Towards energy-efficient hardware acceleration of memory-intensive event-driven kernels on a synchronous neuromorphic substrate

Saunak Saha
Iowa State University

Towards energy-efficient hardware acceleration of memory-intensive event-driven kernels on a synchronous neuromorphic substrate

by

Saunak Saha

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Electrical Engineering (Very Large Scale Integration)

Iowa State University

Ames, Iowa

2019

DEDICATION

To the relentless advancement of science

and the teachers who make it possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I am deeply indebted to all my colleagues, faculty members and staff in the department for their assistance in my work. I would like to express my heartfelt regard to Dr. Joseph Zambreno for the immense support that he has extended towards me, for continually providing accomplished academic guidance and for steering me in the direction of quality research. I convey my sincerest gratitude to Dr. Henry Duwe whose constant support and technical critique was indispensable in my work. The long fruitful discussions with Dr. Duwe have been invaluable, I feel, in my development as a student of engineering. Particular appreciation is expressed to Dr. Nathan Neihart for providing insight and advice in an area of ineptitude and to Dr. Phillip Jones for his regular reviews and questions about my work. Lastly, I also thank the committee for their meticulous review of this thesis, and express my gratefulness to my family without whose patient forbearance this work could not have been completed.

ABSTRACT

Spiking neural networks are increasingly becoming popular as low-power alternatives to deep learning architectures. To make edge processing possible in resource-constrained embedded devices, there is a requirement for reconfigurable neuromorphic accelerators that can cater to various topologies and neural dynamics typical to these networks. Subsequently, they also must consolidate energy consumption in emulating these dynamics. Since spike processing is essentially memory-intensive in nature, a significant proportion of the system's power consumption can be reduced by eliminating redundant memory traffic to off-chip storage that holds the large synaptic data for the network. In this work, I will present CyNAPSE, a digital neuromorphic acceleration fabric that can emulate different types of spiking neurons and network topologies for efficient inference. The accelerator is functionally verified on a set of benchmarks that vary significantly in topology and activity while solving the same underlying task. By studying the memory access patterns, locality of data and spiking activity, we establish the core factors that limit conventional cache replacement policies from performing well. Accordingly, a domain-specific memory management scheme is proposed which exploits the particular use-case to attain visibility of future data-accesses in the event-driven simulation framework. To make it even more robust to variations in network topology and activity of the benchmark, we further propose static and dynamic network-specific enhancements to adaptively equip the scheme with more insight. The strategy is explored and evaluated with the set of benchmarks using a software simulation of the accelerator and an in-house cache simulator. In comparison to conventional policies, we observe up to 23% more reduction in net power consumption.

CHAPTER 1. INTRODUCTION

1.1 Artificial Neural Networks

Cortical information processing in mammals form the rudimentary principles of deep learning, albeit, in a highly abstracted manner. State-of-the-art Artifical Neural Networks (hereafter referred to as ANNs) are mathematical abstractions of how organic matter processes sensory information in the brain. Such a mathematical model is usually tuned specifically to objective tasks like pattern recognition or data generation. Therefore, ANNs perform extremely well in their own specific domains. However, in doing so, they realize a large amount of resources in terms of hardware real-estate and energy consumption. Recently, there has been a steep growth in the size of neural networks [11, 2, 12] owing to their usage in extremely complicated, real-time and data-intensive applications of perception and generation [13, 14, 15, 16, 17, 18]. Fig. 1.1 shows recently developed ANNs and their computational intensity on the ImageNet dataset [19]. Such a growth has two major implications. Firstly, ANN applications can be made commercially feasible only by deploying in the cloud because they require significant processing power and performance. Therefore, resource-constrained embedded and IoT devices cannot afford to process these large algorithms at the edge. These correlations are in direct conflict with a high demand of intelligent algorithms in live embedded autonomous agents and mobile devices. Secondly, this unprecedented growth of complexity in ANNs coupled with a relatively sluggish growth of silicon process technology in the post Moore's law era have together rendered ANN processing to be quite painstaking in general purpose hardware. The computational primitive in deep ANNs is typically either General Matrix-Vector multiply (GEMV) or General Matrix-Matrix multiply (GEMM) which are both massively data-parallel but highly memory-intensive operations. Although Graphics Processing Units (GPUs) can exploit a very high degree of data level parallelism [20], the sequential nature of traditional von Neumann architectures consistently fall short in realizable memory bandwidth. In sum, there

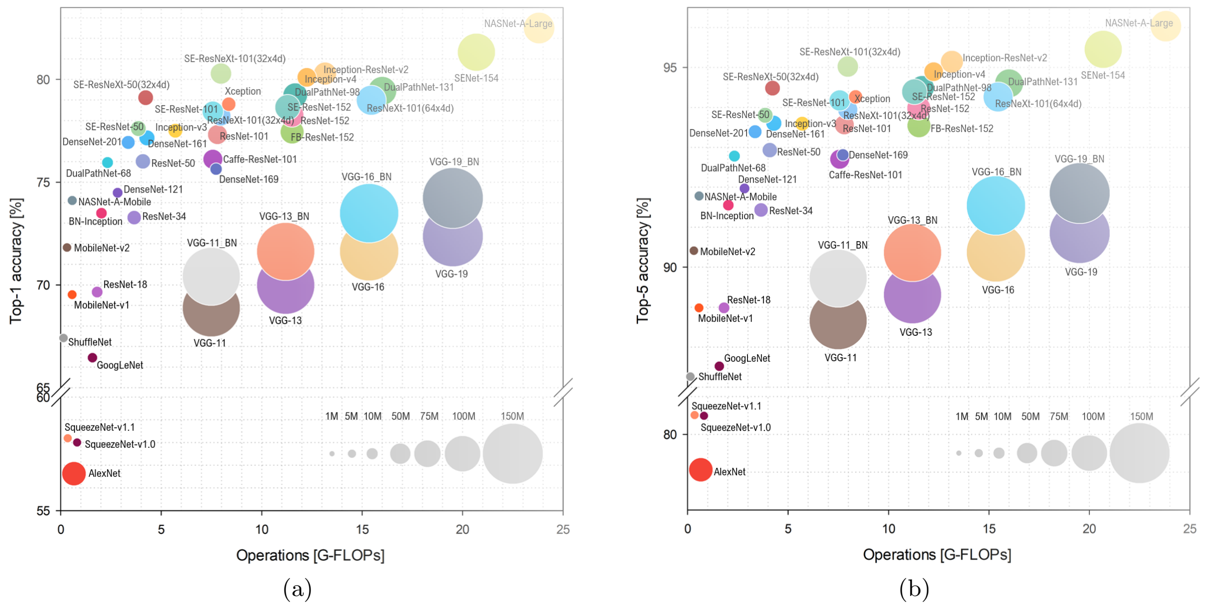
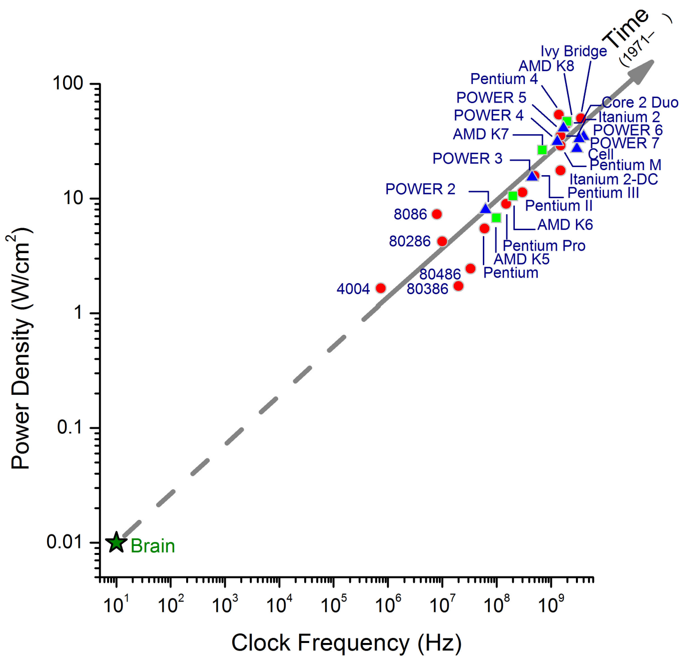


Figure 1.1: (a) Top-1 and (b) Top-5 accuracy in the ImageNet dataset for recent deep ANNs vs. computational complexity (giga-floating point operations per second (GFLOPs) required for a single example inference. The size of each ball is proportional to the complexity of the ANN it represents (figure from [2])

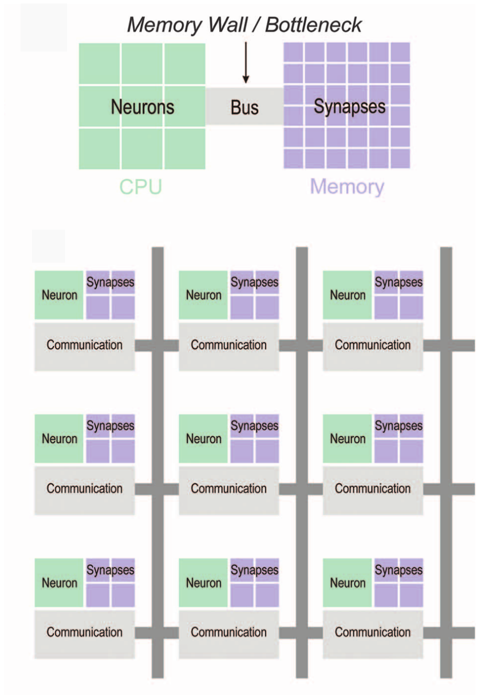
is a need for improved latency and energy-efficiency in ANN processing as well as compact special purpose hardware that can achieve the required memory bandwidth in edge-processing applications.

1.2 Hardware Acceleration

To overcome the inefficiency of sequential computers in processing ANN kernels, interest has gradually shifted towards hardware accelerators. These are special purpose neural network processors that employ dedicated hardware units to compute ANN primitives. For instance, [21] uses a dedicated global Multiply and Accumulate (MAC) unit and a separate activation bank to compute ANN layer activation vectors and the process repeats for all layers in a single forward pass for ANN inference. [22] uses a network of processing elements each having a local memory space, a MAC unit and associated control circuitry to collocate processing and memory in an efficient dataflow for



(a)



(b)

Figure 1.2: (a) Power Density of the brain is orders of magnitude lower than the exponentially rising density of general-purpose processing systems. (b) A conceptual understanding of how the von-neumann bottleneck can be distributed among individual processing elements in neural networks (figures from [3])

inference in deep networks. Similarly, all accelerators use a certain amount of specificity in their hardware that makes them highly selective to, and hence highly efficient in, ANN processing. Accelerators have been designed to tackle the inference in offline-trained ANNs [21, 22, 23, 24, 25, 26, 27] as well as to reduce dependency on GPUs by supporting on-chip online training [28, 29, 30, 31, 32]. There has been a considerable effort to make accelerator hardware extremely compact and low-power so as to make edge processing possible for embedded and mobile devices [33, 34]. Although accelerators have succesfully displaced general purpose hardware to some extent, majority of ANN processing still eludes the edge. In spite of being inspired by the computation of the brain, ANNs always end up consuming orders of magnitude more energy. As a reference, Fig. 1.2a compares the power-density of the brain with commercial general-purpose hardware. Although accelerators strive to eliminate the bottleneck by homogenously distributing processing and memory as shown in Fig. 1.2b, a comparable efficiency has not been achieved. Therefore, there has been a sincere drive towards how to make the basic fundamentals of ANNs more akin to biophysical computation.

1.3 Spiking Neural Networks

Evidence in rudimentary neuroscience progressively hinges towards lower abstraction levels in computational neural network models [35]. Building upon experimental studies on real cortical cultures, biologically plausible models collectively known as Spiking Neural Networks (hereafter referred to as SNNs) have emerged. Fundamentally, the inefficiency of the ANN algorithms vis-a-vis the efficiency of an equivalent SNN can be summarized in the following arguments:

- ANNs use simple processing elements a.k.a. *perceptrons* [36] that perfectly integrate its inputs and apply an analog nonlinear activation if the input exceeds a bias. Biological processing elements a.k.a. *spiking neurons* [37] are imperfect and noisy integrators of input synaptic currents and activate to an all-or-nothing voltage spike upon exceeding a threshold.

- Consequently, ANN perceptrons continuously communicate their activations leading to expensive MAC operations at every discrete timestep while spiking neurons communicate their spikes only when their membrane voltage exceeds the spiking threshold.

- ANNs are generally trained using *error backpropagation* [38] which is a supervised learning algorithm and requires large volumes of labeled data to generate error functions. The basic underlying primitive is *gradient descent* which is especially difficult to model in hardware. SNNs are trained in an unsupervised manner [39], require no gradient computation and is consistent with neuroscientific evidence against a backward pass of error signals in real neurons [40].

- While ANNs require special topologies to learn time-varying features, SNNs have an inherent temporal aspect to their processing since inputs are exposed for a finite time window and generate input spike *trains*. However, SNNs communicate with the environment only using these spike trains that typically require some wrapper hardware [41, 42] or software around the underlying neural network accelerator.

As a direct consequence, biologically plausible SNNs are much closer in philosophy to the computation in biological brains. They give great insight into greater neuroscientific understanding and assist greatly in medical investigation [35]. In addition, the very nature of local computation and sparse communication lead to ultra-low energy consumption, superior error and noise tolerance when deployed on dedicated hardware. As a limitation, they are comparatively much poorer in performance in terms of accuracy in pattern recognition tasks when compared to ANNs. To harvest the best of both worlds, many studies have attempted to hybridize the ANN and SNN approach by trading off some biological plausibility for gains in accuracy. This has been achieved by converting a fully trained ANN into an equivalent SNN for inference [43, 44, 45] or by augmenting the spike signal to make it differentiable [46] for gradient computation. As a result, close to ANN accuracy levels have been demonstrated with very low energy footprint. In this context, it is important to recognize both biologically plausible dynamics and computationally efficient dynamics because any answers to the nature of computation in the brain will come through concerted investigation of how biological systems work and also what leads to large scale learning capability. To this end, large-scale models have been produced [47] with a sound mathematical basis underlying them [48] and experimentally demonstrating their ability to perform complex visuomotor tasks. But perhaps, the most important factor fueling this research as well as facilitating their deployment in resource-constrained embedded applications is the emergence of dedicated brain-inspired hardware that is able to efficiently process SNNs.

1.4 Neuromorphic Computing

The term 'Neuromorphic Computing' was first introduced by Carver Mead in 1990 [49]. Over the last 3 decades, the agenda has been to conceptualize and design substrates that are able to emulate the dynamics of biological networks so as to perform energy efficient, fault-tolerant and real-time processing of neural information reminiscent of the mammalian cortex [50]. This includes

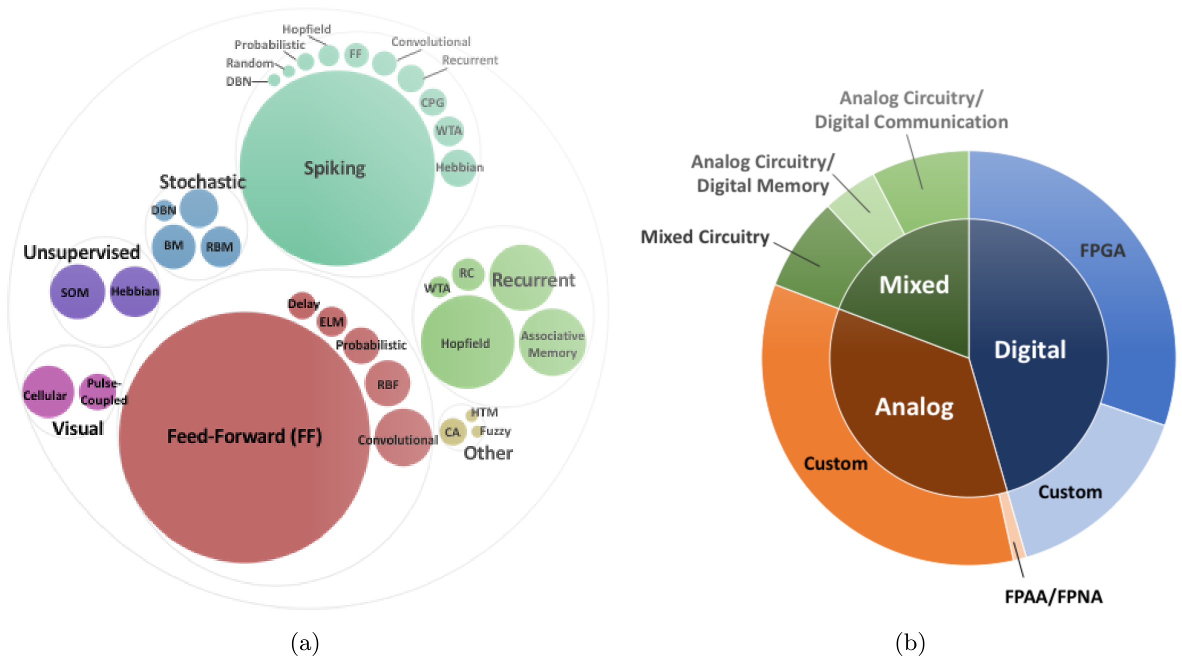


Figure 1.3: Graphical representation of (a) different neural network models on special purpose hardware that was termed 'neuromorphic' sized according to the number of relevant articles found in the literature while and (b) the relative proportions of analog, digital and mixed-signal design philosophies among these implementations (figures from [4])

systems that generate neural representations of real-world phenomenon like [41, 42, 51, 52, 53] as well as systems that process these representations into meaningful pattern recognition semantics or subsequent motor output [4]. There are roughly two major areas, in terms of their motivations, imminent applications and design philosophy, that have fostered within the neuromorphic research community. [54] argues that in the face of thwarted transistor scalability, the only way we can support a single electron-lane channel is by emulating analog behavior typical of ion channels in our brain so as to effectively tolerate noise and thereby squelching minimum energy to eliminate it. Coupled with ultra-low power subthreshold conduction in MOS Devices [55, 56], systems like [57, 58, 59, 60, 61] follow an Analog computation - Digital Communication model to achieve very high realism. The other large chunk of neuromorphic systems like [62, 63, 64, 65, 66, 67] follow a fully digital approach.

It has been noted that digital systems fail to realize the full possibilities of energy efficiency that comes from the naturally analog local computation in biological neural networks. However, this work focuses on a fully digital implementation of a Neural Processing Engine that can support reconfigurable dynamics and topologies of SNNs. The digital end of the spectrum provides us with some advantages when it comes to targetting large scale production in embedded and IoT devices:

First, analog circuits are intrinsically plagued by Process, Voltage and Temperature (PVT) variations and these hinder the capability to extensively scale the devices to state-of-the-art processes and maintain stable working conditions especially when neural applications can be dauntingly large in terms of processing and communication infrastructure required. Second, as mentioned by [62], having a fully digital implementation guarantees a one-one equivalence with a software stack or ecosystem on top of the chip that can support easy mapping of reconfigurable networks efficiently. Lastly, a digital system can be exhaustively evaluated while still under design because of easy availability of CAD tools and quick turnaround time for testing and verification when compared to mixed-signal systems.

1.5 Contributions

Since neural response latency is orders of magnitude higher than the latency of digital circuits, digital neuromorphic accelerators can usually realize faster-than-real-time sensory processing when used to emulate similar timescales as their biological counterparts. Since SNNs are primarily used to cut down energy requirements to ultra-low budgets, the usefulness of such an accelerator is critically dependent on its efficiency. In this work, the concentration is on energy consumption and attempt to control the same without incurring noticeable losses in performance. This contributions of this work can be summarized in the following points:

- CyNAPSE, a fully digital synchronous accelerator core that can efficiently emulate SNN inference with reconfigurable neuronal dynamics and topology, is designed and implemented.

- Since SNNs are primarily a memory-intensive operation, memory access patterns of a diverse set of SNN workloads is studied and it is observed that a very large percentage of the power consumption in accelerating these workloads results from off-chip memory accesses.

- Accordingly, a domain-specific memory management scheme is proposed to reduce off-chip traffic by exploiting temporal locality and cutting off redundant memory accesses. As a result, 13-44% improvement in total power consumption is obtained over the baseline.

This thesis is organized in the following way. Chapter 2 delves into the biophysical, mathematical and some circuit-level background required to understand neuronal dynamics in SNNs. Chapter 3 describes the SNN benchmarks that were used for this work. Chapter 4 meticulously describes the microarchitecture specification of the CyNAPSE neural processing system and also its scheduling, programming and implementation. Chapter 5 motivates and describes the novel memory management scheme, discusses the experimental setup and presents evaluation results. Chapter 6 talks about the scope of future work and concludes.

CHAPTER 2. BACKGROUND

This chapter is dedicated to provide the minimal background required to understand neural computation as it happens in organic neural assemblies and the mathematical modeling associated with the same. A small digression to introduce circuit design for the relevant mathematical models is also provided.

2.1 Biophysical background

The *neuron* is the basic unit of neuroanatomy and is described concisely in Fig. 2.1. The mammalian brain is a spatially dense distribution of neurons of various types and functions. Typically, a neuron consists of a cell-body or *soma* and a large number of neural *processes* to communicate with other neurons. Most of these processes emerge in the form of *dendrites* for communication within local clusters of neurons. Sometimes, processes develop into long cable-like fibres called *axons* to facilitate communication at long spatial separations. Neurons form connections and clusters in various ways. Fig. 2.2 shows sketches of some of the typical neural assemblies observed in mammalian brains. The human brain is an assembly of about 10^{11} neurons and about 10^{14} connections which is a testimony to the complexity of neural circuitry. In this section, I will gradually discuss the electrochemical processes that constitute the working of a typical neuron circuit without going into much detail.

2.1.1 Membrane capacitance

The thin *bilayer membrane* of a neuron is the centre of all electrochemical activity and electrically isolates the neuron from the extracellular fluid [68, 69, 6, 1]. Owing to the high polarizability of water, the energy of an ion like sodium has a much higher energy in the interior of the membrane than outside. This energy barrier prevents ions from entering the bilayer membrane under normal

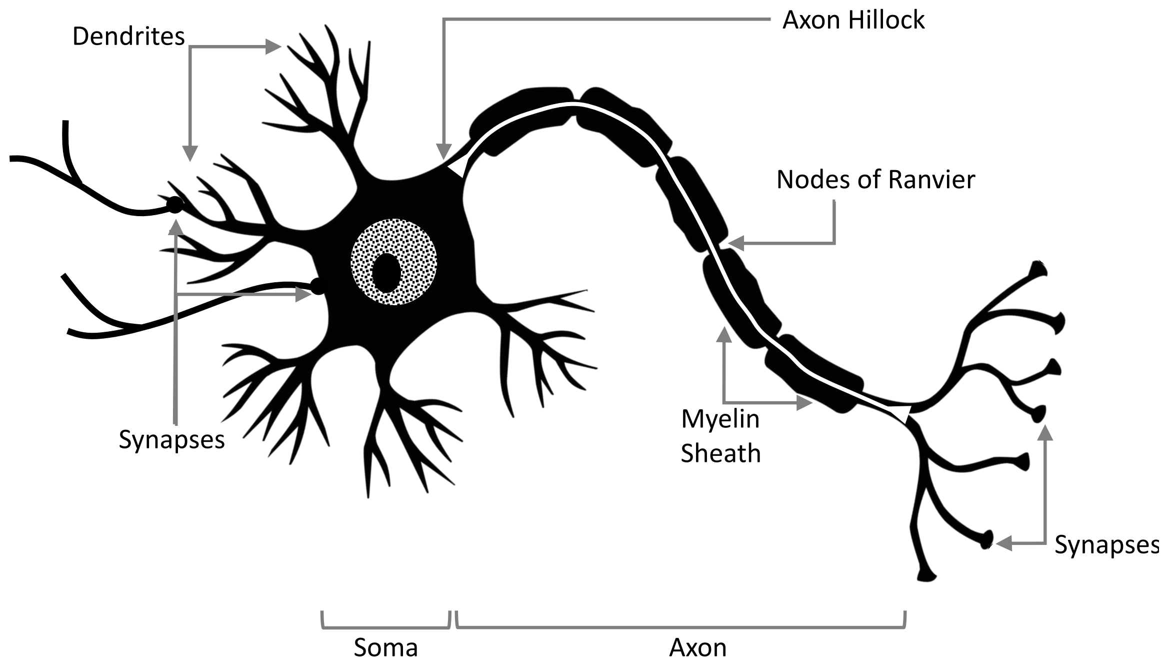
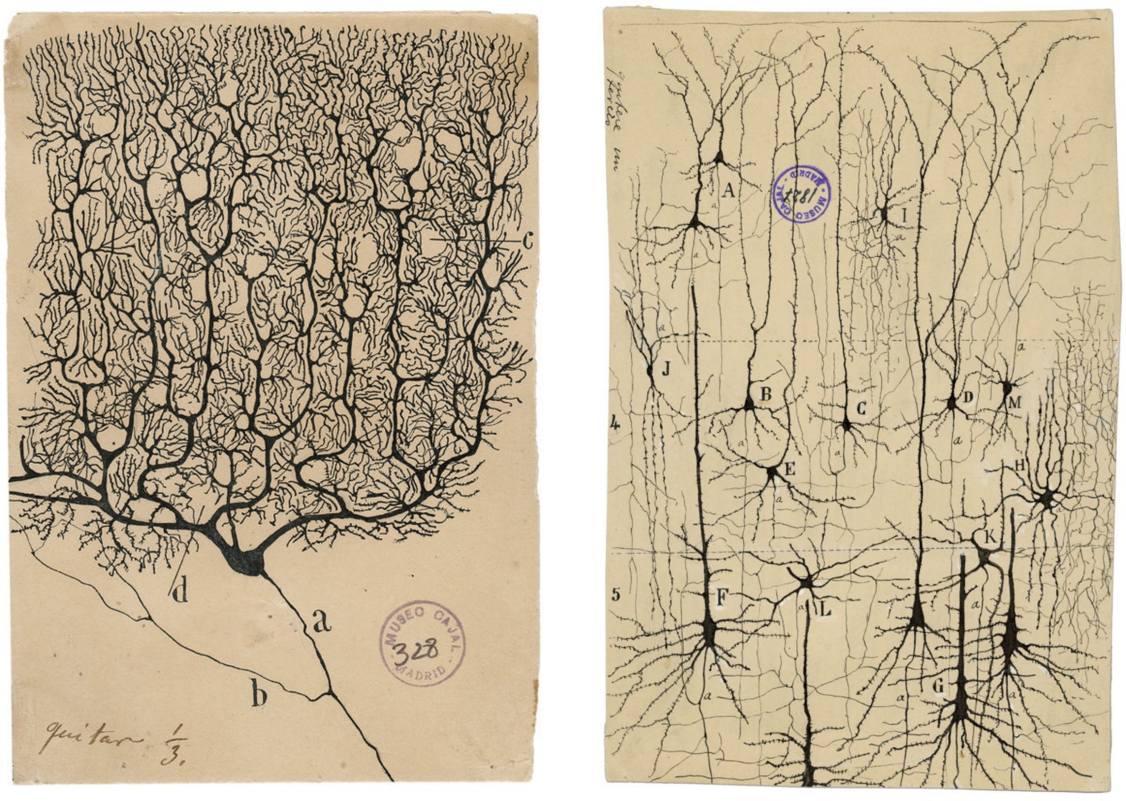


