Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2021-5078016

M.Sc. Thesis

Mapping of Spiking Neural Network Topologies on Neuromorphic Hardware

Shreya Sanjeev Kshirasagar

Abstract

As we move towards edge computing, not only low power but concurrently, critical timing is demanded from the underlying hardware platform. Spiking neural networks ensure high performance and low power when run on specialized architectures like neuromorphic hardware. However, the techniques in use to configure these neural networks on massively parallel neuromorphic crossbar arrays remain sparsely explored. This motivates the research on how neural network topologies encompassing spiking architectures can be configured on a neuromorphic hardware. In this thesis, a unique placement algorithm is devised to map diverse and complex neural network architectures on a connectivity-constrained array with thousands of processing elements(PEs) within seconds. Wide spectra of SNNs with varying complexity are investigated to evaluate the feasibility of mapping on the target neuromorphic architecture involving unique connectivity constraints. The performance of the proposed ALAPIN mapper is validated through time-to-solution for the surveyed SNN schemes with varying network sizes and diverse complexity measures. Experiments show that simple networks converge within 10 milliseconds. With limited resources and as the network architectural complexity increases, hardware constraints become overwhelming to achieve placement solution within a decent time frame. Further experiments are carried out to estimate the resource utilization of each candidate SNN for varying network sizes on target hardware. Liquid state machines use a greater number of synapses for a same number of neurons than the rest of the candidates, with approximately 100% neurons, 30% input resources, and 20% synapses on target hardware.

Mapping of Spiking Neural Network Topologies on Neuromorphic Hardware

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

by

Shreya Sanjeev Kshirasagar
born in Gulbarga, India

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **“Mapping of Spiking Neural Network Topologies on Neuromorphic Hardware”** by **Shreya Sanjeev Kshirasagar** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 24th August 2021

Chairman: prof.dr.ir. T.G.R.M. van Leuken

Advisor: dr.ir. Sumeet Kumar

Committee Members: prof.dr.ir. Stephan Wong

Abstract

As we move towards edge computing, not only low power but concurrently, critical timing is demanded from the underlying hardware platform. Spiking neural networks ensure high performance and low power when run on specialized architectures like neuromorphic hardware. However, the techniques in use to configure these neural networks on massively parallel neuromorphic crossbar arrays remain sparsely explored. This motivates the research on how neural network topologies encompassing spiking architectures can be configured on a neuromorphic hardware. In this thesis, a unique placement algorithm is devised to map diverse and complex neural network architectures on a connectivity-constrained array with thousands of processing elements(PEs) within seconds. Wide spectra of SNNs with varying complexity are investigated to evaluate the feasibility of mapping on the target neuromorphic architecture involving unique connectivity constraints. The performance of the proposed ALAPIN mapper is validated through time-to-solution for the surveyed SNN schemes with varying network sizes and diverse complexity measures. Experiments show that simple networks converge within 10 milliseconds. With limited resources and as the network architectural complexity increases, hardware constraints become overwhelming to achieve placement solution within a decent time frame. Further experiments are carried out to estimate the resource utilization of each candidate SNN for varying network sizes on target hardware. Liquid state machines use a greater number of synapses for a same number of neurons than the rest of the candidates, with approximately 100% neurons, 30% input resources, and 20% synapses on target hardware.

Acknowledgments

From the day I decided to send out an application to TU Delft, I was influenced by Rene's line of research and had a strong urge to work under his supervision. I am forever indebted to my supervisor Prof. Dr. Rene van Leuken for giving me the opportunity to work on a novel and multi-disciplinary topic.

I wish to express my deepest gratitude to Dr. Sumeet Kumar. It has been a sheer pleasure to work under his guidance during the entirety of this thesis work. I had the unique privilege to have access to Sumeet's professional expertise and extremely visionary advice. Sumeet's research foresight, technical depth and devotion to students is a valuable treasure for me. His attention towards detail has been influential during the past ten months. I would like to express my heartfelt gratitude to Prof. Dr. Stephan Wong for agreeing to be a part of thesis committee.

I would also thank my co-advisor and mentor Dr. George Vathakatil Joseph. During our various meetings and discussions, his feedback has helped me think clearly when I was stuck with challenges and motivated me to delve deeper. Although I met Dr. Anushree Mahapatra almost during the final months of my thesis, her EDA expertise and technical discussions were extremely invaluable. It has been a wonderful opportunity to collaborate with all the colleagues at Innatera, especially, Dr. Amir Zjajo, Katarzyna, Darek, Aditya, Charles, Petrutz, and Alba. Thank you for your guidance and a cheerful environment that kept me going during bad days.

