 2206–2214, Aug 2012.

[166] S. Lim, J.-H. Bae, J.-H. Eum, S. Lee, C.-H. Kim, D. Kwon, B.-G. Park, and J.-H. Lee, “Adaptive Learning Rule for Hardware-based Deep Neural Networks Using Electronic Synapse Devices,” *ArXiv e-prints*, Jul. 2017.

[167] I. Boybat, M. Le Gallo, S. R. Nandakumar *et al.*, “Neuromorphic computing with multi-memristive synapses,” *Nature Communications*, vol. 9, no. 1, p. 2514, 2018.

[168] B. Moons and M. Verhelst, “Energy-Efficiency and Accuracy of Stochastic Computing Circuits in Emerging Technologies,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 475–486, Dec 2014.

[169] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, “An Architecture for Fault-Tolerant Computation with Stochastic Logic,” *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, Jan 2011.

[170] S. Ambrogio, P. Narayanan *et al.*, “Equivalent-accuracy accelerated neural-network training using analogue memory,” *Nature*, vol. 558, 2018.

[171] 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, July 2016.

[172] H. Jang, O. Simeone, B. Gardner, and A. Grüning, “An Introduction to Probabilistic Spiking Neural Networks,” 2019.

[173] H. Jang and O. Simeone, "Training Dynamic Exponential Family Models with Causal and Lateral Dependencies for Generalized Neuromorphic Computing," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019.

[174] B. Gardner and A. Grüning, "Supervised learning in spiking neural networks for precise temporal encoding," *PloS one*, 2016.

[175] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Energy Efficient Smartphone-Based Activity Recognition using Fixed-Point Arithmetic," *J. UCS*, vol. 19, 2013.

[176] M. T. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proceedings - Computers and Digital Techniques*, vol. 150, no. 6, pp. 403–411, Nov 2003.

[177] C. Alippi and G. Storti-Gajani, "Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning," in *IEEE International Sympoisum on Circuits and Systems*, June 1991.

[178] S. K. Esser, P. A. Merolla, J. V. Arthur *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016.

[179] G. Snider, "Instar and outstar learning with memristive nanodevices," *Nanotechnology*, vol. 22, no. 1, p. 015201, Dec 2010.

[180] G. Indiveri, B. Linares-Barranco, R. Legenstein *et al.*, "Integration of nanoscale memristor synapses in neuromorphic computing architectures," *Nanotechnology*, vol. 24, no. 38, p. 384010, sep 2013.

[181] R. Dorrance, F. Ren, Y. Toriyama, A. A. Hafez, C. K. Yang, and D. Markovic, "Scalability and Design-Space Analysis of a 1T-1MTJ Memory Cell for STT-RAMs," *IEEE Transactions on Electron Devices*, vol. 59, no. 4, pp. 878–887, April 2012.

[182] H. Yu, K. Lin, K. Lin, C. Huang, Y. Chih, T. Ong, J. Chang, S. Natarajan, and L. C. Tran, "Cycling endurance optimization scheme for 1Mb STT-MRAM in 40nm technology," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Feb 2013, pp. 224–225.

[183] R. Nebashi, N. Sakimura, H. Honjo, S. Saito, Y. Ito, S. Miura, Y. Kato, K. Mori, Y. Ozaki, Y. Kobayashi, N. Ohshima, K. Kinoshita, T. Suzuki, K. Nagahara, N. Ishiwata, K. Suemitsu, S. Fukami, H. Hada, T. Sugibayashi, and N. Kasai, "A 90nm 12ns 32Mb 2T1MTJ MRAM," in *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2009, pp. 462–463,463a.

[184] Q. Dong, Z. Wang, J. Lim, Y. Zhang, M. E. Sinangil, Y. Shih, Y. Chih, J. Chang, D. Blaauw, and D. Sylvester, "A 1-Mb 28-nm 1T1MTJ STT-MRAM with Single-Cap Offset-Cancelled Sense Amplifier and In Situ Self-Write-Termination," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 231–239, Jan 2019.

[185] H. Noguchi, K. Ikegami, K. Kushida, K. Abe, S. Itai, S. Takaya, N. Shimomura, J. Ito, A. Kawasumi, H. Hara, and S. Fujita, "A 3.3ns-access-time 71.2 μW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture," in *IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, Feb 2015.