Figure 2.1: Physiology of a typical single neuron. Synaptic inputs via dendritic tree or direct axons are integrated on the capaciance of the soma. When it exceeds threshold, an action potential is generated at the axon hillock and propagated through the cable. Capacitance of the axon is reduced by the myelin sheaths while nodes of ranvier regenerate the attenuated pulse amplitude at intervals

conditions. However, certain *metabolic pumps* actively expel Na^+ ions and import K^+ ions into the cytoplasm of the soma. This results in a high K^+ concentration within the membrane and a high Na^+ concentration outside (see Table 2.1). This leads to a diffusion of ions due to a concentration gradient and a drift of ions due to the opposing electric field. The equilibrium potential whereby these two currents counterbalance each other is known as the *reversal potential* of that particular ion. Similarly, all ionic species have their own gradients and so, their own reversal potentials, and their own *ion-specific conductances* that contribute to the net membrane current as:

$$I = (V_K - V)G_K + (V_{Na} - V)G_{Na} + (V_{Cl} - V)G_{Cl} \tag{2.1}$$

where G_{ion} and V_{ion} are the conductances and reversal potentials for common ionic species found in neural processes [1] while V refers to the instantaneous membrane voltage. A membrane potential higher than V_K results in a net positive current leaving the membrane. Similarly, a membrane



(a) (b)

Figure 2.2: Impressions by Santiago Ramón y Cajal of (a) Purkinje cells in the cat cerebellar cortex and (b) Pyramidal neurons in the inferotemporal cortex of the human brain. Taken from [5] where Cajal's drawings have been redeciphered as works of art

potential lower than V_{Na} results in a net positive current into the membrane. Assuming that the Cl$^-$ ion current is negligible [70], the membrane voltage V_0 for zero net current is given by:

$$V_0 = \frac{V_K G_K + V_{Na} G_{Na}}{G_K + G_{Na}} \tag{2.2}$$

V_0 is called the *resting potential* of the neuron. Using the reversal values reported in Table 2.1 and G_K as 20 times G_{Na} [70], V_0 is calculated as approximately *-85 milivolts* although, experimental conditions have a profound effect on the absolute value. In neuroscientific terminology, this negatively charged resting state is known as a *polarized* state (not to be confused with electric polarization in a dielectric medium). If an external agent (as discussed later) injects ionic currents

Ionic species	Ionic concentration (mM/l)		Reversal potential (mV)
	Intracellular	Extracellular	
K+	400	10	-92
Na+	50	460	55
Cl-	40	540	-65

Table 2.1: Ionic concentrations and reversal potentials observed in the giant axon of *Loligo* (data from [1])

into the membrane, it is called an *excitatory* signal while an outward current is called an *inhibitory* signal. They are said to *depolarize* and *hyperpolarize* the membrane respectively by causing finite excursions from its resting potential. Precisely, this is done by manipulating one or more of the ionic conductances. As has been observed in a typical axonal fibre [71], both Na^+ and K^+ conductances exponentially rise with the membrane potential. This is key to the generation of a spike.

2.1.2 Action potential

Small amounts of excitatory ionic currents when injected into the membrane charge up the membrane capacitance and depolarize the membrane potential slowly. Concomitantly, the sodium conductance in the axon quickly rises with the voltage of the membrane. Upon reaching a certain depolarized state known as the *threshold potential*, the sodium conductance achieves a self-reinforcing explosion. Due to this, there is a large inward current (owing to high extracellular concentration of Na^+) until the point when the Na^+ reversal is reached. Thereafter, a rather slowly rising but sustained K^+ conductance pulls down the membrane potential via a net outward current of K^+ ions. This sustained current remains in effect until the membrane returns to the neighborhood of the resting potential. However, since the K^+ conductance is slow to react to changes in membrane potential, for sometime following the spike there is an extraneous net outward current that hyperpolarizes the membrane below rest and it is impossible for external agents to depolarize the capacitance at this time. This period of hyperpolarization is called the *refractory period*. This entire episode is collectively termed as the *action potential* generation and abstracted as an all-or-nothing *spike* signal used for all neural computation. Physiologists Alan Hodgkin and Andrew Huxley received the Nobel Prize in 1963 for deciphering and characterizing the action potential in a giant squid axon [71, 70, 72, 73, 74].

2.1.3 Ion channels

It was observed that the Na^+ current into the membrane changes in discrete steps [75]. This discrete behavior is a result of *channels*, molecular aggregates that are selectively permeable to a specific ion, and a *population* of these entities lead to net inward or outward currents. The height of a discrete step is same for a specific membrane voltage and changes linearly with the difference between the membrane potential and the reversal potential of the specific ion. This suggests that the individual behavior of an ion-channel is *ohmic* in nature. On the other hand, the number of steps and width of each step vary exponentially. This is because the rate of opening and the rate of disappearance of these channels are both exponential in voltage and the discrete changes in current is a result of a balance in open and closed ion channels. This explains the exponential voltage-conductance relationship of ionic species in the neural processes [76, 77, 78, 79].

2.1.4 Synapses

All the biophysical machinery described so far helps in generation and propagation of action potentials within a single neuron. This is the underlying mechanism of *communication* in the neural circuitry. However, *computation* requires altering the potential across one membrane through electrochemical activity in a different membrane (not unlike the transistor operation). This provides for the basic computation in all neural processing and the *synapses* are responsible. I will present here a lucid account of essential synaptic behavior that generates active control of the postsynaptic membrane due to presynaptic activity as shown in Fig. 2.3. For detailed accounts of synaptic circuitry, readers can refer to [80, 81].

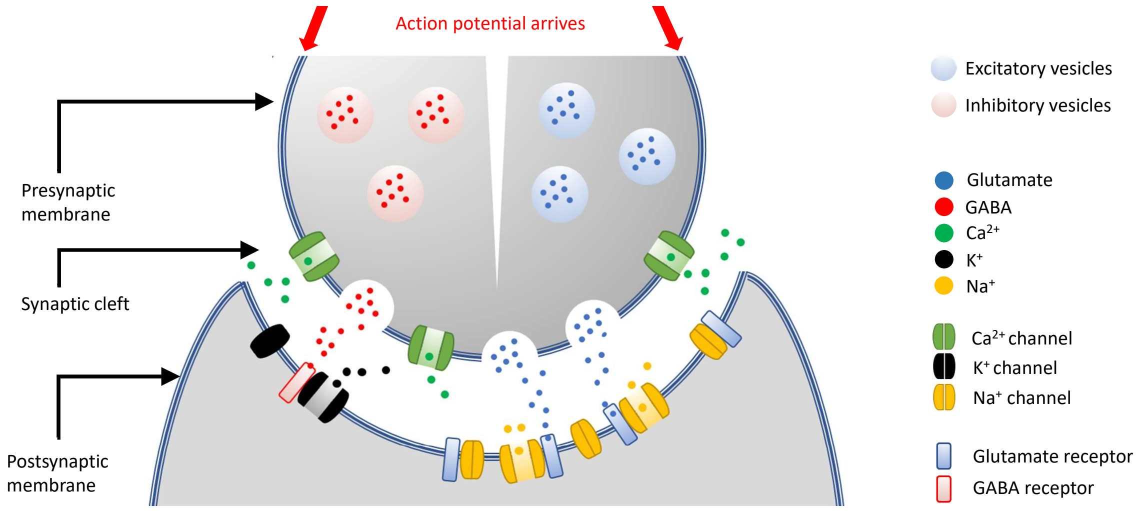


Figure 2.3: Basic operations of a synaptic microcircuit. Upon arrival of an action potential, Ca^{2+} channels open up, vesicles with neurotransmitters are released binding to specific receptors thus opening specific ion channels of the postsynaptic neuron

A depolarization of the axonal fibre causes the concerned synapses to allow opening of calcium channels across the *synaptic cleft*, a seperation of two membranes immersed in the extracellular fluid. As a result, Ca^{2+} ions flow into the presynaptic membrane. This excitation causes release of *neurotransmitter* molecules contained in *vesicles* of different kinds leading to different synaptic species. Neurotransmitters are therefore released in *quanta* of molecules in the vicinity of post-synaptic membrane and enforces opening of ion-channels in the target. The specific ion-channels opened in the target depends on the receptors at post-membrane that these molecules bind to. *Glutamatergic* receptors like *α-Amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid (AMPA)* and *N-methyl-D-aspartate (NMDA)* are responsible for opening Na^{+} channels leading to depolarization of the target membrane, causing *Excitatory post-synaptic currents* or EPSCs. Conversely, some neurotransmitters activate receptors like *Gamma-Aminobutyric acid (GABA)* that lead to opening of K^{+} ion channels and thus, *Inhibitory post-synaptic currents* or IPSCs polarizing the target-membrane further towards rest. The rate of opening of ion channels is, again, exponentially dependent on the presynaptic voltage which triggers the onset of this entire series of electrochemical events.

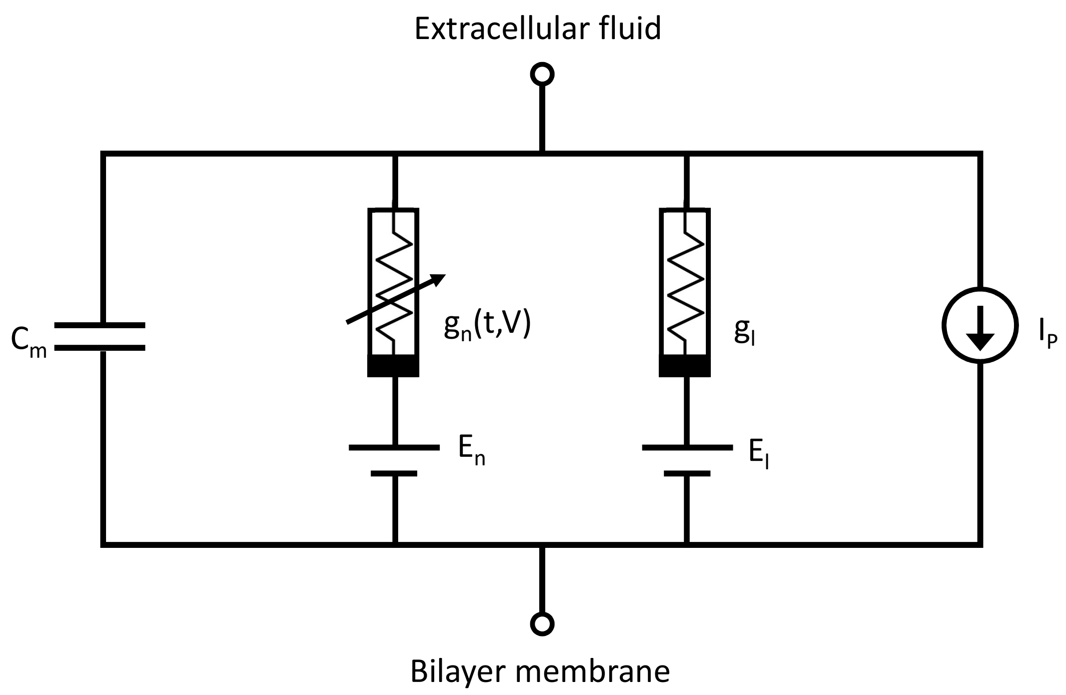


Figure 2.4: Schematic of the Hodgkin-Huxley neuron model. The lipid membrane is represented by the capacitance C_m; voltage-gated ion channels and leak channel are represented by g_n (nonlinear) and g_l (linear) respectively; the corresponding ionic gradients are modeled by E_n and E_l; I_p represents the ionic pumps that maintain background ionic concentration

The dendritic tree of a neuron typically forms a large variety of synaptic connections with axonal fibres from distant neurons or dendritic connections of clustered neurons thus leading to *neural networks* that collectively define the computation of a population of neurons. It is also worth mentioning that the quantal release of neurotransmitter vesicles is not characterized by a constant synaptic strength. In fact, it is the basic site of learning in neural circuits. Biologically plausible learning will be the focus of a future work, but for interested readers, [82, 83, 84, 85, 86, 87] provide biophysical and phenomenological accounts of synaptic learning.

2.2 Mathematical models

2.2.1 The Hodgkin-Huxley model

The behavior of real neurons described so far were very aptly captured by the *Hodgkin-Huxley neuron model*, a mathematical framework for highly detailed neural modeling [74]. Fig. 2.4 shows a simple schematic overview of the model. Mathematically, the current through the capacitance I_c is given by:

$$I_c = C_m \frac{dV_m}{dt} \tag{2.3}$$

where V_m is the membrane voltage. Further, a particular ion-channel current I_{ion} is given by:

$$I_{ion} = g_{ion}(V_m - V_{r_{ion}}) \tag{2.4}$$

where g_{ion} and $V_{r_{ion}}$ are the conductance and reversal potentials of the particular ionic species. Assuming only Na$^+$ and K$^+$ channels to be of importance, the total membrane current can be given by:

$$I = C_m \frac{dV_m}{dt} + g_K(V_m - V_{r_K}) + g_{Na}(V_m - V_{r_{Na}}) + g_l(V_m - V_l) \tag{2.5}$$

where g_l and V_l are the constant leak conductance and leak reversal potential respectively that model flow of current due to the membrane's natural permeability to background ionic concentrations. Hodgkin and Huxley also characterized the behavior of voltage-gated ion channels by strategically manipulating the extracellular fluid ionic concentrations and studying them individually [71]. As a result, they postulated a set of mathematical axioms to exhaustively model the non-linear ion-channel dynamics and linear leak dynamics of the neuron membrane. These equations are depicted below:

$$I = C_m \frac{dV_m}{dt} + \overline{g_K} n^4 (V_m - V_{r_K}) + \overline{g_{Na}} m^3 h (V_m - V_{r_{Na}}) + \overline{g_l}(V_m - V_l) \tag{2.6}$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n \tag{2.7}$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m \tag{2.8}$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h \tag{2.9}$$

where I represents membrane current per unit area. $\overline{g}$ are maximum conductance values for voltage-gated ion and linear leak channels. m, n and h are *gating variables* that point to K^+ and Na^+ channel activations and Na^+ channel inactivation respectively. The α and β parameters are the voltage-dependent rate constants for the ion-channels. They can be expressed, in general as:

$$\alpha_p(V_m) = p_\infty(V_m)/\tau_p \tag{2.10}$$

$$\beta_p(V_m) = (1 - p_\infty(V_m))/\tau_p \tag{2.11}$$

for p = (m,n,h) where p_∞ and $(1 - p_\infty)$ are the steady-state activation and inactivation probabilities for each ion-channel. The above dynamics and its extensions [88, 89, 90] represent faithful neural models that plausibly model, to a great extent, measurements from the voltage-clamp experiments [91]. However, simulation of a large-scale array of these neuron models in complicated network topologies quickly becomes intractable with available computing resources. Motivated by this, there was a gradual interest in developing more tractable models that were able to model biologically plausible dynamics to some extent and also support large scale simulation in software or large scale hardware realization by providing simpler math and compact circuits, respectively. I will discuss these models next.

2.2.2 Phenomenological models

Phenomenological model of neurons compromise some of the biological detail by simplifying some of the more intricate model parameters, making them more suitable for large-scale simulations or emulations. Instead of modeling actual electrochemical processes, such models attempt to empirically reproduce the logical order of phenomenon in generation and propagation of action potentials. An early example of a phenomenological model is the *Fitzugh-Nagumo (FHN) model* [92] that describes the neuron membrane as a system of two ordinary differential equations depicting a dynamical system. Other extensions include [93, 94, 95] and a comparative study of models exist in [96].

A rather popular and simple neuron model is the *Integrate and Fire (IF)* neuron model first postulated by Louis Lapicque [97]. It is simply given by the equation:

$$I(t) = C_m \frac{dV_m}{dt} \tag{2.12}$$

whereby the time-varying membrane current $I(t)$ results from integrating membrane voltage V_m across the capacitance C_m. It is a *point neuron* model that ignores the spatial variability of the dendritic integration process as well as the ion-channel activation and inactivation dynamics. The action potential generation results simply from a thresholding behavior, to which a refractoy period can be added for more plausibilty, as follows:

$$f(I) = \frac{I}{C_m V_{th} + t_{ref} I} \tag{2.13}$$

where f is the firing frequency of the neuron dependent on membrane current I. Although it spares theoreticians from the strenuous parameter fitting required in faithful models, the glaring shortcoming of the IF model is the eternal retention of memory due to the absence of leak conductance which is unlikely in real neural systems.

This problem was solved by a *Leaky Integrate and Fire (LIF)* neuron model by introducing a 'leak' term to the otherwise simple point neuron model that was sufficiently simple to simulate in large assemblies [98, 99]. A simple mathematical expression of the LIF model can be given by:

$$I(t) - \frac{V_m}{R_m} = C_m \frac{dV_m}{dt} \tag{2.14}$$

Now the membrane is associated with finite resistance R_m which makes the membrane potential decay or *leak* in the absence of any input current stimulus through the same membrane capacitance C_m. As a result of this imperfect integration that is typical of ion-channel dynamics in real neurons, the thresholding behavior can be denoted in the frequency-domain as follows:

$$f(I) = \begin{cases} 0, & I \leq I_{th} \\ [t_{ref} - C_m R_m \log(1 - \frac{V_{th}}{I R_m})]^{-1}, & I > I_{th} \end{cases} \tag{2.15}$$

The LIF neuron model has served as the focal point of many large scale simulations and hardware realizations. Therefore, it has received great attention from the computational neuroscientists.

Naturally, there have been attempts to increase its plausibility without affecting its efficiency and a number of specialized neuron models have come up that model one or more observed neural behaviors [96] like the *Quadratic IF* [100], the *Exponential IF* [101] and the *Adaptive exponential IF* [102]. Since our focus is to cater to a wide array of predominantly used neural models, prescriptions from [103, 104, 82, 105, 106] is closely followed and a *generalized* LIF model is adopted that can reconfigure itself along the plausibilty-efficiency line. This model can be aptly described as follows:

$$R_m C_m \frac{dV_m(t)}{dt} = -g_l(V_m(t) - V_{rest}) - g_{Na}(t)(V_m(t) - V_{r_{Na}}) - g_K(t)(V_m(t) - V_{r_K}) \tag{2.16}$$

where $R_m C_m$ can be collectively considered as a single parameter τ_m which denotes the time constant of membrane voltage decay. Eq. 2.16, very similar to Eq. 2.5 models leak and ion-channel dynamics. However, the major bottleneck of the Hodgkin-Huxley model is avoided by having much simpler dynamics for the nonlinear ionic conductances:

$$\tau_{Na} \frac{dg_{Na}(t)}{dt} = -g_{Na}(t) \tag{2.17}$$

$$\tau_K \frac{dg_K(t)}{dt} = -g_K(t) \tag{2.18}$$

and eliminating the need for gating variables altogether. Here, τ_{Na} and τ_K are decay time constants for the conductances. The above set of equations model conductance-based synapses and post-synaptic voltage-dependent current integration, which gives us a good compromise for a plausible and simple neuron model. This is also a point neuron model which has a simple threshold and reset behavior with a finite refractive period modeled by:

$$V_m(t) = \begin{cases} V_{reset}, & t_n \le t \le t_n + t_{ref} \\ V_m(t), & otherwise \end{cases} \tag{2.19}$$

when cascaded with Eq. 2.16 where V_{reset} is a hyperpolarized potential where the K$^+$ conductance levels off and t_{ref} is the refractory period that must be surmounted before LIF dynamics kick in again after each spike for spike train:

$$s(t) = \sum_n \delta(t - t_n) \tag{2.20}$$

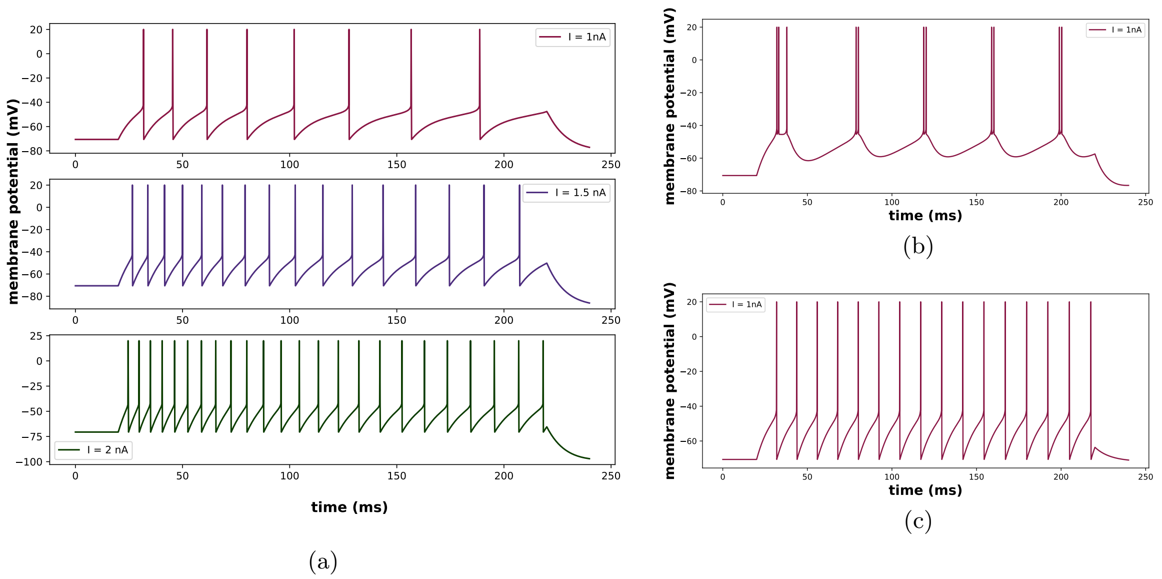


Figure 2.5: Simulation characteristics of the LIF neuron showing membrane potential traces in (a) regular spiking, (b) tonic bursting and (c) fast spiking behaviors

Instead of 20 parameters that need meticulous fitting for equations 2.6 - 2.11, we just have 7 parameters that need fitting (τ_m, τ_{Na}, τ_K, g_l, V_{rest}, V_{reset} and t_{ref}). For biologically plausible models of neurons, the timescale of neural leak and that of voltage-dependent conductances are observed and recommendations of the τ parameters are made. Other parameters are directly observable in vitro or in vivo [82]. Fig. 2.5 show some of the representative spiking behaviors of the conductance-based generalized LIF neuron based on a *Brian 2* [107] simulation of the aforementioned model. Fig. 2.5a shows the *regular spiking* behavior of a neuron with adaptation for an input current of (from top) 1, 1.5 and 2 nA while Fig. 2.5b and Fig. 2.5c show the *bursting* and *fast spiking* behaviors under the influence of a constant synaptic current injection of 1 nA, respectively. These behaviors are representative of faithful point neuron models as described in [96]. While this model is faithful to some extent and preserve computational tractability at the same time, it can be easily configured to other models that are less or more biologically plausible as and when required for simulation performance. For conversion into simpler models, some parameters should be set to certain limits. Simple LIF models with constant current integration can be modeled by making

τ_{Na} and τ_K values extremely small (directly integrate input current). Furthermore, IF models with no leak conductance can be modeled by making τ_m value equal to unity and g_l equal to zero. This feature will be beneficial in 2 out of 3 workloads used in this work as referred to in Chapter 3.

2.3 Silicon neurons and synapses

2.3.1 Neurons

Although efficient neuron models have been developed that retain biological faith and can be simulated in software simulators [107, 108, 109, 110, 111], the interest in custom hardware realizations of neuron circuits have been a matter of great interest [112]. Fig. 2.6 shows an equivalent circuit modelling the ion-channel and leak dynamics of a faithful neuron model [7]. The capacitance C_M integrates the input synaptic current I_{SYN} while bias voltage V_{LEAK} determines the leak conductance of the circuit by a constant leak current (assuming saturation and neglecting output resistance of the transistor). When there is no excitation, the membrane potential V_M gradually leaks to ground (V_{rest} is 0 here) while in presence of excitatory signal, it gradually approaches V_{DD}. However, whenever the membrane potential node exceeds V_{THR}, the differential output rises to the positive rail.The output of inverter $M_4 - M_5$ immediately goes to ground switching on transistor M_2 and the Na^+ current I_{Na} pulls up V_M thus generating an action potential. Concomitantly, inverter $M_6 - M_7$ switches on and starts charging up capacitance C_{REF}. This represents the slow but sustained K^+ conductance rise. C_{REF} is charged up by the current I_{WID} and represents the delay between Na^+ and K^+ channel activations i.e. the width of the spike generated. As soon as the voltage on this node rises to a sufficient value, it switches on M_3 and thereby pulls down V_M back to rest. Since the gate of an nFET has infinite impedance, the capacitance C_{REF} discharges through the current I_{REF}. Therefore, this current represents the delay between K^+ channel activation and inactivation and therefore, the refractory period of the neuron.

