Design Methods for Large Scale Photonic Spiking Neural Networks

Milad Eslaminia

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

December 2022

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Milad Eslaminia**

Entitled: **Design Methods for Large Scale Photonic Spiking Neural Networks**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________ Chair
Dr. Glenn Cowan

______________________ External Examiner
Dr. Arash Mohammadi

______________________ Examiner
Dr. Glenn Cowan

______________________ Supervisor
Dr. Sébastien Le Beux

Approved by ______________________
Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

December 6th 2022

____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Design Methods for Large Scale
Photonic Spiking Neural Networks

Milad Eslaminia

Silicon Photonics is a promising technology to develop neuromorphic hardware accelerators. Most optical neural networks rely on wavelength division multiplexing (WDM), which calls for power-hungry calibration to compensate for the non-uniformity fabrication process and thermal variations of microring resonators (MRR). This imposes practical limitations on neuromorphic photonic hardware since only a small number of synaptic connections per neuron can be implemented. As a result, the mapping of neural networks (NN) on a hardware platform requires the pruning of synaptic connections, which drastically affects the accuracy.

In this work, we address these limitations from two directions. First, we proposed a method to map pre-trained NN on an all-optical spiking neural network (SNN). The technique relies on weight partitioning and unrolling to reduce synaptic connections. This method aims to improve hardware utilization while minimizing accuracy loss. The resulting neural networks are mapped on an architecture we propose, allowing us to estimate accuracy and energy consumption. Results show the capability of weight partitioning to implement a realistic NN while attaining a 58% reduction in energy consumption compared with unrolling. Second, a synaptic weighting architecture is proposed to implement weighting while reducing the number of required MRRs by half thus simplifying the calibration requirements. The architecture was simulated to demonstrate its capability of performing synaptic weighting. These methods together introduce design directions that can work around constraints of photonic spiking neural network architectures and help reach toward realizing large-scale photonic spiking neural networks.

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Sebastien Le Beux, for his invaluable patience and feedback throughout my studies and research. I am grateful to my parents. Their belief in me has kept my motivation high during the hardships in the past few years. I would like to thank my brothers who supported me and made everything possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Neural Networks (ANN) are computing systems made of simple but highly interconnected processing elements called neurons [1]. These elements, loosely based on models of biological neurons, process information by their dynamic response based on external inputs. In contrast to conventional von Neuman machines, which are suitable for processing given instructions based on centralized processing and memory, ANNs are highly parallel [2, 3]. ANNs are deployed in an ever-growing number of applications [2], which led to challenges related to design complexity, computation power, and energy efficiency. This has brought the research community to focus on neuromorphic engineering [4]. The goal of Neuromorphic engineering is to create hardware that closely replicates the functionality of a biological neural system and attempt to match Machine Learning algorithms to hardware that is similarly massively distributed and parallel [5, 6].

Biological systems like the brain perform complex tasks at very low energy costs. Biological neurons communicate asynchronously through action-potentials and perform spike-based processing [7]. These action potentials are generated based on the dynamic system modeling of the neuron. Spiking Neural Networks (SNN) is a class of neural networks where each processing node or neuron communicates through a sequence of spikes, hence allowing closer implementation of biological systems. SNNs promise advantages such as low power consumption, fast inference, event-based processing, and analog computation. SNNs are applicable to all the same problems solvable by non-spiking neural networks [8] such as signal processing, event detection, classification, and object recognition [9, 10].

Electronic neuromorphic hardware typically relies on a shared time-division multiplexed digital bus, for which the design involves a trade-off between available bandwidth and the interconnects size [11]. To overcome the high latency and low bandwidth limitations of electronic interconnects, silicon photonics has been proposed as an interconnect backbone to accelerate neural networks [12]. Matrix multiplication is at the core of Neural Network models. Neurons are interconnected through synaptic weights. The input to post-synaptic neurons is the result of the summation and attenuation of the output of each presynaptic neuron by synaptic weights. Photonic implementations of linear operations such as matrix multiplication [13] can be performed in parallel and at lower energy cost [14]. Considering the success of photonic and optical interconnects in delivering high data rates in sectors such as data centers and telecommunication sectors, the technology is likely to become mainstream for the implementation of neural networks dedicated to applications with high bandwidth requirements such as wide-band radio frequency information processing [15], and ultra-fast inference [16]. Applications beyond the reach of digital electronics could then be the target for photonic neural networks and they could complement conventional electronic devices.

Photonic components such as excitable lasers have been demonstrated to behave similarly to Leaky-Integrate and Fire Model (LIF) [17]. The LIF model implements a simplified model of the dynamic systems of a biological neuron. Microring resonators (MRR) with a phase-change material (PCM) have demonstrated similar integration and spiking behavior described by this model [18]. These features make silicon photonics a promising candidate for the implementation of SNNs.

1.1 Problem Statement

Neural Networks are highly parallel computing models. These models consist of individual processing nodes called neurons. Large numbers of neurons placed in interconnected layers enable the highly parallel computation capabilities of neural networks. To replicate this attribute on neuromorphic hardware and map neural network models for accelerated processing, neurons and corresponding interconnect need to be represented one-to-one on the hardware. In other words, to realize neuromorphic hardware, it is necessary to realize an architecture for the neurons and a feasible way to interconnect these neurons to create larger networks.

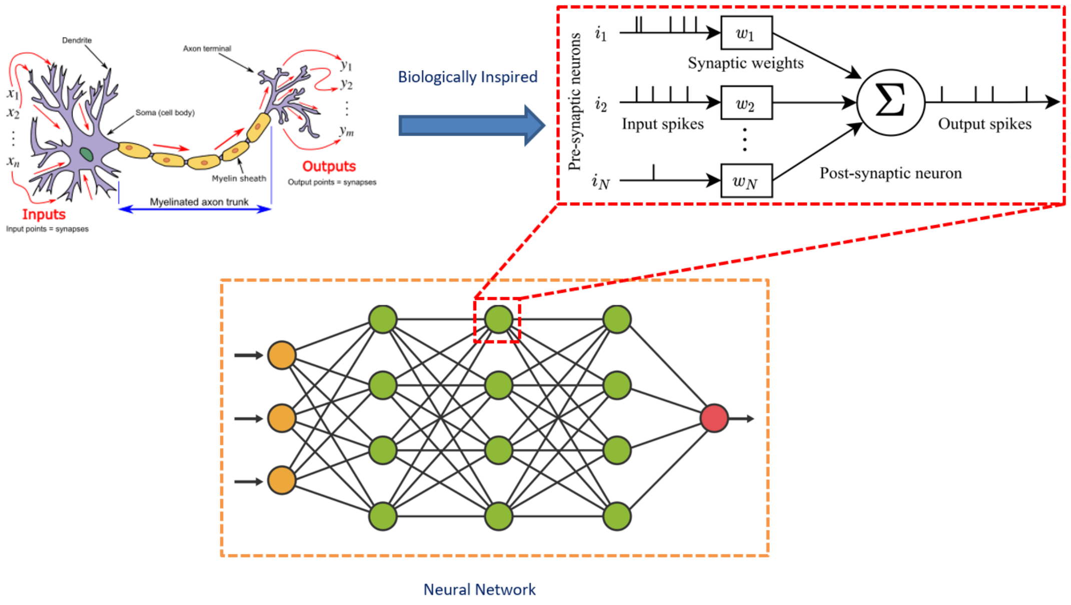


Figure 1.1: Neural networks consist of a large number of interconnected neurons. This figure is reproduced from [19] with modifications.

Photonic Spiking Neural Network architectures rely on wavelength division multiplexing (WDM) to collect and distribute signals between the neurons and the layers in a neural network. Operation of WDM relies on a careful calibration of MRRs to individual resonance wavelengths. This is necessary to reliably transmit and extract signals that are propagating on different wavelengths. Calibration of MRRs is challenging as they are easily influenced by environmental factors like temperature. Minute variations in these factors can cause a drift in the resonance wavelength of these MRRs and result in the degradation of WDM to reliably extract a signal at the destination. Neuromorphic applications with many neurons and their individual synapses require a greater number of MRRs. This imposes a practical limitation on the scale of a neural network model that current photonic architectures can accommodate. To address this limitation, we need to find potential answers to these questions:

- How a neural network model is affected by the limited synaptic connectivity on the target hardware?
- What are the potential methods to work around or mitigate these shortcomings?

- Do these methods require a new hardware design or can they be implemented through software preprocessing?
- How to evaluate and compare the impact of these limitations and the benefits of the mitigation methods?

These questions motivate us to explore design methods to work around these limitations and propose viable solutions to realize larger neural network models with the limitations of current photonic architectures.

1.2 Thesis Objectives

We can address this problem in two directions. First, we can consider the neural network model to be mapped onto the hardware. By modifying and reshaping the model, we potentially can work around the limitation of the target neuromorphic hardware and create models that do not exceed the number of available hardware resources. Second, we can consider the hardware itself and propose new designs that can accommodate a larger number of neurons and synaptic connections with fewer MRRs. It is to be expected that taking either of these directions will come at a penalty to model accuracy and energy consumption. It will be necessary to evaluate the proposed solutions and explore design space to arrive at trade-offs between these factors and this could lead us toward a better understanding of design challenges and goals when scaling photonic spiking neural network architectures to large networks. The objectives of this thesis are the following:

- The main objective of this thesis is to explore design methods that scale all-optical neural network architectures to accommodate large neural network models.
- The design methods should consider the physical limitations and characteristics of optical hardware.
- Provide evaluation and comparisons between the proposed and existing approaches to explore design space.

1.3 Thesis Contribution

The main contributions of this thesis are:

- A design flow for partitioning and mapping neural network models to a photonic SNN hardware.
- Distribution of the neurons in a neural network model over identical and interconnected neuron clusters.
- An architecture for all-optical neuron clusters that realize the distribution of the neural network over these clusters.
- Evaluation and comparison of partitioning and mapping methods in terms of accuracy and estimated energy consumption to explore the design space.
- A proposed photonic synaptic weighting architecture that can perform synaptic weighting with a reduced number of MRRs.
- Validating the proposed synaptic weighting architecture using optical simulation in OptSim.
- A design flow for transforming and mapping synaptic weight from a source pre-trained neural network model to the proposed architecture.
- A quantization method for synaptic weights that more closely models how accurately weights are represented on photonic synapses.

The proposed design flow for weight partitioning and mapping has been accepted by 30th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-Soc) 2022 and presented at the conference in October 2022. The title of the paper is:

- Milad Eslaminia and Sébastien Le Beux, ”Toward Large Scale All-Optical Spiking Neural Networks”.

1.4 Thesis Organization

The thesis is organized into six chapters. Chapter 1 introduces the topic and discusses the problems and contributions of this thesis. Chapter 2 discusses the related works and other background information related to this thesis. Chapter 3 proposes partitioning and mapping methods for scaling an existing photonic SNN. Chapter 4 proposes a synaptic weighting architecture that can perform synaptic weighting with a reduced number of MRRs. Chapter 5 presents simulation results of the methods proposed in chapters 3 and 4. Finally, chapter 6 outlines the conclusion of the work and future directions that could be pursued.

Chapter 2

Related Works

2.1 Spiking Neural Networks

Deep ANN and convolutional neural networks (CNN) have demonstrated great success in applications such as classification, clustering, pattern recognition, and prediction in various fields. On certain tasks such as image classification [20], these models are approaching human performance and could explain how the human visual system performs similar tasks. Given these similarities, there has been a great interest to create models that more closely resemble biological systems [21, 22, 23, 24, 25].

While ANNs may rely on transmitting floating-point data between each compute unit in the network, spiking neural networks (SNN) communicate through sequences of spikes over time, and they are functionally like the biological neurons in the brain [9]. Conventional ANNs learn through standard training methods based on the backpropagation of error signals [26]. Based on studies, learning in the brain is closer to unsupervised learning methods such as Spike-timing-dependent plasticity (STDP) [27]. SNNs offer solutions to solve certain problems, such as event detection, classification, and signal processing [9, 28]. SNNs can be applied to all the same problems that other non-spiking artificial neural networks can but with significant advantages in computational power and energy efficiency [29, 11, 30].

In this section, fundamental concepts related to the theory of SNNs are introduced. First, we introduce the readers to a representation of information using encoded spike sequences. Then neuron structure, its neural functions, and the dynamic models describing a spiking neuron are introduced. Finally, we introduce the learning process in SNN, and we illustrate how spiking neurons can be used to perform simple tasks.

2.1.1 Spiking Signals

In SNN, data representation is fundamentally different from other artificial neural networks. In artificial neural networks, input data are represented as vectors of floating-point numbers and are processed using matrix multipliers. In contrast, SNN takes sequences of pulses over periods of time as its input [31]. The spike train effectively forms a binary sequence and the information is encoded in the frequency of the spikes or the time gaps between them over a specific period [28]. Fig. 2.1 illustrates two main strategies for spike coding through rate coding and temporal coding.

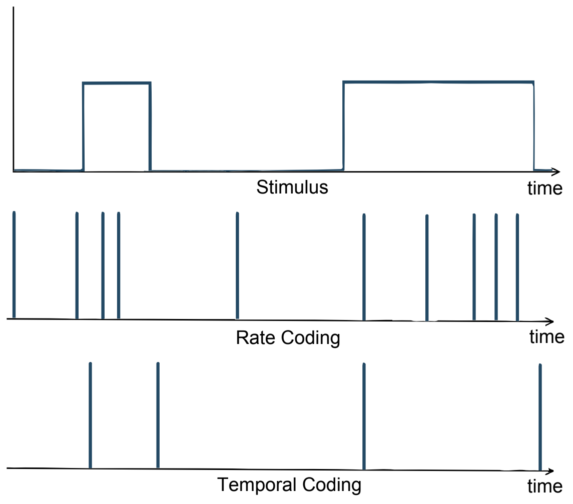


Figure 2.1: Main strategies to code stimulus data into spike trains. With rate coding, the frequency of the spikes correlates with the occurrences of the action potentials. With temporal Coding, the information is encoded in the timing of the spikes

Input data for SNNs are encoded in the form of Poisson-distributed spike trains. In this encoding method, spike firing rates are proportional to the intensity of the input signal [32].

2.1.2 Spiking Neuron Model

Neurons are the elementary processing units in a nervous system and a neural network. A typical neuron can be divided into three parts called dendrites, soma, and axon. The dendrites are analogous to the inputs of a component that gather signals from other neurons [33]. These inputs are transmitted to soma which performs the non-linear processing step of the neurons. If the total inputs that arrived at soma exceed a threshold, an output is generated by the axon and transmitted to other neurons. The connections between neurons are called synapses. Fig. 2.2 demonstrates this process.

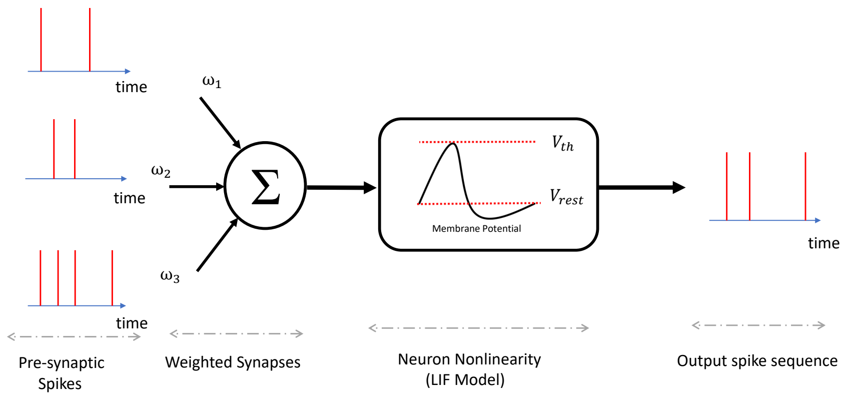


Figure 2.2: The functional structure of a biological neuron and its operation based on Leaky Integrate-and-Fire model.

These neuronal signals are short electrical pulses that are called action potentials or spikes. A sequence of these spikes forms a spike train that contains information on the density of spikes and their timings. These action potentials or spikes are separated over time, and it is not possible for the neuron to generate another spike in a period following the first spike. This period of inactivity where it is difficult or impossible for the neuron to generate a spike is called the refractory period.

These electric spikes arrive at the postsynaptic neuron and can cause changes in the potential difference between the interior and surroundings of the cell. This quantity is called the membrane potential. When the neuron is not receiving any spikes, it stays at the resting state denoted by resting potential V_{rest}. After a spike arrives at the neuron, the potential changes from this resting state, and after a period it decays back to the resting state. The synapses carrying these spikes that increase or decrease the membrane potential are called excitatory or inhibitory synapses, respectively. If the membrane potential exceeds a specific threshold due to incoming received spikes, the neuron generates a spike output and then the membrane potential returns to the resting state. This behavior of membrane potential can be seen in Fig. 2.3.

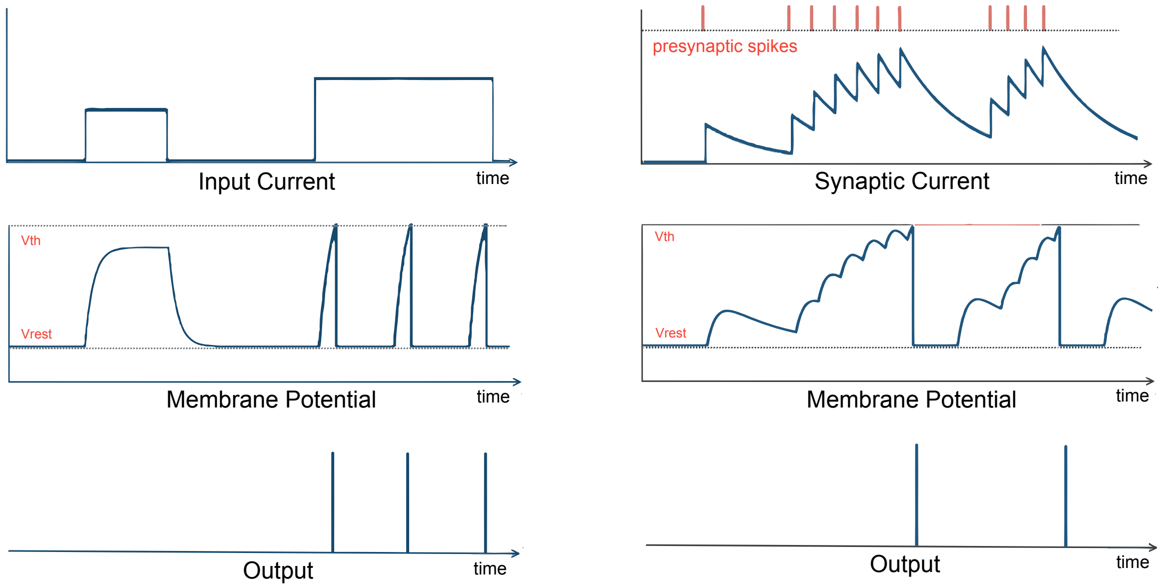


Figure 2.3: Behavior of a LIF neuron with, Left: An input current source and, Right: Presynaptic spikes and synaptic current. Membrane potential ($V(t)$) rises from resting potential (V_{rest}). In absence of synaptic current, membrane potential decays back to resting potential. If the input current is strong enough so that membrane potential reaches the threshold potential (V_{th}), the neuron generates an output spike and returns to a resting state. In this period, called the refractory period, the neuron cannot output additional spikes. After the refractory period passes, the neuron can resume accumulating potential and output additional spikes.

In these experiments, the neuronal dynamics can be described as a summation or integration process over the action potentials followed by an activation mechanism triggered after the potential reaches a specific voltage. To describe this behavior, various mathematical models are proposed, such as Hodgkin-Huxley [34]. This model is a set of nonlinear differential equations that approximate the dynamic systems of a neuron and its membrane potential. The leaky Integrate-and-Fire (LIF) model [35] is a simplified model of this behavior and is described following equation (1).

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -\left[V(t) - V_{\text{rest}}\right] + RI(t) \tag{1}$$

With the condition that if $V(t) > V_{th}(t)$ then $V(t) = V_{rest}$, where $V(t)$ is the membrane potential at time t, $V_{th}(t)$ is the threshold potential, V_{rest} is the resting potential, $I(t)$ is the input current and τ_m is the time constant that models the leaky integration, analogous to an RC circuit in Fig. 2.4. In this model, the neuron is represented as a parallel combination of a resistor and a capacitor alongside a current source that acts as synaptic input to charge the capacitor and produce a potential.