To the most dynamic and level-headed personality, my late grandmother, I am largely indebted for having sown the seed of education in our family. Continuing your legacy, I promise to let no future generations starve from education. I give my sincere thanks to my Mom and Dad. I could not have achieved any of this without your sacrifices and support throughout the journey.

I would like to thank my friends and housemates who have been supportive throughout, especially Avinash for many technical discussions we had over the last year. I would like to thank my friends from Master's course without whom life would have been dull in Delft for me. I would like to thank my fellow master students Roy, Sybold, and Pretha, who have been along the journey of thesis.

I have to thank Shreyas for showing love and support throughout this journey, with big doses of patience. Its restoration of strength and life in your presence. Finally, to everyone who has encouraged and been there for me during this journey, thank you from the bottom of my heart.

Shreya Sanjeev Kshirasagar
Delft, The Netherlands
24th August 2021

Contents

List of Figures

List of Tables

Introduction 1

1.1 Problem description

Spiking neural networks (SNNs) are the 3rd generation of artificial neural networks (ANNs) that employ spiking neurons as compute units. Lately, SNNs have received immense attention due to their ability to closely replicate the biological neural behaviour. The rich temporal pattern in spiking neurons, mimicking the brain, is what makes a spiking neural network best suited for time-series analysis which is much needed for sensory data processing. The applications of SNNs are vividly large. Applications like pattern recognition, pattern classification, involving RADAR, audio, or LiDAR sensor data are heavily benefited with the deployment of SNNs at the sensor edge. This necessitates the use of low-power embedded hardware to deploy AI at the sensor edge. The conventional hardware architectures like Von Neumann do not cater well to the behaviour of spiking neural networks, due to the precise spike timings that SNNs work on leads to early performance bottlenecks and this behaviour is approximated on traditional Von Neumann architecture. As we move towards edge computing, ultra low power and concurrently, critical timing is demanded. Running SNNs on edge compute devices ensures low power and high performance with a good accuracy. To accomplish this task, a new underlying hardware like neuromorphic architecture is emerging.

For data centers or cloud based applications, systems like SpiNNaker or BrainScaleS provide insights to neuromorphic research community through real-time simulations and hardware emulations respectively. Nevertheless, these gigantic processing systems custom made for running SNN applications are not viable when the focus lies on edge applications demanding low-power, low-area and high performance. At the sensor edge, power constraints dictate the design of the embedded hardware, thus leading to a new architectural revolution seen through neuromorphic hardware. The overarching theme of neuromorphic engineering is to impart the cognitive abilities to an electronic device by implementing this architecture in silicon. The advancement in this field is to try and implement the biologically inspired spiking neural networks on a dedicated neuromorphic hardware to achieve high energy efficiency and high performance.

The process of compilation for a traditional compute architecture is well established. However, there is no neuromorphic assembly language or well defined and generalised instruction set to translate the neural network that can be understood by the underlying neuromoprhic hardware, due to the fact that the hardware designs are just emerging. The spiking neural network topologies are colossal and complex in nature and the target neuromoprhic hardware has thousands of PEs(processing elements)/resources. There are numerous configurations of placement possible for a single network on a given hardware, thus the problem falls in NP complexity class. Respecting the constraints and obtaining a solution, is primarily a big challenge. Achieving a valid placement solution is time-consuming and days/months of human effort is needed. This engenders devising a mapping algorithm that takes into account an increasing complexity of the hardware constraints, and efficiently performs placement of neural networks. Having an automatic way of mapping neural networks will also benefit faster design space exploration allowing designers to reiterate over the hardware design for better occupancy of resources, or design upgrades of specific neuromorphic components in silicon.

1.2 Objective

There is ongoing research in mapping spiking neural networks on neuromorphic platforms using different techniques like particle swarm optimization methods[58], greedy algorithmic approaches[60] etc. The research efforts so far have been mostly focused on partitioning the large network and multi-core mapping. There is sparse literature on mapping spiking neural networks topologies onto low-level primitives of hardware. As the problem in this work is a new subset of hardware architecture, there is a need for new subset of algorithm to perform the placement of the network.