Although the circuit of Fig. 2.6, gives a good description of emulating spike generation in silicon, it is still not the most efficient hardware realization. Since the most critical requirement of these circuits is to operate at extrememely low power and the timescales of neural activity are

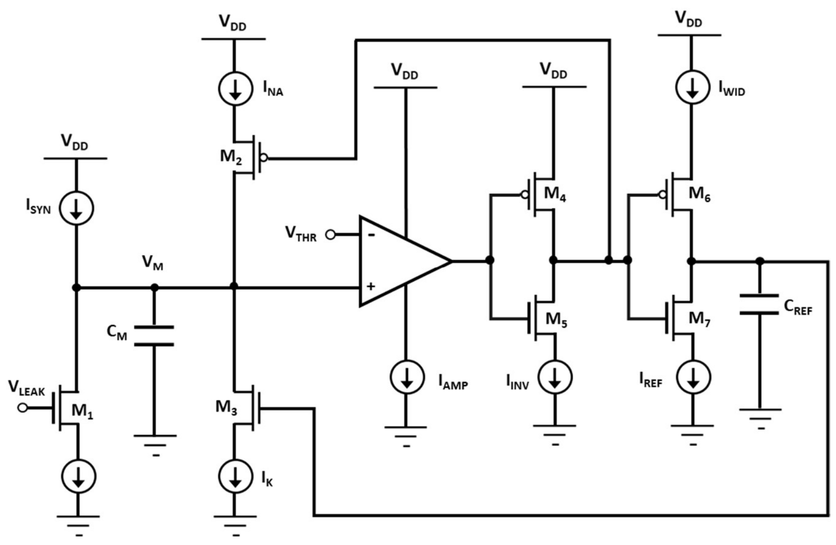


Figure 2.6: Schematic of spike generation circuit that models leak and ion-channel dynamics in a plausible and compact way. (Inspired from [6] and [7])

much slower than natural transistor operation, we can make suitable compromises of speed for efficiency. Operating MOSFETs in the *subthreshold* regime has been a very popular technique used to precisely make this tradeoff. In this region, MOSFETs draw extremely litte current thus dissipate extremely low power, have an exponential dependence of drain current on gate voltage which is an attractive primitive for neural computation and easily achieve saturation leading to large swing of voltages required for multi-stage circuits [6]. Current-mode subthreshold dynamics have been used extensively to create compact, low-power and robust circuits for neuromorphic designs [55].

One of the popular state-of-the-art design styles following the same underlying philosophy of Fig. 2.6 is the *Differential-Pair Integrator (DPI)* neuron shown in Fig. 2.7. This neuron has been used in large-scale systems like [59, 60] for extremely low power emulation of spiking networks. The input stage of this neuron is a *differential pair* formed by transistors $M_2 - M_3$. A background current set by V_{rest} is the input to the node V_{dp} integrating into the membrane capacitance C_{mem} and represents the stable resting potential of the membrane. On a synaptic input, I_{inj} is added to

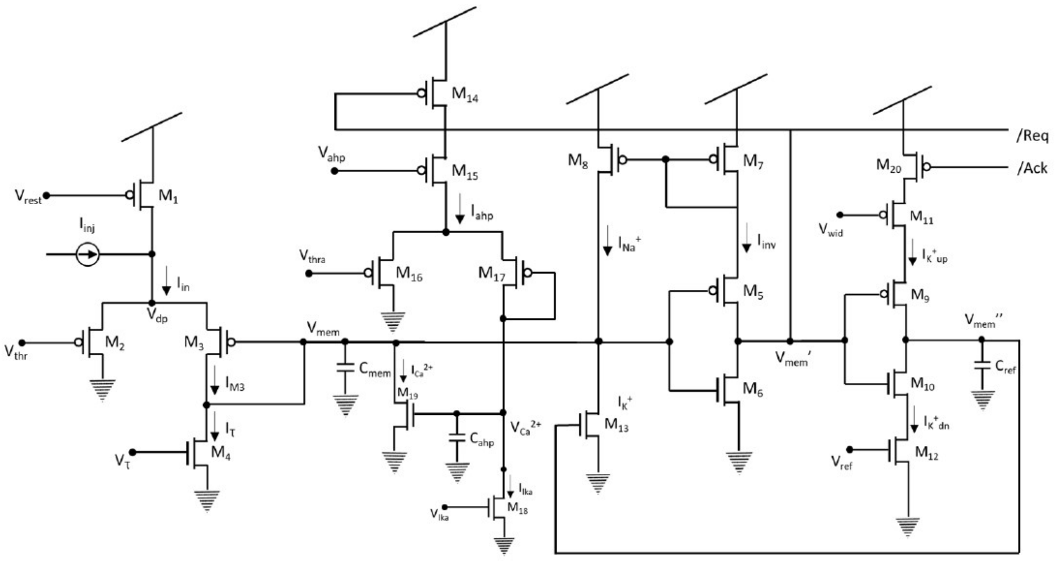


Figure 2.7: Schematic of a Differential-pair Integrator (DPI) Neuron with spike-frequency adaptation. (Inspired from [8])

this background current. A constant leak conductance is set by the bias V_τ. The membrane voltage V_{mem} can be quantified by a current I_{mem} drawn by a (fictitious) nFET tied to the membrane node as follows:

$$I_{mem} = \frac{I_{in} I_{gain}}{I_\tau}(1 - e^{\frac{-t}{\tau}}) \tag{2.21}$$

where I_{gain} is a current drawn by a fictitious nFET biased by V_{thr}. Hence, this gain serves to set an implicit spiking threshold for the neuron. τ can be manipulated by setting the capacitance C_{mem} and V_τ. For further details regarding the *differential input* circuit, see [113]. When the membrane voltage V_{mem} exceeds the switching threshold of the inverter $M_5 - M_6$, a positive feedback loop is activated via current mirror $M_7 - M_8$ supplying a current I_{Na^+} into the membrane node. It pulls up the membrane to generate an action potential. The second cascaded inverter $M_9 - M_{10}$ works with current sources biased by V_{wid} and V_{ref} in a similar way as before to set the spike-width and refractory period respectively by setting the K^+ channel activation and inactivation through capacitance C_{ref}. An additional behavior modeled by this circuit is the effect of Ca^{2+}

channels. This is known as *spike frequency adaptation*, a local adaptation mechanism in neurons by which it becomes more and more difficult for a neuron to threshold with each spike in a certain timeframe [104, 102]. It is important in competitive learning of neurons and gives a good chance to each neuron to modulate learning in its pre-synapses. This circuit achieves this using a second differential pair $M_{14} - M_{17}$. Each time the Na$^+$ channel is activated, transistor M_{14} is switched on thereby drawing a *after-hyperpolarization* current I_{ahp} into the calcium capacitance C_{ahp} and results in a constant current drawn from the membrane node. This current also has a leak conductance set by V_{lka} and therefore has similar dynamics as the membrane current but in a different timescale. With each spike, more current is drawn by the calcium node and hence makes it more difficult for the membrane to spike. The circuit interfaces to a digital communication infrastructure that routes spikes using the *Address Event Representation (AER)* protocol [114, 115, 116]. Accordingly, it is supplied with *Req* and *Ack* signals to notify spike and start K$^+$ activations, respectively. On a high level, the operation of this circuit can be mathematically described as follows:

$$\tau \frac{d}{dt} I_{mem} = -I_{mem}\left(1 + \frac{I_g}{I_\tau}\right) + I_{mem\infty} + I_{Na^+} \tag{2.22}$$

$$\tau_{ahp} \frac{d}{dt} I_{Ca^{2+}} = -I_{Ca^{2+}} + I_{Ca^{2+}max} r(t) \tag{2.23}$$

where I_{mem} is the current drawn by a readout nFET transistor (not shown in Fig. 2.7) biased by the membrane voltage V_{mem} i.e. $I_{mem} = I_0 e^{\frac{kV_{mem}}{U_T}}$; $I_{mem\infty}$ is given by $\left(\frac{I_{in}}{I_\tau}\right) I_0 e^{\frac{kV_{thr}}{U_T}}$; $I_{Ca^{2+}}$ is the current drawn by the nFET transistor M_{19} biased by the calcium conductance bias $V_{Ca^{2+}}$ i.e. $I_{Ca^{2+}} = I_0 e^{\frac{kV_{Ca2+}}{U_T}}$; $I_{Ca^{2+}\infty}$ is given by $\left(\frac{I_{ahp}}{I_{lka}}\right) I_0 e^{\frac{kV_{thra}}{U_T}}$; $r(t)$ is given by:

$$r(t) = \begin{cases} = 1, & if\ V_{mem}^{'}\ is\ low \\ = 0, & if\ V_{mem}^{'}\ is\ high \end{cases}$$

and the usual meanings hold for subthreshold conduction in MOSFETs i.e. I_0 is the leakage current, U_T is the thermal voltage and k is the subthreshold slope factor [6, 113]. Fig. 2.8 shows an implementation of the DPI Neuron circuit in the Cadence Virtuoso custom IC design environment marked up with all functional compartments. Figs. 2.9a and 2.9b show the manipulation of refractory periods and the spike-frequency adaptation mechanisms for the circuit, respectively.

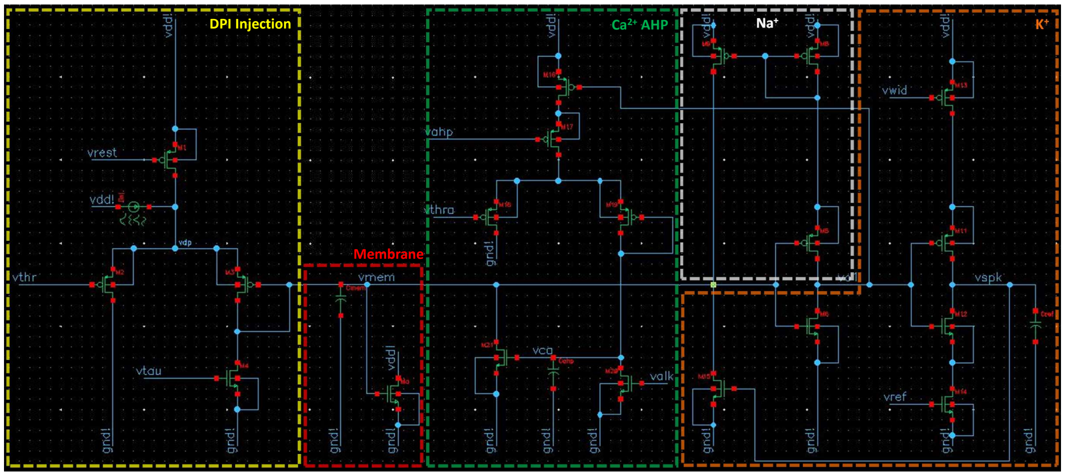
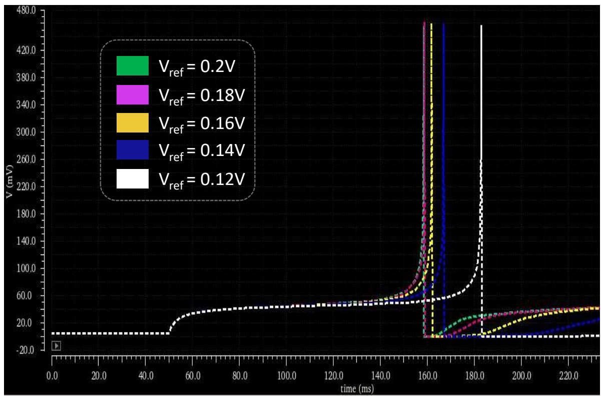


Figure 2.8: Implementation of the DPI Neuron circuit in Cadence analog design environment using a TSMC .18um process and simulated using Spectre. The injection and ion-channel compartments are marked in the schematic

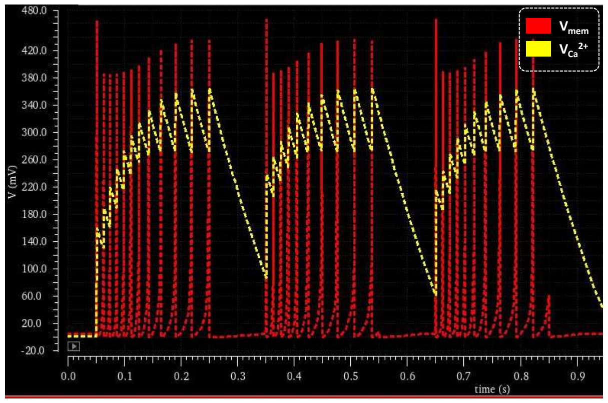
2.3.2 Synapses

Synapses are the most numerous elements in any neural network topology and hence are one of the central aspects of neural systems design. Accordingly, it is one of the most interesting design problems in neuromorphic engineering. With point neuron models, the synapses *linearly* sum all input currents into the post-synaptic neuron's membrane capacitance. In case the neurons are *multi-compartmental* models, synaptic dynamics must also support spatial summation to emulate the dendritic properties accuractely [117, 118]. The differential-pair integrator described as the neuron input stage can also be used as a linear first-order filter to provide synaptic dynamics for highly biologically plausible modelling. It is a compact, low-power option and also has significant gain for very narrow spike pulses to convert these digital pulses into a continuous input current for integration into the neuron without requiring any pulse-extender circuits. For a detailed account on synapse circuits and the DPI synapse itself, see [119].

As described in this chapter, there are numerous approaches for hardware realizations of neural systems. In Chapter 4, a purely digital full-custom implementation of the generalized point neuron



(a)



(b)

Figure 2.9: (a) The shape of action potential produced by the DPI Neuron circuit. It shows how refractory periods can be modulated by controlling the bias V_{ref}. (b) Spike frequency adaptation in the DPI neuron circuit. The plot shows the calcium voltage and membrane voltage spikes produced with decreased frequency within its timescale

model will be described that follows closely from the biological, mathematical and circuit equivalents discussed in this chapter.

CHAPTER 3. BENCHMARKS

In this chapter, building on the elementary concepts introduced in Chapter 2, the workloads for the CyNAPSE system will be described. Since the target is a hardware accelerator, it's microarchitecture is tightly coupled to the specific functionality it is expected to emulate. Therefore, a high-level description of the wide array of workloads is provided before discussing the microarchitecture.

3.1 Spiking Neural Networks

As surmised in Chapter 1, Spiking Neural Networks or SNNs are neuroscientifically plausible, low-level abstractions of real cortical networks and their dynamics. Neuron models, such as described in Chapter 2, are used as processing units in these networks. These neurons are associated with synapses each unique to a post-synaptic and pre-synaptic neuron and "connects" them, so to speak. Synapses can be both *excitatory* or *inhibitory* in the sense that they can add or subtract membrane voltage of the post-synaptic neuron when a spike passes through them. Similarly, neurons can also be excitatory or inhibitory depending on what kind of synapse they trigger. The spike is usually a digital all-or-nothing signal, an abstraction of the action potential generated in plausible neuron models. In a network-level abstraction, usually the amplitude of the spike or its own physiology does not contain any meaningful information. The information is contained either in the precise timing of a spike or in its frequency or rate. Accordingly, neural data can be temporal coded or rate coded [48]. However, unlike classical neural networks, SNNs do not understand floating point inputs and neither do they output posterior probabilities. They talk in the language of spikes i.e. the inputs and outputs are spike trains and all semantic content surrounding an SNN benchmark must be derived from spike trains.

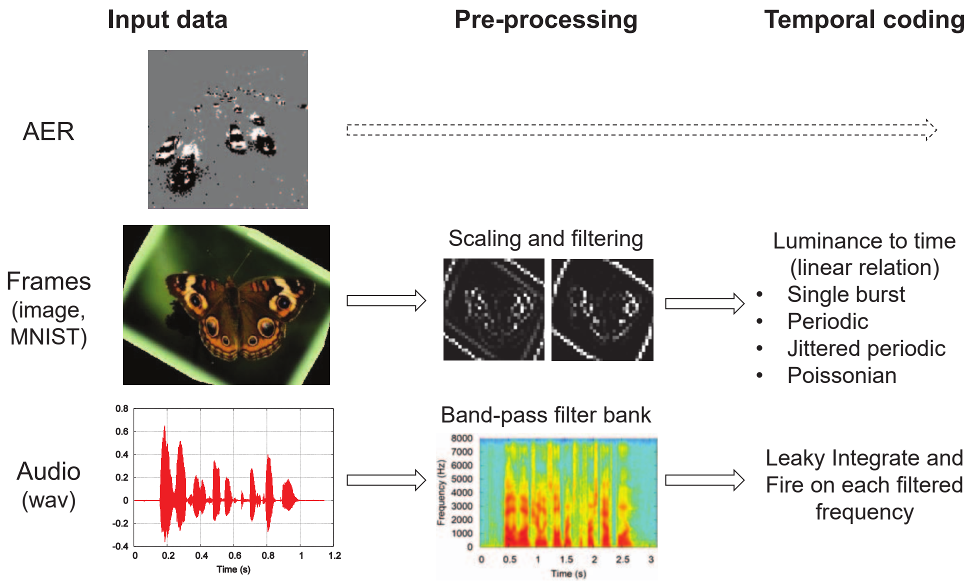


Figure 3.1: Different kinds of input data, preprocessing and temporal coding methods amenable for processing by SNN benchmarks (figure from [9])

3.1.1 Spike Inputs

Real-world time-varying or spatial data can be converted into neural response data that model the response of sensory neurons, for example, in the retina or cochlea corresponding to vision and audio stimuli, respectively. This process of conversion can be done in specialized hardware or by simply a random process routine run in software. Fig 3.1 lists some ways spike input is generated. Events in Address Event Representation (AER) format can be directly processed by any SNN benchmark in the CyNAPSE and many other neuromorphic substrates that use AER protocol [114, 115, 116]. A direct conversion is facilitated by silicon implementations of vision and auditory sensors like silicon retina [41, 120, 121] and cochlea [42, 122, 123] or a dynamic vision sensor [124, 125, 126]. Alternatively, a soft conversion of real-time data to spike trains is possible which is the approach pursued in this work. For spatial data like images or individual video frames, random processes can be used to convert pixel intensity into mean spike rates introducing some randomness in the process to best encode sensory response. In this work, all images have been converted using *poissonian* spike generation routine [127] provided by the Brian 2 simulator [107]. Time-varying data like audio signals can also be converted into essentially spatial data by frequency domain filtering and making spiking neurons sensitive to particular bands, thus making spike signatures for those bands.

3.1.2 Inference

Inference, as in classical neural networks, requires forward pass of data through the SNN. (In fact, the biological plausibility of a backward pass altogether is highly debated). However, as discussed in Chapter 1, there are many important differences. Firstly, every testing example needs to be simulated not once but multiple times equalling the number of *timesteps* simulated in one example *exposure* period. The timestep or resolution of a network is the minimum timestep for processing in this network (in time units) and a single example exposure is the time that network runs for one example (also in time units). Therefore, each example is simulated for $\left(\frac{exposure}{resolution}\right)$ timesteps. Secondly, in each of these timesteps, a single input unit has a finite probability of spiking which is often directly proportional to the pixel intensity of the input frame (no. of units = no. of pixels in frame). Since computation in all units is local, a forward communication is only produced when there is a spike. For the input layer, synaptic weights are communicated when the poisson process generates an input spike. For subsequent layers, communication results only upon thresholding of spiking neurons that integrate these synaptic weights in a perfect or leaky manner. In this way, inference in SNNs is very different from classical networks where all examples are simulated only once but for a full forward pass through the entire network. While the latter is fast, the former is vastly more energy efficient because spiking is essentially a *sparse* event by nature.

3.1.3 Output handling

SNNs output spike trains as a signature of the network's activity over the entire exposure period of a testing example. This signature is used to draw semantic information about the network's inference. Mapping of output units to output classes varies among benchmarks. In general, handling routines monitor spike signatures of each neuron and infer from the highest spiking neuron or

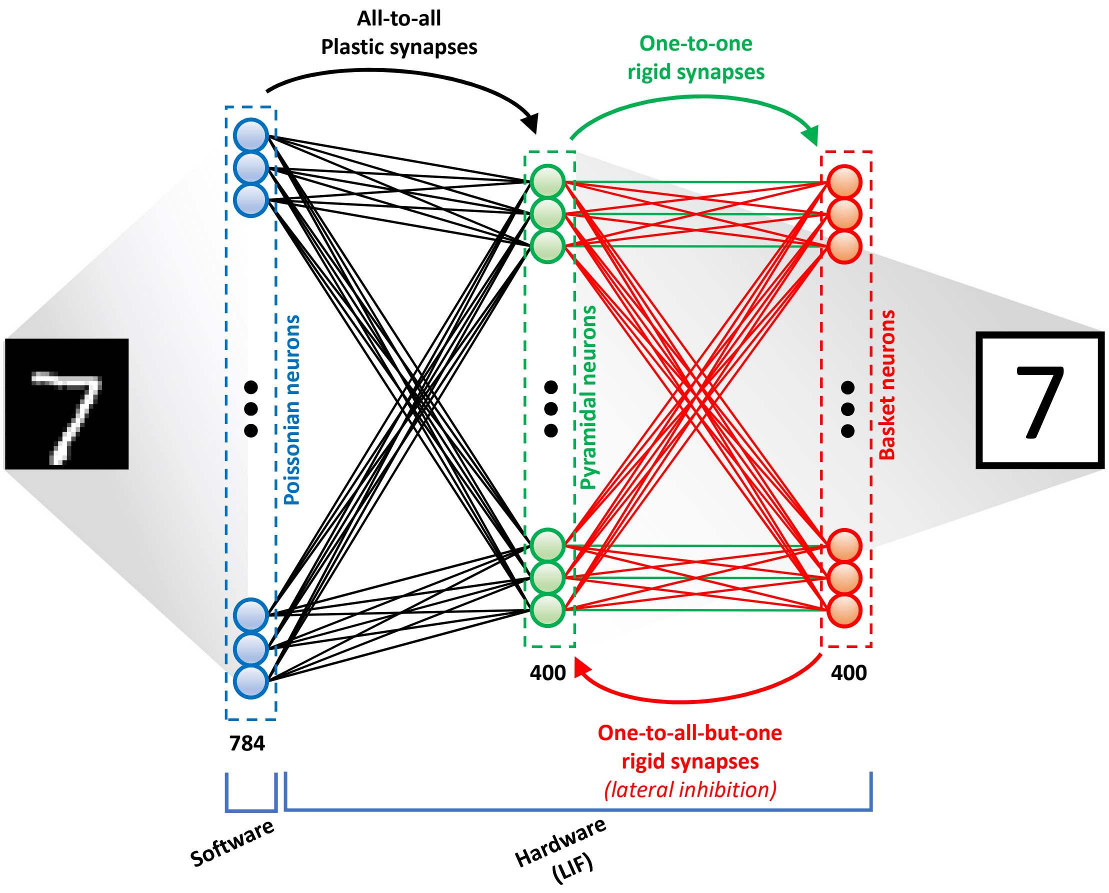


Figure 3.2: Overview of the SCWN Benchmark architecture

population. In the following section, the three benchmarks used for this study will be described in terms of their architecture and simulation parameters. Additionally, a layer-wise spike signature of each network, collected across multiple example stimuli, will be provided which will denote the underlying temporal activity of each network and establish their differences. As will be explained in later chapters, the temporal spiking activity of a network is an important factor that determines how much processing effort is required.

3.2 Benchmark I

3.2.1 Architecture

The first benchmark is a *Spiking Competitive Winner-take-all Network* (hereafter referred to as SCWN) inspired from the work in [106]. It is a recurrent neural network topology with three layers and 1584 total neurons as shown in Fig. 3.2. The input layer consists of 784 input units corresponding to the 28x28 pixel input field of the MNIST digit recognition dataset [128]. These input units are *poissonian* neurons that are excitatory in nature but solely required for spike generation and are modeled exclusively in software. The generated spikes are then sent via learned (plastic) synapses in an all-to-all topology to the processing neurons in the subsequent layer which are modeled in hardware. The second layer consists of 400 *pyramidal* or excitatory Leaky Integrate and Fire (LIF) neurons. These neurons are modeled exactly as denoted by Eq. 2.16, i.e. it has all of the excitatory (Na^+), inhibitory (K^+) and leak channels. They have a one-one connection with corresponding neurons in the third layer via rigid synapses with a constant weight. The third layer has 400 LIF neurons which are *basket* cells or inhibitory. They provide an inhibitory signal back to all pyramidal cells that do not supply an input to them, again via rigid synapses, all with the same constant weight. The ratio of these two constant weights along with this recurrent topology simulate what is known as *lateral inhibition*. Through lateral inhibition, a certain pyramidal neuron has the power to inhibit all other pyramidal neurons when it strongly spikes and therefore a *winner-take-all* philosophy is established. This makes it highly suitable for *recurrent competitive learning* which has extensive neuroscientific support [129, 130, 84]. The network also promotes *homeostasis*. Homeostasis allows for adaptive thresholds that consolidate firing frequency of pyramidal neurons during learning by giving all competitors a fair chance of winning [131]. The parameters used for the pyramidal and basket cells are highly motivated by biological evidence and manipulated only slightly to benefit performance [82].