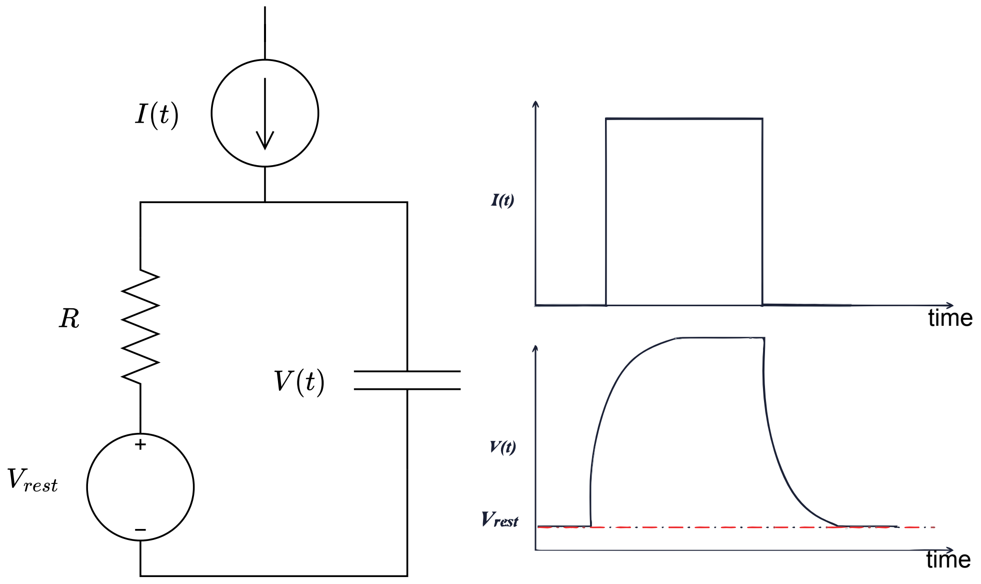


Figure 2.4: Electrical model of the leaky integrate and fire model. In response to a step input current, the membrane potential responds with a smooth trace

The LIF model is an effective and relatively simple model [36, 34] to describe several biologically observed phenomena as well as representing an accurate computational model for existing neural network algorithms [20].

2.1.3 Spike-Timing Dependent Plasticity (STDP)

STDP is a phenomenon experimentally discovered in biological neurons by Bi and Poo [27] and then adapted for learning in event-based networks. STDP is an unsupervised learning algorithm based on the time relation between post and presynaptic spikes. Any given synapse in the network possesses a weight with which it connects the output of presynaptic neurons to postsynaptic ones. If a spike has occurred in the post-synaptic neuron at a specific time window after a presynaptic spike, it could be considered positive or excitatory stimulation. The synaptic weight under STDP learning increases to signify this event. Similarly, the weight decreases if the post-synaptic spike has occurred before a pre-synaptic spike is detected and this denotes an inhibitory stimulation. In this way, STDP is a specific form of the more general Hebbian learning rule. The extent the weight is adjusted in these events is a function of the time between post and presynaptic spikes and follows the STDP curve for learning shown in Fig. 2.5. There are many theoretical models of STDP with formulations that differ in weight dependence and weight changes [37].

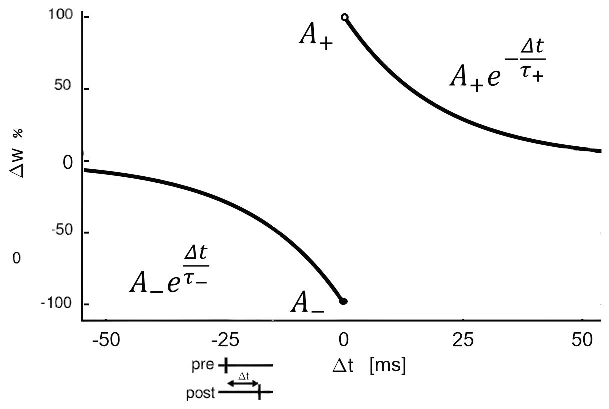


Figure 2.5: STDP curve. The change in synaptic weight ($\Delta\omega$) as a function of the pre- and post-spike timing (Δt).

This figure is reproduced from [37]

2.2 Silicon Photonics

Silicon photonics aims to use existing CMOS fabrication processes to design optical hardware integrated into a chip. Photonic integrated circuits (PIC) being compatible with the CMOS industry, it is a promising route towards cost-effective optical accelerators [38]. Furthermore, the high refractive index of silicon allows for close spacings and sharp bends in optical interconnect, hence leading to compact optical circuits. In this section, primary photonic components that are used for the implementation of neural networks are introduced.

2.2.1 Waveguides

Waveguides can be considered as wires for light in a photonic circuit and enable low-loss light transmission between optical components. The high refractive index difference between the core and surrounding medium can confine the optical waves in the waveguide. Modes are the solutions to the field equations in a given waveguide geometry. Modes do not interact with each other, and each propagates with its own wave number depending on the propagation frequency and geometry of the medium. In this way, each mode is subject to an effective refractive index when propagating in a waveguide.

2.2.2 Microring Resonators (MRR)

A typical ring resonator is composed of a looped optical waveguide in a way that resonance can occur when the optical path is multiple times the carrying wavelengths. In this way, a ring resonator is capable of multiple resonance wavelengths. The distance between resonance wavelengths depends on the length of the resonator: the smaller the ring radius, the larger the distance between the resonances. The high refractive index of silicon allows for micrometer-scale radius rings. Ring resonators are one of the most important components in silicon photonics as they are already used to implement filters, switches, modulators, and wavelength division multiplexing (WDM).

A ring resonator usually consists of a looped waveguide and a straight waveguide transmitting an optical signal. Common MRR configurations are represented in Fig. 2.6. In the all-pass configuration, the input signal is transmitted to the output with very low attenuation when the ring is off resonance; in case resonance occurs, the input wave is coupled into the ring cavity, hence leading to

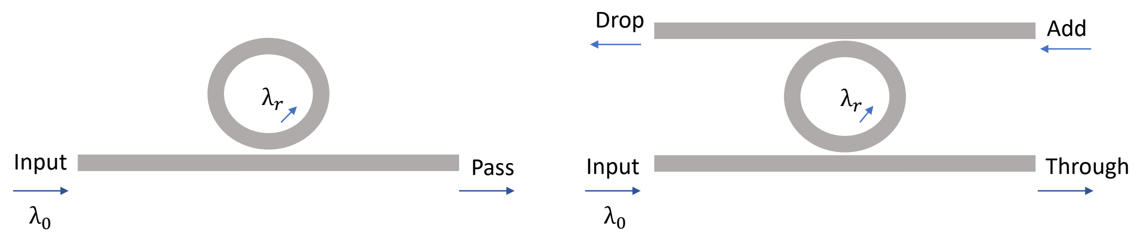


Figure 2.6: The primary configuration of ring resonators. Left: All-pass. Right: Add-Drop configurations.

2.2.3 Wavelength Division Multiplexing (WDM)

It is highly inefficient to use a separate physical communication channel for each connection path in the network. Multiplexing allows the transmission of multiple signals into one channel, thus increasing the aggregated network bandwidth. The common method of multiplexing in optics and photonics is wavelength division multiplexing (WDM). WDM multiplexes signals into distinct carrier wavelengths. At the receiver's end, by filtering for certain wavelengths using for instance ring resonators, the originals signals can be recovered. This way all available bandwidth can be used and by reconfiguring the multiplexing system, a different number of transmission links can be implemented using the same physical hardware.

2.2.4 Phase-Change Materials (PCM)

Phase-Change Materials (PCM) can change their state between amorphous and crystalline states by absorbing sufficient energy. The change in phase is accompanied by a change in the refractive index. This is suitable for controlling the attenuation of crossing optical signals or their routing. These properties give photonic integrated circuits (PIC) non-volatile reconfiguration capability, thus allowing PCM-based photonic hardware to be used for different applications.

2.2.5 Lasers

Lasers are light sources for a photonic circuit. Laser light is preferred to thermal light sources due to its narrow spectral line width, which makes it suitable for precise optical computation and transmission. This narrow band is achieved with high optical output power, typically in the range of mW for on-chip interconnects. In comparison, a filtered thermal light source also has a narrow spectral width but wastes a significant portion of optical power. Among numerous already demonstrated integrated lasers, Vertical-cavity surface-emitting laser (VCSEL) is one of the most promising technology. A VCSEL is a semiconductor-based laser that emits light from its upper surface. This allows several diodes to be implemented on the same wafer, allowing scaling to large 2-dimensional arrays.

2.3 Optical Neural Networks

Neurons accumulate incoming pre-synaptic inputs that over time could result in firing spikes or action potentials. This process can be divided into three neural functions: weighting function, integration function, and activation function. This section is divided into subsections that each discuss different neural functions along with existing implementations of these functions with photonic hardware.

2.3.1 Weighting Function

The fundamental function of a neural network is to learn from the observed sample input data during the learning process and access this information for future input data. To this end, the learned information needs to be stored in a memory in a manner that can be updated as well. This function is achieved using the weights of synapses. In neural networks, updating of these weights enables the learning process, which aims to achieve better performances through neural plasticity.

In order to implement this functionality, it is necessary to control the signal transmissions in each synapse and perform excitatory and inhibitory functions. The work in [39, 40] proposes an implementation of this weighting function using a bank of microring resonators (MRR) to dynamically control the transmission over synapses between neurons, as illustrated in Fig. 2.7. The MRR weight bank is a controllable spectral filter. The MRRs are parallel-coupled and can independently control the transmission of a single WDM input. The output corresponds to the weighted sum of the inputs. By relying on the resonance qualities of the MRR bank, this approach is highly sensitive to temperature and fabrication variations, which are difficult to control and require calibration.

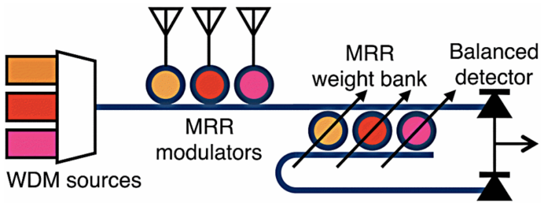


Figure 2.7: A microring weight bank. Input signals are modulated onto WDM carriers and multiplexed. Each input can be weighted by one tunable MRR corresponding to its wavelength. The figure is reproduced from [40].

MRR weight control can be achieved through feed-forward [41] or feedback [39] based approaches. The calibration phase in the feed-forward approach requires a suite of external measurement equipment such as optical spectrum analyzers, oscilloscopes, and pattern generators. Although the same calibration mechanism could be fully integrated on the chip, this approach is vulnerable to fluctuations in temperature after calibration as well and they are thus only feasible for small-scale models. The feedback approach in [39], does not require an external optical probing setup and as a result, has a simpler calibration and control phase and it is not vulnerable to environmental fluctuations. The feedback capabilities are achieved by embedding an in-ring photo-conductive heater within the waveguide of each MRR weight.

The work in [42] studies the factors limiting the performance and channel count in the MRR weight bank by introducing a quantitative parameter called the independent weighting factor. It establishes a trade-off between channel count and required signal power and quantifies the degree to which weights can be set independently. The authors developed a parametric transmission model for MRR weighting banks with generalized matrix transfer theory and simulated their model with symbolic programming methods in MATLAB to create a parametric simulator for waveguide circuits. They developed a parametric transmission model of a weighting bank with 8 MRRs and fit the model to measured experimental data.

Phase-Change Materials such as $Ge_2Sb_2Te_5$ (GST) can change their state from amorphous to crystalline by receiving or releasing specific amounts of energy. This change in phase results in a change in the refractive index of the material. The work in [43] realized a photonic synapse by integrating islands of PCMs onto waveguides. It is demonstrated that a specific number of optical pulses can change the state of the integrated PCM material in waveguides to create a photonic memory of up to eight levels [44]. The schematic of this synapse can be seen according to Fig. 2.8. An optical circulator connects the output of the synapse to the post-synaptic neuron from ports 2 to 3. Port 1 is used to apply input pulses to change the synaptic weight.

2.3.2 Integration Function

A neuron must be able to integrate the incoming presynaptic spikes over time and forward the value to the activation function. This summation determines the value of membrane potential over

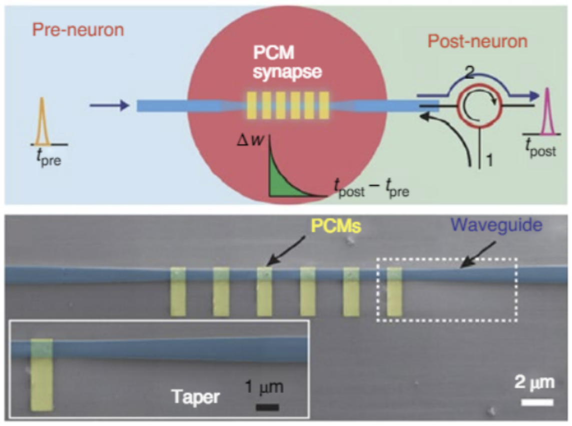


Figure 2.8: Schematic of a PCM synapse. Synaptic weight changes by the number of pulses. This figure is reproduced from [43].

time according to the operating model of the neuron such as the Leaky Integrate and Fire model. It is important that the integration method is scalable to a large number of incoming synaptic inputs as it defines the scalability of the model to the whole network. The work in [14] proposes a simple scheme for parallel photonic neural interconnects called broadcast and weight that can be combined with WDM to create scalable networks. Broadcast- and-weight weighting is accomplished by using spectral filter banks on WDM multiplexed channels. The total power of each weighted signal is detected using photodetectors and used as the input for the excitable laser. (Fig. 2.9). The photodetectors output an electric current that is proportional to the total detected power of the weighted WDM inputs and represents their sum. This electronic signal can modulate a laser and demonstrate inhibitory behavior.

The work in [18] proposes an integration unit implemented using a GST embedded on a ring resonator. This integration unit consists of two ring resonators creating a bipolar neuron capable of both excitatory and inhibitory behavior and can receive both positive and negative weighted inputs (2.10). The energy efficiency and speed of this method are dictated by the properties of the GST material during read and write cycles. Pulses of amplitudes proportional to the positive and negative weighted sums are fed to respective ring resonators. The two ports integrate positive and negative weighted sums and the output is considered as the membrane potential. Modulating the resonant wavelength offers the possibility of scaling to larger networks through wavelength multiplexing.

The summation can be achieved using WDM multiplexers. The work in [45] utilizes this scheme

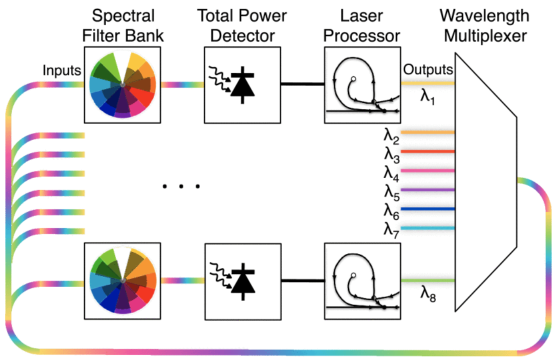


Figure 2.9: Schematic of the electro-optical implementation of integration. Power detectors generate an electrical signal that is proportional to the total power of the inputs from weighted synapses and corresponds to the sum of the input.

This figure is reproduced from [14].

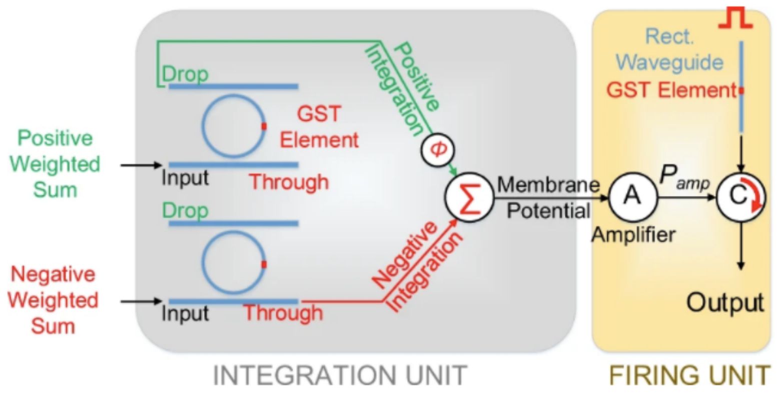


Figure 2.10: Schematic of the bipolar integration unit using properties of GST embedded on a ring resonator.

This figure is reproduced from [18].

for building an all-optical spiking neural network. The schematic of the proposed all-optical neuron is shown in Fig. 2.11. Weighted presynaptic inputs arrive on different wavelengths. An on-chip multiplexer combines these pulses onto one waveguide that is connected to the soma of the neuron. This multiplexer is realized using ring resonators in an add-drop configuration. The resonance of each of these rings must not overlap to prevent a coupled wavelength in the main waveguide to couple out in other rings. For this purpose, the radii of the rings are designed to be slightly different. The difference in the radii leads to different resonance states for each of the weighted inputs. The transmission ratio from the input port to the main waveguide can be tuned by adjusting the gaps between the input waveguides, the ring, and the main drop waveguide.

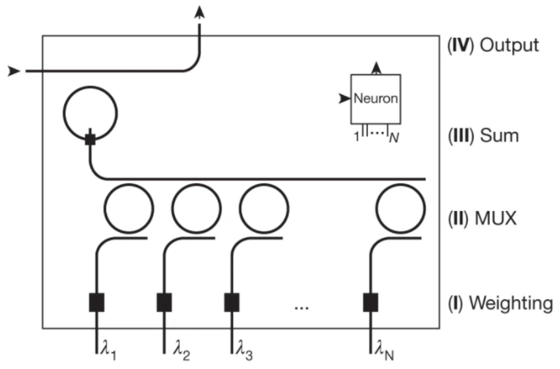


Figure 2.11: Schematic of the integration function implemented using WDM multiplexers in an all-optical PCM-based network implementation.
This figure is reproduced from [45]

2.3.3 Activation Function

In a spiking neural network, presynaptic spikes accumulate in a post-synaptic neuron and increase the membrane potential of the neuron until it reaches a specific threshold. At this point, the neuron fires a spike, and the membrane potential is reset. The activation function of the neuron determines when a neuron will fire a spike and it decides the signal propagation time in the network. The work in [45] utilizes the properties of PCM embedded on a ring resonator as an activation function. The PCM cell is deposited on top of the waveguide crossing to tune the resonance condition. This combination simulates the behavior of the rectified linear unit activation function (ReLU) according to Fig. 2.12. This all-optical neuron implementation can scale to larger networks by using wavelength multiplexing.

It has been demonstrated that an Excitable Laser can exhibit behavior like Leaky Integrate and Fire mode, which is suitable to mimic a biological neuron. An excitable system has one stable state at which it can indefinitely stay at rest and if excited over a certain threshold, the system emits a spike and decays gradually over a period of time back to the resting state. The work in [46] has demonstrated excitable behavior near the threshold by utilizing VCSELs with saturable absorbers. The system dynamics of a two-section laser consisting of a gain section and a saturable absorber can be described by Yamada model [47]. These differential equations describe the dynamic behavior of the system using the gain ($G(t)$), the absorption ($Q(t)$), and laser intensity ($I(t)$). Nahmias et al. [17] Demonstrated that the dynamics of the system near-threshold can be approximated by equation

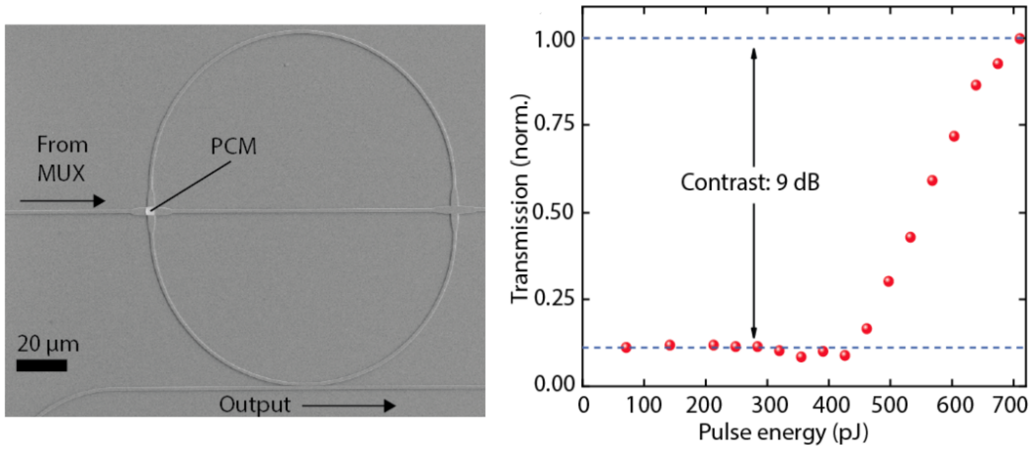


Figure 2.12: Schematic of the activation unit using PCM material embedded on a ring resonator. By exploiting the properties of the PCM, a nonlinearity for transmission ratio is achieved that resembles the ReLU function.
This figure is reproduced from [45].