[186] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015.

[187] C. J. Lin, S. H. Kang, Y. J. Wang, K. Lee, X. Zhu, W. C. Chen, X. Li, W. N. Hsu, Y. C. Kao, M. T. Liu, W. C. Chen, YiChing Lin, M. Nowak, N. Yu, and Luan Tran, "45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell," in *2009 IEEE International Electron Devices Meeting (IEDM)*, Dec 2009, pp. 1–4.

[188] S. Moradi and R. Manohar, "The Impact of On-chip Communication on Memory Technologies for Neuromorphic Systems," *Journal of Physics D: Applied Physics*, vol. 52, no. 1, p. 014003, Oct 2018.

[189] S. R. Kulkarni, D. V. Kadetotad, S. Yin, J. Seo, and B. Rajendran, "Neuromorphic hardware accelerator for snn inference based on stt-ram crossbar arrays," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 438–441.

[190] L. Steffen, D. Reichard, J. Weinland, J. Kaiser, A. Roennau, and R. Dillmann, "Neuromorphic Stereo Vision: A Survey of Bio-Inspired Sensors and Algorithms," *Frontiers in Neurorobotics*, vol. 13, p. 28, 2019.

[191] L. Camunas-Mesa, C. Zamarreno-Ramos, A. Linares-Barranco, A. J. Acosta-Jimenez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An Event-Driven Multi-Kernel Convolution Processor Module for Event-Driven Vision Sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 2, pp. 504–517, Feb 2012.

[192] H. Akolkar, C. Meyer, X. Clady, O. Marre, C. Bartolozzi, S. Panzeri, and R. Benosman, "What Can Neuromorphic Event-Driven Precise Timing Add to Spike-Based Pattern Recognition?" *Neural Computation*, vol. 27, no. 3, pp. 561–593, 2015, pMID: 25602775.

[193] A. E. G. Huber and S. Liu, "On Send-on-Delta sampling of bandlimited functions," in *2017 International Conference on Sampling Theory and Applications (SampTA)*, July 2017, pp. 422–426.

[194] M. Miskowicz, “Send-On-Delta Concept: An Event-Based Data Reporting Strategy,” *Sensors (Basel, Switzerland)*, vol. 6, no. 1, pp. 49–63, Jan 2006.

[195] J. D. Reiss, “Understanding Sigma-Delta Modulation: The Solved and Unsolved Issues,” *J. Audio Eng. Soc*, vol. 56, no. 1/2, pp. 49–64, 2008.

[196] Z. Pan, Y. Chua, J. Wu, M. Zhang, H. Li, and E. Ambikairajah, “An Efficient and Perceptually Motivated Auditory Neural Encoding and Decoding Algorithm for Spiking Neural Networks,” *Frontiers in Neuroscience*, vol. 13, p. 1420, 2020.

[197] S. Yin, S. Venkataramanaiah, G. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, and J. sun Seo, “Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations,” in *2017 IEEE Biomedical Circuits and Systems Conference, BioCAS 2017 - Proceedings*, vol. 2018-January, 3 2018, pp. 1–4.

[198] S. Chakraborty, A. Garg, and M. Suri, “True Random Number Generation From Commodity NVM Chips,” *IEEE Transactions on Electron Devices*, pp. 1–7, 2020.

[199] C. Huang, W. C. Shen, Y. Tseng, Y. King, and C. Lin, “A Contact-Resistive Random-Access-Memory-Based True Random Number Generator,” *IEEE Electron Device Letters*, vol. 33, no. 8, pp. 1108–1110, Aug 2012.

[200] H. Jiang, D. Belkin, S. E. Savel’ev *et al.*, “A novel true random number generator based on a stochastic diffusive memristor,” *Nature Communications*, vol. 8, no. 1, p. 882, 2017.

[201] T. Tanamoto, N. Shimomura, S. Ikegawa, M. Matsumoto, S. Fujita, and H. Yoda, “High-Speed Magnetoresistive Random-Access Memory Random Number Generator Using Error-Correcting Code,” *Japanese Journal of Applied Physics*, vol. 50, no. 4, p. 04DM01, apr 2011.