The network has been learned using *Spike Timing Dependent Plasticity (STDP)*, an unsupervised learning technique that is very tightly coupled with learning observed in real neurons from

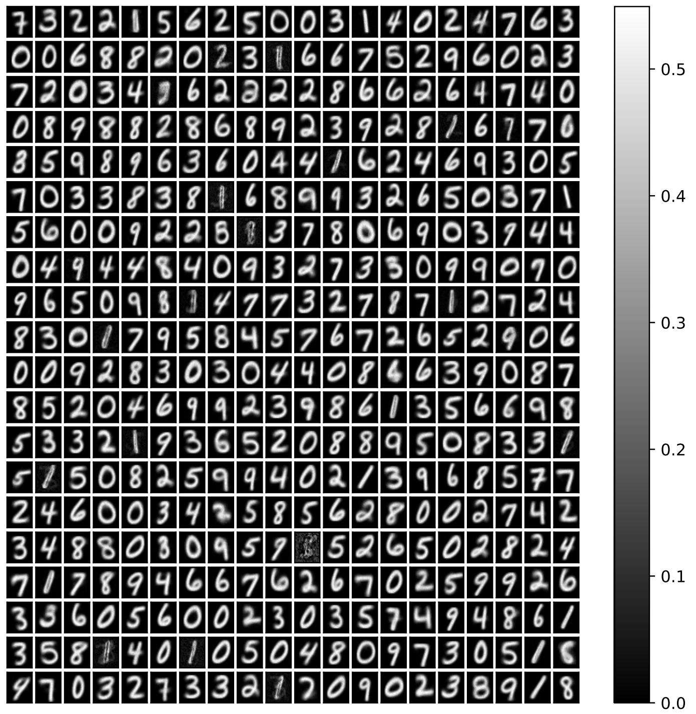


Figure 3.3: Receptive fields of the pyramidal neurons in the SCWN showing the input sensitivity of these neurons. Due to the *Hebbian* nature of learning, receptive fields can be identified as readable patterns

hippocampal cultures [39, 85]. As a result of this learning, neurons in the pyramidal layer adjust their pre-synaptic weights to make them selectively sensitive to a certain input pattern. These patterns closely resemble the *receptive fields* of the V1 simple cells in the retinal sensory pathway [132] and determine what digit fires a neuron the most, as shown in Fig. 3.3. Sometimes, as was done is [106], this sensitivity is statistically verified during part of the learning routine to make the whole process semi-supervised.

3.2.2 Simulation

An example digit is presented to the network for an exposure time of 350 miliseconds with a resolution of 0.5 miliseconds, and a maximum firing frequency of 63.75 Hz. During that time,

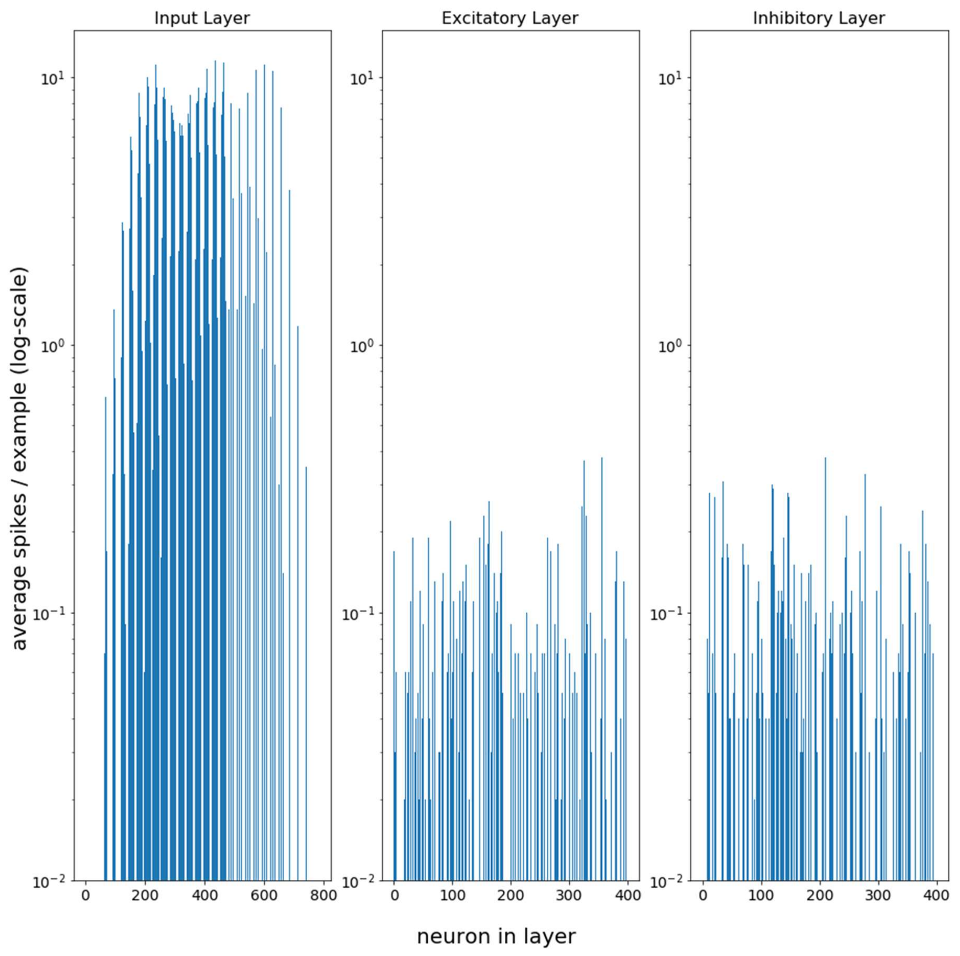


Figure 3.4: Layer-wise spiking activity in the SCWN network

all spikes produced by the pyramidal neurons are monitored for inference. Once these spikes are collected, the number of total spikes against each class is measured using neuron class assignments. Every neuron has been trained in a semi-supervised manner to determine the class of digit it is sensitive to but there are multiple neurons sensitive to a certain digit thus covering different morphologies of a certain digit found in the dataset. Based on the spiking activity, an inference is made. Since this simulation causes significant excursion from the resting state of a neuron, a 150 milisecond resting period is afforded to ensure that the neurons return to rest before presenting the next example. This network architecture achieves a maximum classification accuracy of 95.0% [43]. Fig. 3.4 shows the overall layer-wise activity distribution in the network.

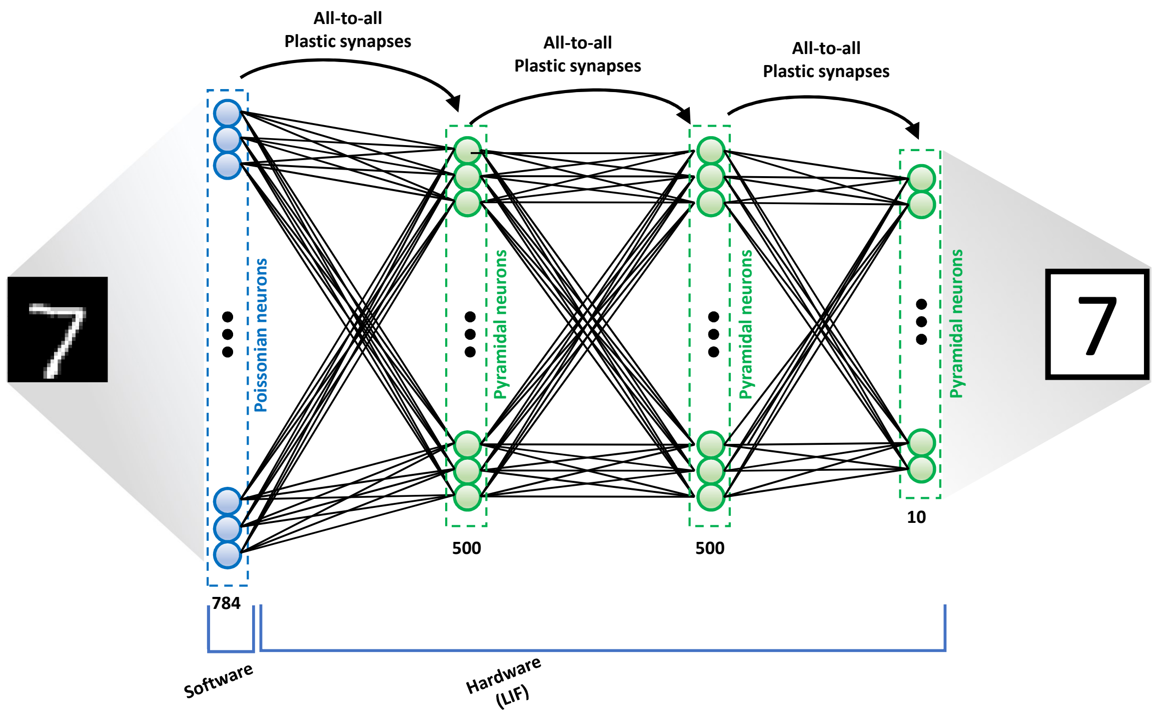


Figure 3.5: Overview of the SDBN Benchmark architecture

3.3 Benchmark II

3.3.1 Architecture

The second benchmark is a *Spiking Deep Belief Network* (hereafter referred to as SDBN) inspired from the work in [133]. It is a completely feed-forward fully connected topology with four layers and a total of 1794 neurons as shown in Fig. 3.5. As any typical network solving the MNIST task, the input layer consists of 784 poissonian neurons modeled only in software and generate poisson spike trains for consumption of the subsequent layers. The subsequent *hidden* layers consists of 500 pyramidal LIF neurons. The first three layers have all-to-all fully connected synapses between them with learned (plastic) weights. However these neurons are modeled as a subset of the total LIF behavior of Eq. 2.16. These neurons have a leak conductance but do not have separate ion-channel dynamics. They support only direct constant synaptic current integration as discussed in

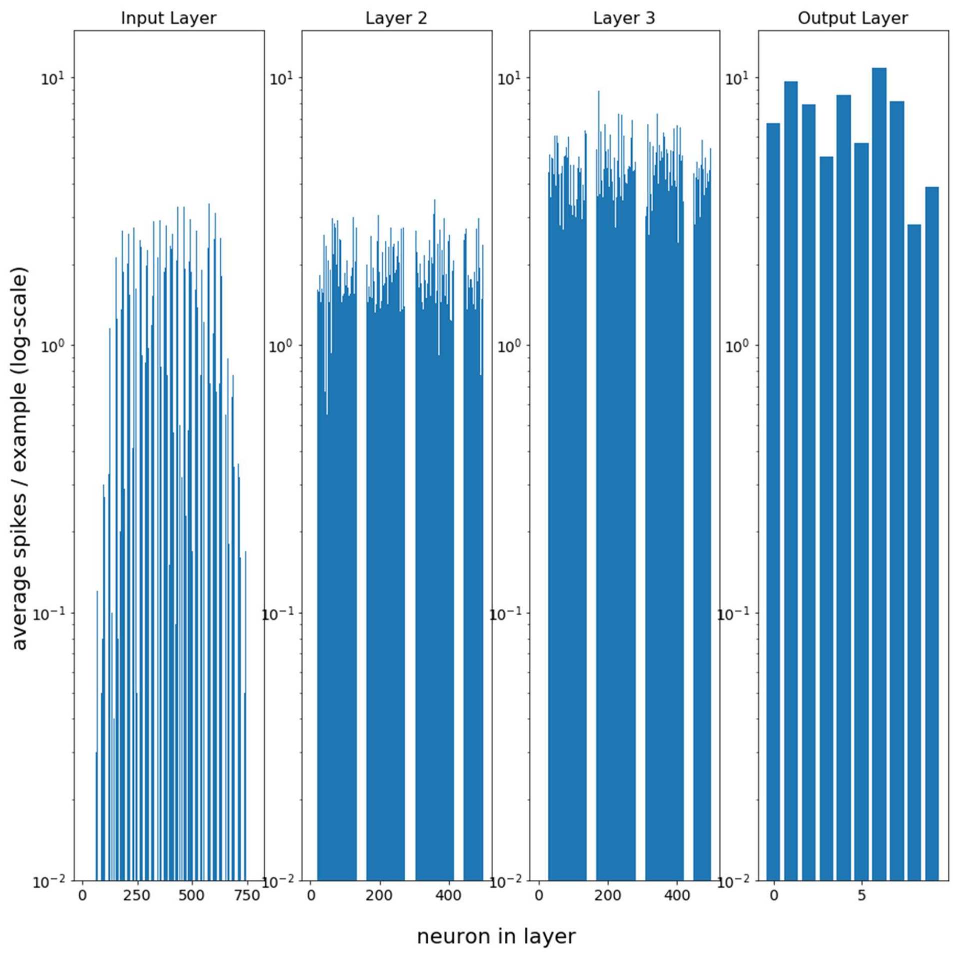


Figure 3.6: Layer-wise spiking activity in the SDBN network

Section 2.2.2. The output layer consists of 10 such LIF neurons and each denote a single output class (digits). This network has been learned as a classical feed-forward Restricted Boltzmann Machine using *Contrastive Divergence (CD)* [134, 135] and then converted into the spiking domain with spiking LIF neurons. The standard output spike-handling is performed but unlike the previous benchmark, there is only one neuron per output class.

3.3.2 Simulation

An example digit in this benchmark network is presented for an exposure time of 1 second for producing spikes at a maximum frequency of 6 Hz while a resolution of 1 milisecond is exercised.

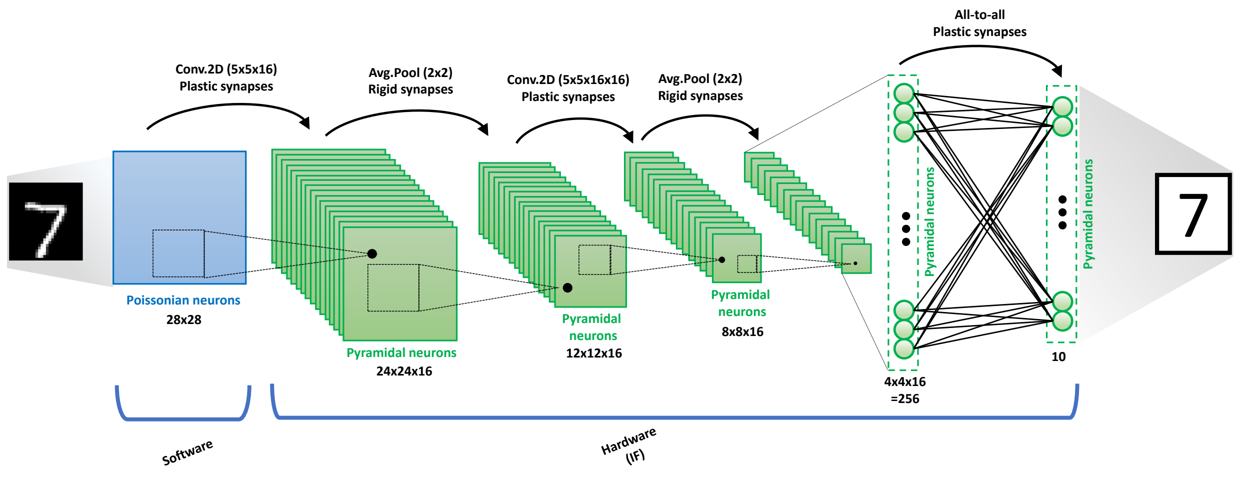


Figure 3.7: Overview of the SCNN Benchmark architecture

All spikes produced by the output layer is monitored and used to infer the most probable class. This network achieves a maximum accuracy of 92% on the MNIST digit classification task. Fig. 3.6 shows the spike signatures from this network benchmark.

3.4 Benchmark III

3.4.1 Architecture

The third benchmark is a *Spiking Convolutional Neural Network* (hereafter referred to as SCNN). It is a feed-forward deep neural network with convolutional, pooling and dense (fully connected) layers as shown in Fig. 3.7. It has a total of 6 layers and 13584 neurons. Just like the SCWN and the SDBN benchmarks, 784 input poissonian neurons are modeled in software. All the other 12810 processing neurons are simple *Integrate and Fire (IF)* neurons. These neurons do not require any leak or ion-channel dynamics in their modelling and are accommodated accordingly using the generalized model (see Section 2.2.2). The second layer is a *convolutional* layer with 16 (24x24) pyramidal output maps that receive sparse input connections through 16 (5x5) filters containing learned (plastic) synapses. The third layer is a *subsampling* layer with 16 (12x12) pyramidal output maps that receives fixed (rigid) synaptic weights required to perform average pooling. The

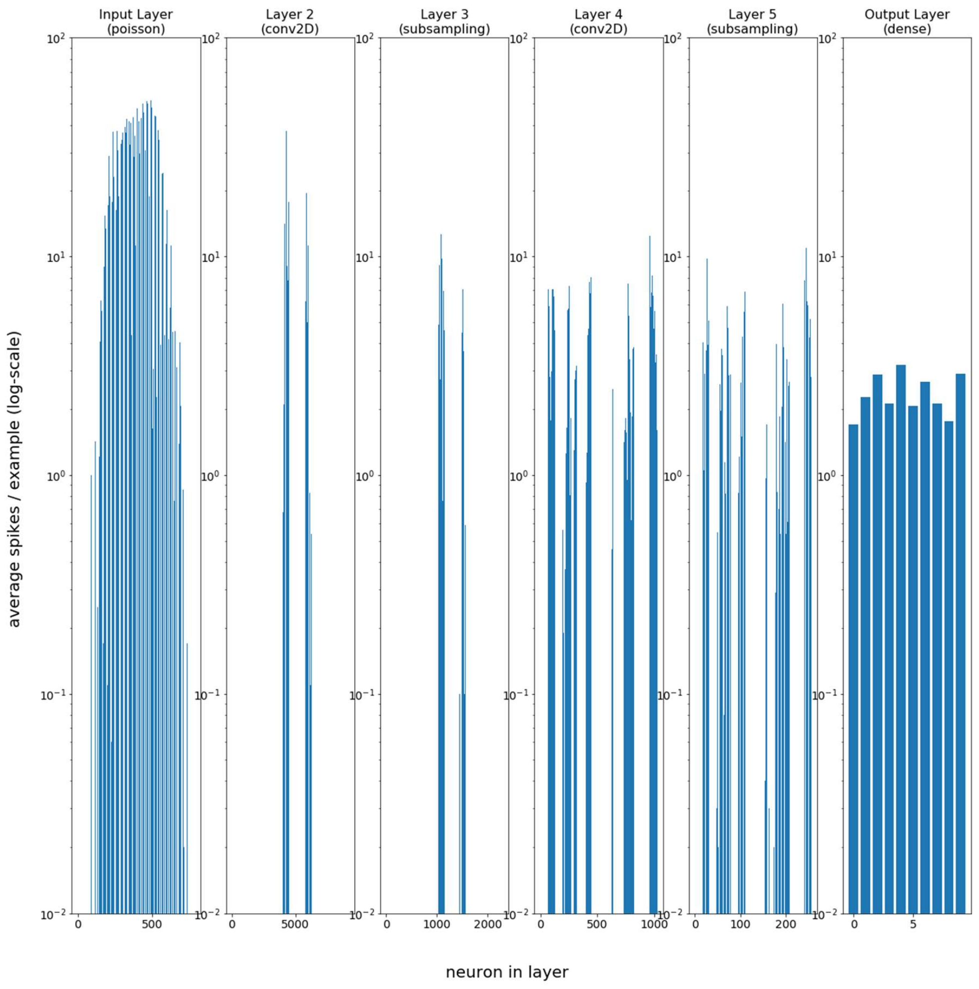


Figure 3.8: Layer-wise spiking activity in the SCNN network

fourth layer is also a convolutional layer having 16 (8x8) output maps through 16 (5x5x16) filters. Similarly, the fifth layer is a subsampling layer having 16 (4x4) output maps. These maps are flattened out and fully-connected to a final output layer of 10 pyramidal neurons with an all-to-all connection of plastic synapses between them.

The network has been trained as a classical neural network with analog (ReLU) activation maps using standard *Error Backpropagation (BP)* [38] with certain restrictions. Following training within these limits, the network is converted into an equivalent spiking network using the prescription provided in [43] wherein all the activation maps are switched to simple IF neurons.

Spiking Competitive Winner-Take-All Network (SCWN)								
Layers	Neurons	Synapses	Neuron Model	Max. Input Freq.	Exposure	Resolution	Training	Max. Accuracy
3	1584	473600	LIF	63.75 Hz	500 ms	0.5 ms	STDP - WTA	95%
Layer	Input Layer			Excitatory Layer (forward single)			Inhibitory Layer (recurrent dense)	
Spke Fraction	97.8%			1.1%			1.1%	
Spiking Deep Belief Network (SDBN)								
Layers	Neurons	Synapses	Neuron Model	Max. Input Freq.	Exposure	Resolution	Training	Max. Accuracy
4	1794	647000	LIF	6 Hz	1000 ms	1 ms	CD	92%
Layer	Input Layer		Layer 2 (dense)		Layer 3 (dense)		Output Layer (dense)	
Spke Fraction	15.6%		23.7%		59.0%		1.7%	
Spiking Convolutional Neural Network (SCNN)								
Layers	Neurons	Synapses	Neuron Model	Max. Input Freq.	Exposure	Resolution	Training	Max. Accuracy
6	13594	652800	IF	1000 Hz	100 ms	1 ms	Backpropagation	97%
Layer	Input Layer		Layer 2 (conv2D)	Layer 3 (subsampling)	Layer 4 (conv2D)		Layer5 (subsampling)	Output Layer (dense)
Spike Fraction	47.2%		35.7%	7.6%	7.7%		1.7%	0.1%

Table 3.1: Spiking neural network benchmarks used for this study

3.4.2 Simulation

An example digit is presented to this equivalent spiking network for a very short exposure period of 100 miliseconds under a resolution of 1 milisecond, but with a relatively high maximum frequency of 1 KHz. All spikes produced by the output layer is monitored for inference of the most probable output class. A maximum accuracy of 97% on the MNIST testing dataset is achieved using this topology, although, the deviation from the original analog neural network is negligible as pointed out in [43]. The spiking activity of the network is shown in Fig. 3.8.

3.5 Summary

Although the three benchmarks are SNNs driven by the same inspiring philosophy of biological neural networks, there are significant differences amongst them. While the SCWN is much more biologically plausible, the SDBN and SCNN gradually lose plausibility as the approach begins to take the best from both worlds - sparse efficient spike based processing and robust accurate statistical inference of ANNs. These differences are easily seen in the layer types, the weight values, the inference topology and most importantly, the spike footprint of the network. The spiking activity in all these networks are very different from each other and will directly influence their memory access patterns in a neuromorphic accelerator.

For analysis of these workloads, as explained in later results, the spiking activity of all layers of these benchmarks were monitored using a software simulator that periodically dumps internally produced spikes within the network. The input events are available from the source during simulation. This data along with a summary of most important characteristics of each benchmark or our application, is provided in a succinct manner in Table 3.1

CHAPTER 4. THE CyNAPSE ARCHITECTURE

In this chapter, I will present the *CyNAPSE neuromorphic accelerator*, a hardware acceleration fabric that emulates the neural dynamics introduced in Chapter 2 and accelerates neural inference in spiking neural networks like the benchmarks discussed in Chapter 3. The system is reconfigurable both in terms of neural dynamics and network topology which makes it a flexible environment for emulating SNNs of different types and functions. In the following sections, the overall system architecture of the accelerator will be discussed, including a detailed overview of neuron circuits. Subsequently, I will discuss scheduling, control, programmability and the details of implementation.

4.1 System Overview

4.1.1 Overall hardware architecture

Fig. 4.1 shows the overall architecture of the accelerator. The system contains three circular FIFO queues that hold spike events. Whether a spike is generated off-chip in software (input software-generated spikes) or on-chip in hardware (internal network-generated spikes), all spikes are universally encoded in the *Address Event Representation* (AER) format [116]. In this format, each event is associated with a Biological timestamp (BT) and a Neuron Address (N_{ID}) of the neuron that produced this event. The *Input FIFO queue* enlists all poissonian input spikes from an off-chip environment in a streaming manner. Once the network itself generates some spike, it gets enlisted in the *Auxiliary FIFO queue.* If they are produced by the output layer, they are sent to the *Output FIFO queue.* The system contains two routing state machines in the *Input spike router* and the *Internal spike router.* The input router takes an event from either queue at the front-end and looks up the weights associated with it and routes them to all relevant target neurons. In this process, it interfaces with the *Memory Controller* that interfaces with CyNAPSE's memory hierarchy to bring the relevant synaptic weights from an *Off-chip DRAM* storage back

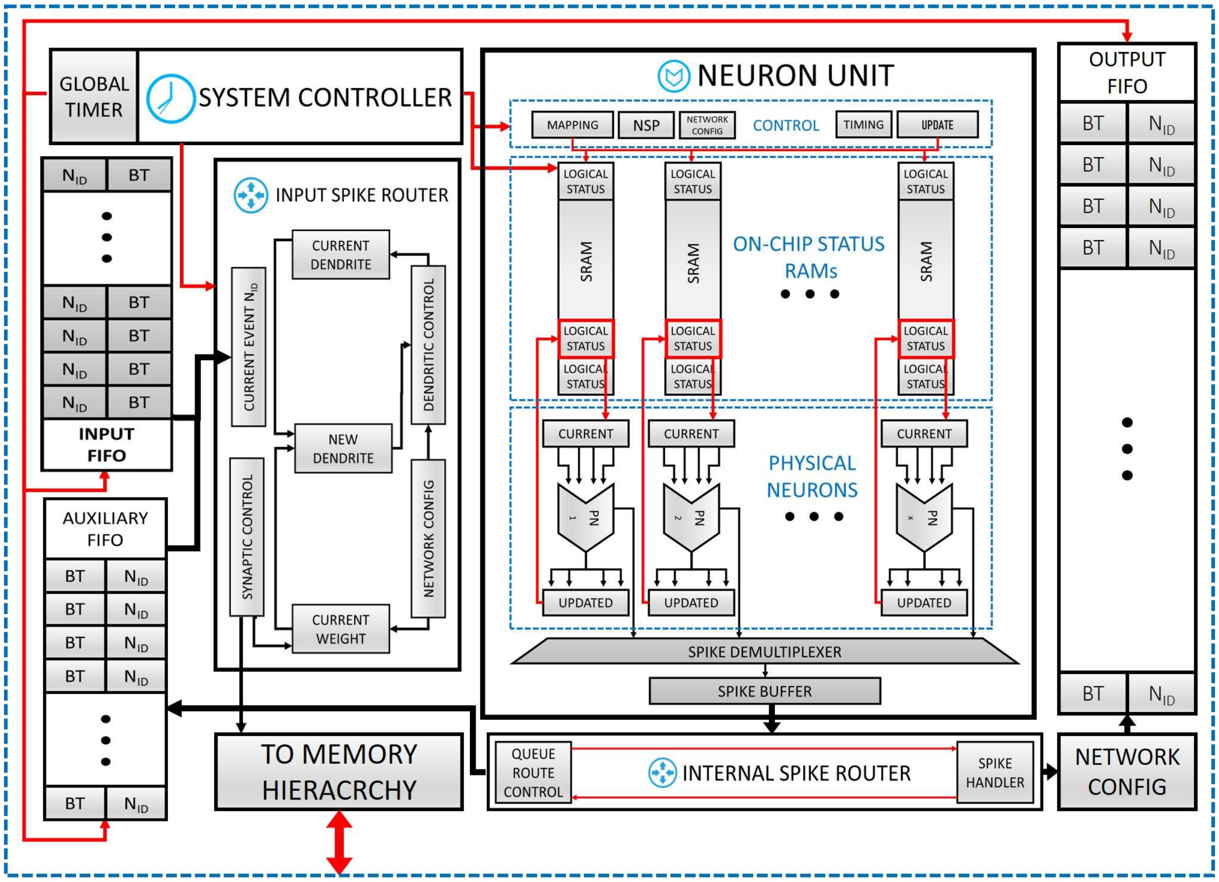


Figure 4.1: The CyNAPSE microarchitecture. It has a neuron-unit with on-chip circuits emulating LIF neurons, dendritic-tree SRAMs, an input spike router, an internal spike router , FIFO queues holding AER events and a system controller

for further routing. The internal router just routes spikes produced on-chip to the auxiliary queue for further processing. The system contains a *Neuron Unit* that is equipped with neuron circuits and dendritic tree RAMs that efficiently perform the LIF operation by updating neuron statuses at every timestep of the simulation and producing spikes whenever a neuron thresholds. However, not all supported neurons are modeled on-chip. Rather, the total number of supported (*logical*) neurons are multiplexed into a small number of on-chip (*physical*) neurons and it is required to store the status associated with every logical neuron in the SRAM columns. Spikes produced in the neuron unit are filtered and passed into the internal spike router. There is a *System Controller* with a global timer that synchronizes the working of every unit and enforces control dependencies among them according to the user-defined timing resolution and simulation times for the network kernel running on it.