(2).

$$\frac{\mathrm{d}G(t)}{\mathrm{dt}} = -\gamma_G(G(t) - A) + \theta(t) \tag{2}$$

Where $G(t)$ represents the gain, A is the gain bias, γ_G is the gain carrier relaxation rate, and $\theta(t)$ represents the spiking inputs in the form of a series of impulses at different time intervals. If $G(t) > G_{thresh}$, a spike is released and $G(t)$ is set to G_{rest}, the resting state gain. When comparing this equation with the Leaky Integrate-and-Fire neuron model in equation (1), a close analogy can be established where the gain behaves similarly to the membrane potential of the neuron. Table 2.1 summarizes the works studied in this section.

Table 2.1: Comparison of Photonic implementations of neural functions and their features. Note: ”O” denotes all-optical and ”O/E” denotes Optoelectric designs.

Neural Function	Implementation	Optical Components	WDM Compatible	Design	Reference
Weighting	Weighting banks (Spectral Filter)	MRR	Yes	O/E	[14][42]
	PCM Synapse	PCM waveguide	Yes	O	
Integration	All-Optical bipolar weighting	PCM on Ring Resonators		O	[7]
	Photodiodes (Total Power Detection)	MRR	Yes	O/E	[14]
	WDM Multiplexer				
		MRR	Yes	O	[45]
Activation	Excitable Laser	VCSEL with saturable absorbers	Yes	O/E	[17]
	PCM embedded on a ring resonator	PCM on a ring resonator	Yes	O	[45]

2.3.4 Scaling Photonic Neural Networks

The work in [45] used WDM techniques to demonstrate a scalable photonic spiking neural network with self-learning capabilities. The self-learning capabilities follow a simplified STDP where the timing between incoming and output spike pulses are fixed in this model. The model increases the synaptic weights of all inputs that contributed to a spike output and decreases the synaptic weights that did not contribute. This is achieved by using a feedback waveguide from the output of the neuron and channeling part of the energy back to the synaptic PCM weights to implement the Hebbian learning rule. Fig. 2.13 shows this proposed scaling implementation.

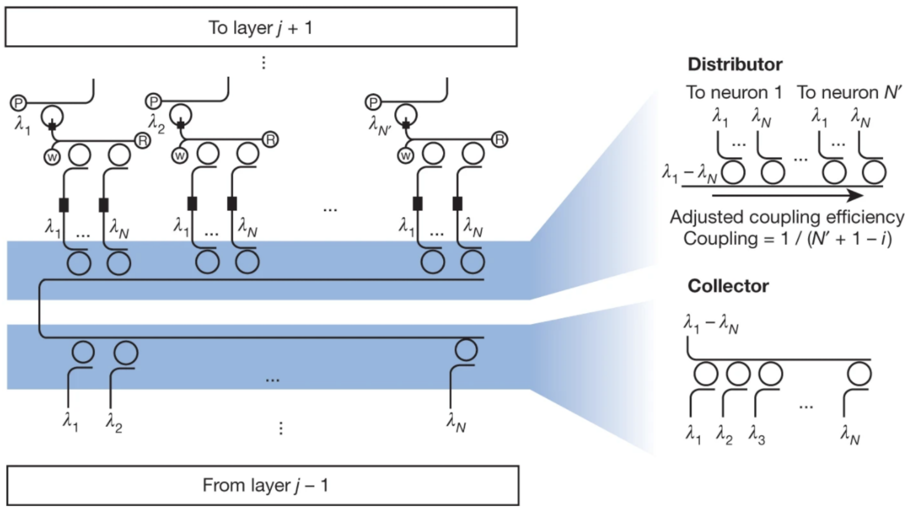


Figure 2.13: Schematic of a single layer and photonic circuit model of distributor and collector components.
This figure is reproduced from [45].

Each layer in this architecture is composed of Distributor and Collector components which are implemented using a photonic circuit shown in Fig. 2.14. The collector is responsible for gathering all the incoming outputs from the previous layer. This is achieved by multiplexing the incoming weighted inputs into a single waveguide. This multiplexer is designed using a series of parallel ring resonators. The distributor is responsible for equally distributing these collected inputs to all neurons in this layer by demultiplexing. The authors have demonstrated their proposed all-optical network by fabricating a single-layer network with 4 neurons and 15 connected synapses per neuron. The design is then applied to the experimental task of detecting 4 different patterns of 3 by 5 pixels denoting different alphabets. The schematic of this network is shown in Fig. 2.14. The model is simulated for larger networks for digit recognition tasks and a network with hidden layers applied to a simple task of detecting the language in a given text reaching accuracy exceeding 90%.

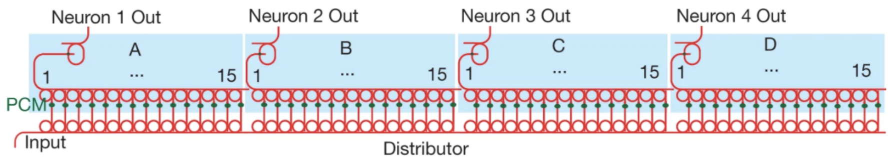


Figure 2.14: Schematic of the all-optical network.
This figure is reproduced from [45].

The work in [48] implemented STDP for unsupervised pattern learning by using VCSEL elements along with vertical-cavity semiconductor optical amplifiers (VCSOA). In the encoding part of the network, the inputs are encoded into spikes with different spike timing generated by the input VCSEL neurons. The proposed network can learn one spike pattern at a time. The learning is achieved by using an STDP array circuit constructed by VCSOA elements [49]. A three-port optical circulator is used to inject the optical pulses into the VCSOA. The reflective output of the VCSOAs is filtered to beams with different wavelengths to create the extent of weight change.

The work in [43] proposes an all-optical STDP learning scheme compatible with PCM synapses. The presynaptic pulse is split with an optical coupler (OC) so that 50% of the presynaptic signal is connected to one input (1) of an interferometer via a phase modulator (PM). The other 50% of the postsynaptic signal is connected to the other input (2) of the interferometer.

The output signal of the interferometer is used to update the synaptic weight. In Fig. 2.15, the behavior of presynaptic (black) and postsynaptic (blue) signals with no time delay $(\Delta t = 0)$ and the net output power of the interferometer as the switching signal (red) is demonstrated. As the time delay between pre- and post-synaptic pulses increases, the number of output pulses with power above the threshold increases, and a larger number of pulses are sent to the PCM synapse, demonstrating the capability to change synaptic weights according to the timing of the spikes.

In this section, various proposed models to implement STDP learning with optical and photonic elements were discussed. These models can adjust synaptic weights depending on the timing of pre- and post-synaptic spikes. The proposed models intend to replicate the STDP curve from the generated spikes by using characteristics of SOAs and other elements. Furthermore, some of the

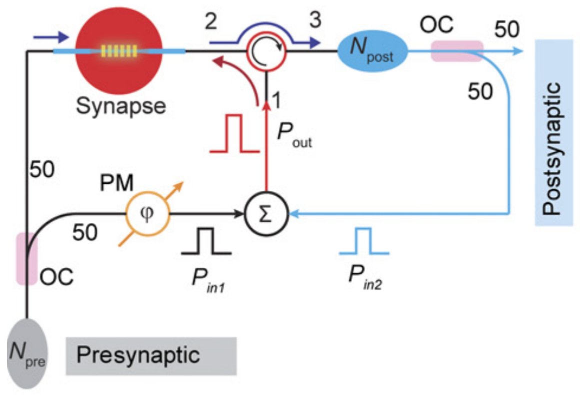


Figure 2.15: Schematic and behavior of all-optical STDP using PCM synapse. This figure is reproduced from [43].

basic test applications of these proposed networks were covered.

2.4 Design and Simulation of Neuromorphic Silicon Photonic Circuits

The design and fabrication of neuromorphic optical accelerators involve components with heterogeneous behaviors involving optics and electronics domains. Simulation is needed to validate the system behavior and to optimize circuits. The models must consider relevant physical aspects and realities of the circuit to produce useful results. At the same time, physical simulation of the system through numerical solvers is computationally complex, too slow, and hence does not scale to a large number of components neural network applications demand. To this end, simplified behavioral models are needed to replicate key characteristics and complex behavior of these components, while allowing to scale up to larger circuits. Currently, one of the main challenges is the lack of software enabling the comprehensive design and simulation of photonic neural networks. An ideal simulation platform should simultaneously capture the complex dynamics of the underlying photonic components and accommodate the design and exploration of large-scale network architectures. The work in [50] proposes a design flow for neuromorphic photonics architectures using behavioral models. It takes the hardware into account from the early stages of the design to training and initial inference tests. The design steps are illustrated in Fig. 2.16. The top four steps would result in a neural network graph with trained weights.

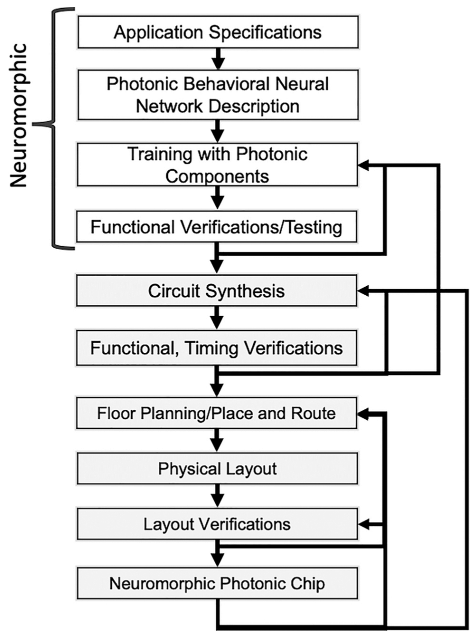

Neuromorphic
Application Specifications
Photonic Behavioral Neural Network Description
Training with Photonic Components
Functional Verifications/Testing
Circuit Synthesis
Functional, Timing Verifications
Floor Planning/Place and Route
Physical Layout
Layout Verifications
Neuromorphic Photonic Chip

Figure 2.16: Design flow for neuromorphic systems.
This figure is reproduced from [50].

In this section, first, the behavioral models and designs proposed for modeling photonic components are introduced. The design approaches using these models to design and simulate a circuit are discussed. Finally, the tools available for developing and optimizing spiking neural networks are discussed.

2.4.1 Behavioral Models for Silicon Photonic Components

Electromagnetic simulation is typically used for the design and validation of photonic components such as waveguides, resonators, couples, and others. They rely on numerical solvers such as Lumerical [51] based on finite element method (FEM), finite difference time domain methods (FDTD), or beam propagation methods. They can simulate the propagation of electromagnetic waves in given geometries and materials. Simulating larger or more complicated topologies using these methods is computationally challenging and extremely slow. While it is possible to physically simulate photonic components through the methods mentioned earlier, physical simulations do not scale to design networks involving multiple heterogeneous components, especially when they involve phase-change materials [52]. The accurate and scalable simulation of neural networks, which are typically composed of thousands or more photonic components, is thus challenging. As such, behavioral models that enable fast simulation of photonic components that are still capable of expressing accurate operating characteristics are required.

The work in [53] proposes an equivalent circuit model emulating the dynamics of a laser neuron. The laser neuron rate equations are typically solved using numerical methods for differential equations such as the Runge-Kutta method and lack any analytical solution. This abstraction alongside efficient SPICE analysis is used to demonstrate the accurate behavior of the laser neuron.

The work in [52] proposes a behavioral model for the simulation of PCM-based components. The model allows representing the heating of phase-change cells, crystallization and amorphization of PCM materials, and light propagation in the integrated devices. The simulations of the writing, erasing, and reading process from a PCM cell involve simplified heat transfer, phase-change, and electromagnetic models. The result is a model that can be computed in seconds as opposed to several hours that would be required using conventional finite-element models with comparable accuracy in transient responses of photonic components.

The work in [38] studies the design flows of photonic circuits. The layout of the circuit is built from reusable cells where certain parts of the geometry of the component can be parameterized. These so-called "PCells" are programmed in scripting languages and can be paired with other software for the simulation of photonic circuits. A key advantage of the approach is the availability of existing models for building elements in commercial photonic design kits. The work in [54], proposes the use of hardware description language to create a parametric model based on experimentally validated data.

The work in [55] has developed an open-source toolbox for the simulation of photonic components and systems in Simulink and includes a library of common building blocks of photonic circuits. These models are built based on a compact phase shifter model [56] that can be used to replicate the physical behavior of optical components such as MZ Interferometers and microring resonators. These models can be readily used in MATLAB for the simulation of silicon photonic systems.

2.4.2 Circuit Simulation Approaches

In electronics, circuit simulation typically relies on SPICE modeling of the components based on Kirchhoff's law and Modified Nodal Analysis for voltage and current. The simulation of photonic circuits is fundamentally different since it involves optical waves. Optical waves are defined by an amplitude and a phase. The waves propagate alongside waveguides and can travel in both directions due to reflections; therefore, photonic circuits cannot be modeled by effort-flow formalism used in other physical domains such as electronic circuits and mechanics [22]. This makes it difficult to simulate circuits involving optical and electronic components such as neuromorphic architectures inside the same framework. The work in [38] studies the design flow for photonic circuits and describes four possible approaches to this problem. The work in [54] reviews these four general approaches in the scope of the simulation of neuromorphic circuits:

(1) "Co-simulation using separate electronic and photonic circuit simulators."

In this approach, simulators work in lockstep with signals being converted and exchanged from one simulator to the other. Exchanging signals and data between an electronic and photonic circuit simulator scales poorly for larger-scale neuromorphic architectures.

(2) "Partitioning and simulation using separate electronic and photonic circuit simulators."

The circuit is split into electronic and optical partitions to be simulated separately in domain-specific environments. According to the defined type of information shared by the environments, the parts are simulated in the right order over the time domain. The outputs and signals from each partition are used as input to the other in order, hence allowing to replicate the whole system. Since photonic neuromorphic architectures involve optical-electrical-optical conversions, separate simulation of electronic and optical components is not suitable for this purpose. Additionally, systems that include feedback loops between the two partitions or operate in both directions are not suitable for this approach.

(3) "Simulate photonics and electronics together in a photonic circuit simulator. "

Simulating electronic circuits into a photonic simulator is possible in some environments [38] but it requires electronic designers to abandon their familiar, reliable, simulation environment.

Current photonic tools still do not support the simulation of optical and electronic elements for large-scale circuits required for neural networks and might not include support for necessary electronic building blocks.

(4) "Simulate photonics and electronics together in an electronic circuit simulator."
Modeling photonic circuits inside an electronic circuit simulator has been demonstrated in [53] for the SPICE analysis of a laser neuron. An example of this approach for the simulation of a spiking neural network is shown in Fig. 2.17. The physical model simulator SIMPEL is used to simulate the behavior of a laser neuron in a circuit. The resulting physical parameters are used in Nengo [57] neural framework to predict the performance of the circuit to accomplish a task. Photonic quantities can be mapped to electronic quantities using circuit models in Verilog, thus leading to a working environment for optoelectronic circuit simulation for a predefined subset of applications and circuits.

2.4.3 Neural Modeling Software and Engineering Frameworks

While the tools and approaches covered so far enable the design and simulation of silicon photonic circuits, they are not suitable for the design of neural functions for a specific application Neural Engineering Frameworks propose a methodology to create and simulate optimized large-scale neural functions that can leverage biologically inspired models for neurons and demonstrate their cognitive abilities. By using these frameworks, one can create a neural function, consisting of a population of neurons and their synaptic weights, based on a given application specification.

The Neural Engineering Framework (NEF) proposed by [58] follows three main principles of "Representation", "Transformation" and "Dynamics" for the construction of large neural models and for fast and scalable simulations. In such a framework, a population of neurons represents a vector of real numbers that change over time (representation). The weights between this population of neurons are defined by the linear or nonlinear functions to be applied on these vectors through linear decoding (transformation) and therefore saving significant compute time during the simulation. These vectors then can be considered as a set of state variables to model a dynamic system (dynamics). In a sense, this approach takes a high-level description of a neural function and combines it with anatomical constraints to produce a detailed model of a population of neurons trained to produce the desired output.

These frameworks can potentially be used by photonic researchers for system-level simulation and training neuromorphic photonics in conjunction with behavioral models and circuit design flow approaches to assess and optimize the performance of the architecture. As an example, Nengo [57] is a neural engineering framework with graphical and scripting capabilities that support the simulation of large-scale spiking neural networks. Nengo takes advantage of TensorFlow framework and supports Python scripting and enables quick and accessible prototyping to researchers. As previously discussed, Nengo has been jointly used with SIMPEL for the simulation of photonic spiking neural networks involving lasers. However, the simulation of networks involving other types of components such as ring resonators and PCM remains to be investigated.

BINDsNET [59] is another framework created as a Python package to support the simulation of Spiking Neural Networks by extending existing Pytorch capabilities that is popular and established among Machine Learning development communities and enables rapid prototyping of spiking neural networks on machine learning applications and with the help of existing optimized code path from Pytorch and with the advantage of familiar scripting language. For a comparison of various neural frameworks that support spiking neural networks refer to [59].

Photonics neuromorphic hardware is sensitive to imperfections in manufacturing processes, thermal variations, and aging, which can lead to a drift in their parameters after extended periods of use. For example, WDM multiplexers commonly used in neuromorphic architectures are built using microring resonators and rely on a careful calibration of resonance frequencies depending on the size of gaps or thermal constraints. These imperfections in hardware should be considered in the training algorithm and design stage. These parameters should be exposed to the network trainer and different neuromorphic processors should be able to perform the same function despite varying parameters in hardware [54].

The neural networks should be modeled at both behavioral and physical abstraction levels. Meanwhile, the behavioral layer simulates the flow of information throughout the network, the information representation and their encoding, and the network learning process.

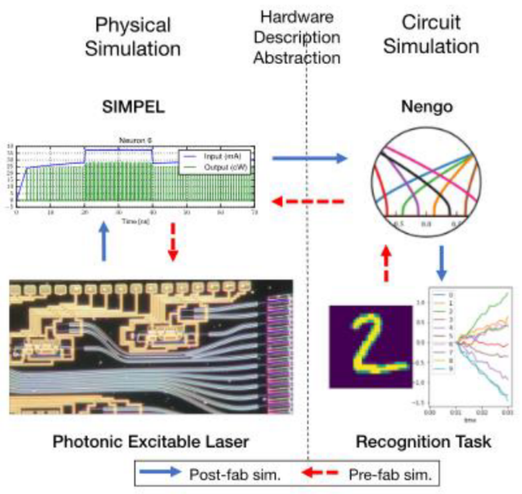


Figure 2.17: Simulation approach for a spiking neural network using SIMPEL and Nengo. SIMPEL can simulate the behavior of the component (in this case an excitable laser neuron) and create variables to be used for neural network simulation in Nengo to evaluate the performance and accuracy on a task such as digit recognition.
This figure is reproduced from [54].

The photonic neuromorphic processors should be represented and implemented by users who do not have any background in photonics. For this purpose, the hardware should be described through a specification that breaks down the photonic processor into different levels of detail and hierarchy from individual parts to lower-level photonic components. Programmers should be able to design their desirable neural function to a certain specification that photonics engineers will be able to design their components toward as well. This would allow programmers to define the target neural functions independently from the underlying photonic hardware and the photonic engineer will have the freedom to design the processor that meets a specification with different approaches that in the end could serve the same application but with different performance, power efficiency, and throughput.

The work in [54] proposes the use of hardware description languages (HDL) to describe the circuits; the objective is twofold. First, HDL allows for scalable simulation of the network by the. Second, it provides a set of specifications that photonics engineers can target during the hardware design. Such an approach can be extended to support higher-level programming languages such as Python or C to describe the behavior of the hardware in an algorithmic way. This higher-level coding then can be synthesized into HDL. Then the resulting HDL code could be assembled onto a photonic neuromorphic processor, hence following a design flow similar to the one used to program FPGAs.

The physical behavior and response of photonic components such as microring resonators or excitable lasers heavily rely on environmental factors like temperature or physical design tolerances that demand careful calibration and control of the components for producing predictable desired behavior. When scaling up to larger systems with numerous components, the speed and viability of the simulation become more important. By using behavioral models or other simplified building blocks that represent these photonic components, faster and more viable computer-assisted designs are made. However, while the existing approach already demonstrates the feasibility of using the behavioral model in large-scale simulation, key hardware characteristics such as thermal sensitivity are to be considered. Table 2.2 compares the capabilities of various simulations tool.

Table 2.2: Comparison of Simulations Tools for Photonic SNNs

Design and Simulation stage	Lumerical [51]	SPICE Analysis [53]	Synopsys [60]	SiPh Simulink [55][56]	Nengo [57]	BindsNET [59]	Neurophox[61][62][63]
Behavioral Models for Silicon Photonic components	Yes	Yes	Yes	Yes			Yes
Silicon Photonic Circuit Simulation	Yes		Yes	Yes			
Neural Network Modeling					Yes	Yes	
Optical Neural Networks							Yes

2.5 Conclusion

Spiking Neural Networks are considered to be the third generation of neural networks. They are diverging from artificial neural networks by incorporating time dependence into the architecture. In SNNs, each neuron and synapse obey dynamical rules analogous to the biological brains. SNNs promise higher energy efficiency and could be more suited for event-driven applications compared with ANNs. This leads to fast-growing research and development efforts to design efficient hardware to implement SNNs, which remain to be found. Current implementations of silicon photonic networks have been demonstrated but with low neuron counts. The need to increase the number of neurons requires scalable hardware enable to improve the processing capability of photonics neural accelerators and to broaden application domains. Current architectures mostly rely on external electronic control systems, which is one of the main limits for scalability. Indeed, since the number of programmable parameters increases quadratically with the neurons count, the large-scale integration of electronic controllers and their co-packaging alongside photonic hardware is the main challenge.

Although today's neuromorphic silicon photonic networks cannot reach the neuron density and stability of current CMOS implementations, they promise high bandwidth and low latency processing beyond the capabilities of digital electronic circuits. This raises the need for suitable applications allowing to fully take advantage of features neuromorphic silicon photonics circuits can provide. We found that control systems requiring fast decision-making involving systems with rapidly changing parameters could potentially be a suitable application.

Neuromorphic silicon photonic architectures are analog processors requiring a direct mapping of a given application to the hardware. Since optical components are sensitive to temperature, fabrication process variation, and aging, their actual behavior can not be fully predicted at design time, thus leading to different results and performance. This calls for robust hardware and neural functions able to compensate for the variation at run-time. The validation of such hardware required domain-specific standardized benchmarks such as pattern recognition that remain to be developed. As an example, pattern recognition based on the character was used in [45] to evaluate a photonic network composed of 15 neurons. Software for the deployment of machine learning algorithm and applications to silicon photonic processors are currently lacking. The standard approach involves the development of applications using generic neural network frameworks, such as PyTorch and Tensorflow, and the conversion to machine code for differing digital processors through compilers. Considering the diversity of photonic components required to design optical neural networks, their complex behavior, and their sensitivity to environmental variation, domain-specific neural engineering software is needed to design the hardware by taking into consideration the characteristics of the target hardware and similarly converting abstracted applications for specific analog photonic processors.

Chapter 3

Toward Large Scale All Optical Neural Networks

3.1 Introduction

Most optical spiking neural networks rely on WDM to transmit signals in the network. WDM involves accurate calibration of microring resonators (MRR). The resonance wavelength of these MRRs is easily influenced by environmental factors, which leads to challenges. This imposes a practical limit on photonic neuromorphic hardware where only a small number of synaptic connections per neuron can be accommodated.

To map large NNs to neuromorphic hardware, it is necessary to partition the network into smaller clusters than can be mapped onto hardware neurons. There have been various works proposed for partitioning and mapping NN models to crossbar-based electronic neuromorphic hardware [64, 65, 66]. These methods largely aim to reduce the number of synapses per neuron in the NN model and partition the network according to hardware resource constraints, such as synapses per neuron.

In this work, we propose a method to efficiently map pre-trained NN on an all-optical SNN architecture. The method relies on two algorithms leveraging weight partitioning and unrolling to reduce synaptic connections in a pre-trained NN model. The efficiency of the method is evaluated on the all-optical SNN architecture we propose. Results show that our method allows exploring the trade-off between classification accuracy and power consumption.

3.2 Design Flow

To efficiently map a NN on the targeted optical architecture, the network is partitioned, where each partition respects the constraints of the targeted hardware. We propose the design flow illustrated in Fig. 3.1. The inputs of the flow are:

- a NN designed and trained using an existing neural engineering framework (NEF) [67][68]. Such NN typically performs a specific cognitive task such as classification. It is represented by a graph where vertices and links represent neurons and synaptic connectivity respectively.
- A library of hardware components allowing to model of the behavior of the targeted optical accelerator [69, 70, 71]. Physically accurate modeling of the silicon photonic components involving PCM is challenging [69] and not suitable for performing large-scale simulation that is required for the implementation of neural network. The waveguides embedded with islands of PCM can provide varying but small numbers of transmissions. This feature enables performing different weighting on the input signals. By quantizing the weights in the neural network, we can represent the weighting that these synapses can provide. Similarly, other components of the optical neural network such as the activation unit can be represented by their behavior and their effects on signals. This helps to simplify the simulation for larger neural network models. Section 3.5.1 further describes the models that we used in this work.

A key characteristic we use is the number of neurons and synapses per neuron. According to the all-optical network proposed in [45], our architecture is built upon the number of calibrated MRRs. Hence, taking into account the neuron count per computing unit allows us to ensure that

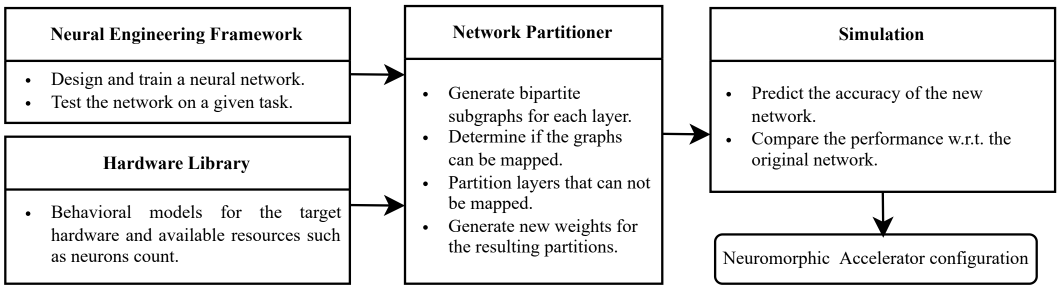

Neural Engineering Framework
Design and train a neural network.
Test the network on a given task.
Hardware Library
Behavioral models for the target hardware and available resources such as neurons count.
Network Partitioner
Generate bipartite subgraphs for each layer.
Determine if the graphs can be mapped.
Partition layers that can not be mapped.
Generate new weights for the resulting partitions.
Simulation
Predict the accuracy of the new network.
Compare the performance w.r.t. the original network.
Neuromorphic Accelerator configuration

Figure 3.1: Proposed design flow.

i) the hardware architecture is realistic and ii) each graph partition can be mapped. To overcome the rather limited computing capability of each neuron cluster, the design of a realistic accelerator requires flexible interconnected units. To satisfy the connectivity constraints, the partitioning involves the duplication of neurons and the introduction of additional layers in the NN. Since the network structure is changed, new synaptic weights are defined from the initially trained weights, which we achieve without retraining. Indeed, while retraining would lead to better accuracy, it is also a time-consuming task. Finally, the performance of the resulting partitioned network mapped on the hardware is estimated using system-level simulation.

3.3 Hardware Architecture

The optical NN we target is composed of neuron clusters, where each cluster contains N neurons with N synaptic inputs. The cluster is based on the all-optical NN demonstrated in [45]. Compared to [45], we propose to use a multi-output collector and a multi-input distributor on the interface of a cluster. Our goal is to better scale by enabling the mapping of NNs on multiple interconnected neuron clusters. For this purpose, each cluster has three input and three output waveguides, each one enabling the transmission of multiple signals using WDM. The multi-input distributor is depicted in Fig. 3.2. By configuring the MRRs in the inputs, we can select the inputs to be distributed to the neurons in a cluster. The MRRs on the output is calibrated to evenly distribute the signals among the neurons. The schematic of the multi-output collector is depicted in Fig. 3.3. By configuring the MRRs in the collector, it is possible to transmit the output of any neuron connected to the collector to neighboring neuron clusters.

The schematic of a neuron cluster composed of N neurons, a distributor, and a collector is illustrated in Fig. 3.4. Each of N neurons in the cluster can accommodate N synapses, therefore the cluster is referred by a $N \times N$ cluster. For flexibility purposes, we assume that the signal wavelength emitted by a neuron can be configured, which can be achieved using tunable lasers and ring tuning. Finally, since we aim at implementing forward propagation only, we connect the output of each cluster toward the input of three neighboring clusters, following the pattern illustrated in Fig. 3.5

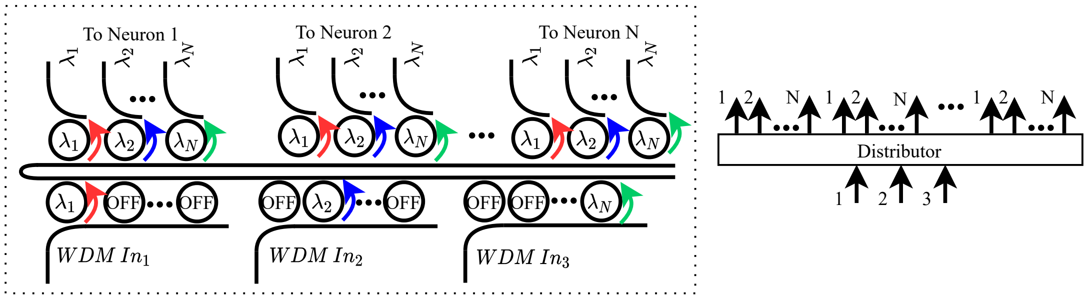


Figure 3.2: Schematic of the multi-input distributor.

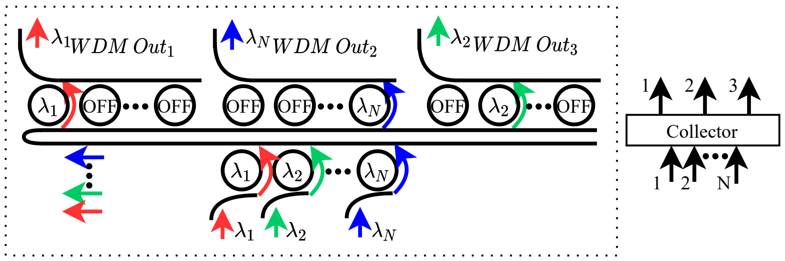


Figure 3.3: Schematic of the multi-output collector.

The number of neurons and synapses per neuron, N, is directly related to the number of calibrated MRRs. Hence, taking into account the neuron count per neuron cluster allows to ensure that i) the hardware architecture is realistic and ii) each graph partition can be mapped. To satisfy the connectivity constraints, the partitioning involves the duplication of neurons and the introduction of additional layers in the NN. Since the network structure is changed, new synaptic weights are defined from the initially trained weights, which we achieve without retraining. Indeed, while retraining would lead to better accuracy, it is also a time-consuming task. Finally, the performance of the resulting partitioned network mapped on the hardware is estimated using system-level simulation.

In this way, the target hardware can theoretically process N number of neurons in each cluster where each can accept N synaptic inputs. To this end, it is necessary to partition and distribute the target neural network in a way that conforms to the specifications of the clusters.

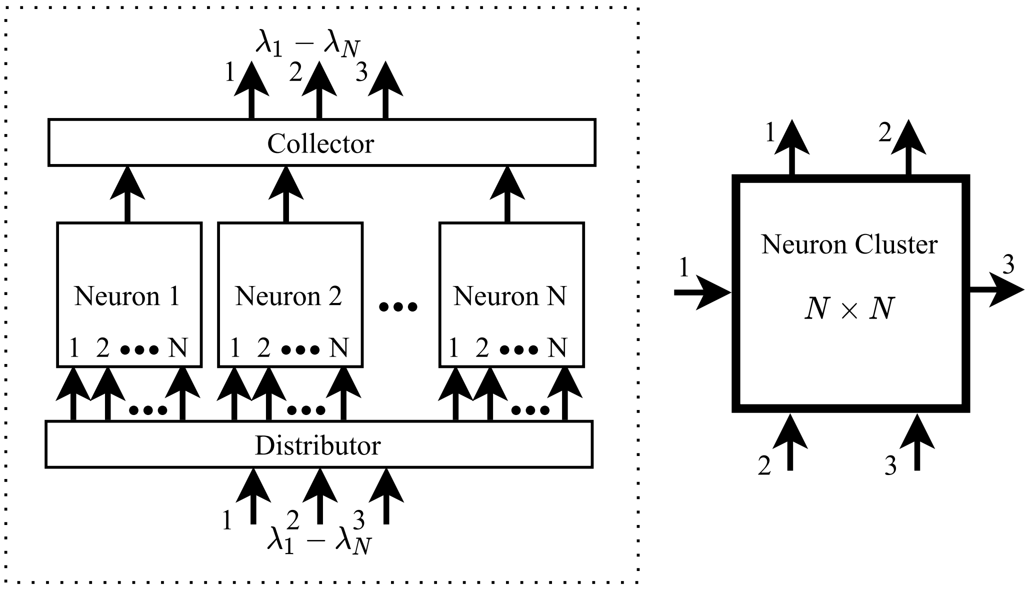


Figure 3.4: Schematic of a $N \times N$ neuron cluster.

3.4 Energy Consumption Estimation

In this work, energy estimation is used for the purpose of comparison between different models only and extrapolated based on the number of neurons used in each configuration. The main sources of energy consumption in this architecture are for switching the activation units in each neuron and programming the synaptic weights in all neuron clusters. The PCM synaptic weights are programmed before operation and do not require a continuous energy source, therefore to estimate operating power consumption it is only necessary to estimate the energy required for switching of the activation units [45]. To this end, the combined energy of the input pulses, subjected to attenuation in the distributor and PCM synapses, reaching the activation unit must be enough to switch the state of the PCM. The energy at each PCM-synapse, E_{syn} can be estimated by $E_{syn} = E_{in} \cdot (1 - L_{MRR}) \cdot C$, where E_in is the energy of the input pulses to the distributor, L_{MRR} is the percentage of losses in the coupling MRR and C is the measured coupling. By neglecting the losses in the PCM-synapse itself, the energy that reaches the PCM-cell on the activation unit after the second MRR can be estimated by $E_{act} = E_{syn} \cdot (L_{insertion}) \cdot C$, where $L_{insertion}$ is the insertion losses in the add-drop MRR configuration. The approximate values for the parameters are extracted from the experimental results in [45]. L_{MRR} can be estimated to be 9.3% to 29.0% depending on the position of then

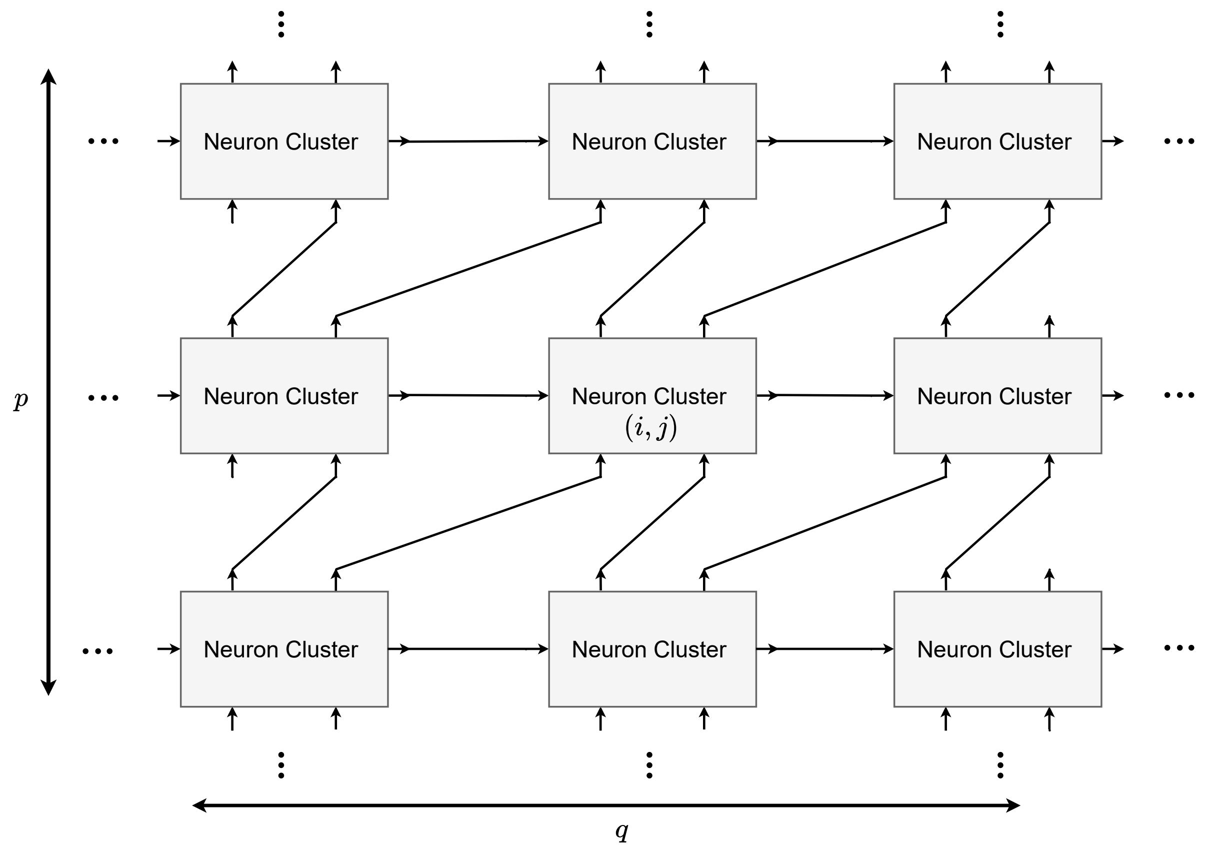


Figure 3.5: 2D organization of neuron clusters with size $p \times q$. The data flow according to the indicated directions.

neuron, C is estimated to be 0.165 for E_{syn} and 0.8 for E_{act}. The operation of a network with 4 neurons in [45] is estimated to consume 7.5 nJ per cycle.

3.5 Network Partitioning Algorithm

The main goal of the partitioning method is to efficiently distribute the neurons and their synapses onto the clusters by considering hardware constraints. To achieve this, we have implemented the following key features:

- Every layer is partitioned in a way that each partition fits on a single cluster. These clusters are then connected through the introduction of additional layers to recreate the original function of the layer with similar input and output sizes.

- The partitions are allocated to clusters by considering inter-partition dependencies. Careful wavelength allocation is achieved to minimize the number of clusters required to propagate dependencies. The multi-output collectors and multi-input distributors have been configured accordingly.

3.5.1 Problem Formulation

We define the problem by representing each layer of the network as a bipartite graph $G(U, V, E)$ where U and V denote each of the two parts of the graph and represent the input and outputs of the layer and E denotes the edges (links) in the graph corresponding to the synaptic weights between the two parts. Therefore, the parameters of the model are as follows:

- F_{in}: A matrix of pre-trained weights of a Fully Connected (FC) model used as input to the algorithm.
- F_o: A matrix of pre-trained weights that is mappable to a neuron cluster, representing the output of the algorithm.
- p, q: The number of neuron clusters in each axis resulting in a total of $p \times q$ clusters.
- $u = |U(G)|$: number of input neurons in layer G.
- $v = |V(G)|$: number of output neurons in the layer G.
- N: number of neurons per cluster. In this work, we assume N also equals the number of wavelengths and hence to the number of possible fan-in synaptic connections per neuron.
- w: The number of possible different weight values for quantization. It corresponds to the states of a PCM-cell synapse [72] and the available levels of signal transmission ratio on each synapse. These ratios are controlled by configuring the states of the PCM-cells which in return create varying levels of attenuation and therefore different weights that can physically be represented on the hardware PCM-synapse. For example, the PCM synapse demonstrated in [72] can provide 10 levels of signal transmission ratio.
- $T_{p \times q}$: Scaling factor for synaptic weights in a cluster, which enables calibration of the sensitivity of the PCM-cell in the activation units [45]. In terms of hardware implementation, this

scaling can be obtained by either i) adapting the laser power of the source neuron or ii) tuning ring resonators in the destination collector to extract the required signal power. We assume all neurons in a given cluster have the same scaling factor. This scaling factor along with w allows quantizing the weights from the source network to $T_{i,j} \times w$ unique levels.

```
Input: $F_{in}$ input FC network, $N, p, q, w$
Output: $F_o$ mapped configuration containing the information for every synaptic weight.
  $F_o \leftarrow Initialize(p, q, N)$
  for every layer $G$ in $F_{in}$ do
      $u \leftarrow |U(G)|$
      $v \leftarrow |V(G)|$
      if $v > N$ or $u > N$ then
          ▷ Check if synaptic connections to a neuron exceed the number of neurons per cluster and find the number of partitions.
          $Partitions \leftarrow \lceil u/N \times \lceil v/N \rceil$
          $W_{Partitions \times N \times N}$
                                              ▷ Create an array to store the weights of each group.
          $W \leftarrow SBA(\mathrm{G}, Partitions, N)$
                                              ▷ Partition the weights using SBA.
          $W, T_{Partitions} \leftarrow Quantize(W, w)$
                                              ▷ Quantize the weights and determine a scaling factor for each partition.
          $G_{partitioned} \leftarrow Initialize(W, T_{Partitions})$
          ▷ Create a graph that represents the partitioned state of the layer. This determines the state of connections between neuron clusters.
          $F_o \leftarrow Map(G_{partitioned}, W, T_{Partitions})$
                                              ▷ Write $W$ and $T$ to proper positions based on $G_{partitioned}$.
      else
          $W, T \leftarrow Quantize(G)$
          $F_o \leftarrow Map(G, W, T)$
  Predict accuracy performance with $F_o$
  Calculate the number of allocated neurons and clusters in $F_o$
```

Figure 3.6: Network partitioning algorithm.