4.1.2 Neuron design

The on-chip neuron circuitry implements the generalized integrate and fire model of Eq. 2.16. For a digital custom circuit emulating these dynamics with a resolution of Δt, the equation can be rewritten in discrete-time format as follows:

$$\Delta V_m = (V_m[t+1]-V_m[t]) = \Big\{-g_l(V_m[t]-V_{rest})-g_{Na}[t](V_m[t]-V_{r_{Na}})-g_K[t](V_m[t]-V_{r_K})\Big\}\frac{\Delta t}{\tau_m} \quad (4.1)$$

or,

$$V_{mem}[t+1] = V_{mem}[t] - I_{leak}[t] + EPSC[t] - IPSC[t] \quad (4.2)$$

with

$$EPSC[t] = \Big(\frac{V_{r_{Na}} - V_{mem}[t]}{\tau_m}\Big)\Delta t g_{Na}[t] \quad (4.3)$$

$$IPSC[t] = \Big(\frac{V_{mem}[t] - V_{r_K}}{\tau_m}\Big)\Delta t g_K[t] \quad (4.4)$$

$$I_{leak}[t] = \Big(\frac{V_{mem}[t] - V_{rest}}{\tau_m}\Big)\Delta t g_l \quad (4.5)$$

where $EPSC$ and $IPSC$ are the excitatory and inhibitory post-synaptic currents respectively resulting from leaky Na$^+$ and K$^+$ conductances and I_{leak} is the leak current from the constant leak conductance. The sign inside the expression for the excitatory current is intentionally reversed to denote the typically highly positive Na$^+$ reversal potential, thereby signifying its existence as a 'positive supply rail' supplying inward current into the membrane capacitance. The K$^+$ reversal potential works like the negative rail in that regard. Ofcourse, if a simpler model of the neuron is required and the ionic conductances are set to zero, the parameters can be set accordingly to realize membrane dynamics at any given voltage range.

Similarly as the membrane potential, discrete-time expressions for the voltage-gated ionic conductances take the following form:

$$g_{Na}[t+1] = g_{Na}[t] - \Big(\frac{g_{Na}[t]}{\tau_{Na}}\Big)\Delta t + \sum_i S_i w_{i_{exc}} \tag{4.6}$$

$$g_K[t+1] = g_K[t] - \Big(\frac{g_K[t]}{\tau_K}\Big)\Delta t + \sum_i S_i w_{i_{inh}} \tag{4.7}$$

where S_i are the spiking activities of all pre-synaptic neurons while $w_{i_{exc}}$ and $w_{i_{inh}}$ are the weights of excitatory and inhibitory synapses. In effect, therefore, when an excitatory neuron spikes, the weights are added to the sodium conductance while an inhibitory spike affects the potassium conductance. Finally ofcourse, all the membrane update Eqs. 4.3- 4.2 are skipped when the neuron is in its refractory period while conductance updates can continue. Also, at every timestep, a thresholding operation takes place. This can be represented as:

$$V_{mem}[t+1] = \begin{cases} V_{reset}, & if\ V_{mem}[t+1] \geq V_{thresh} \\ V_{mem}[t+1], & otherwise \end{cases} \tag{4.8}$$

$$S[t+1] = \begin{cases} 1, & if\ V_{mem}[t+1] \geq V_{thresh} \\ 0, & otherwise \end{cases} \tag{4.9}$$

where $S[t+1]$ is the spiking activity of the neuron in question and V_{reset} is the hyperpolarized voltage of the neuron which it resets to, after generating an action potential.

Fig. 4.2 shows the implementation of the generalized LIF neuron. The input logical status corresponds to the state of each logical neuron that multiplexes onto the given physical circuitry. The K^+ and Na^+ ion channels integrate the dendritic inputs from the routing cycles of the last timestep and the respective leaks to give the output conductance values. Simultaneously, the membrane potential is also updated with the leak, excitatory and inhibitory currents using the conductances of the current timestep and subsequently checked for thresholding. Additionally, a refractory counter checks if the neuron is in refractory period using the status t_{ref} and skips membrane updates if a non-zero t_{ref} is observed. If V_m exceeds threshold, a spike bit is released

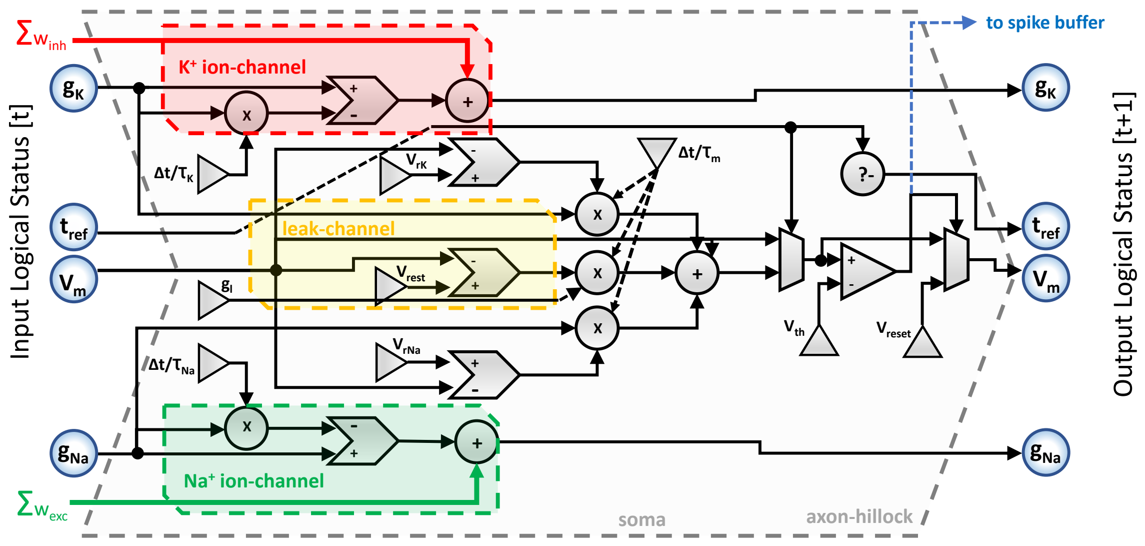


Figure 4.2: The full-custom digital generalized integrate and fire neuron. The different channels are regions are marked. All parameters shown in gray circular units are reconfigurable in nature and are loaded from a global parameter file

into the spike buffer. Subsequently, the next input logical status is accepted until the dendritic tree is completely exhausted of valid logical statuses.

4.2 Scheduling and control flow

4.2.1 Core control

CyNAPSE works in a co-processor interface with a CPU that can supply software-generated spikes and handle output spikes to perform inference. Alternatively, it can also interface with a spiking sensor in the front-end and a motor actuator in the back-end in an embedded environment. In any case, the input spikes are enqueued into the input FIFO in an online fashion. After the system has been initialized and programmed with the particular kernel (neuronal dynamics, parameters, synaptic weights and network topology), an event from the top of the queue is dequeued only if the timestamp matches the biological time of the system as indicated by the global timer. The input

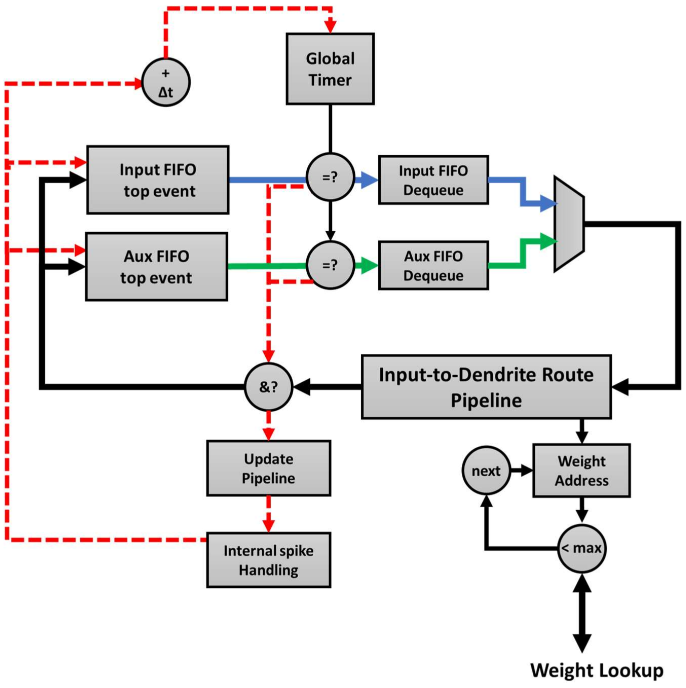


Figure 4.3: Control flow of SNN emulation in the CyNAPSE Core

router reads the address of the neuron that produced this event and performs a memory lookup in its memory hierarchy for this neuron's post-synaptic weights. It reads the current dendritic status of each post-synaptic neuron from the dendritic SRAMs and adds the new weight to this value. Following that, it looks for the next synaptic connection by this neuron. When all connections for this particular neuron have been serviced, the routing cycle is completed. In a pipelined manner, all events that correspond to the current biological time of the system, are routed to the appropriate dendrites. When an event of a new timestep is encountered in the input queue, a similar service is performed on the auxiliary queue. When there are no more events in either queue that belongs to this timestep, the system switches to its update cycle.

In the update cycle, all logical status are refreshed through accessing any one of the physical neurons. After all the logical neuron statuses have been updated, any spikes that have been produced in this update cycle are passed into the internal router. The system then asks the internal spike handler to return all these events to the auxiliary queue for routing in the next timestep while any output layer events are also sent to the output queue to be dequeued by an off-chip spike

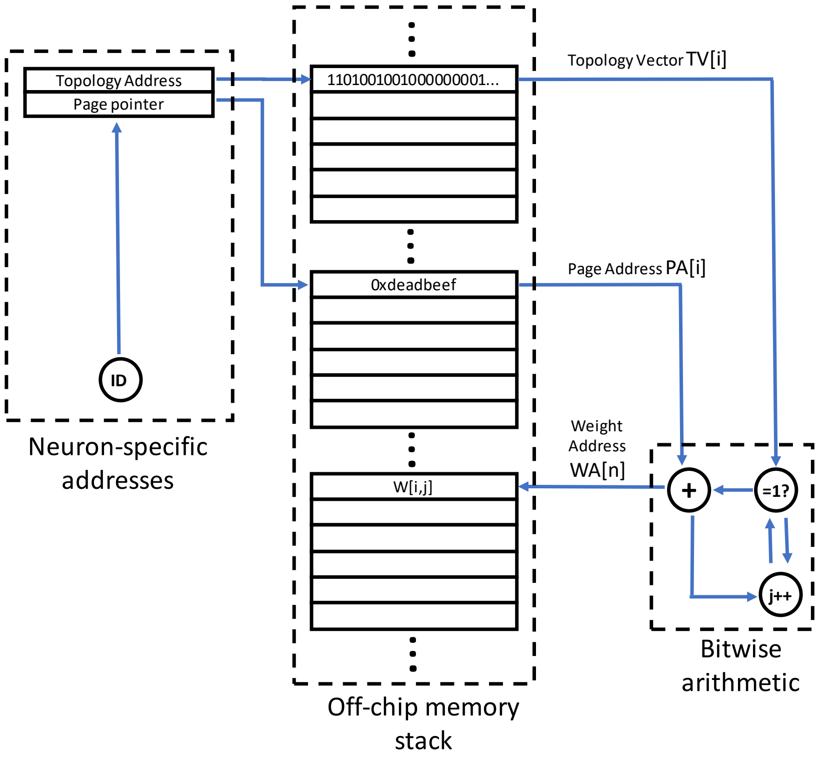


Figure 4.4: The control flow of a single synaptic weight lookup by the input spike router

handling prompt. This essentially completes the processing of one timestep (resolution) of the simulation. A barrier synchronization ticks the global timer to the next timestep and simulation resumes. Fig 4.3 graphically describes the core scheduling.

4.2.2 Memory control

For N supported logical neurons, the number of possible logical synapses would be N^2 requiring $O(N^2)$ memory. However, as networks get deeper and with more sparse layers, realistically most of these synapses are not used at all. Therefore, storing these synapses in a fixed table of weights in an external DRAM is highly inefficient. By adding another layer of indirection in the memory access path, a large amount of storage efficieny can be gained. This is done by storing synaptic weights in *pages*. However, unlike conventional virtual address translation, this scheme works best without fixed page offset size, i.e. pages can be of variable sizes, since many neurons can be sparsely connected and many, densely. To make most efficient use of memory, pages of variable sizes are enabled. Therefore, these pages are marked by their starting address called a *page pointer* which is required by the router to translate to the exact address. Since *all* post-synaptic weights will be accessed when it spikes, every spike is a full page access and no offset is explicitly required. The size of a page, or the number of weights in it, is first determined by a connectivity table called the *topology matrix*. A *topology vector* pertaining to the particular neuron is extracted and every connection in that vector instructs the router to access the next weight in memory starting with that neuron's page pointer. So, the input router's memory cycle consists of three accesses for each event i.e. (in chronological order) a topology vector, a page pointer, and all weights of that page using a simple bitwise arithmetic on the topology vector. This flow is shown in Fig. 4.4. In Chapter 5 we exploit this dataflow to make informed architectural choices towards optimizing memory accesses in SNN processing.

4.3 Programming and Reconfigurability

A single CyNAPSE core supports a maximum of N logical neurons and N^2 synapses. However, as discussed before, a smaller number X of actual physical neurons are implemented on-chip. This also requires X dendritic tree SRAMs on-chip each holding $\left({}^{N}\!/_{X}\right)$ logical dendritic trees statuses. The CyNAPSE interface provides pin inputs that help specify the network topology and neuron model parameters and other parameters that will decide the benchmark details. At the same time, the synaptic table must be updated in the above expected format for the accelerator to perform continuous synaptic lookups. This is followed by mapping of logical neurons and statuses into the SRAMs and physical neuron units. Mapping is done in a 'next physical neuron every logical neuron' fashion so that most of the load is divided as best possible among all on-chip neurons and SRAMs and fastest inference is ensured. When mapping completes, there is a cue for inference to start and starts when there is a response from the off-chip control environment. Thereafter the accelerator expects simulation as long as there are events in the Input FIFO and keeps producing output spikes for consumption by the spike handler. N and X are design-time configurable parameters. A

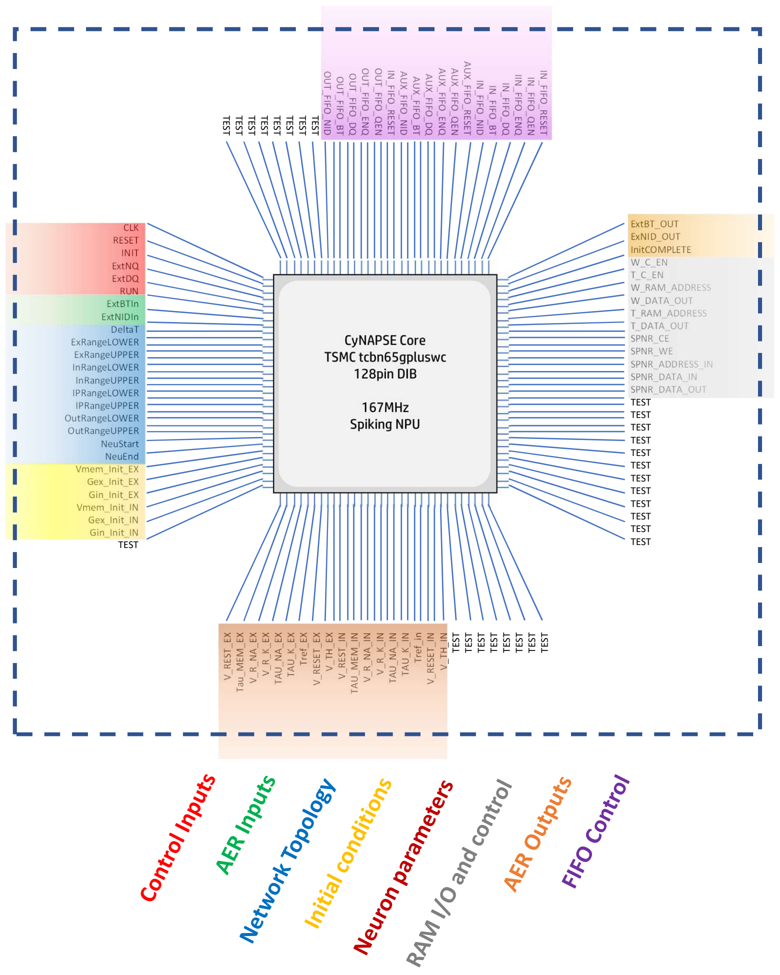


Figure 4.5: A pin diagram of the CyNAPSE Core ($N = 16384$, $X = 64$) showing an overview of the programming signals to reconfigure the network topology, neural dynamics, initial data and simulation data

CyNAPSE Core can be generated by configuring N and X but all neural parameters and network topologies within these limits can be run by reconfiguring the CyNAPSE fabric, remapping the logical-to-physical relationships and writing the synaptic and neuron metadata into memory. The network can have all pyramidal neural dynamics or can have seperate pyramidal and basket cell layers (see Chapter 3 for details). The pin diagram of a CyNAPSE Core is shown in Fig. 4.5.

Technology	TSMC .065μm CMOS		
Worst-case Clock frequency	167 MHz		
Supply voltage	0.9 V		
Max. Logical # neurons	16K		
Max. Logical # synapses	256K		
Physical # neurons	64		
Synaptic storage	Off-chip DRAM		
Max. Synaptic resolution	8-bytes		
Arithmetic	Fixed-point		
Neuron-model	Reconfigurable generalized LIF		
Core logic area	16.4 mm^2		
Benchmarks	SCWN	SDBN	SCNN
Core power dissipation (mW)	157.72	184.03	405.952
Total power dissipation (mW)	530.123	573.48	1455.397

Table 4.1: Characteristics of synthesized CyNAPSE core used for experiments

4.4 Implementation details

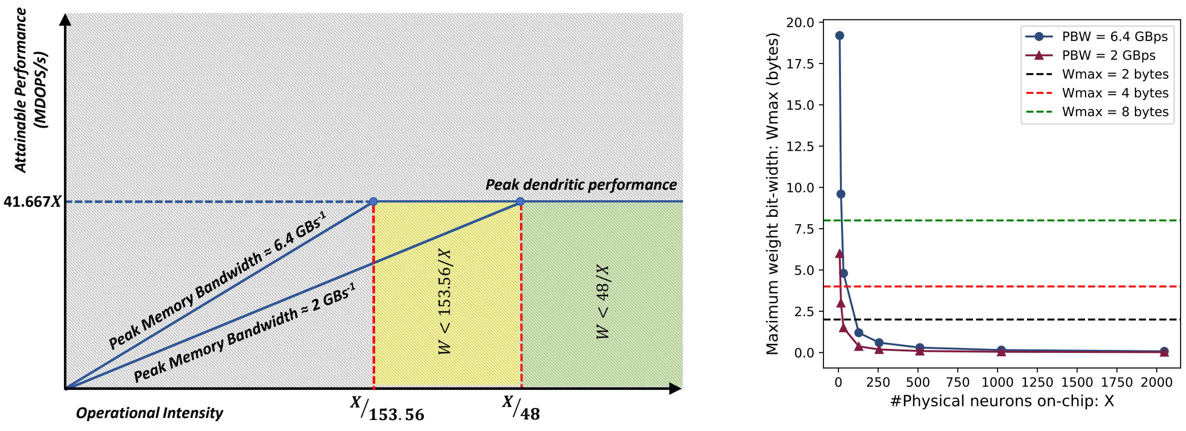
The CyNAPSE Core has been completely implemented in fully synthesizable Verilog HDL and functionally verified for all three benchmarks described in Chapter 3 using ModelSim. Although the memory (external DRAM Synaptic weights, on-chip SRAM dendritic trees and FIFOs) were not synthesizable, they were modeled in RTL for verification. The entire implementation is provided for public use at the CyNAPSE RTL Repository. The logic portion of the core was synthesized to a TSMC 65nm library using a supply voltage of 0.9V using the Cadence SOC Encounter RTL Compiler. As will be discussed in later chapters, measuring power consumption is the prime experimental focus of this work. From synthesized and verified netlists, representative activity files were dumped to characterize and estimate power consumption of the core's logic portions. Synopsys PrimeTime was used to annotate and estimate power from VCD and SAIF files. For the memory structures, Ramulator [136], DRAMPower [137] and CACTI-P [138] were used to calculate off-chip and on-chip power consumption. The details of further experimental setup are discussed in Section 5.3. Lastly, Table 4.1 lists the characteristics of the synthesized CyNAPSE Core that was used for further experiments in this study.

CHAPTER 5. ADAPTIVE MEMORY MANAGEMENT

As seen in the microarchitecture discussion in Chapter 4, the dendritic memory resides close to the neurons holding relevant synaptic data for the current timestep. This data is periodically refreshed via neuron circuits during the update cycle. However, all synaptic data cannot be stored off-chip because with deeper networks, these parameters very quickly grow out of storage resource limits. Hence, CyNAPSE provisions off-chip DRAM synaptic storage and regularly goes there to fetch weights. DRAM is reliable, fast and maximizes storage efficiency, an important factor for larger and deeper network capabilities. But how does that affect its energy efficiency? In this chapter, I will discuss CyNAPSE's power consumption and set up a motivation for solving the problem at hand. Following that, I will discuss caching, its applicability to CyNAPSE's memory system and conventional cache management policies. This will lead to a detailed description of the proposed memory management scheme. A description of the experimental setup will be provided before presenting the results of evaluation for the proposed policy.

5.1 Power consumption profile

For each spike generated within a Spiking Neural Network (SNN), whether from the input or from within the network, all post-synaptic weights of the relevant neuron ID are loaded and added onto the relevant dendritic trees. Depending on the average connectivity of the network, the amount of time spent in the routing cycle can vary but it is always a large majority of the simulation time. Fig. 5.1a presents a roofline analysis [139] of CyNAPSE's routing cycle while Fig. 5.1b shows the maximum weight data-width for a configuration to have compute-bound performance. The result is not counterintuitive in that most artificial neural network kernels themselves are quite memory-bound even on distributed processing hardware [21, 27, 140, 141]. Spiking networks, on top of that, have completely local computation and extremely sparse global communication. With practical



(a) (b)

Figure 5.1: (a) Roofline model showing constrained performance of CyNAPSE's routing cycle under various conditions. As we move towards lower steady-state bandwidths, higher weight bit-widths(W) and higher physical neurons on-chip(X), performance is much more likely to be memory-bound. (b) confirms this hypothesis. It shows how maximum weight bit-widths for compute-bound performance vary against the physical number of neurons for two peak bandwidths (PBW): the pin bandwidth and a lower steady-state bandwidth. The area above either curve is memory-bound while the area below is compute-bound. It is easy to see that for most practical configurations, CyNAPSE will have heavily memory-bound performance

data-widths, therefore, SNNs require very little compute performance and puts great pressure on the memory path.