By considering $p \times q$ clusters with N neurons per cluster, there is a total of $p \times q \times N^2$ synaptic weights to configure. These weights are represented by a matrix F_o where each entry corresponds to a PCM synapse in the architecture. Unused synaptic connections are assigned to 0 and are discarded during simulation. The algorithm to generate F_o is illustrated in Fig. 3.6. The input to the algorithm is a pre-trained FC model where each layer G is individually tested to be mapped to the hardware. If $u > N$ or $v > N$ (i.e. if the number of synapses exceeds the number of cluster inputs or outputs), then partitioning is required. The partitioning relies on Sorting Based Algorithm (SBA) presented in [64] and [65] and summarized as follows. The input and outputs neurons are divided into $\lceil u/N \rceil$ and $\lceil v/N \rceil$ groups with size N respectively for a total of $\lceil u/N \rceil \times \lceil v/N \rceil$ partitions. Since partitioning introduces additional layers and neurons that impact the accuracy, partitions are defined by sorting synaptic weights according to their magnitudes. This allows to use of the same quantization and scaling factor $(T_{i,j})$ for a given cluster, thus limiting the accuracy reduction. The resulting matrix F_o corresponds to the architecture configuration, for which accuracy is evaluated by simulation.

3.5.2 Mapping Example

Fig. 3.7 and Fig. 3.8 provide an example, demonstrating the process of converting a given layer in a NN through unrolling and weight partitioning respectively. The neural functions represent the connectivity in a layer between the inputs x_i and output neurons y_j with synaptic weights w_{ij}. In this example, 7 inputs are connected to 3 neurons and the target hardware can only accommodate $N = 3$ synapses per neuron (3×3 neuron clusters). y_1 does not exceed 3 synapses and can be directly mapped to a cluster without modification in each method while y_2 and y_3 require 7 and 4 synapses respectively. The unrolling process decomposes y_2 and y_3 to a series of neural computing units (f_i) each with 3 synapses. All of the connectivity from the inputs are preserved which could maintain accuracy performance and model quality. The weight partitioning process, groups the inputs based on the magnitude of w_{ij} (g_i) so that the strongest weights are mapped to shared clusters while the smallest weights are pruned and are not represented on the hardware. In this example, $x_4 \cdot w_{43}$, $x_6 \cdot w_{63}$ and $x_7 \cdot w_{72}$ are pruned which could lead to degraded model quality. The unrolling scheme requires 6 neurons in total located across 6 separate clusters and requires injection of inputs in subsequent layers while the weight partitioning requires 5 active neurons across 3 neuron clusters. This leads to reduced energy consumption and size of the circuit at the expense of reduced accuracy performance and model quality.

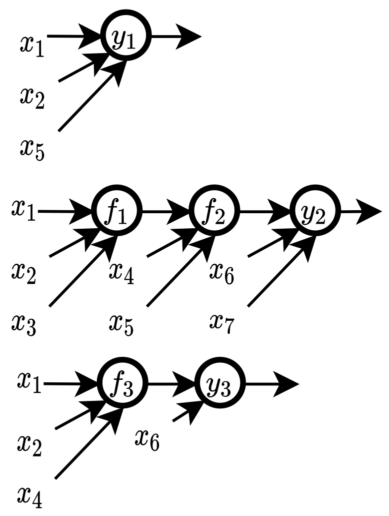
Neural Functions

$$y_1 = x_1 \cdot w_{11} + x_2 \cdot w_{21} + x_5 \cdot w_{51}$$
$$y_2 = x_1 \cdot w_{12} + x_2 \cdot w_{22} + x_3 \cdot w_{32} + x_4 \cdot w_{42} + x_5 \cdot w_{52} + x_6 \cdot w_{62} + x_7 \cdot w_{72}$$
$$y_3 = x_1 \cdot w_{13} + x_2 \cdot w_{23} + x_4 \cdot w_{43} + x_6 \cdot w_{63}$$

Unrolling

$$f_1 = x_1 \cdot w_{12} + x_2 \cdot w_{22} + x_3 \cdot w_{32}$$
$$f_2 = f_1 + x_4 \cdot w_{42} + x_5 \cdot w_{52}$$
$$y_2 = f_2 + x_6 \cdot w_{62} + x_7 \cdot w_{72}$$

$$f_3 = x_1 \cdot w_{13} + x_2 \cdot w_{23} + x_4 \cdot w_{42}$$
$$f_4 = f_3 + x_6 \cdot w_{63}$$
$$y_3 = f_4$$



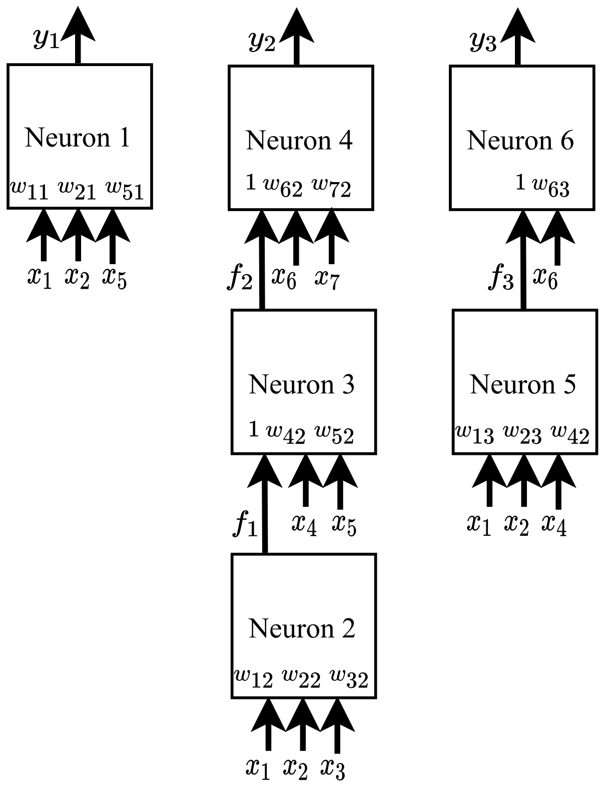


Figure 3.7: Illustration of SNN partitioning with unrolling

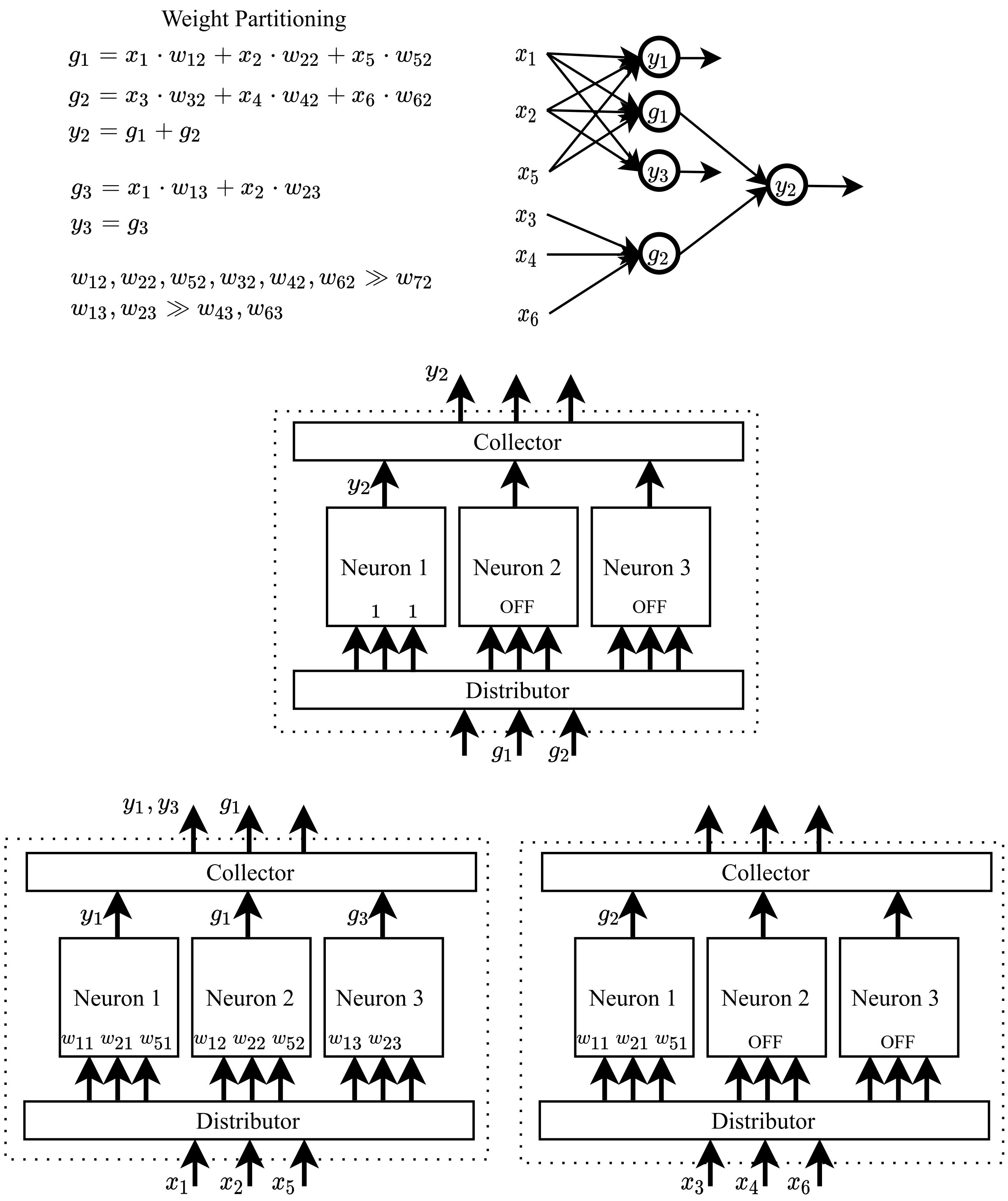


Figure 3.8: Illustration of SNN partitioning with weight partitioning on target hardware.

3.6 Conclusion

This chapter explores the mapping optimization of pre-trained neural networks on neuromorphic photonic hardware accelerators. For this purpose, we developed a method to efficiently partition and distribute NN models onto instances of neuron clusters. We also proposed an architecture of neuron clusters where they communicate with a configurable interconnect. The mapping method together with the neuron cluster architecture realizes a larger neural network model by using instances of existing photonic SNN hardware.

Chapter 4

Synaptic Weighting with Photonic Interconnects

4.1 Introduction

Microring resonators are the fundamental building blocks of the photonic circuit. The very large architectures required for implementing neural networks would require an increasingly larger number of MRRs and with that, the need for precise calibration of potentially several thousands of MRRs rises. This gives importance to designs that aim to reduce the number of MRRs while attempting to maintain the core functionality of the neural network architecture. The architecture relies on a set of N MRRs to de-multiplex the WDM input and then subjects each individual wavelength to weighting by using PCM cells. These weighted inputs are combined by using N additional MRRs into a single waveguide that leads to the activation unit. This approach would require $2 \times N$ MRRs to weigh and connect N inputs to a single neuron. By eliminating the first set of N MRRs and combining PCM-cells with the waveguide we could reduce the number of MRRs to N. With this goal in mind, the architecture proposed in [45] was modified, according to Fig. 4.1, to halve the number of MRRs.

In this approach, the inputs are subjected to attenuation by all the PCM cells on their path. It is clear then the state of the PCM-cells (y_i) and the resulting attenuation in this design differs from the states in the original design (x_i). In order to properly weigh the inputs then it is necessary to find the equivalent states for these PCM cells. The objective of this chapter is to propose an architecture for neural networks based on this idea and propose an interconnect and a method to find the equivalent values for the PCM cells.

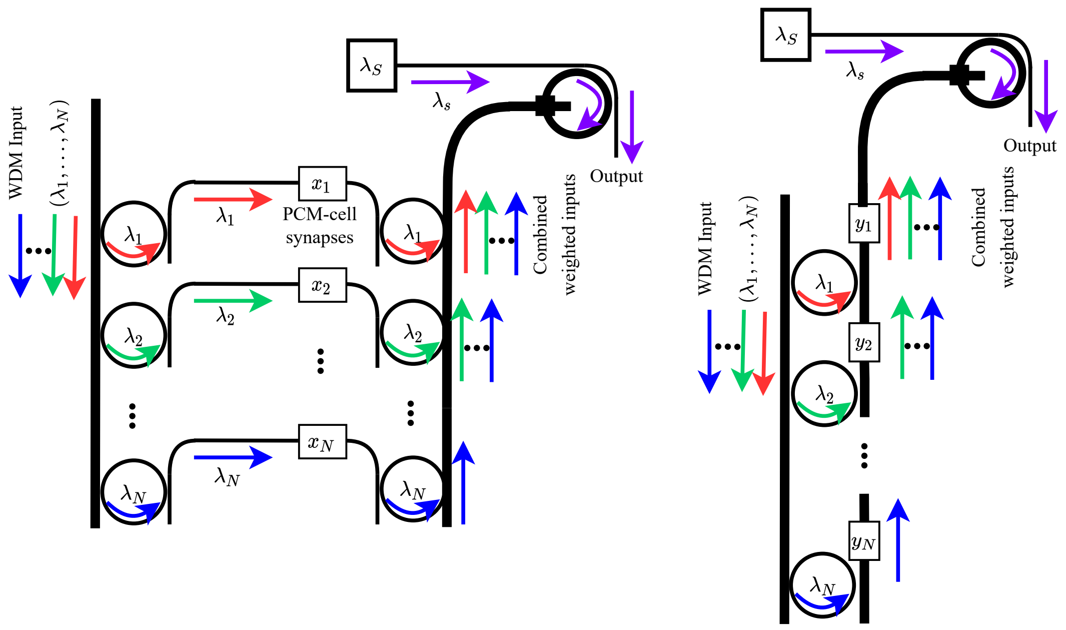


Figure 4.1: Comparison between synaptic weighting proposed by Feldmann et al and the proposed method. The number of MRRs is reduced to half.

4.2 Architecture

Fig. 4.2 illustrates the proposed interconnect for synaptic weighting in a layer with N inputs and N output neurons. The output of the neurons from the previous layer, similar to the work in [45], are combined into a waveguide by the collector and directed to the layer as a WDM input with wavelengths λ_1 to λ_n. The MRRs on the path are calibrated to distribute the input power at each wavelength equally between each of the n output neurons (f_i). The input signals are coupled into the waveguides leading toward each of the f_i's and are subjected to different levels of attenuation. This total effective attenuation depends on the position of the MRR with the corresponding wavelength. In this example, the input λ_1 is subjected to attenuation by only y_{11} PCM-cell when propagating toward f_1 but when it is propagating toward f_n, it is facing the effective attenuation (x_{nn}) caused by $y_{1n}, y_{2n}, \cdots$ and y_{nn}. In this way, by assigning wavelengths to MRRs in higher or lower positions with respect to each of f_i's, we can create stronger or weaker synaptic weighting. The level of attenuation caused by each of PCM-cells (y_{ij}) can also be adjusted to further increase the accuracy of the weights.

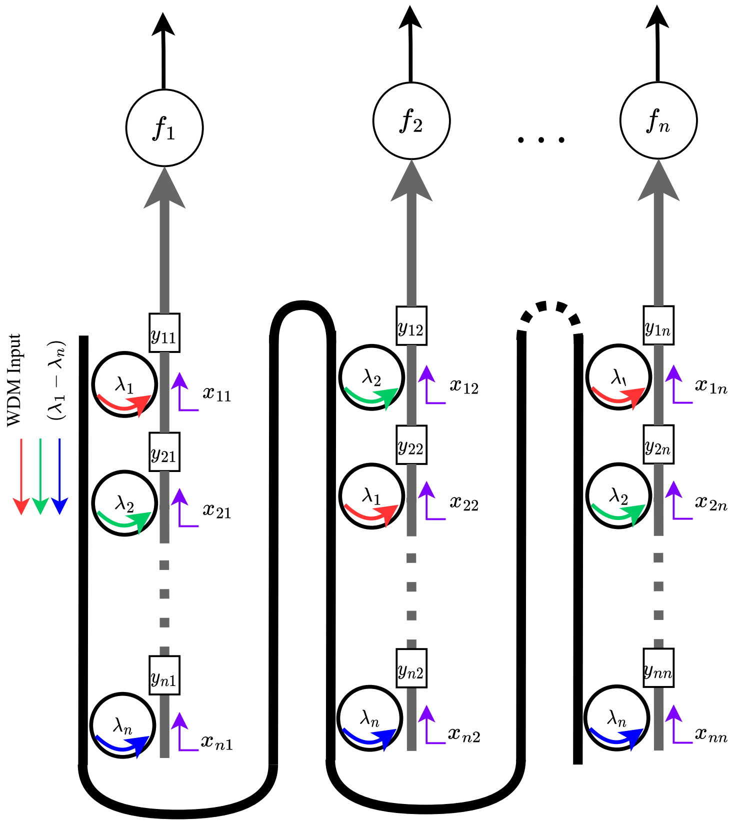


Figure 4.2: Proposed architecture.

The design in [45] with N neurons each with M synapses required $2 \times M \times N$ MRRs to realize all synaptic interconnects. In contrast, in this design the number of MRRs are reduced to $M \times N$. Fig. 4.3 illustrates how this interconnect can be repeated to create an entire neural network encompassing many hidden layers.

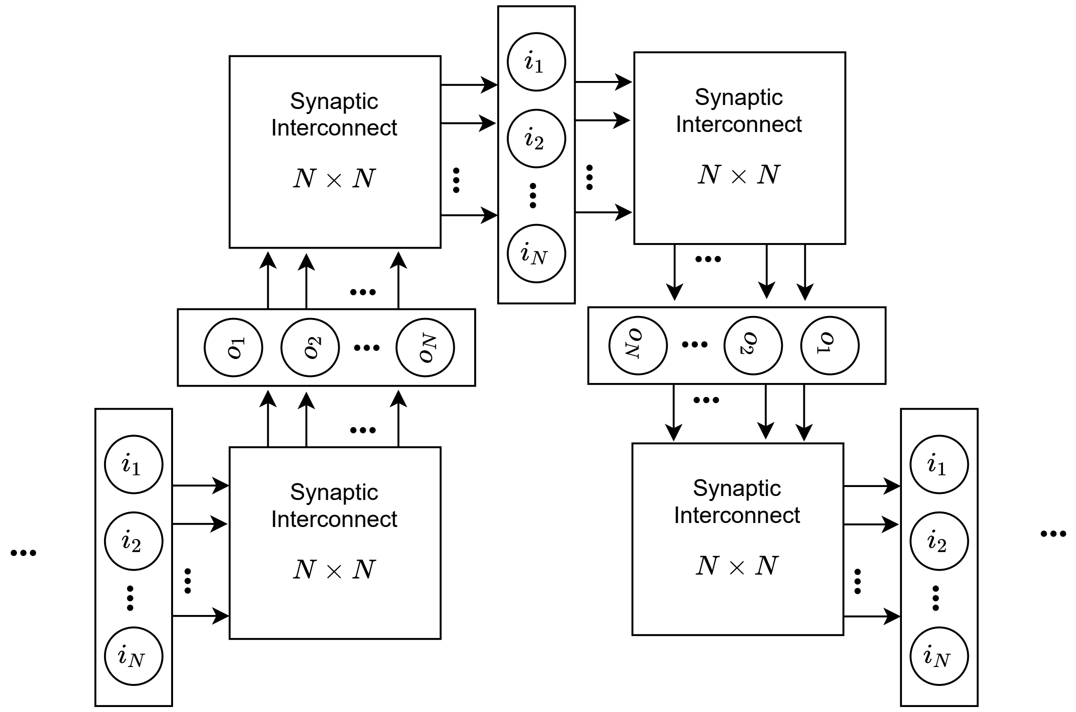


Figure 4.3: Architecture for large neural network using the proposed synaptic weighting architecture.

4.3 Wavelength assignment and PCM adjustments

Finding the optimal position for each of the wavelengths and calibration of PCM-cells based on the weights of the original neural network is the main goal of this section and it can be achieved by following a similar design flow described in Section 3.2. In this method, y_{ij} represents the attenuation in dB caused by $i-th$ PCM-cell leading to f_j. The effective attenuation between input i leading to f_j, x_{ij}, equals to the sum of all attenuation in the path or:

$$x_{ij} = y_{ij} + y_{i-1j} + ... + y_{1j} \quad (dB)$$

$$\forall i, j \in 1, \cdots, n$$

In this way, the inputs placed farther from the neurons will be subjected to higher attenuation which corresponds to weaker synaptic weights. Conversely. inputs placed closer to each neuron will represent stronger connectivity. The mapping method has to select and assign inputs accordingly to create the closest representation of the original weights in a NN model. The proposed method is illustrated in Fig. 4.4. The input to this method is a Weight matrix (W). W is a $n \times n$ matrix containing the pre-trained weights of the input NN model where ω_{ij} is the $i-th$ input to $j-th$ neuron in a hidden layer.

Each wavelength represents a single input to the layer thus the same wavelength can be used only once when coupling to a waveguide connected to the same neuron. The placement of the MRR with a matching wavelength signifies the strength of the weight corresponding to that input. Sorting the synaptic weights allows for determining the positions of the wavelengths. The same wavelength must be assigned to all MRRs that correspond to an input. The sorted weight matrix SW is a $n \times n$ matrix and it is formed by sorting each column of the matrix W in descending order. The wavelength matrix Λ is a $n \times n$ matrix where λ_{ij} represents the wavelength of the $i-th$ MRR coupling toward neuron f_j. In this way, the wavelength matrix will maintain the information that what input each entry in matrix SW corresponds to.

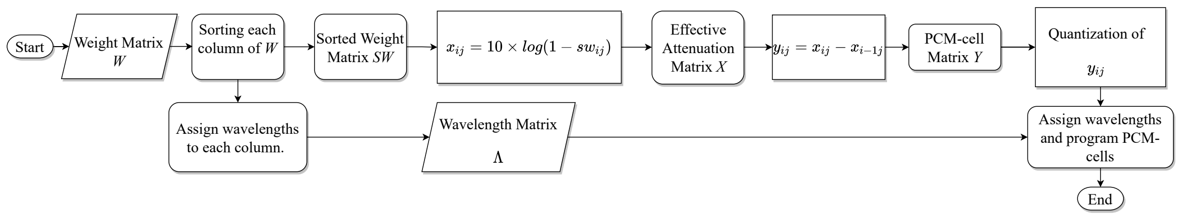

Start
Weight Matrix W
Sorting each column of W
Sorted Weight Matrix SW
$x_{ij} = 10 \times log(1 - sw_{ij})$
Effective Attenuation Matrix X
$y_{ij} = x_{ij} - x_{i-1j}$
PCM-cell Matrix Y
Quantization of y_{ij}
Assign wavelengths to each column.
Wavelength Matrix Λ
Assign wavelengths and program PCM-cells
End

Figure 4.4: The flowchart for the proposed method for wavelength assignment and programming of PCM-cells.

The original weights from the input NN model are assumed to be normalized floating-point values. In the next steps, the weights must be transformed to values that the PCM synaptic weights can physically represent. To this end, we need to review the capabilities of the PCM synapse demonstrated by [72] and illustrated in Fig. 4.5-A. The experiment on a PCM synaptic weighting sample demonstrated 11 levels of transmission ratios corresponding to 11 levels of weighting. The transmission change depends on the number of pulses with the same energy used to program the PCM-cells. This dependence is illustrated by Fig. 4.5-B. The model

$$\Delta T = \Delta T_0 + A \times \exp\left(B \times N_p\right)$$

can be fitted to this graph to model the relation between transmission ratio (ΔT) and the number of pulses N_p required for programming and A and B are parameters of the model. In this way, we could estimate how to program each PCM-cell to the desired transmission ratio by finding the number of required pulses. By assuming that original weights correspond to target transmission ratios for each input, we can calculate the attenuation required for each input and by calculating the differences of each subsequent weight, we can determine the desired attenuation in each of the PCM-cells in this architecture and program them accordingly. The effective attenuation matrix X is a $n \times n$ matrix where x_{ij} is the total attenuation seen from before PCM-cell y_{ij} in dB as illustrated in Fig. 4.2. This matrix is formed from matrix SW via $X = 10 \times log(1 - SW)$. We can now form the PCM-cell matrix Y where y_{ij} is the target attenuation for $i-th$ PCM-cell toward neuron f_j and it is given by:

$$y_{ij} = x_{ij} - x_{i-1j}$$

Finally to program the PCM-cell, we need to determine the number of pulses required for achieving the closest possible attenuation in each PCM-cell. The number of pulses N_p needs to be an integer value between 0 and 10000 according to the work in [72]. This step results in the quantization of y_{ij} according to the fitted model:

$$y_{ij} = C_1 + C_2 \times N_p$$

Where C_1 and C_2 are constant parameters of the model. The experimental values for parameters A and B are presented in [72] and can be used to derive C_1 and C_2.

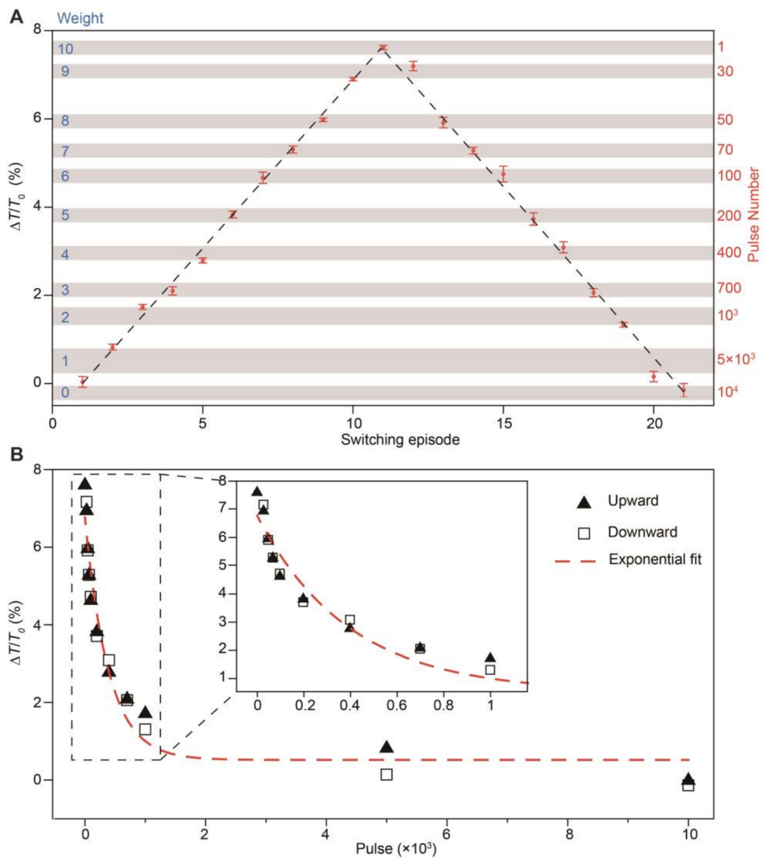


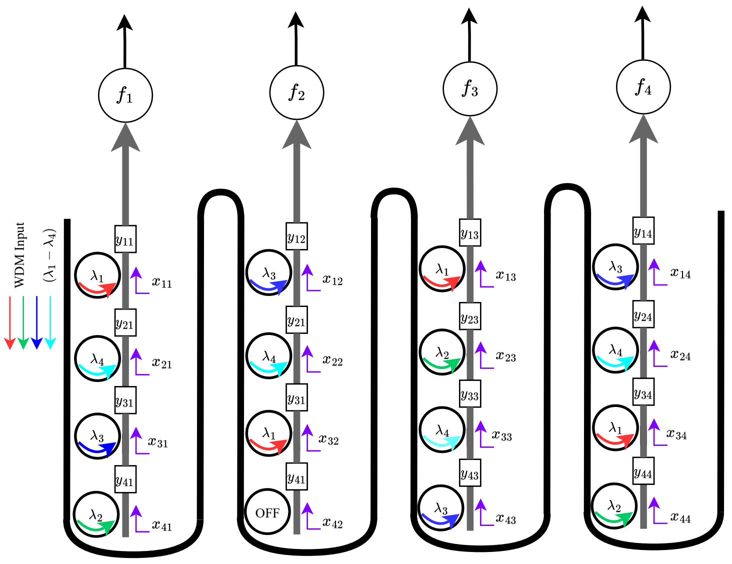
Figure 4.5: (A) Optical transmission change of sample 2 shows 11 weights with pre-determined numbers of 20 ns optical pulse (216 pJ) with upward and downward sweeps. The corresponding number of pulses for each level is illustrated in the figure. (B) The dependence of the transmission change on the pulse number.
This figure is reproduced from [72]

4.4 Example

In this section, we use an example to help better illustrate the wavelength assignment and PCM adjustments method. In this example, the weight matrix W is a 4×4 matrix that represents a fully connected layer with 4 neurons according to Fig. 4.6. Each column of this matrix is then sorted in descending order to form matrix SW. Here, we can form a wavelength matrix Λ by assigning the same wavelength to entries that correspond to the same input. These are entries that are in the same row in the original weight matrix W. Effective attenuation matrix X is then formed from matrix SW via $x_{ij} = 10 \times log(1 - sw_{ij})$. In the next step, the difference between entries in each column is calculated to form PCM-matrix Y via $y_{ij} = x_{ij} - x_{i-1j}$. Finally, we quantize the entries of matrix Y following the described method to program PCM-cells. With this information, we can configure the interconnect to represent this layer

$$W = \begin{bmatrix} 1 & 0.33 & 0.5 & 0.33 \\ 0.25 & 0 & 0.5 & 0.25 \\ 0.33 & 1 & 0.25 & 1 \\ 0.5 & 0.5 & 0.33 & 1 \end{bmatrix} \Rightarrow SW = \begin{bmatrix} 1 & 1 & 0.5 & 1 \\ 0.5 & 0.5 & 0.5 & 1 \\ 0.33 & 0.33 & 0.33 & 0.33 \\ 0.25 & 0 & 0.25 & 0.25 \end{bmatrix} \Rightarrow X = \begin{bmatrix} 0 & 0 & 3.01 & 0 \\ 3.01 & 3.01 & 3.01 & 0 \\ 4.81 & 4.81 & 4.81 & 4.81 \\ 6.02 & NaN & 6.02 & 6.02 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} (\lambda_1) & (\lambda_3) & (\lambda_1) & (\lambda_3) \\ (\lambda_4) & (\lambda_4) & (\lambda_2) & (\lambda_4) \\ (\lambda_3) & (\lambda_1) & (\lambda_4) & (\lambda_1) \\ (\lambda_2) & (OFF) & (\lambda_3) & (\lambda_2) \end{bmatrix} \qquad Y = \begin{bmatrix} 0 & 0 & 3.01 & 0 \\ 3.01 & 3.01 & 0 & 0 \\ 1.8 & 1.8 & 1.8 & 4.81 \\ 1.41 & NaN & 1.41 & 1.41 \end{bmatrix}$$



WDM Input
$(\lambda_1 - \lambda_4)$
f_1
f_2
f_3
f_4
y_{11} y_{21} y_{31} y_{41}
y_{12} y_{21} y_{31} y_{41}
y_{13} y_{23} y_{33} y_{43}
y_{14} y_{24} y_{34} y_{44}
x_{11} x_{21} x_{31} x_{41}
x_{12} x_{22} x_{32} x_{42}
x_{13} x_{23} x_{33} x_{43}
x_{14} x_{24} x_{34} x_{44}
λ_1 λ_4 λ_3 λ_2
λ_3 λ_4 λ_1 OFF
λ_1 λ_2 λ_4 λ_3
λ_3 λ_4 λ_1 λ_2

Figure 4.6: An example illustrating the process of determining values of y_{ij} and assigning wavelengths.

4.5 Conclusion

Large architectures required for implementing NN would require an increasingly larger number of MRRs. In this chapter, a synaptic weighting design was introduced that aims to reduce the number of MRRs. Compared with the work in [45], this architecture reduces the number of required MRRs by half. The weights of a NN model are not directly mappable to these synapses. The attenuation on each PCM-cell needs to be derived from the original weights of the NN model. To this end, a flow for assigning wavelengths to the inputs and finding the desired attenuation for each waveguide with PCM-cell was introduced.

Chapter 5

Results and Discussion

In this chapter, the experiments and results based on the proposed design methods in chapters 3 and 4 are presented. In section 5.1, the weight partitioning method introduced in Chapter 3 is applied to a pattern recognition task. In section 5.2, we study the impact of weight partitioning and unrolling on the accuracy of the model and hardware utilization using a fully connected model for image classification. In section 5.3, we validate the synaptic weighting hardware proposed in chapter 4 using simulation. Finally, in section 5.4, we compared this new synaptic weighting hardware with the baseline design.

5.1 Pattern Recognition with Weight Partitioning

We consider a fully connected layer with 12 inputs and 4 outputs used to memorize four patterns, as illustrated in Fig. 5.1. Each input pattern is composed of 4×3 black and white pixels. To illustrate the method, we consider from 2 to 12 neurons per cluster in the architecture. Ten levels of PCM-cell weighting were used for all scenarios. Table 5.1 reports, for each case, the mapping results we obtain in terms of Active Neuron count, Neuron Cluster count, number of partitions, and number of Recalled Patterns. The Active Neuron count corresponds to the number of active input probes required for generating the spiking output of neurons. Cluster count is an indicator of energy efficiency since only used clusters require laser emission and ring calibration. The number of partitions is an indicator for both accuracy and computing latency since the more partitions, the more neurons and PCM synapses to be crossed. Finally, the number of patterns successfully recalled allows for finding the minimum number of neurons per cluster required to map an application.

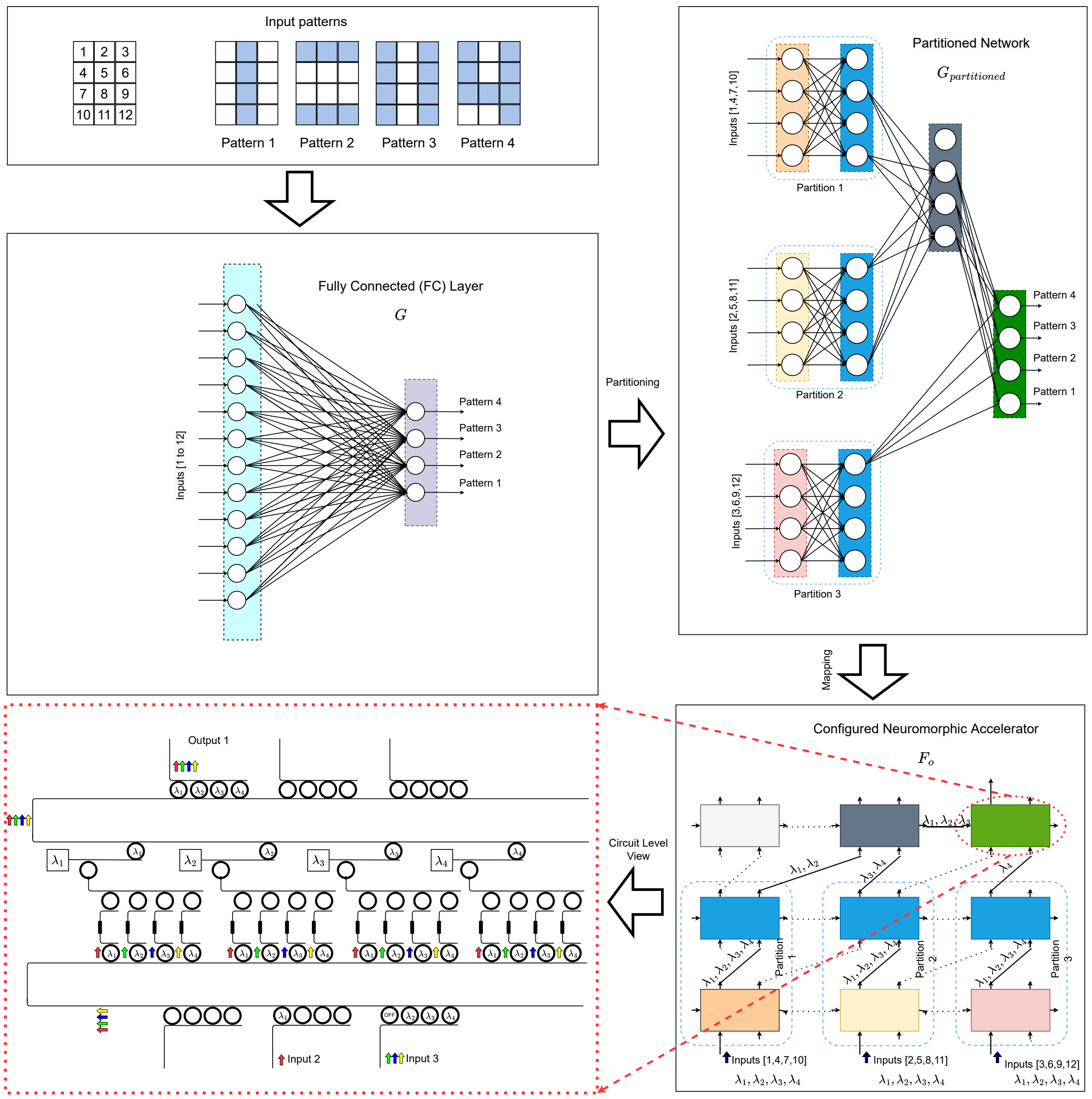


Figure 5.1: Demonstration of pattern recognition example. 12-4 FC layer is trained to memorize four different patterns. This layer is partitioned with $M = 4$ to be mapped to target hardware with 4 neurons per cluster and 4 synapses per neuron. The partitions are mapped to the neuron clusters with matching colors. Each of the indicated neuron clusters is configured to enable interconnectivity. The dotted lines indicate unused connections. The circuit-level view of the indicated output cluster is demonstrated. The inputs are routed by configuring the MRRs in the collector and distributor.

Table 5.1: Mapping Result for Pattern Recognition Example

M	**Neurons**	**Clusters**	**Partitions**	**Recalled**
2	44	32	12	1
3	40	20	8	2
4	24	8	3	3
6	24	5	2	4
8	24	5	2	4
10	24	5	2	4
12	16	2	1	4

For $M = 12$, the source neural network can be directly mapped on a single cluster. The four patterns can be recalled since all synaptic connections are implemented. Reducing M implies removing some synaptic connections but, in some cases, this can be achieved without impacting the network performance. Indeed, since inputs pixels are not of equal significance (e.g. pixels 2, 5, 8, and 11 can be used to reliably distinguish between the four patterns), the significance of the presence (or absence) of key pixels correlates to the magnitude of the synaptic weight to the output neurons. This allows recalling all patterns for 6, 8, and 10 neurons per cluster. For $M = 3$, the number of output neurons exceeds the number of neurons per cluster, which calls for a significant increase in partitions (8) and neuron clusters (20). In addition to expected energy and latency overhead, one of the patterns can not be recalled. For $M = 2$, each neuron only receives 2 fan-in connections which result in only one pattern to be recalled.

This study demonstrates the ability of our method to i) efficiently map neurons on clusters and ii) explore the design space. It allows for maximizing energy efficiency by reducing neuron count overhead and MRR calibration requirements while reaching the targeted accuracy.