However, it is not clear how largely memory-bound performance affects the total power consumption of CyNAPSE. A configuration with $N = 16384$ (sufficient for all of our benchmarks) is taken and the power consumption of the system is measured (see Section 4.4 for details) for each of the three benchmark networks. This includes power consumed by the logic portions of the core, the on-chip dendritic SRAMs, the on-chip FIFOs and the off-chip DRAM storage. The resulting profile is shown in Fig. 5.2. It can be seen that the most significant share of the system's power consumption results from retrieval of synaptic weights from a remote DRAM storage. One can allocate more on-chip neurons to increase performance by making the kernels less-memory bound but the power consumption still results highly from the memory access path. Therefore, regardless of hardware configuration, both the importance of memory power consumption and the opportunity

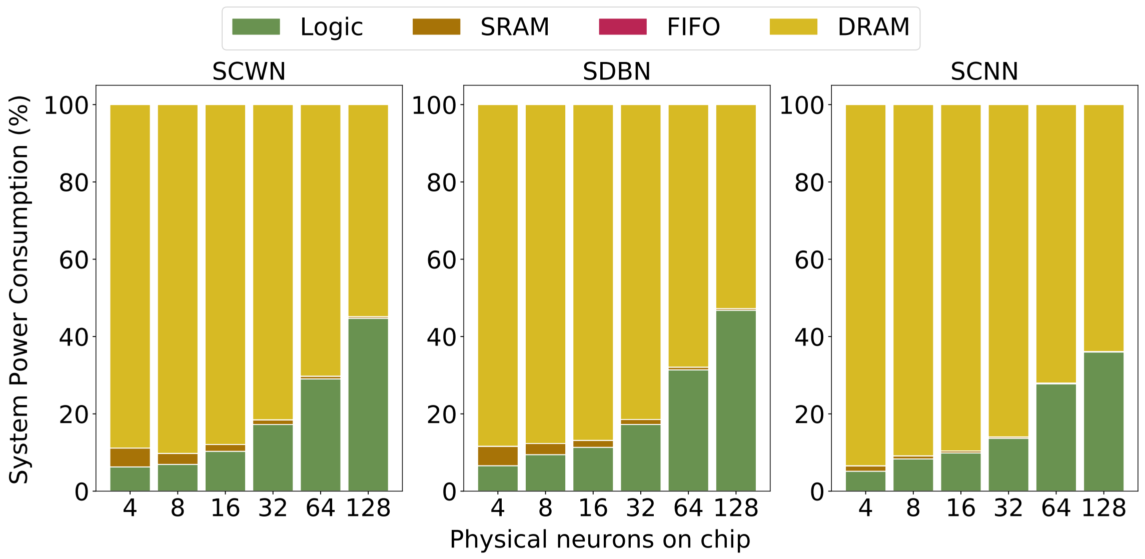


Figure 5.2: Net system power consumption of CyNAPSE for each benchmark showing various consumption sources

to optimize this process in such a system is quite clear. This motivates the need for architectural exploration in this area to find possible solutions to offset the bottleneck in an efficient manner. As has been argued, weight bit-widths are an important factor contributing to the memory bottleneck. It is the same for power consumption of the system. Although algorithmic optimizations like pruning and quantization of synaptic weights [142, 143] might help in this regard, they lead to finite degradation in accuracy of the network. For spiking networks the allowable margin in accuracy is low and therefore, the attempt is to make microarchitectural optimizations that are agnostic to accuracy so that algorithmic changes are still compatible but not necessary. To that end, this work is focussed on studying the memory access patterns and data locality in SNN processing to cleverly mitigate redundant accesses in a workload-aware manner.

5.2 Energy-efficient memory management techniques

5.2.1 Cache management policies

In general purpose computers, caches consistently exploit temporal and spatial locality of memory accesses to reduce redundant data and instruction retrieval from main memory by storing frequently accessed items close to processing [144]. This reduces memory traffic, by distributing it over (usually, multiple levels of) small local storage thereby improving performance and sometimes, energy-efficiency. Even in multicore computing, each core is closely coupled with its own *L1* cache and have shared later-level caches to improve core efficiency as well as overall system efficiency [145, 146]. Since caches are fast but highly constrained storage, data needs to be temporally evicted to make way for new data allocation. In a *direct-mapped cache* with only one memory block allocated to a particular *index* (global memory addresses separated by regular intervals), this is a trivial problem. However, these caches miss exploitation of locality by being so strict and have given way to *set-associative caches* that allocate multiple blocks or *ways* for a single index (set) [147, 148]. Associativity can range from two-way till fully associative where there is a single set and all cache blocks are ways of that set. With an efficiency in storage, comes a non-trivial decision of which way to allocate new incoming data on. For a general-purpose computer, this is difficult since it appeals to multiple application domains and kernels have highly variable memory access patterns. This results in higher than normal misses in cached data and the performance naturally degrades. While caches have benefited greatly from clever structural, functional and compile-time adjustments [149, 150, 151, 152, 153, 154, 155, 156, 157], the amount of improvement in performance and/or net power dissipation is still highly dependent on the replacement policy. A good strategy of replacement is, therefore, a significant research problem in caches and their effectiveness is often critically dependent on it [158].

Conventional cache policies are based on temporal access history to drive replacement decisions. A classic example is *Least Recently Used (LRU)*, a policy still used in many modern architectures because it of its great cost-to-performance ratio. Even less hardware cost is incurred in implementing *random* replacement policy, wherein a random way is selected for eviction. While they perform reasonably well for general purpose workloads to a certain extent, it has been pointed out that the degree of associativity in a cache limits their capacity to model *unique references* to the same cache block [159]. In the subject of replacement policies, the early work of L.A Belady [160] laid down the guidelines for an *optimal replacement policy* for virtual memory systems which hold for hardware caches as well. Simply put, Belady's policy suggests replacing the block that is re-referenced farthest in time from the present. However, this requires a practically infeasible view of the future accesses. Building on the same, policies like DIP [161], RRIP [162], LIRS [163] have explored speculative architectural techniques for general purpose processors to equip set-associative caches with the ability to predict re-reference of every cache block and therefore make replacement decisions based on dynamically collected past access data. However, because of the nature of event-driven simulation, CyNAPSE can indeed exploit some forward visibility in memory accesses. Because there exists a finite number of future spike events listed in the input FIFO queue, there is a scope to improve locality by (pre)fetching neuron metadata and post-synaptic weights related to the neuron ID in these events. The amount of forward visibility is proportional to the relative difference in latency between allocating its cache and processing one spike event in entirety. Also, for neural inference, there is no requirement of writing to cache (or memory), which in effect is equivalent to having just an instruction cache in the hierarchy. Therefore, given a domain-specific simulation framework, a domain-specific memory management policy is proposed to capture memory behavior specific to the SNN kernels that conventional policies fail to account for.

5.2.2 Proposed management strategy

As discussed in Section 4.2.1, the simulation of the SNN kernel starts off as soon as the Input FIFO is populated with initial events from an off-chip poissonian or natural spike source. Thereafter, as the nature of FIFO dictates, further events are enqueued at the FIFO write pointer only upon the dequeue of one event from the read pointer. The dequeued neuron ID induces its own synaptic lookup process from the input router through the memory hierarchy. At the same time, the queue already contains F more events where F is the length of the input FIFO. The first level cache in the hierarchy can look into the queue to monitor these events and accordingly allocate its contents in a deterministic way for accesses that are guaranteed in the future by tagging the allocated blocks with a *reuse score*: number of times the block is expected to be used in the future. As such, two times for each event may be defined: a *read-time* i.e. when an event is read by the cache for prefetching, but not actually dequeued from the FIFO queue and *route-time* i.e. when this event is eventually dequeued for synaptic lookup and dendritic placements. Before the start of simulation, the cache is warmed-up with events in the queue to give the *reading*, its necessary head start over the *routing*. This cost in latency is amortized in a short while owing to improvements in performance and energy consumption throughout simulation. However, this is not usually done to the full extent of the length of the queue (FIFO lengths can be very large since they don't typically demand much hardware resources or energy expenditure). Rather, a certain *lookahead distance* is selected carefully to exploit reuse without incurring sizeable *thrashing* (unwanted eviction) from events that are re-referenced later. After warming up, there is one read per completed route and the simulation proceeds accordingly. The policy can be described in detail using each type of cache hit/miss scenarios that is typical in cache accesses [147] as follows.

5.2.2.1 Compulsory miss at warm-up and read-time

When an event is monitored off the queue at its read-time, all relevant memory addresses corresponding to its metadata and synaptic data (see Section 4.2.2) are generated using network kernel information. So, an unallocated way in the cache is now tagged and the import from main memory (or next level cache) is started. At warm-up, there is no contention from the routing because it has not started yet but queue read requests need to be serviced even when there is. This requires a cache with two independent read-write ports. Although, depending on the steady state DRAM bandwidth, multiple read requests can be served within the regime of one routing cycle, this work studies a one-to-one ratio to keep the design sufficiently simple to achieve and evaluate the benefit from it. A future focus would be to consider multiplying this ratio. A *compulsory miss* means the concerned block was encountered for the first time since the last time that the cache was flushed [147]. Thus, all blocks with a compulsory miss are marked with a reuse score of one, which basically refers to one guaranteed access at route-time to this block in the future.

5.2.2.2 Hit at read-time

After a while, there can be a hit in the cache of an already allocated block that has been encountered before, whether from the same neuron ID or from a closely residing one in the memory. On a hit, the only course of action within the cache is to increase the reuse score of the block by one. No further import request is issued to the next level and the next event in the queue is monitored.

5.2.2.3 Capacity or conflict miss at read-time

Depending on cache capacity, there will be misses at a certain point in the simulation at read-time. This is either due to limited capacity or limited associativity [147]. This requires eviction of a way in order to make room for new data. However, there are potential issues that concern read-time replacements. Accordingly, three different approaches are proposed:

- *Conservative approach*: Blocks that have multiple guaranteed accesses can be thrashed by blocks that do not end up with a lot of reuse and will therefore lead to heightened thrashing at read-time and redundant memory traffic at route-time leading to unnecessary energy consumption. Hence, conservative approach does not allow any read-time replacements.

- *Aggressive approach*: By disallowing read-time replacement totally, there will be loss of reuse from blocks that do end up generating a great amount of reuse and will also lead to redundant traffic at multiple routes. Hence, aggressive approach allows all read-time replacements by evicting the way with the lowest reuse score.

- *Intelligent approach*: Read-time replacements are only allowed when the minimum reuse score in the concerned set is less than a certain reconfigurable *reuse threshold*. This, theoretically, cuts down on missing reuse blocks while also limiting the introduction of thrashable blocks because of low reuse score.

Under any scheme, if and when a read-time replacement scheme actually takes place, the new block is similarly tagged with a reuse score of one.

5.2.2.4 Compulsory miss at route-time

Since all blocks are encountered at least once before the route, there are no compulsory misses at route-time. There are misses only when the read-time policy, for instance, opts out of replacement or is thrashed by a later block before one of its route-times is reached.

5.2.2.5 Hit at route-time

A hit at route-time essentially means that one of the guaranteed future accesses to a particular block has now been realized. This is therefore associated with the decrement of the reuse score by one.

5.2.2.6 Policy miss at route-time

The management policy, as discussed above, can opt-out or thrash read out blocks before any or all of their route-accesses are served. This leads to a miss at route-time. This requires the router to issue an import request at route-time to the next level for the relevant block/s. In order to allocate this in the cache, the simple scheme of evicting the lowest reuse score block is proposed. This is because, in the baseline scheme, allocation is compulsory. This is revised in a later discussion. If an allocation does take place, the block is used up instantly since it has been requested at route-time. So, the reuse score of the freshly imported block is a zero as there are no guaranteed future accesses to this block. The entire baseline scheme is summarized in Fig. 5.3.

5.2.3 Network-adaptive enhancements

The proposed scheme works precisely because the input events are generated at a significantly higher throughput than the expected latency of completing a single route-cycle for a particular event. However, besides the input events, spikes generated within the network have to be served

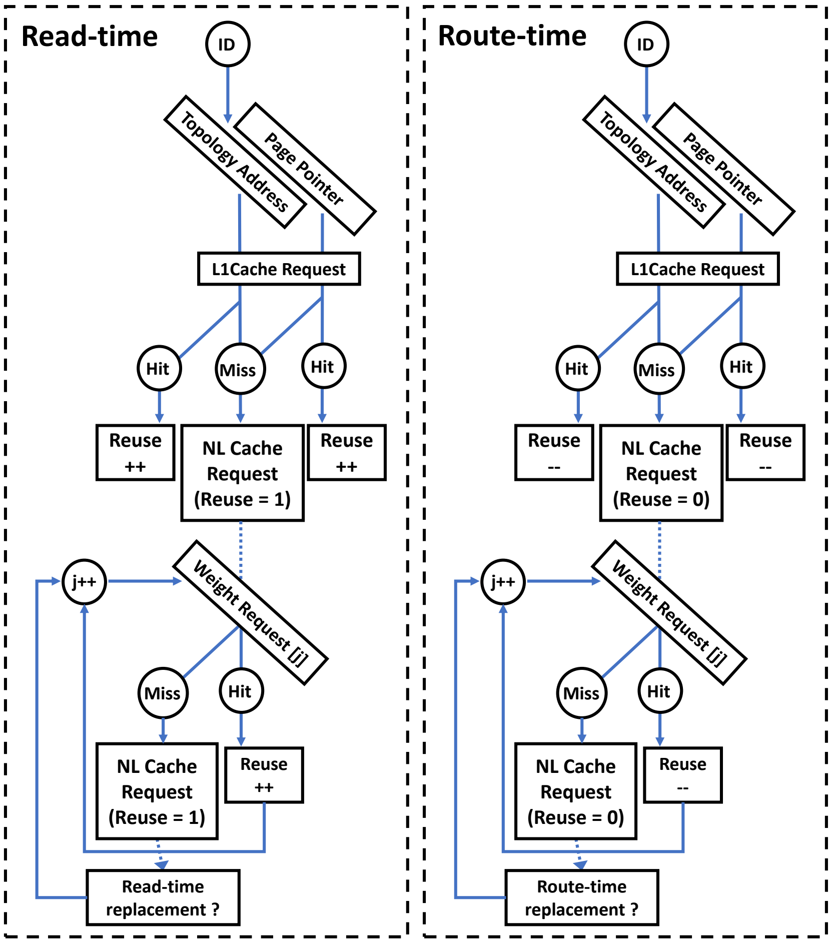


Figure 5.3: Baseline memory control strategy in read-time and route-time access of cache

as well and are produced in the timestep directly preceding their routing. This gives very poor forward visibility for internally generated events. As shown in Chapter 3, the input acitivity can have varying relative importance to internal activity and therefore, some benchmarks with significant internal activity might not benefit from the proposed scheme at all. This motivates the need for network-specific enhancements that adaptively equip the scheme to changing activity patterns throughout simulation.

The CyNAPSE core is programmed with compile-time network information like layer types (conv2d, dense, subsampling), excitatory and inhibitory neuron ranges, neuron parameters etc. This is *static* information that can provide network-specific enhancements rightaway and adap-

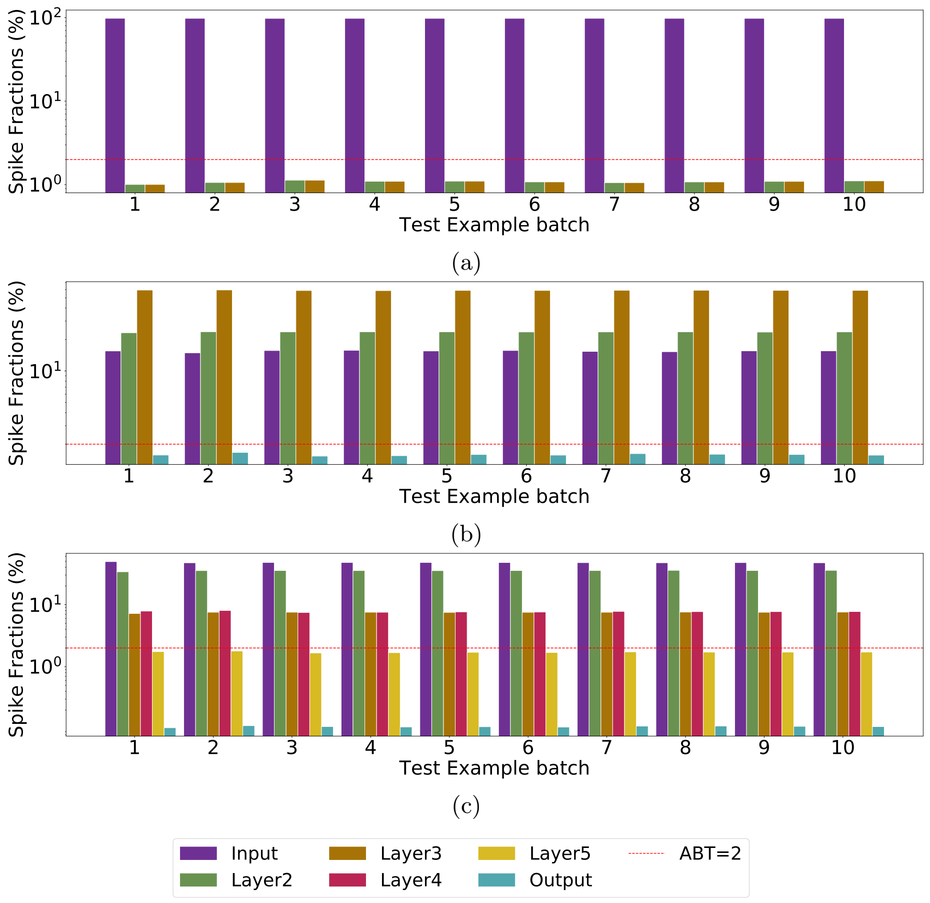


Figure 5.4: Dynamic spike statistics generated by CyNAPSE software simulator to adaptively handle memory requests.(a), (b) and (c) show layer-wise activity *fractions* for the SCWN, SDBN and SCNN benchmarks respectively with time and how they compare to the activity bypass threshold (ABT)

tively extend our proposed strategy. Furthermore, behavioral diagnostics can provide *dynamic* information like spiking activity of different layers to identify high-activity as well as dormant regions of a network. The source of all this information is the auxiliary queue, where all internally generated spikes reside at some point. In this work, queue statistics are dumped after the simulation of a single *batch* of example stimuli and dynamic information is computed. This progressively helps to improve the policy. Spiking activity, for example, is collected at a layer-by-layer granularity. Individual neuron spiking information can also be collected but this leads to very high storage and logic overhead which can offset the energy savings expected. Therefore, a layer granularity for diagnostics was chosen. Fig 5.4 shows the dynamic statistics collected over 10 batches of test examples for each network benchmark while simulating on the CyNAPSE software simulator (see Section 5.3). Two techniques are proposed in this regard, to extend the management strategy.

5.2.3.1 Cache bypassing

All three benchmarks have significantly varying spike signatures throughout their layers. The SCWN benchmark, for instance, shows very little internal activity compared to the high degree of input activity generated for it. The distribution is highly skewed in favor of the poissonian spikes in the input layer and the corresponding input neuron IDs. Therefore, this network is inherently suited for our baseline scheme and should benefit greatly from it. However, the SDBN and SCNN benchmarks have different distributions to this and to each other. The internal activity is considerably higher and in case of the SDBN, even higher than the input activity. For networks that have considerable internal activity, a bypass scheme in the cache is proposed. Neurons belonging to sparse activity layers are allowed to bypass cache allocation so that neurons in high activity layers are not thrashed by contention. This information can be static: for example, output layer neurons for feed-forward networks that do not have any post-synaptic connections, or dynamic: for example, low activity neurons in simulation. For dynamic information, an *Activity Bypass Threshold (ABT)* for average layer activity is maintained below which, all neurons of the concerned layer are granted a bypass request and above which, allocation at route-time is enforced. On a bypass request, the memory control does not allocate a cache way and issues a one-time retrieval request for all meta and synaptic data belonging to that neuron ID.

5.2.3.2 Line protection

Similar to extremely dormant regions in a network, there can be internal regions of heightened activity. For examples, layers 2 and 3 of the SDBN and layer 2 of the SCNN, show relatively high activity when compared to other layers. The baseline management scheme does not cover these neurons and can lead to redundant accesses when compared to conventional replacement policies.

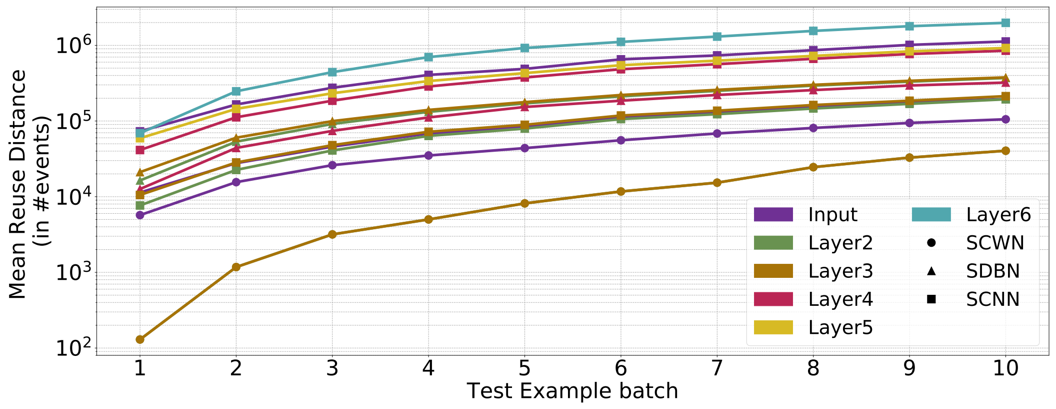


Figure 5.5: Layer-wise mean reuse distances shown for all layers in all benchmarks

To avoid that, a line protection scheme is proposed that protects the cache lines that hold the data corresponding to the neuron IDs belonging to layers of high activity (all layers above ABT). This is done by tagging these lines with a *probable reuse score* dynamically determined from network diagnostics collected in the software simulator. For reference, the mean reuse distances of different layers in all three benchmarks are shown in Fig. 5.5. The probable reuse score should be inversely proportional to this distance to effectively account for the expected number of reuses within a time window.

5.3 Experimental Infrastructure

The implementation details of the low level design was discussed in Section 4.4. Here, I will discuss the experimental setup for high-level exploration of the memory subsystem. A CyNAPSE software simulator has been built for this purpose. This software simulation models CyNAPSE's neuron model and core architecture in an object-oriented fashion with discrete timestepped simulation. In other words, it maintains a one-one equivalence with the hardware architecture thereby confirming accurate hardware results. However, the simulation is not cycle-accurate. Therefore, no performance or energy consumption measures are taken directly from it. Instead, the simulator is used to dump statistics and diagnostic information about specific benchmarks using which, it can better adapt the caching scheme. The cache is simulated by an in-house cache simulator which can be easily interfaced with the software simulation tool. The cache simulator provides statistics like tag array and data array accesses per spike, hit rate, miss profile etc. By using the software simulator, memory traffic can be dumped into an address trace file. This address trace can be converted into a DDR3 command trace using Ramulator [136]. Ramulator sets up a config with the speed, architecture and organization of the DRAM and generates JEDEC standard command traces relevant to that memory chip. These command traces are then routed to DRAMPower 3.1 [137] in a similar organization, speed and architecture configuration, to estimate the energy consumption of these traces. For this work, a 256MB DDR3 x8 DRAM with a 1600MHz pin bandwidth was used which provides sufficient storage for all the benchmarks used. Although, the precision requirements of each network can be different, 8-byte precision is used in all benchmarks to have a fair comparison of memory access patterns.

Using the energy consumption of traces and timing information from the core netlist simulations, power consumption of the memory subsystem was calculated. For the logic portions, as discussed in Section 4.4, activity files annotated with the benchmarks provide power consumption. In a cached configuration, the power consumption is measured using the same experimental flow, except, cache simulator statistics like tag and data array accesses are coupled with CACTI's UCA cache energy estimates to model net power consumption of caches [138].

For evaluating the dynamic adaptive enhancements, the simulator provides simple routines to dump FIFO queue contents after each batch, calculate the statistics required, feeds them into the cache simulator and restarts the simulation from the last checkpoint after forwarding the cache contents. Fig. 5.6 summarizes the experimental infrastructure across low-level design and high-level exploration. One limitation of this experimental setup is the insufficient simulation time. Owing to very long individual simulations, only a subset of the MNIST dataset with 100 test examples (in 10 batches) was used. However, the 100 examples were chosen uniformly to contain equal number of random examples from all classes so as to offset any bias in the simulation infrastructure.

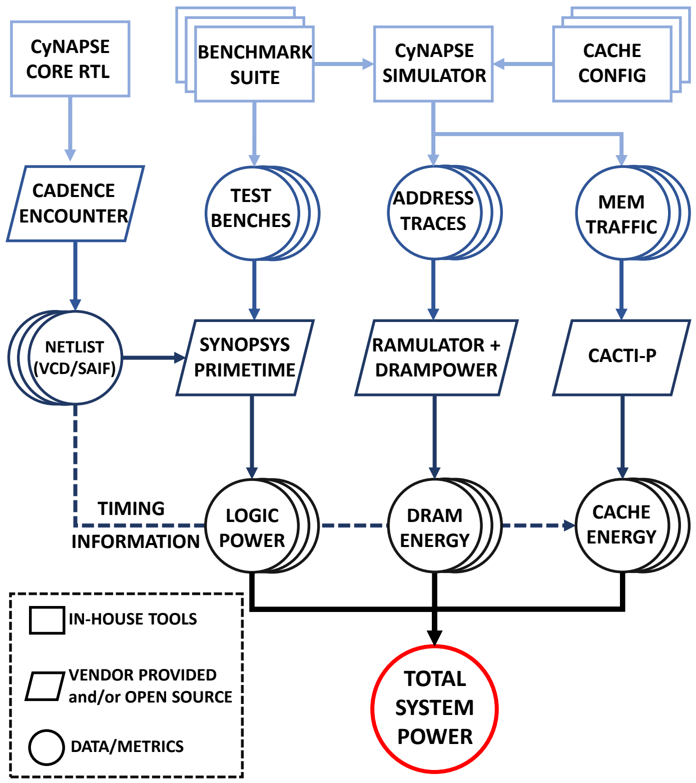


Figure 5.6: Experimental infrastructure and flow of data between tools

5.4 Results

Three design parameters for caches, in general, were explored first, with conventional cache management policies. These are *block-size*, *associativity* and *cache depth* (number of cache blocks). After exploring all these design parameters using a binary search and staying within area constraints of the total logic chip area, the configuration with the best return-on-investment was selected. For the benchmarks used in this study, on average, this configuration was a 256 KB 4-way set-associative cache with 64 byte blocks. This was selected as the operating point for all comparisons to ensure fairness of evaluation and similarly provisioned alternatives in this work. In this section, I will first validate the assumption from Section 5.2.2.3 about read-time replacements and attempt to explain the results of exploration. Using this verdict, I will evaluate the effectiveness of the proposed scheme in general, for each benchmark, in comparison to conventional management schemes in reducing

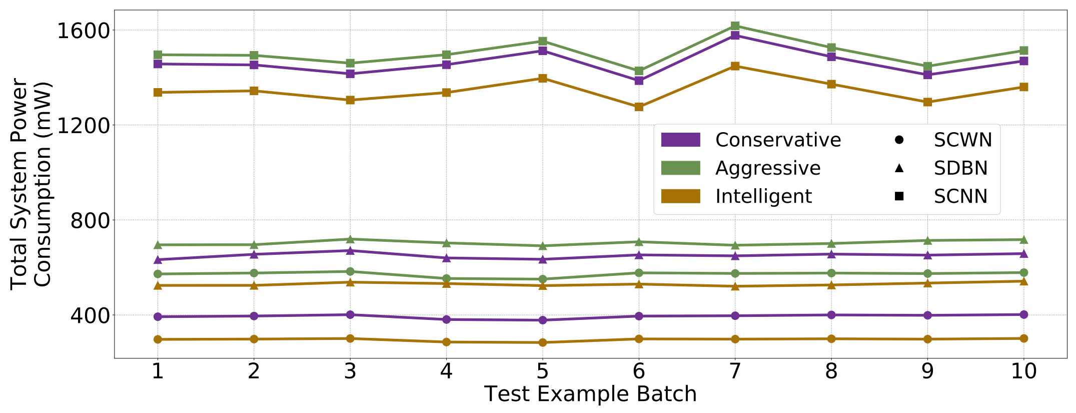


Figure 5.7: Experimental results of exploring different read-time replacement policies for each benchmark

total system power consumption. At the same time, the relative benefits of extending the policy to include dynamic adaptive network-specific enhancements are also quantified.

5.4.1 Read-time replacement

As mentioned before, the CyNAPSE software simulator provides real-time statistics in the core. Similarly, the cache simulator provides hooks to select and dump cache contents at regular intervals as desired. It also provides a log of replacement decisions and an image of the set before and after the decision. Using the average reuse scores for evicted blocks, a minimum reuse threshold was fixed for each benchmark. This is required to validate the intelligent approach to read-time replacements. Fig. 5.7 shows the results of experiments on read-time replacement policies.

For all three benchmarks, the intelligent approach outperforms the conservative and aggressive approaches. Aggressive approach opts for replacing all conflicts which essentially nullifies locality by ignoring reuse cores of blocks already allocated when reading from the queue. Conservative approach also ignores reuse opportunities by totally opting out of any replacement at all. This leads to redundant accesses at route-time. However, unlike conservative, aggressive approach also leads to severe ping-ponging of blocks at read-time which particularly worsens the situation. Therefore, conservative approach performs better than aggressive, on average for all benchmarks. For bench-

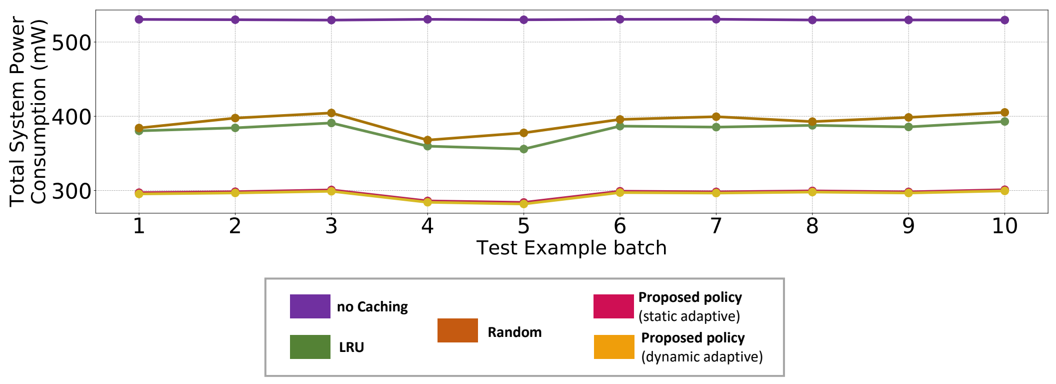


Figure 5.8: Comparative analysis of replacement policies towards savings in net system power consumption for the SCWN benchmark

marks with long reuse distances (e.g. SCNN), the loss due to read-replacement policies is much smaller than benchmarks with shorter reuse distance (e.g. SCWN).

5.4.2 LRU vs Random vs Proposed policy

This section evaluates total system power consumption as a function of time (test example batches) for all three benchmarks. With similarly provisioned cache configurations, the conventional replacement policies are compared with the proposed policy.

5.4.2.1 SCWN

Fig. 5.8 shows the evaluation for the SCWN benchmark. The SCWN has a much lower overall spiking activity when compared to the others. On an average, it produces 2.14404 spikes per timestep (including internal spikes) or 2144.04 spikes per test example. To induce a stable inference, each input poissonian neuron needs to generate multiple spikes. This makes SCWN highly amenable to neuron data reuse and exploitation of temporal locality. In short timescales, it is well captured by LRU. For every example, there are also some *winner* pyramidal neurons in the winner-take-all circuit that will generate higher than usual activity when inhibiting the basket cell neurons. 85-90% of all cache misses are profiled as capacity misses so they are not limited by associativity of the

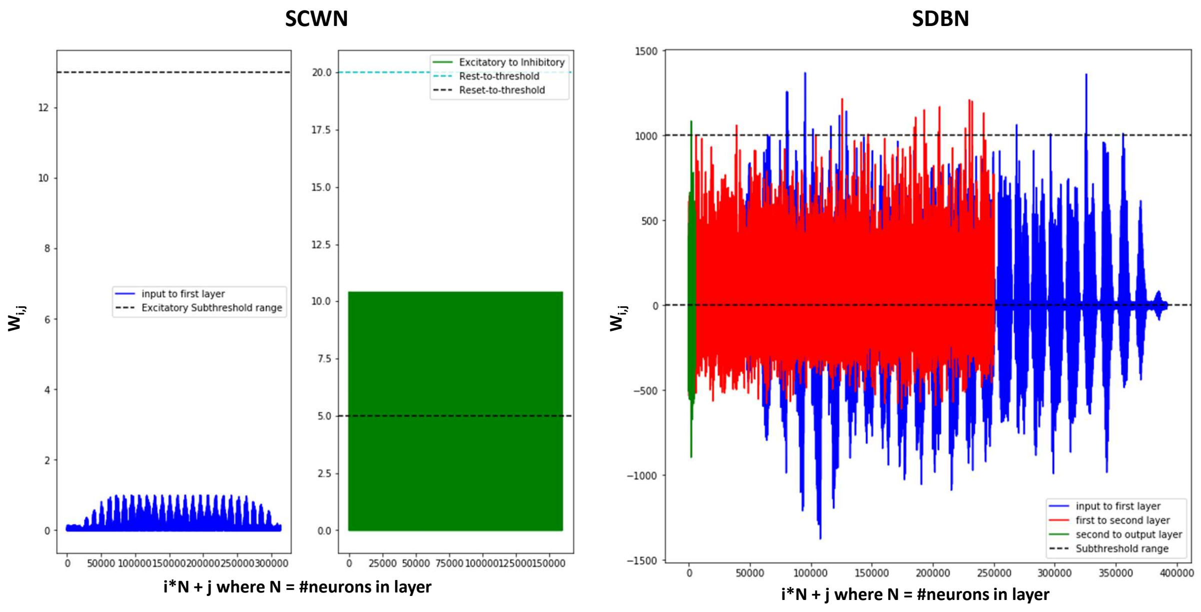


Figure 5.9: Difference in distribution of synaptic weights in SCWN and SDBN showing large synaptic weights reaching up to the subthreshold ranges for the latter, and sufficiently small weights for the former.

cache. Random replacement, on the other hand, cannot capture reuse beyond example digits, but if a cache is sufficiently associative, it can still handle conflicts close to LRU.

The proposed policy collects reuse information for all layers in the SCWN from reuse distances both within and beyond a single example stimulus. An activity bypass threshold (ABT) of 2% is set explicitly and dynamic network-adaptive scheme is evaluated. Since SCWN is largely dominated by input events that represents 97.8% of the network's spike signatures, the baseline scheme itself performs very well in this benchmark only with static adaptive enhancements (with cache bypass requests). With the dynamic enhancements, the improvement is very little since it only appeals to the remaining 2.2% of the internal activity.

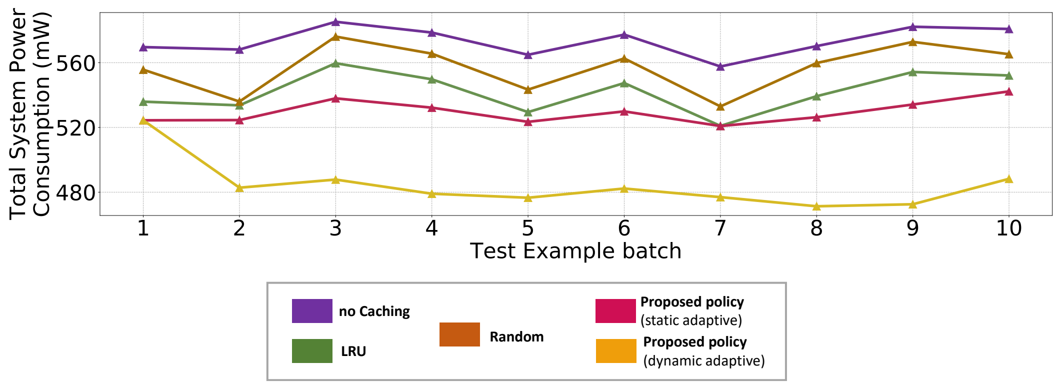


Figure 5.10: Comparative analysis of replacement policies towards savings in net system power consumption for the SDBN benchmark

5.4.2.2 SDBN

The spike signatures from the SDBN benchmark are significantly different from the SCWN. The input frequency is larger much larger and as shown in Fig. 5.9, the weights of the SDBN are close to or even exceed the rest-to-threshold (or reset-to-threshold) range of the internal neurons making it very easy for them to spike. Therefore, the input events (3.99056 spikes per timestep or 3990.56 spikes per example digit) ends up inducing very high internal spiking that multiplies with deeper layers. Particularly, the third layer has a very high spiking activity. Due to low input activity, most of the reuse is captured well by LRU. So relative benefits from switching to the proposed policy are modest. Fig. 5.10 shows the trend.

However, on applying dynamic adaptive schemes with the same ABT, a much greater relative savings in system power consumption is observed. Reuse scores inversely proportional to dynamically observed reuse distances are progressively applied to the internal neurons for line protection. Most neurons in the third layer benefit highly from these enhancements and therefore, a marked improvement is observed for this benchmark.

5.4.2.3 SCNN

The convolutional neural network has a very high overall activity throughout the network. It produces a total of 219.2259 spikes per timestep or 21922.59 spikes per test example on an average.

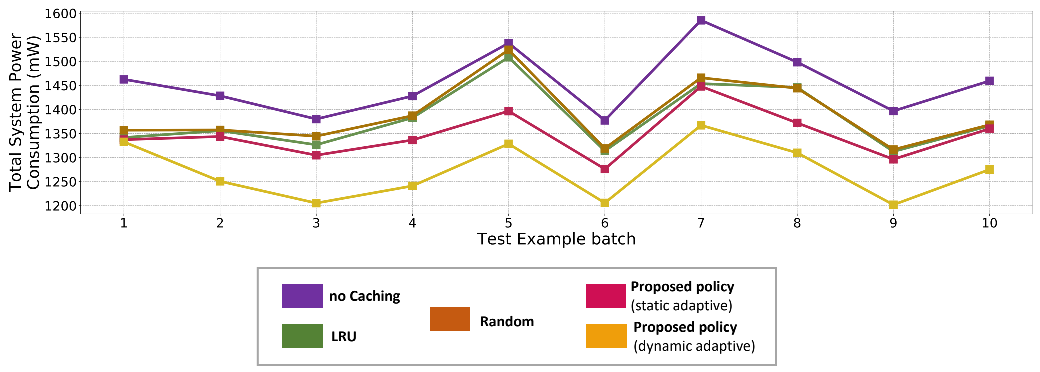


Figure 5.11: Comparative analysis of replacement policies towards savings in net system power consumption for the SCNN benchmark

It has a much longer neuron reuse distance in all layers when compared to the other benchmarks. With a cache of limited capacity, conventional cache policies find it very difficult to capture any locality. A healthy fraction of the input activity is, nevertheless, targeted by the proposed policy. Fig. 5.11 shows a comparative analysis for this benchmark which shows the proposed policy clearly outperforming LRU and random.

It can be noted that there is a great difference between the static adaptive scheme and dynamically enhanced scheme for the SCNN for similar reasons as the SDBN. However, many neurons in the processing conv2D and average pooling layers are inactive for all stimuli because of the commonly known observation of *sparse activations* in convolutional neural networks [164]. Therefore, very few neurons request allocation under a line protected enhancement. Therefore, SCNN benefits much lesser in percentage when compared to the relative savings in the SDBN benchmark.

5.5 Summary

The proposed policy outperforms the conventional cache management policies for all benchmarks. In general, for benchmarks having a high degree of biological plausibility, the normalized synapses lead to sparser activity in deeper layers and hence the proposed scheme with static enhancements are very well suited to the kernel. In trained-converted classical networks, however, the activity can have erratic patterns which requires dynamically adapting the scheme to perform

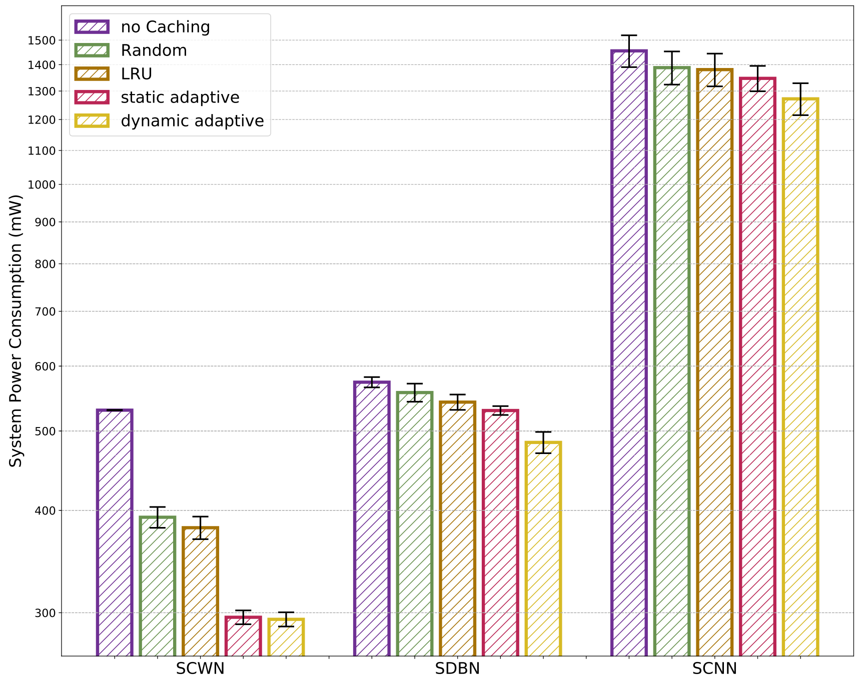


Figure 5.12: A graphical summary of the evaluation

satisfactorily for these kernels. Even so, the distribution of layer types causes great variation in the effectiveness of the scheme. The results are summarized graphically in Figure 5.12 and numerically in Table 5.1.

Table 5.1: Relative energy savings achieved using different policies

Benchmark	**LRU v/s baseline**	**Random v/s baseline**	**Proposed Policy (static adaptive) v/s baseline**	**Proposed Policy (dynamic adaptive) v/s baseline**	**Proposed Policy v/s LRU**
SCWN	28.13%	25.99%	44.13%	44.45%	22.71%
SDBN	5.46%	2.88%	7.65%	15.55%	10.67%
SCNN	5.12%	4.59%	7.4%	12.61%	7.9%

CHAPTER 6. FUTURE-WORK AND CONCLUSIONS

6.1 Extensions

In this work, a single operating point was compared by carefully allocating similar provisions to all considered policies. However, not all structural configurations in caches react similarly to all replacement policies. One of the imminent future extensions will be to study various operating points and explore the design space further.

As mentioned in Section 5.2.2.1, depending on steady state DRAM bandwidth, more than one read requests can be served within the time a single event route is completed. This provides the scope of a higher number of total reads within a route cycle. This can be studied in simulation and possibly checked for coherency and other issues if sizable efficiency gains are observed.

Lastly, individual parameters like lookahead lengths, reuse thresholds, bypass thresholds etc. can be explored for potentially interesting experimental waypoints.

6.2 Architectural enhancements

A possible future work is to look at processing larger benchmarks with increased processing demands. This will require greater compute performance and parallel processing could be key. A multi-core CyNAPSE architecture can be explored with coarse-grain multiprocessing that can share neuron-processing of a single layer in a single timestep across multiple nodes and thereby complete simulation of a timestep faster since there are no data hazards between different neurons within a timestep. Apart from faster inference, this might also save more energy by saving more memory traffic at each node with reduced local storage and more hierarchical caching. Fig. 6.1 shows a conceptual sketch of the possible hierarchy in such a multi-core system. With this, opportunities to explore a number of architectural design points will appear, like interconnect architectures, network-on-chip, communication protocols etc.

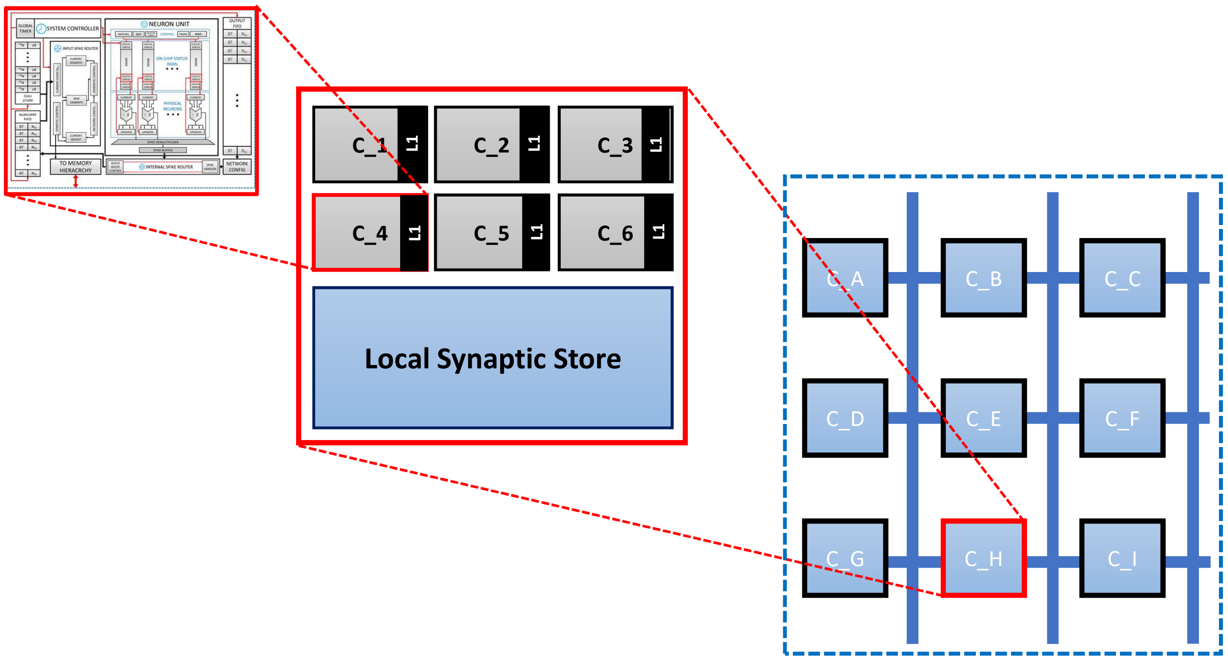


Figure 6.1: Conceptual diagram showing a possible avenue of future work. The multi-core system could consist of individual processing clusters (C_A, C_B etc.) and communication infrastructure connecting these clusters. Each cluster could contain multiple CyNAPSE cores (C_1, C_2 etc.) with their private L1 caches and a local synaptic storage adding another large reservoir to the multilevel memory hierarchy.

Another possible optimization within the CyNAPSE core could be towards controlling leakage power dissipation since majority of its logic power consumption is in the idle state. This includes simple techniques in the CAD process like gating or architectural techniques like sleep modes in SRAMs [165] or drowsy caches [153] to reduce the core power consumption.

A software stack with a tailor-made compiler infrastructure can lead to large performance and energy improvements. Compiler-driven optimizations on neural network inference have already been demonstrated [166, 167]. A parser for a high-level language for SNN description coupled with a domain-specific compiler stack can be highly beneficial for the project in the future.

Besides neuromorphic acceleration, the memory management policy presented in this work can, in general, be applied to any event-driven framework, with relevant modifications. It can be considered for an execution model with queue-based input instructions at its front end. Any simulation hardware with these characteristics such as embedded performance and energy counters [168] or general purpose emulators [169] can be benefit from this scheme, if allocation latency at read time can be tolerated by the latency of each individual instructions through the pipeline.

6.3 Learning

As discussed in Chapter 2, plasticity in the synaptic strength or weights are the primary site of learning in real neurons. The benchmarks used in this study, however, have all been offline trained and accelerated at inference using the CyNAPSE fabric. Online and incremental learning is very important in modern applications as it makes a dramatic improvement in overall performance and reduces pressure on datacenters by bringing more computation to the edge. A very lucrative avenue of future work is to equip CyNAPSE with online learning capabilities and then study the resulting memory access patterns and make recommendations in a similar way to reduce the memory bottleneck and energy consumption.

6.3.1 Evolving neural networks

On the one hand, there has been a sustained interest in evolving spiking neural networks [170, 171, 172, 173, 174] because of the amount of neuroscientific support from observable evolving networks in vitro and further from in vivo data. On the other hand, early research in Neuroevolution had generated great interest in simultaneous learning of weights and topology using genetic algorithms [175, 176, 177, 178, 179, 178, 180]. Since then, a number of works [181, 182, 183, 184, 185] have reported evolving neural networks using either an adaptation of [180] or fundamentally different approaches. However, the benefits of hardware acceleration in this space is inconclusive for purely biologically plausible networks or hybrid SNN-ANN approaches.

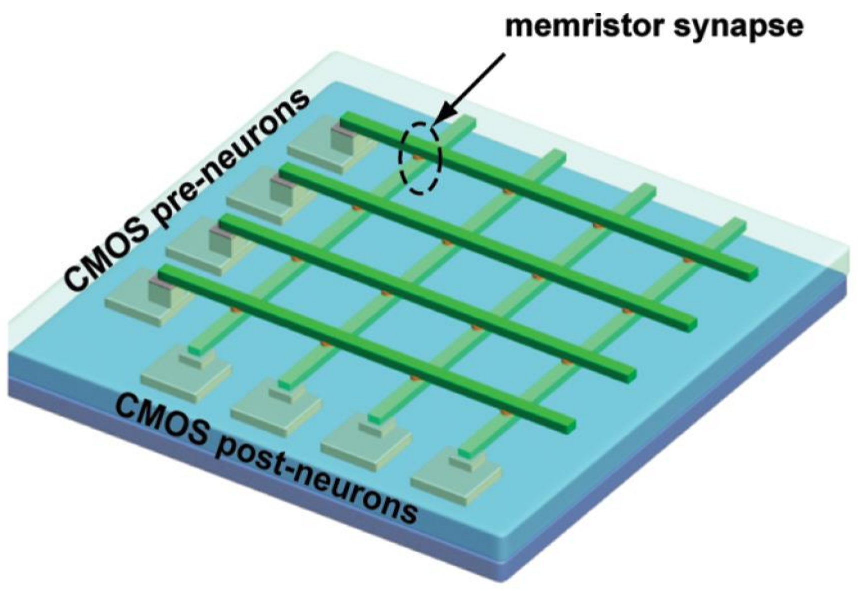


Figure 6.2: Schematic of a synaptic *crossbar* consisting of CMOS neurons integrated with memristive devices sandwiched within CMOS interconnects (figure from [10])

6.3.2 Emerging Devices

Memristive devices [186, 187, 188] have emerged as an attractive solid-state nanoscale candidate for synaptic implementations [10, 189]. They are dense, reliable and gradually being made compatible with CMOS processes for large-scale fabrication. They have been used in neuromorphic implementations in diverse ways [190, 189]. CyNAPSE uses off-chip DRAM storage because of the large amount of storage demanded by large networks. There is a scope of multiplexed, off-chip and on-chip storage of weights using dense devices whose cost can be amortized through the instinctive low-cost learning they provide for spiking biologically plausible systems [191, 192, 193, 194]. Fig. 6.2 shows the implementation of a memristive synapse based neural network in hardware. Other emerging devices like PCM [195, 196], Ferroelectric FETs [197, 198] and STT-RAMs [199, 200] can also be considered for online learning synaptic implementations.