5.2 Image Classification with Weight Partitioning

5.2.1 Evaluation Method

We evaluated our method using Keras framework[68]. We used KerasSpiking to convert the model to a spiking model. We approximated the behavior of the neurons using rectified linear unit (ReLU) activation function. ReLU function closely resembles the behavior of the activation unit demonstrated in [45]. We consider a fully connected multi-layer perceptron with $784 - 100 - 10$ neurons for digit recognition on MNIST [73] data set with $60,000$ training samples and $10,000$ testing samples. The baseline accuracy for this network was 88.53%. The goal of the evaluation is to estimate the required size of the cluster N for weight partitioning while avoiding great losses in accuracy performance. This allows exploring the design space, here the number of neurons per cluster, in order to find the best mapping and possibly identify the limit of the hardware.

5.2.2 Discussions

We evaluated the accuracy loss of the network and neuron count overhead against the baseline over a range of a number of neurons per cluster (N). The results are presented in Fig. 5.2. The accuracy of the baseline is obtained for a number of neurons per cluster equal to the number of inputs in the data set ($N = 784$). This also corresponds to the number of wavelengths in the optical neuromorphic accelerator, which is not realistic since dense WDM can typically accommodate up to 80 wavelengths. Decreasing the number of wavelengths in the accelerator is thus required, which leads to partitioning and hence neuron count overhead. It is worth noting that the partitioning does not immediately affect the accuracy performance. This is possible since the number of synaptic connections from the neurons in the input layer to the neurons in the hidden layer does not exceed N in these instances. From $N = 98$, the hidden layer with 100 neurons can no longer be mapped onto a single cluster, which induces additional partitions. From 56 neurons per cluster, the neuron count overhead would reach 145% for an accuracy loss of 47% for the weight partitioning approach.

As expected, energy consumption is correlated to the number of active neurons. The energy consumption is estimated by using the experimental values provided by [45] and extrapolated to the number of active neurons in considered models. Unrolling maintains every connectivity found in

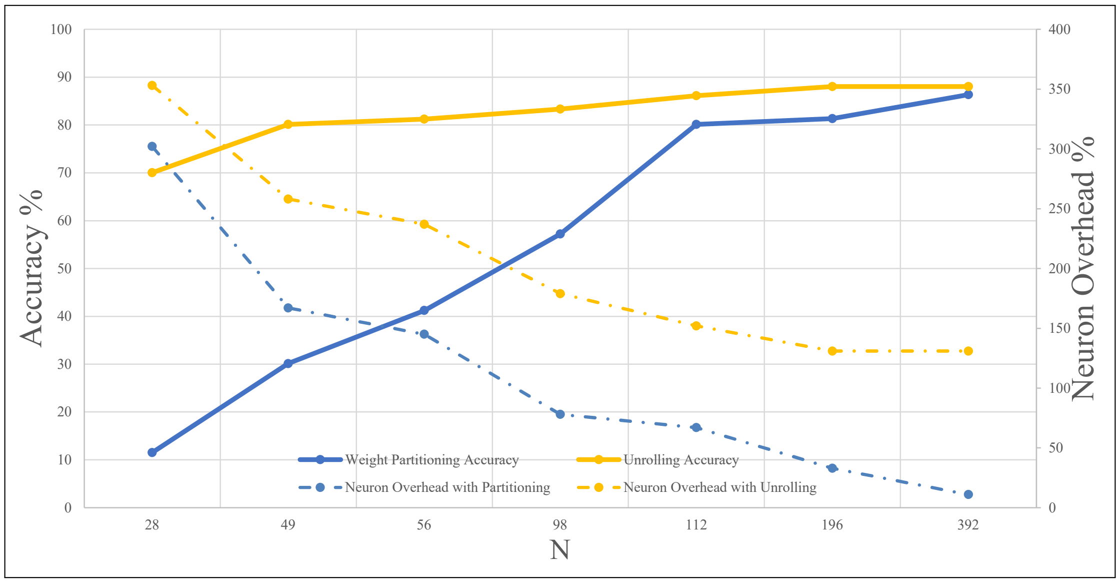


Figure 5.2: Comparison of Weight Partitioning and Unrolling methods for image classification on MNIST.

the original model. To map the target hardware requires separate instances of the inputs and output neurons to be presented in the hardware since it is not trivial to transmit the inputs and outputs to and from the neurons freely to any other neuron. This means that each of the 100 neurons in the hidden layer is required to be unrolled and mapped individually. This results in significant neuron count overhead (237% for $N = 56$).

In contrast, the weight partitioning method aims to create compact models but that can lead to significant losses in accuracy performance. Hence, while unrolling only suffers a 7.3% loss in accuracy, it reaches 47.3% for weight partitioning. However, for realistic implementations of WDM multiplexers of $N = 56$, the continuous energy consumption is estimated to be 1677 nJ per cycle, compared with 3973 nJ per cycle for unrolling, which corresponds to a significant 58% reduction in energy consumption.

This presents the ability of the proposed flow to investigate different mapping strategies with the aim of exploring accuracy and energy trade-off. We showed that the proposed method allows for exploring key design trade-offs involving accuracy and energy consumption. In our future work, we plan to evaluate the latency overhead induced by additional network layers from unrolling, which will call for transient simulations. We also plan to investigate the power overhead induced by the calibration of MRRs and the programming of PCM-based synapses. Considering the significantly different results we obtain with unrolling and partitioning, two main research directions will be explored. First, at the algorithmic level, we plan to combine both partitionings and unrolling in order to satisfy accuracy constraints while minimizing power consumption. Second, at the architecture level, we will explore the use of multistage network topologies, which are regular and could ideally map optical SNN.

5.3 Simulation and Validation of Synaptic Weighting Architecture

The architecture proposed in chapter 4 was simulated inside Synopsys OptSim with four inputs according to Fig. 5.3. OptSim is a software tool for the design and simulation of optical systems at the signal propagation level. The goal is to validate the design and its capability to perform synaptic weighting. This schematic implements the synaptic weighting with four laser inputs. Each laser is modulated with a pseudo-random binary sequence (PRBS). These four inputs are then under WDM multiplexing to replicate the WDM input to the synapses. For MRRs are used to de-multiplex the WDM input and extract the individual signals from the WDM input. To this end, the wavelength of each laser source needs to be adjusted to match the wavelength where each corresponding MRR has the maximum transmission.

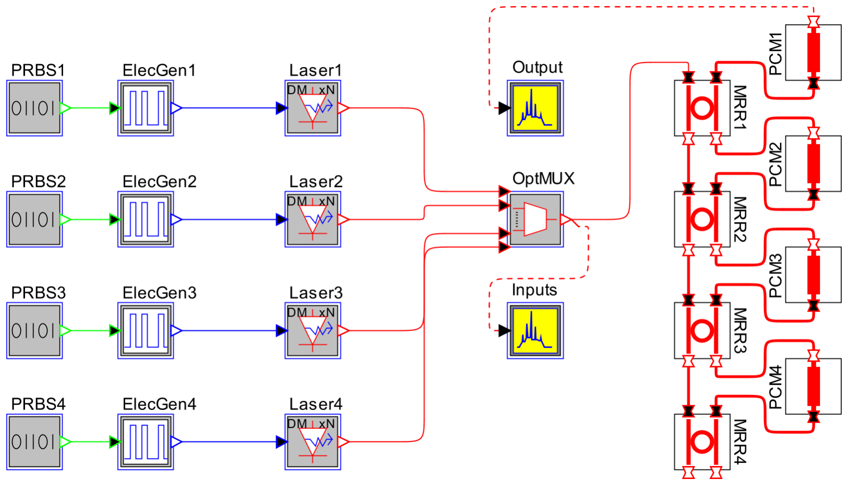


Figure 5.3: Schematic of the circuit used for design validation with OptSim.

Table 5.2: Parameters for OptSim Schematic.

Microring#	Length (μm)	Laser Source	$\lambda(nm)$
MRR1	21	Laser1	1575
MRR2	24	Laser2	1565
MRR3	27	Laser3	1557
MRR4	30	Laser4	1551

The wavelength spectrum of the four MRRs in the schematic is shown in Fig. 5.4. The MRRs are designed to reduce overlapping spectrum. This allows the maximization of the coupling of the input signals to the desired target MRR and minimizes the coupling of the adjacent signals.

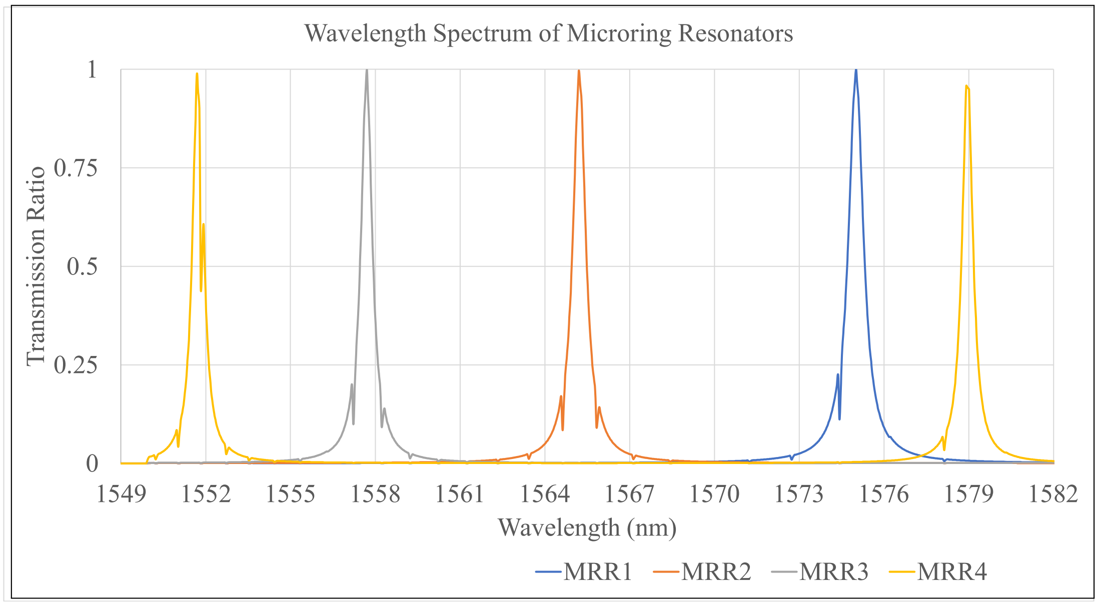
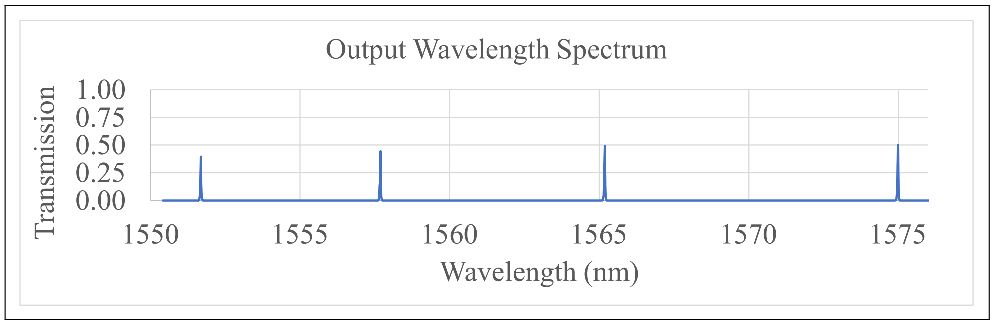


Figure 5.4: Wavelength spectrum of the four MRRs.

The wavelength of the four laser sources is adjusted to correspond with the maximum transmission through each of the four MRRs. The parameters of these components are presented in Table 5.2. Four waveguides (PCM1-4) are placed on the signal path between the MRRs. These waveguides represent the attenuation caused by PCM-cells to produce the synaptic weighting on the input signals.

The schematic was tested with three sets of target weights. The weights are values in $[0, 1]$ and can be considered as transmission ratios for each of the four inputs to the schematic. These weights are converted to losses in dB to better illustrate their impact on the inputs. For example, the weight of 1 implies that the corresponding input should reach the output without any losses. In contrast, the weight of 0.5, suggests $3dB$ of attenuation is desired.

First, we test equal weighting according to Fig. 5.5. To test this scenario, we assume four equal weights of 0.5. This is a scenario where all four inputs should be subjected to the same levels of attenuation and therefore resulting in identical signal power at each wavelength in the output. This means that we desire to achieve a transmission ratio of 0.5 or 3dB of attenuation on each individual input signal. By setting the insertion loss of PCM1 to 3dB, we achieve this goal on input 1. Since inputs 2, 3, and 4 will also be subjected to attenuation by PCM1, we configure PCM2-to-4 to a state with minimal losses. The values for the attenuation caused by PCM synapses are extrapolated based on the work in [72] that was presented in chapter 4. The attenuation in waveguides PCM1 to PCM4 is adjusted according to the values in the $Att.$ column to implement the target weights. In this way, the accumulated losses for each input should be equal to the corresponding target weight. We can measure the output power spectrum at the output port on the schematic in Fig. 5.5. The observed transmission is then compared with the target weights to calculate errors. We can observe that as we move down to the subsequent inputs, the weighting error increases. The input signal has to travel through additional PCM synapses and MRRs before reaching output.

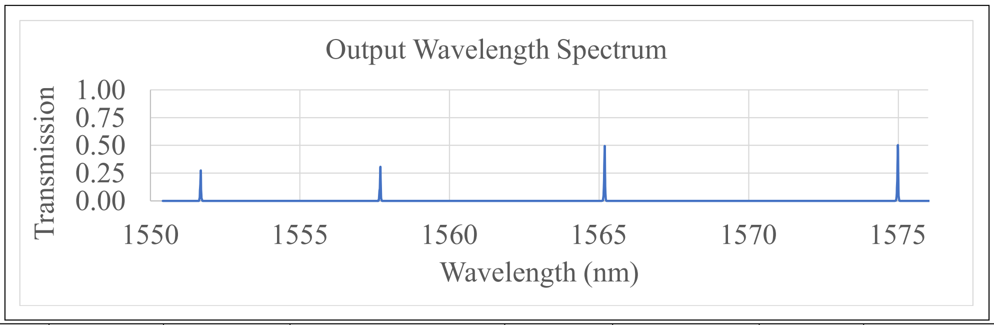


Input#	$\lambda(nm)$	Weights	Target Att.(dB)	PCM#	Att. (dB)	Trans.	$\pm$ Error (Δe)
1	1575	0.5	3.01	PCM1	2.91	0.50	0.00
2	1565	0.5	3.01	PCM2	0.30	0.49	0.01
3	1557	0.5	3.01	PCM3	0.30	0.44	0.06
4	1551	0.5	3.01	PCM4	0.30	0.40	0.10

Figure 5.5: Test with identical weights for all inputs.

Next we assume that $w_1 = w_2 = 0.5$ and $w_3 = w_4 = 0.33$, according to Fig. 5.6. The goal is to further observe the impact of this design on inputs placed farther from the output neuron. These inputs are subjected to higher attenuation. We can determine that inputs 1 and 2 are to be subjected to 3dB losses while inputs 3 and 4 are to be subjected to 4.81 dB. Similar to the previous test, we can satisfy this condition for inputs 1 and 2 by configuring PCM1 to 3dB of attenuation while adjusting PCM2 for maximum transmission. Since input 3 will be subjected to 3dB of attenuation by PCM1, we need to satisfy the remaining 1.81dB by configuring PCM3 to this level of attenuation. In the end, since input 4 will be subjected to similar losses as input 3, we only have to configure PCM4 to maximum transmission.

The output power spectrum is illustrated in Fig. 5.6. As expected, we can observe that the weighting error increases the farther we move toward inputs farther from the output but overall the

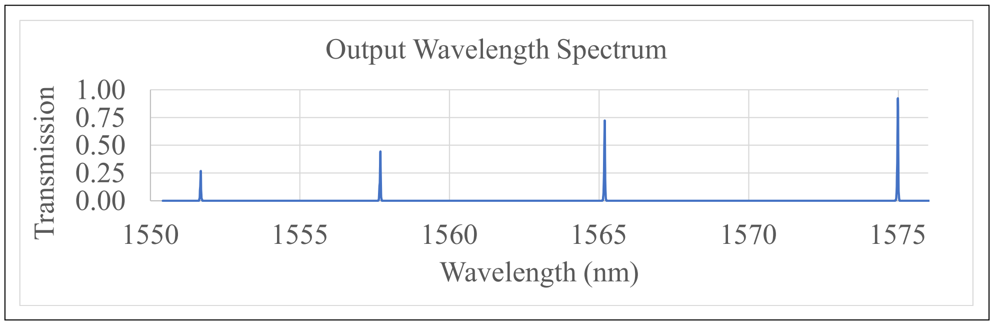


Input#	$\lambda(nm)$	Weights	Target Att.(dB)	PCM#	Att. (dB)	Trans.	$\pm$ Error (Δe)
1	1575	0.5	3.01	PCM1	2.91	0.50	0.00
2	1565	0.5	3.01	PCM2	0.30	0.49	0.01
3	1557	0.33	4.81	PCM3	1.88	0.30	0.03
4	1551	0.33	4.81	PCM4	0.30	0.28	0.05

Figure 5.6: Test with $w_1 = w_2 = 0.5$ and $w_3 = w_4 = 0.33$

desired weighting and relative ratios between the four inputs are achieved.

Finally, we test the scenario where each weight is unique. We assume input 1 is weighted by 1 where it is intended to reach the output with the maximum transmission. Inputs 2, 3, and 4 are weighted by 0.75, 0.5, and 0.33 respectively. The weights are sorted in descending order and assigned to inputs in this order.



Input#	$\lambda(nm)$	Weights	Target Att.(dB)	PCM#	Att. (dB)	Trans.	$\pm$ Error (Δe)
1	1575	1	0	PCM1	0.30	0.92	0.08
2	1565	0.75	1.25	PCM2	1.30	0.72	0.03
3	1557	0.5	3.01	PCM3	1.89	0.44	0.06
4	1551	0.33	4.81	PCM4	1.89	0.17	0.18

Figure 5.7: Test with unique weights for each input.

In this way, the weakest weight, or the highest attenuation, will be assigned to lower inputs. We configure PCM1 for maximum transmission to satisfy the weighting for input 1. PCM2 is then configured to provide approximately 1.25dB of attenuation to satisfy the weighting for input 2. To satisfy 3dB of attenuation for input 3, we configure PCM3 to 1.75 dB. In this way, together with PCM2, they provide 3dB of attenuation to input 3. Finally, by configuring PCM4 to 1.81dB attenuation, we satisfy 4.81dB attenuation for input 4. The output power spectrum is illustrated in Fig. 5.7. We can observe that the corresponding transmission for each wavelength and as a result the desired weighting was achieved.

This simulation demonstrates the ability of this design to perform synaptic weighting on the inputs while utilizing half as many MRRs for this function. This saving comes at the expense of increased weighting errors which calls for a comparison between the performance of the two approaches. In the following section, we attempt to compare both architectures on larger models to study the impact of the weighting error on model accuracy.

5.4 Comparison of Synaptic Weighting Architecture with Baseline

In the previous section, we demonstrated the ability of the proposed architecture to perform the weighting functions with simulation. We observed an error in the weights after mapping to the hardware which increases as we look toward weights that are placed farther from the output. This necessity a comparison with the baseline architecture to identify the potential losses in the accuracy of the neural network when processing a large model such as the one we used for the image classification task in section 5.2.

The model used for this comparison is similar to the one used in section 5.2. We evaluated our method using Keras framework[68]. We used KerasSpiking to convert the model to a spiking model. We approximated the behavior of the neurons using rectified linear unit (ReLU) activation function. ReLU function closely resembles the behavior of the activation unit demonstrated in [45]. We consider a fully connected multi-layer perceptron with $784-100-10$ neurons for digit recognition on MNIST [73] data set with $60,000$ training samples and $10,000$ testing samples. The samples were pre-processed to a binary format to reduce the sensitivity of the network to the precision of weights when mapping the model to target hardware with limited weight precision.