6.4 Conclusion

In this thesis, CyNAPSE was presented in design and implementation. It is a reconfigurable acceleration fabric for processing spiking neural networks. Although it has a sparse computation that leads to inherent efficieny over classicial neural networks, the computation is still losing quite a lot of power by going to an off-chip storage and retrieving synaptic weights. By using an application-specific caching strategy, up to 44% power savings was achieved over the baseline and it outperformed LRU by up to 22%. Because these benchmarks are so different in their original architecture, their training and conversion and come with varying degrees of biological realism and spike activities, the effectiveness of the scheme differs for every workload.

With this work, I expect to make future recommendations on network-specific neuromorphic hardware acceleration which can best manage the memory bottleneck of CyNAPSE architecture while still enjoying the benefits of a simple, scalable and efficient digital design. I also expect to contribute to this field by enabling a more resourceful and easily available platform for neuromorphic applications.

BIBLIOGRAPHY

[1] B. Katz, *Nerve, muscle, and synapse.* McGraw-Hill, 1966.

[2] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018.

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

[4] 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," *arXiv preprint arXiv:1705.06963*, 2017.

[5] L. W. Swanson, E. Newman, A. Araque, and J. M. Dubinsky, *The beautiful brain: the drawings of Santiago Ramón y Cajal.* Abrams, 2017.

[6] C. Mead, *Analog VLSI and Neural Systems.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[7] A. Van Schaik, "Building blocks for electronic spiking neural networks," *Neural networks*, vol. 14, no. 6-7, pp. 617–628, 2001.

[8] P. Livi and G. Indiveri, "A current-mode conductance-based silicon neuron for address-event neuromorphic systems," in *2009 IEEE international symposium on circuits and systems.* IEEE, 2009, pp. 2898–2901.

[9] O. Bichler, D. Roclin, C. Gamrat, and D. Querlioz, "Design exploration methodology for memristor-based spiking neuromorphic architectures with the xnet event-driven simulator," in *2013 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH).* IEEE, 2013, pp. 7–12.

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

[11] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: a comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[12] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[13] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[14] W. De Mulder, S. Bethard, and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech & Language*, vol. 30, no. 1, pp. 61–98, 2015.

[15] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[16] S. Mahdavifar and A. A. Ghorbani, "Application of deep learning to cybersecurity: A survey," *Neurocomputing*, 2019.

[17] M. Paganini, L. de Oliveira, and B. Nachman, "Accelerating science with generative adversarial networks: an application to 3d particle showers in multilayer calorimeters," *Physical review letters*, vol. 120, no. 4, p. 42003, 2018.

[18] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[20] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE micro*, vol. 28, no. 2, pp. 39–55, 2008.

[21] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[22] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[23] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "Diannao family: energy-efficient hardware accelerators for machine learning," *Communications of the ACM*, vol. 59, no. 11, pp. 105–112, 2016.

[24] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "Dadiannao: A neural network supercomputer," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 73–88, 2016.

[25] 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," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[26] A. Podili, C. Zhang, and V. Prasanna, "Fast and efficient implementation of convolutional neural networks on fpga," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2017, pp. 11–18.

[27] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[28] Y. Li and A. Pedram, "Caterpillar: Coarse grain reconfigurable architecture for accelerating the training of deep neural networks," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2017, pp. 1–10.

[29] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "F-cnn: An fpga-based framework for training convolutional neural networks," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2016, pp. 107–114.

[30] B. Fleischer, S. Shukla, M. Ziegler, J. Silberman, J. Oh, V. Srinivasan, J. Choi, S. Mueller, A. Agrawal, T. Babinsky *et al.*, "A scalable multi-teraops deep learning processor core for ai trainina and inference," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 35–36.

[31] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pj/decision 3.12 tops/w robust in-memory machine learning classifier with on-chip training," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 490–492.

[32] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 26.

[33] G. Desoli, N. Chawla, T. Boesch, S.-p. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh *et al.*, "14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 238–239.

[34] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[35] R. C. O'Reilly and Y. Munakata, *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. MIT press, 2000.

[36] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," *Machine learning*, vol. 37, no. 3, pp. 277–296, 1999.

[37] W. Gerstner, "Spiking neurons," MIT-press, Tech. Rep., 1998.

[38] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[39] 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.

[40] D. G. Stork, "Is backpropagation biologically plausible," in *International Joint Conference on Neural Networks*, vol. 2. IEEE Washington, DC, 1989, pp. 241–246.

[41] M. Mahowald, "The silicon retina," in *An Analog VLSI System for Stereoscopic Vision.* Springer, 1994, pp. 4–65.

[42] B. Wen and K. Boahen, "A silicon cochlea with active coupling," *IEEE transactions on biomedical circuits and systems*, vol. 3, no. 6, pp. 444–455, 2009.

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

[44] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[45] E. Hunsberger and C. Eliasmith, "Spiking deep networks with lif neurons," *arXiv preprint arXiv:1510.08829*, 2015.

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

[47] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[48] C. Eliasmith and C. H. Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* MIT press, 2004.

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

[50] D. Attwell and S. B. Laughlin, "An energy budget for signaling in the grey matter of the brain," *Journal of Cerebral Blood Flow & Metabolism*, vol. 21, no. 10, pp. 1133–1145, 2001.

[51] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 x 128 120db 30mw asynchronous vision sensor that responds to relative intensity change," *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*, pp. 2060–2069, 2006.

[52] C. Brändli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240× 180 130 db 3μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014.

[53] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-based neuromorphic systems.* John Wiley & Sons, 2014.

[54] K. Boahen, "A neuromorph's prospectus," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 14–28, 2017.

[55] A. G. Andreou, K. A. Boahen, P. O. Pouliquen, A. Pavasovic, R. E. Jenkins, and K. Strohbehn, "Current-mode subthreshold mos circuits for analog vlsi neural systems," *IEEE Transactions on neural networks*, vol. 2, no. 2, pp. 205–213, 1991.

[56] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri, "Neuromorphic electronic circuits for building autonomous cognitive systems," *Proceedings of the IEEE*, vol. 102, no. 9, pp. 1367–1388, 2014.

[57] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[58] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2019.

[59] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in neuroscience*, vol. 9, p. 141, 2015.

[60] 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*, vol. 12, no. 1, pp. 106–122, 2017.

[61] T. Yu, J. Park, S. Joshi, C. Maier, and G. Cauwenberghs, "65k-neuron integrate-and-fire array transceiver with address-event reconfigurable synaptic routing," in *2012 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2012, pp. 21–24.

[62] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[63] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2012.

[64] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[65] D. Neil and S.-C. Liu, "Minitaur, an event-driven fpga-based spiking network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, 2014.

[66] J. Shen, D. Ma, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, and G. Pan, “Darwin: a neuromorphic hardware co-processor based on spiking neural networks,” *Science China Information Sciences*, vol. 59, no. 2, pp. 1–5, 2016.

[67] Y. Kim, Y. Zhang, and P. Li, “A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 11, no. 4, p. 38, 2015.

[68] A. L. Lehninger, “The neuronal membrane.” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 60, no. 4, p. 1069, 1968.

[69] L. J. Gentet, G. J. Stuart, and J. D. Clements, “Direct measurement of specific membrane capacitance in neurons,” *Biophysical journal*, vol. 79, no. 1, pp. 314–320, 2000.

[70] A. L. Hodgkin and A. F. Huxley, “Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo,” *The Journal of physiology*, vol. 116, no. 4, pp. 449–472, 1952.

[71] A. L. Hodgkin, A. F. Huxley, and B. Katz, “Measurement of current-voltage relations in the membrane of the giant axon of loligo,” *The Journal of physiology*, vol. 116, no. 4, pp. 424–448, 1952.

[72] A. L. Hodgkin and A. F. Huxley, “The components of membrane conductance in the giant axon of loligo,” *The Journal of physiology*, vol. 116, no. 4, pp. 473–496, 1952.

[73] A. Hodgkin and A. Huxley, “The dual effect of membrane potential on sodium conductance in the giant axon of loligo,” *The Journal of Physiology*, vol. 116, no. 4, p. 497, 1952.

[74] 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, 1952.

[75] B. U. Keller, R. P. Hartshorne, J. A. Talvenheimo, W. A. Catterall, and M. Montal, “Sodium channels in planar lipid bilayers. channel gating kinetics of purified sodium channels modified by batrachotoxin.” *The Journal of general physiology*, vol. 88, no. 1, pp. 1–23, 1986.

[76] J. M. Dubois, M. F. Schneider, and B. I. Khodorov, “Voltage dependence of intramembrane charge movement and conductance activation of batrachotoxin-modified sodium channels in frog node of ranvier.” *The Journal of general physiology*, vol. 81, no. 6, pp. 829–844, 1983.

[77] M. I. Behrens, A. Oberhauser, F. Bezanilla, and R. Latorre, “Batrachotoxin-modified sodium channels from squid optic nerve in planar bilayers. ion conduction and gating properties.” *The Journal of general physiology*, vol. 93, no. 1, pp. 23–41, 1989.

[78] M. Eisenberg, J. E. Hall, and C. Mead, “The nature of the voltage-dependent conductance induced by alamethicin in black lipid membranes,” *The Journal of membrane biology*, vol. 14, no. 1, pp. 143–176, 1973.

[79] B. Hille, “Ionic channels in excitable membranes. current problems and biophysical approaches,” *Biophysical Journal*, vol. 22, no. 2, pp. 283–294, 1978.

[80] G. M. Shepherd, *The synaptic organization of the brain.* Oxford university press, 2003.

[81] G. Shepherd, "Microcircuits in the nervous system." *Scientific American*, vol. 238, no. 2, pp. 93–103, 1978.

[82] F. Jug, "On competition and learning in cortical structures," Ph.D. dissertation, ETH Zurich, 2012.

[83] M. F. Bear and R. C. Malenka, "Synaptic plasticity: Ltp and ltd," *Current opinion in neurobiology*, vol. 4, no. 3, pp. 389–399, 1994.

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

[85] A. Morrison, M. Diesmann, and W. Gerstner, "Phenomenological models of synaptic plasticity based on spike timing," *Biological cybernetics*, vol. 98, no. 6, pp. 459–478, 2008.

[86] W. M. Kistler, "Spike-timing dependent synaptic plasticity: a phenomenological framework," *Biological cybernetics*, vol. 87, no. 5-6, pp. 416–427, 2002.

[87] D. Wang, "A neural model of synaptic plasticity underlying short-term and long-term habituation," *Adaptive Behavior*, vol. 2, no. 2, pp. 111–129, 1993.

[88] M. Forrest, "Can the thermodynamic hodgkin-huxley model of voltage-dependent conductance extrapolate for temperature?" *Computation*, vol. 2, no. 2, pp. 47–60, 2014.

[89] K. Pakdaman, M. Thieullen, and G. Wainrib, "Fluid limit theorems for stochastic hybrid systems with application to neuron models," *Advances in Applied Probability*, vol. 42, no. 3, pp. 761–794, 2010.

[90] Q. Zheng and G.-W. Wei, "Poisson–boltzmann–nernst–planck model," *The Journal of chemical physics*, vol. 134, no. 19, p. 194101, 2011.

[91] D. Johnston and S. M.-S. Wu, *Foundations of cellular neurophysiology.* MIT press, 1994.

[92] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical journal*, vol. 1, no. 6, pp. 445–466, 1961.

[93] R. Rose and J. Hindmarsh, "The assembly of ionic currents in a thalamic neuron i. the three-dimensional model," *Proceedings of the Royal Society of London. B. Biological Sciences*, vol. 237, no. 1288, pp. 267–288, 1989.

[94] C. Morris and H. Lecar, "Voltage oscillations in the barnacle giant muscle fiber," *Biophysical journal*, vol. 35, no. 1, pp. 193–213, 1981.

[95] H. R. Wilson, "Simplified dynamics of human and mammalian neocortical neurons," *Journal of theoretical biology*, vol. 200, no. 4, pp. 375–388, 1999.

[96] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[97] N. Brunel and M. C. Van Rossum, "Lapicque's 1907 paper: from frogs to integrate-and-fire," *Biological cybernetics*, vol. 97, no. 5-6, pp. 337–339, 2007.

[98] C. Koch and I. Segev, *Methods in neuronal modeling: from ions to networks.* MIT press, 1998.

[99] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge university press, 2002.

[100] P. E. Latham, B. Richmond, P. Nelson, and S. Nirenberg, "Intrinsic dynamics in neuronal networks. i. theory," *Journal of neurophysiology*, vol. 83, no. 2, pp. 808–827, 2000.

[101] L. Badel, S. Lefort, R. Brette, C. C. Petersen, W. Gerstner, and M. J. Richardson, "Dynamic iv curves are reliable predictors of naturalistic pyramidal-neuron voltage traces," *Journal of Neurophysiology*, vol. 99, no. 2, pp. 656–666, 2008.

[102] 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.

[103] R. Jolivet, T. J. Lewis, and W. Gerstner, "Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy," *Journal of neurophysiology*, vol. 92, no. 2, pp. 959–976, 2004.

[104] R. Jolivet, A. Rauch, H.-R. Lüscher, and W. Gerstner, "Integrate-and-fire models with adaptation are good enough," in *Advances in neural information processing systems*, 2006, pp. 595–602.

[105] W. Gerstner and R. Naud, "How good are neuron models?" *Science*, vol. 326, no. 5951, pp. 379–380, 2009.

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

[107] D. F. Goodman and R. Brette, "The brian simulator," *Frontiers in neuroscience*, vol. 3, p. 26, 2009.

[108] M. L. Hines and N. T. Carnevale, "The neuron simulation environment," *Neural computation*, vol. 9, no. 6, pp. 1179–1209, 1997.

[109] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, p. 48, 2014.

[110] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in neuroinformatics*, vol. 2, p. 11, 2009.

[111] J. M. Eppler, M. Helias, E. Muller, M. Diesmann, and M.-O. Gewaltig, "Pynest: a convenient interface to the nest simulator," *Frontiers in neuroinformatics*, vol. 2, p. 12, 2009.

[112] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in neuroscience*, vol. 5, p. 73, 2011.

[113] S.-C. Liu, J. Kramer, G. Indiveri, T. Delbruck, and R. Douglas, *Analog VLSI: circuits and principles.* MIT press, 2002.

[114] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, "Silicon auditory processors as computer peripherals," in *Advances in Neural Information Processing Systems*, 1993, pp. 820–827.

[115] S. R. Deiss, R. J. Douglas, A. M. Whatley *et al.*, "A pulse-coded communications infrastructure for neuromorphic systems," *Pulsed neural networks*, pp. 157–178, 1999.

[116] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000.

[117] J. V. Arthur and K. Boahen, "Recurrently connected silicon neurons with active dendrites for one-shot learning," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, vol. 3. IEEE, 2004, pp. 1699–1704.

[118] D. P. Northmore and J. G. Elias, "Building silicon nervous systems with dendritic tree neuromorphs," *Pulsed neural networks*, pp. 135–156, 1998.

[119] C. Bartolozzi and G. Indiveri, "Synaptic dynamics in analog vlsi," *Neural computation*, vol. 19, no. 10, pp. 2581–2603, 2007.

[120] K. A. Boahen and A. G. Andreou, "A contrast sensitive silicon retina with reciprocal synapses," in *Advances in neural information processing systems*, 1992, pp. 764–772.

[121] T. Delbruck, "Silicon retina with correlation-based, velocity-tuned pixels," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 529–541, 1993.

[122] S. Wang, T. J. Koickal, G. Enemali, L. Gouveia, L. Wang, and A. Hamilton, "Design of a silicon cochlea system with biologically faithful response," in *2015 International Joint Conference on Neural Networks (IJCNN).* IEEE, 2015, pp. 1–7.

[123] A. v. Schaik, T. J. Hamilton, and S.-C. Liu, "Silicon cochleas," *Event-Based Neuromorphic Systems*, pp. 71–89, 2015.

[124] M. Yang, S.-C. Liu, and T. Delbruck, "A dynamic vision sensor with 1% temporal contrast sensitivity and in-pixel asynchronous delta modulator for event encoding," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 9, pp. 2149–2160, 2015.

[125] D. P. Moeys, F. Corradi, C. Li, S. A. Bamford, L. Longinotti, F. F. Voigt, S. Berry, G. Taverni, F. Helmchen, and T. Delbruck, "A sensitive dynamic and active pixel vision sensor for color or neural imaging applications," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 123–136, 2017.

[126] T. Delbruck, "Neuromorophic vision sensing and processing," in *2016 46th European Solid-State Device Research Conference (ESSDERC)*. IEEE, 2016, pp. 7–14.

[127] D. Heeger, "Poisson model of spike generation," *Handout, University of Standford*, vol. 5, pp. 1–13, 2000.

[128] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database," *URL http://yann. lecun. com/exdb/mnist*, 1998.

[129] W. Zieglgansberger, E. D. French, G. R. Siggins, and F. E. Bloom, "Opioid peptides may excite hippocampal pyramidal neurons by inhibiting adjacent inhibitory interneurons," *Science*, vol. 205, no. 4404, pp. 415–417, 1979.

[130] M. C. Van Rossum, G. Q. Bi, and G. G. Turrigiano, "Stable hebbian learning from spike timing-dependent plasticity," *Journal of neuroscience*, vol. 20, no. 23, pp. 8812–8821, 2000.

[131] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, "Connectivity reflects coding: a model of voltage-based stdp with homeostasis," *Nature neuroscience*, vol. 13, no. 3, p. 344, 2010.

[132] N. C. Rust, O. Schwartz, J. A. Movshon, and E. P. Simoncelli, "Spatiotemporal elements of macaque v1 receptive fields," *Neuron*, vol. 46, no. 6, pp. 945–956, 2005.

[133] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in neuroscience*, vol. 7, p. 178, 2013.

[134] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[135] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, p. 1, 2010.

[136] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[137] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," *URL: http://www. drampower. info*, vol. 22, 2012.

[138] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2011, pp. 694–701.

[139] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2009.

[140] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[141] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[142] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[143] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2016.

[144] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach.* Elsevier, 2011.

[145] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, "Multi-core cache hierarchies," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 3, pp. 1–153, 2011.

[146] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," in *32nd International Symposium on Computer Architecture (ISCA'05).* IEEE, 2005, pp. 408–419.

[147] M. D. Hill and A. J. Smith, "Evaluating associativity in cpu caches," *IEEE Transactions on Computers*, vol. 38, no. 12, pp. 1612–1630, 1989.

[148] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design MIPS Edition: The Hardware/Software Interface.* Newnes, 2013.

[149] N. J. Richardson and C. A. Stack, "Pipelined non-blocking level two cache system with inherent transaction collision-avoidance," Feb. 2003, uS Patent 6,519,682.

[150] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No. 99TH8477).* IEEE, 1999, pp. 273–275.

[151] T.-F. Chen and J.-L. Baer, "Effective hardware-based data prefetching for high-performance processors," *IEEE transactions on computers*, vol. 44, no. 5, pp. 609–623, 1995.

[152] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture.* IEEE, 1999, pp. 248–259.

[153] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *ACM SIGARCH Computer Architecture News*, vol. 30, no. 2. IEEE Computer Society, 2002, pp. 148–157.

[154] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Acm Sigplan Notices*, vol. 37, no. 10. ACM, 2002, pp. 211–222.

[155] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Compiler-directed instruction cache leakage optimization," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture.* IEEE Computer Society Press, 2002, pp. 208–218.

[156] R. Nishtala, R. W. Vuduc, J. W. Demmel, and K. A. Yelick, "When cache blocking of sparse matrix vector multiply works and why," *Applicable Algebra in Engineering, Communication and Computing*, vol. 18, no. 3, pp. 297–311, 2007.

[157] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *ACM SIGARCH Computer Architecture News*, vol. 18, no. 2SI. ACM, 1990, pp. 364–373.

[158] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni, "A survey on cache management mechanisms for real-time embedded systems," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, p. 32, 2015.

[159] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture.* IEEE, 2012, pp. 389–400.

[160] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[161] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 381–391, 2007.

[162] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture.* IEEE Computer Society, 2010, pp. 175–186.

[163] S. Jiang and X. Zhang, "Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 31–42, 2002.

[164] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA).* IEEE, 2017, pp. 27–40.

[165] H. Jiao, Y. Qiu, and V. Kursun, "Variations-tolerant 9t sram circuit with robust and low leakage sleep mode," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS).* IEEE, 2016, pp. 39–42.

[166] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: end-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, pp. 1–15, 2018.

[167] M. W. Moskewicz, F. N. Iandola, and K. Keutzer, “Boda-rtc: Productive generation of portable, efficient code for convolutional neural networks on mobile computing platforms,” in *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2016, pp. 1–10.

[168] F. Bellosa, “The benefits of event: driven energy accounting in power-sensitive systems,” in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*. ACM, 2000, pp. 37–42.

[169] J. Bauer, M. Bershteyn, I. Kaplan, and P. Vyedin, “A reconfigurable logic machine for fast event-driven simulation,” in *Proceedings 1998 Design and Automation Conference. 35th DAC.(Cat. No. 98CH36175)*. IEEE, 1998, pp. 668–671.

[170] S. G. Wysoski, L. Benuskova, and N. Kasabov, “Evolving spiking neural networks for audio-visual information processing,” *Neural Networks*, vol. 23, no. 7, pp. 819–835, 2010.

[171] N. Kasabov, “Evolving spiking neural networks and neurogenetic systems for spatio-and spectro-temporal data modelling and pattern recognition,” in *IEEE World Congress on Computational Intelligence*. Springer, 2012, pp. 234–260.

[172] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri, “Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition,” *Neural Networks*, vol. 41, pp. 188–201, 2013.

[173] S. Soltic and N. Kasabov, “Knowledge extraction from evolving spiking neural networks with rank order population coding,” *International Journal of Neural Systems*, vol. 20, no. 6, pp. 437–445, 2010.

[174] S. Schliebs, M. Defoin-Platel, S. Worner, and N. Kasabov, “Integrated feature and parameter optimization for an evolving spiking neural network: Exploring heterogeneous probabilistic models,” *Neural Networks*, vol. 22, no. 5-6, pp. 623–632, 2009.

[175] D. Chen, C. Giles, G. Sun, H. Chen, Y. Lee, and M. Goudreau, “Constructive learning of recurrent neural networks,” in *IEEE International Conference on Neural Networks*. IEEE, 1993, pp. 1196–1201.

[176] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *IEEE transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[177] J. urgen Branke, “Evolutionary algorithms for neural network design and training,” in *Proceedings of the 1st Nordic Workshop on Genetic Algorithms and its Applictions*. Citeseer, 1995.

[178] F. Gruau, D. Whitley, and L. Pyeatt, “A comparison between cellular encoding and direct encoding for genetic neural networks,” in *Proceedings of the 1st annual conference on genetic programming*. MIT Press, 1996, pp. 81–89.

[179] X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[180] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[181] T. Desell, "Large scale evolution of convolutional neural networks using volunteer computing," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 127–128.

[182] Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks for image classification," *arXiv preprint arXiv:1710.10741*, 2017.

[183] S. S. Tirumala, S. Ali, and C. P. Ramesh, "Evolving deep neural networks: A new prospect," in *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2016, pp. 69–74.

[184] J. Wu, J. Long, and M. Liu, "Evolving rbf neural networks for rainfall prediction using hybrid particle swarm optimization and genetic algorithm," *Neurocomputing*, vol. 148, pp. 136–142, 2015.

[185] F. Ruehle, "Evolving neural networks with genetic algorithms to study the string landscape," *Journal of High Energy Physics*, vol. 2017, no. 8, p. 38, 2017.

[186] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[187] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, p. 80, 2008.

[188] J. M. Tour and T. He, "Electronics: The fourth element," *Nature*, vol. 453, no. 7191, p. 42, 2008.

[189] T. Chang, Y. Yang, and W. Lu, "Building neuromorphic circuits with memristive devices," *IEEE Circuits and Systems Magazine*, vol. 13, no. 2, pp. 56–73, 2013.

[190] C. Sung, H. Hwang, and I. K. Yoo, "Perspective: A review on memristive hardware for neuromorphic computation," *Journal of Applied Physics*, vol. 124, no. 15, p. 151903, 2018.

[191] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "Stdp and stdp variations with memristors for spiking neuromorphic learning systems," *Frontiers in neuroscience*, vol. 7, p. 2, 2013.

[192] B. Linares-Barranco, T. Serrano-Gotarredona, L. A. Camuñas-Mesa, J. A. Perez-Carrasco, C. Zamarreño-Ramos, and T. Masquelier, "On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex," *Frontiers in neuroscience*, vol. 5, p. 26, 2011.

[193] J. A. Pérez-Carrasco, C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "On neuromorphic spiking architectures for asynchronous stdp memristive systems," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 1659–1662.

[194] D. Querlioz, O. Bichler, and C. Gamrat, “Simulation of a memristor-based spiking neural network immune to device variations,” in *The 2011 International Joint Conference on Neural Networks*. IEEE, 2011, pp. 1775–1781.

[195] 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, 2012.

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

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

[198] Y. Pan, P. Ouyang, Y. Zhao, W. Kang, S. Yin, Y. Zhang, W. Zhao, and S. Wei, “A multi-level cell stt-mram-based computing in-memory accelerator for binary convolutional neural network,” *IEEE Transactions on Magnetics*, no. 99, pp. 1–5, 2018.

[199] H. Mulaosmanovic, J. Ocker, S. Müller, M. Noack, J. Müller, P. Polakowski, T. Mikolajick, and S. Slesazeck, “Novel ferroelectric fet based synapse for neuromorphic systems,” in *2017 Symposium on VLSI Technology*. IEEE, 2017, pp. T176–T177.

[200] M. Jerry, P.-Y. Chen, J. Zhang, P. Sharma, K. Ni, S. Yu, and S. Datta, “Ferroelectric fet analog synapse for acceleration of deep neural network training,” in *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2017, pp. 6–2.