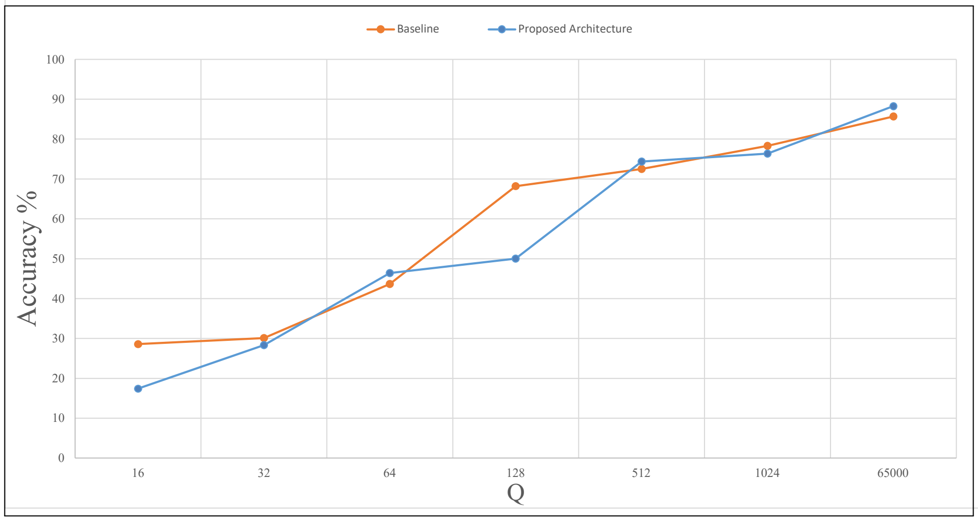


Figure 5.8: Comparison of proposed architecture and baseline with different values of Q.

The goal of the comparison is to estimate the losses in accuracy in the proposed method compared with the baseline photonic synapse architecture in [72]. We introduced a parameter Q. Q is the number of different weightings that each PCM synapse on the waveguide can realize. For example, a PCM synapse with $Q = 8$ can be configured to provide eight unique weights. The quantization process follows the method introduced in section 4.3 and the resulting weights are sampled based on the number for parameter Q. We apply this quantization to both synaptic architecture and compared the accuracy of the resulting model against parameter Q.

We evaluated the accuracy of the network for both photonic synaptic architectures over a range of parameters Q. The results of the comparison are presented in Fig. 5.8.

For large values of Q ($Q = 65000$), the accuracy between the two architectures is comparable. A large number of samples allow for representing the weights with enough precision that it will not lead to notable losses in accuracy.

As we reduce the parameter Q, we can observe a gradual loss in accuracy but the two architectures remain comparable. For $Q = 128$, the proposed architecture results in 19% lower accuracy compared with the baseline. For smaller values of Q ($Q < 64$), the two architectures converge to comparable accuracy. Overall, the two architectures can perform synaptic weighting on this task with comparable accuracy despite the proposed architecture requiring half as many MRRs.

5.5 Conclusion

In this chapter, we provided various experiments and tests to evaluate the methods presented in chapters 3 and 4. We began by providing a pattern recognition experiment to illustrate the weight partitioning method and its impact on the performance of the model. To evaluate the impact of the unrolling and weight partitioning methods on larger models, we used a feed-forward model for image classification on the MNIST dataset and compared the resulting accuracy and hardware overhead to arrive at a trade-off between model accuracy and energy consumption on the hardware.

We simulated the proposed synaptic weighting architecture in OptSim to validate its ability to perform synaptic weighting. The architecture with one output neuron and four inputs were tested. The error for transmission ratio was calculated to provide a comparison with the baseline. Finally, we evaluated the effect of the quantization resulting from PCM synapses on a large model for image classification. We compared the accuracy against the parameter Q. This parameter represents the different number of possible weights after quantization. The result of the test suggests that the proposed method provides comparable accuracy to the baseline over a range of parameter Q.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The thesis aimed to explore potential design methods that realize the mapping of larger neural network models on photonic spiking neural network hardware architectures. These large models can contain thousands of neurons and synapses per neuron that need to be individually mapped onto the target neural network hardware. Therefore, the size of models that can be mapped is limited by the number of neurons and synapses that the hardware can accommodate. Photonic architectures rely on WDM techniques to transmit signals to and from individual neurons. This calls for the addition of microring resonators that are individually calibrated to specific wavelengths. This becomes more challenging and unfeasible with the increased number of synapse neurons.

Unrolling and weight partitioning methods aim to map the neural network model to the hardware with a reduction in the number of required synapses per neuron. These methods reshape the neural network models so that can be they mapped onto more restricted hardware at the expense of model accuracy.

In this work, these methods were adapted to an all-optical spiking neural network architecture. Neuron clusters were introduced to facilitate the weight partitioning method. The larger neural network models are then distributed over these clusters that communicate through photonic interconnects. The two methods were evaluated on an image classification test. In this experiment, the resulting model accuracy with each method was compared when considering a varying number of synapses per neuron. We conclude that for a larger number of synapses, the weight partitioning method results in models with comparable accuracy to unrolling while demanding up to 300% fewer hardware resources. In contrast, when the number of synapses is severely restricted, the unrolling method can maintain accuracy. This allows us to explore the design space by modifying the number of synapses per neuron. We proposed a new synaptic weighting architecture that can perform a weighting function with half as many microring resonators compared with the architecture in [45]. The proposed design with four input sources and one neuron was simulated with OptSim to validate its capability to perform synaptic weighting.

The proposed architecture takes advantage of serialized weights to reduce the number of micro rings. This results in PCM weights that cannot be directly mapped to a neural network model. This necessitated the creation of a design flow to transform the weights from the original neural network model to a format that can be mapped onto the proposed architecture. In order to better represent the PCM weight, a quantization method based on experimental data in [72] was proposed. This allows us to model and represent synaptic weights more closely to the accuracy of the physical PCM synapses. Mapping to the proposed method results in a larger quantization error compared with the original architecture. We expected this would lead to lower model accuracy and it was necessary to evaluate its impact when dealing with many input sources.

To evaluate the architecture for this end, we used an image classification task. The model was mapped onto the original and the proposed architectures and the resulting model accuracy was compared. The goal of this comparison was to explore the design space and limitations of PCM synapses in representing the values of weights. The PCM synapses can represent a limited number of weighting values. The work in [72] has experimentally demonstrated 15 levels of weighting using PCM synapses which is significantly lower than what is required to achieve an accuracy of better than 50% in this application. This allows us to create a design goal for improving the PCM synapses and their capabilities to a level where they will be able to perform more sophisticated neural network applications. The proposed methods for unrolling and weight partitioning together with the proposed synaptic weighting architecture allow us to realize a path toward large-scale photonic spiking neural networks and work around the physical restrictions and limitations inherent to existing photonic architectures.

6.2 Future Work

In this work, we considered each layer in the neural network model independently and applied the weight partitioning and unrolling methods without considering their impact on the overall topology of the network. As an extension of this work, we can analyze the connectivity between the layers in the overall network. In this approach, we can identify paths from inputs to outputs with higher connectivity and temporal spiking activity. These paths may contribute more to the overall neural function of the model and the resulting accuracy compared. Prioritizing the preservation of these paths on the final model could lead to improved overall model accuracy at a lower cost in hardware resources and energy consumption.

We compared applying unrolling and weight partitioning methods to the entire neural network model. One approach to achieve improved trade-offs is the combined use of weight partitioning and unrolling methods at the same time. In this approach, we can identify which method results in better accuracy – energy trade-offs in each hidden layer and apply unrolling or weight partitioning methods accordingly.

One of the limitations of this work is that we did not consider the impact of latency in the network after unrolling and partitioning. We also did not consider the accuracy of the PCM weights and their impact on weight partitioning and unrolling methods. These are aspects that we could consider in future comparisons and design methods. The latency can be added to the model for the spiking neurons and interconnects. The latency will impact the spike timing in encoded inputs and neurons and this impact can be studied as the network is scaled to larger sizes. The PCM synapses can represent weights with limited precision. We provided a quantization method to approximate the limited precision of PCM synapses in chapter 4. This method could be expanded and applied to the weight partitioning and unrolling methods described in chapter 3.

We prioritized using pre-trained neural network models as the input to our design flow and avoided re-training which is computationally expensive and time-consuming. One of the limitations that we identified was the different levels of weighting possible with PCM-synapses and how the resulting errors in representing weights impact the accuracy. We can potentially address this limitation by considering the characteristics of the PCM synapses during the training process. Quantization-aware training methods can be used to compensate for the limitations of the synaptic weights during the training process and improve model accuracy.

Fabrication imperfections on photonic neural networks lead to imperfect routing and non-ideal nonlinearities. These imperfections in turn impact the performance of the hardware. It will be necessary to take these factors into account and potentially correct them during training or with other potential mitigation measures which can be explored in future works.

Bibliography

[1] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc Natl Acad Sci U S A*, 79, 8:2554–2558, 1982.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521, 7553:436–444, 2015.

[3] Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. *Elements of Artificial Neural Networks*. MIT Press, Cambridge, MA, USA, 1996.

[4] Don Monroe. Neuromorphic computing gets ready for the (really) big time. commun. *ACM*, 57, 6:13–15, 2014.

[5] Steve Furber. Large-scale neuromorphic computing systems. *J. Neural Eng*, 13, 5:051001, 2016.

[6] Giacomo Indiveri and Timothy K. Horiuchi. Frontiers in neuromorphic engineering. *Front. Neurosci*, 5, 2011.

[7] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575, 7784:607–617, 2019.

[8] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Front. Neurosci*, 12, 2018.

[9] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *Int J Comput Vis*, 113, 1:54–66, 2015.

[10] Priyadarshini Panda and Kaushik Roy. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. In *2016 International Joint Conference on Neural Networks (IJCNN*, page 299–306, 2016.

[11] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345, 6197:668–673, 2014.

[12] Paul R. Prucnal and Bhavin J. Shastri. *Neuromorphic Photonics*. CRC Press, 2017.

[13] J.W. Goodman, A.R. Dias, and L.M. Woody. Fully parallel, high-speed incoherent optical method for performing discrete fourier transforms. *Opt. Lett., OL*, 2, 1:1–3, 1978.

[14] Alexander N. Tait, Mitchell A. Nahmias, Bhavin J. Shastri, and Paul R. Prucnal. Broadcast and weight: An integrated network for scalable photonic spike processing. *Journal of Lightwave Technology*, 32, 21:4029–4041, 2014.

[15] Alexander N. Tait, Philip Y. Ma, Thomas Ferreira Lima, Eric C. Blow, Matthew P. Chang, Mitchell A. Nahmias, Bhavin J. Shastri, and Paul R. Prucnal. Demonstration of multivariate photonics: Blind dimensionality reduction with integrated photonics. *Journal of Lightwave Technology*, 37, 24:5996–6006, 2019.

[16] Tyler W. Hughes, Momchil Minkov, Yu Shi, and Shanhui Fan. Training of photonic neural networks through in situ backpropagation and gradient measurement. *Optica*, 5, 7:864, 2018.

[17] Mitchell A. Nahmias, Bhavin J. Shastri, Alexander N. Tait, and Paul R. Prucnal. A leaky integrate-and-fire laser neuron for ultrafast cognitive computing. *IEEE J. Select. Topics Quantum Electron*, 19, 5:1–12, 2013.

[18] Indranil Chakraborty, Gobinda Saha, Abhronil Sengupta, and Kaushik Roy. Toward fast neural computing using all-photonic phase change spiking neurons. *Scientific Reports*, 8, 1:12980, 2018.

[19] B. J. Shastri. Silicon photonics for machine learning and neuromorphic computing. *Optical/Photonic Interconnects for Computing Systems*, 2022.

[20] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons, 2015.

[21] Jithendar Anumula, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. Feature representations for neuromorphic audio spike streams. *Frontiers in Neuroscience*, 12, 2018.

[22] Joseph M. Brader, Walter Senn, and Stefano Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput*, 19, 11:2881–2912, 2007.

[23] Peter U. Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci*, 9, 2015.

[24] Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, Charlie Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *Science*, 338:6111, 2012.

[25] Emre Neftci, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Front. Neurosci*, 7, 2014.

[26] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. california univ san diego la jolla inst, 1985.

[27] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472, 1998.

[28] Filip Ponulak and Andrzej Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta Neurobiologiae Experimentalis*, 71, 4:409–433, 2011.

[29] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN*, page 1–8, 2015.

[30] Martino Sorbaro, Qian Liu, Massimo Bortone, and Sadique Sheik. Optimizing the energy consumption of spiking neural networks for neuromorphic applications. *Front. Neurosci*, 14, 2020.

[31] S. Thorpe, A. Delorme, and R.Van Rullen. Spike-based strategies for rapid processing. *Neural Netw*, 14:6–7, 2001.

[32] David Heeger. Poisson model of spike generation, 2000.

[33] Alexander N. Tait, Mitchell A. Nahmias, Yue Tian, Bhavin J. Shastri, and Paul R. Prucnal. Photonic neuromorphic signal processing and computing. In Makoto Naruse, editor, *Nanophotonic Information Physics: Nanointelligence and Nanophotonic Computing*, page 183–222. Springer, Berlin, Heidelberg, 2014.

[34] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Bltn Mathcal Biology*, 52, 1:25–71, 1990.

[35] L.F. Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907. *Brain Research Bulletin*, 50, 5:303–304, 1999.

[36] Roman M. Borisyuk and Galina N. Borisyuk. Information coding on the basis of synchronization of neuronal activity. *Biosystems*, 40, 1:3–10, 1997.

[37] Abigail Morrison, Ad Aertsen, and Markus Diesmann. Spike-timing-dependent plasticity in balanced random networks. *Neural Comput.*, 19(6):1437–1467, June 2007.

[38] Wim Bogaerts, Martin Fiers, and Pieter Dumon. Design challenges in silicon photonics. *IEEE Journal of Selected Topics in Quantum Electronics*, 20, 4:1–8, 2014.

[39] Alexander N. Tait, Hasitha Jayatilleka, Thomas Ferreira De Lima, Philip Y. Ma, Mitchell A. Nahmias, Bhavin J. Shastri, Sudip Shekhar, Lukas Chrostowski, and Paul R. Prucnal. Feedback control for microring weight banks. *Opt. Express, OE*, 26, 20:26422–26443, 2018.

[40] A. N. Tait, T. Ferreira de Lima, A. X. Wu, E. Zhou, M. A. Nahmias, B. J. Shastri, M. P. Chang, and P. R. Prucnal. Silicon microring weight banks for multivariate rf photonics. In *2016 IEEE Photonics Conference (IPC)*, pages 11–12, 2016.

[41] Alexander N. Tait, Thomas Ferreira de Lima, Mitchell A. Nahmias, Bhavin J. Shastri, and Paul R. Prucnal. Multi-channel control for microring weight banks. *Opt. Express*, 24(8):8895–8906, Apr 2016.

[42] Alexander N. Tait, Allie X. Wu, Thomas Ferreira Lima, Ellen Zhou, Bhavin J. Shastri, Mitchell A. Nahmias, and Paul R. Prucnal. Microring weight banks. *IEEE Journal of Selected Topics in Quantum Electronics*, 22, 6:312–325, 2016.

[43] Zengguang Cheng, Carlos Ríos, Wolfram H.P. Pernice, C.David Wright, and Harish Bhaskaran. On-chip photonic synapse. *Science Advances*, 3, 9:1700160, 2017.

[44] Carlos Ríos, Matthias Stegmaier, Peiman Hosseini, Di Wang, C.David Wright Torsten Scherer, Harish Bhaskaran, and Wolfram H.P. Pernice. Integrated all-photonic non-volatile multi-level memory. *Nature Photon*, 9, 11:725–732, 2015.

[45] J. Feldmann, N. Youngblood, C.D. Wright, H. Bhaskaran, and W.H.P. Pernice. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature*, 569, 7755:208–214, 2019.

[46] Joshua Robertson, Yahui Zhang, Yahui Zhang, Matěj Hejda, Julián Bueno, Shuiying Xiang, and Antonio Hurtado. Image edge detection with a photonic spiking vcsel-neuron. *Opt. Express, OE*, 28, 25:37526–37537, 2020.

[47] Johan L.A. Dubbeldam and Bernd Krauskopf. Self-pulsations of lasers with saturable absorber: dynamics and bifurcations. *Optics Communications*, 159, 4:325–338, 1999.

[48] Shuiying Xiang, Yahui Zhang, Junkai Gong, Xingxing Guo, Lin Lin, and Yue Hao. Stdp-based unsupervised spike pattern learning in a photonic spiking neural network with vcsels and vcsoas. *IEEE Journal of Selected Topics in Quantum Electronics*, 25, 6:1–9, 2019.

[49] Shuiying Xiang, Junkai Gong, Yahui Zhang, Xingxing Guo, Yanan Han, Aijun Wen, and Yue Hao. Numerical implementation of wavelength-dependent photonic spike timing dependent plasticity based on vcsoa. *IEEE Journal of Quantum Electronics*, 54, 6:1–7, 2018.

[50] Thomas Ferreira de Lima, Alexander N. Tait, Armin Mehrabian, Mitchell A. Nahmias, Chaoran Huang, Hsuan-Tung Peng, Bicky A. Marquez, Mario Miscuglio, Tarek El-Ghazawi, Volker J. Sorger, Bhavin J. Shastri, and Paul R. Prucnal. Primer on silicon neuromorphic photonic processors: architecture and compiler. *Nanophotonics*, 9(13):4055–4073, 2020.

[51] Lumerical interconnect - pic design and simulation software, 2021.

[52] Santiago G.-C. Carrillo, Emanuele Gemo, Xuan Li, Nathan Youngblood, Andrew Katumba, Peter Bienstman, Wolfram Pernice, Harish Bhaskaran, and C.David Wright. Behavioral modeling of integrated phase-change photonic devices for neuromorphic computing applications. *APL Materials*, 7, 9:091113, 2019.

[53] Bhavin J. Shastri, Mitchell A. Nahmias, Alexander N. Tait, Ben Wu, and Paul R. Prucnal. Simpel: Circuit model for photonic spike processing laser neurons. *Opt. Express, OE*, 23, 6:8029–8044, 2015.

[54] Thomas Ferreira Lima, Hsuan-Tung Peng, Alexander N. Tait, Mitchell A. Nahmias, Heidi B. Miller, Bhavin J. Shastri, and Paul R. Prucnal. Machine learning with neuromorphic photonics. *J. Lightwave Technol., JLT*, 37, 5:1515–1534, 2019.

[55] Sen Lin, Sajjad Moazeni, Krishna T. Settaluri, and Vladimir Stojanovi. Silicon photonics simulink toolbox, 2017.

[56] Sen Lin, Sajjad Moazeni, Krishna T. Settaluri, and Vladimir Stojanović. Electronic–photonic co-optimization of high-speed silicon photonic transmitters. *J. Lightwave Technol., JLT*, 35, 21:4766–4780, 2017.

[57] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7:48, 2014.

[58] Chris Eliasmith and Charles Anderson. *Neural Engineering*. The MIT Press, 2002.

[59] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Front. Neuroinform*, 12, 2018.

[60] Optsim circuit, 2021.

[61] Sunil Pai, Ben Bartlett, Olav Solgaard, and David A.B. Miller. Matrix optimization on universal unitary photonic devices. *Phys. Rev. Applied*, 11, 6:064044, 2019.

[62] Sunil Pai, Ian A.D. Williamson, Tyler W. Hughes, Momchil Minkov, Olav Solgaard, Shanhui Fan, and David A.B. Miller. Parallel fault-tolerant programming of an arbitrary feedforward photonic network, 2019.

[63] Ian A.D. Williamson, Tyler W. Hughes, Momchil Minkov, Ben Bartlett, Sunil Pai, and Shanhui Fan. Reprogrammable electro-optic nonlinear activation functions for optical neural networks. *IEEE Journal of Selected Topics in Quantum Electronics*, 26, 1:1–12, 2020.

[64] J. S. Yang Y. Kang and J. Chung. Weight partitioning for dynamic fixed-point neuromorphic computing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(11):2167–2171, November 2019.

[65] C. Kim, J. A. Abraham, W. Kang, and J. Chung. A neural network decomposition algorithm for mapping on crossbar-based computing systems. *Electronics*, 9:1526, September 2020.

[66] A. Balaji et al. Enabling resource-aware mapping of spiking neural networks via spatial decomposition. *IEEE Embedded Systems Letters*, 13(3):142–145, September 2021.

[67] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, and D. Rasmussen et al. Nengo: A python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7, 2013.

[68] others F. Chollet. Keras. *GitHub. Available for*, 2015.

[69] S. G. C. Carrillo al. Behavioral modeling of integrated phase-change photonic devices for neuromorphic computing applications. *APL Materials*, 7(9):091113, 2019.

[70] B. J. Shastri, M. A. Nahmias, A. N. Tait, B. Wu, and P. R. Prucnal. Simpel: Circuit model for photonic spike processing laser neurons. *Opt. Express*, 23(6):8029–8044, 2015.

[71] Lumerical Inc.

[72] Z. Cheng, C. Rios, W. H. P. Pernice, C. D. Wright, and H. Bhaskaran. On-chip photonic synapse. *Sci. Adv*, 3(9):1–6, 2017.

[73] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, November 2012.