The underlying target platform employed in this thesis work is unique and due to the design of neuromorphic hardware, the resources on the neurosynaptic crossbar and at the I/O interface have a distinct and an intelligent way of sharing resources. The design introduces a new set of connectivity constraints that doesn't allow for straightforward placement of the network graph. There is a need for a new algorithmic approach to carry out placement accounting for all the hardware constraints imposed by the new neuromorphic hardware design. The objective of this thesis work is to primarily devise a placement algorithm to allocate resources on the hardware, map complex topologies of spiking neural networks using a single mapper within a decent time-frame. Different varieties of spiking network topologies are investigated for the feasibility of mapping on the neuromorphic hardware, resources they demand specific to the target hardware, and the time-to-solution for each of the network topologies.

1.3 Research Questions

1. How can diverse and complex neural network architectures be placed on a connectivity constrained array with thousands of processing elements(PEs)?

2. Given a set of neural network topologies and hardware constraints, how quickly and effectively can the mapper converge to a valid placement solution?

3. How well can the mapping algorithm accommodate a wide spectra of neural network topologies accounting for all the constraints imposed by target hardware?

4. How can the estimate of resource utilization of each candidate SNN topology on the target hardware influence decision of choosing a specific neural network for an application?

1.4 Contributions

This thesis work presents a mapping pipeline for SNNs with an algorithmic approach. The SNNs are converted into a graphical representation readable by the custom built mapper for the target hardware platform. The contributions of this thesis are as follows:

- A placement algorithm, called ALAPIN mapper that maps intermediate representation of the complex spiking neural network topologies strictly adhering to the hardware constraints imposed by target neuromorphic architecture within a decent time frame. The proposed algorithm is adaptable to different configurations of hardware architectures with reconfiguration capability.
- The proposed mapping algorithm is designed to accommodate a wide spectra of neural network topologies.
- Given the topology of a neural network, the proposed mapper is capable of providing different placement solutions, strictly respecting all the hardware constraints for every solution produced.
- An automatic network generation tool is designed to create wide range of SNN candidate topologies to evaluate the proposed methodology seamlessly. The tool is designed to generate very large networks taking into account different network architectural complexity measures.
- Synthetic networks generated are employed in evaluation of the feasibility of mapping the candidate SNNs complying to connectivity constraints of the underlying target hardware. This evaluation also provides an early projection of the different design upgrades needed in the neuromorphic hardware to accommodate a vast variety of spiking based neural networks with non-threshold activation functions.
- A translation engine is designed to convert a neural network description to an intermediate representation(IR).
- The proposed method has an accurate estimate mechanism for the resources demanded by the candidate SNNs and the maximum reachable parameters of the neural architecture on target hardware.

1.5 Outline

This thesis work introduces an efficient realization of compile-time mapping of complex architectures of Spiking Neural Networks onto a massively parallel neuromorphic array. Predominantly, it provides new insights into the placement techniques for mapping neural networks and the resources utilised by different SNNs on the target hardware. Main themes of the thesis work are placement technique, spiking neural networks and neuromorphic processor architecture. The outline of the entire thesis work is divided into six chapters.

Chapter 2, 3 introduces the necessary background for spiking neural network and their working. It provides an overview of the various topologies of spiking neural networks, which will be evaluated in depth in Chapter 5. Following which, there is a literature study on different neuromorphic systems and the mapping strategy, specifically, on few neuromorphic hardware available publicly.

Chapter 3, 4 describes and explains the hardware architecture used in this thesis work in detail, in addition, introducing hardware imposed constraints that innately restricts the mapper in terms of achieving a valid mapping within decent time-frame. This chapter also presents the proposed methodology employed in strategic mapping of neural networks onto the priorly explained target hardware.

Chapter 5 investigates the characteristics of the proposed *ALAPIN* Mapper through various spiking neural network schemes and the complexity imposed by these neural architectures onto the proposed mapper. This chapter also provides insights into the parameter space of candidate neural networks and the feasibility of mapping a particular neural network model on the hardware.

Chapter 6 summarizes the thesis work, addressing the research questions briefly. This report concludes with discussion of the future work for efficient deployment of spiking neural networks on neuromorphic hardware serving the greater purpose of achieving energy efficient and high performance for inference at the edge seamlessly.

Part I
Literature Review and Background

2 Neuromorphic Paradigm

2.1 Pre-requisite

Inception of a giant automaton carved in bronze to guard Europa from invaders is the rightful cradle of Artificial Intelligence[4]. Although AI was coined for the first time in 1956 by John McCarthy, the first decade of the 21st century has immensely popularised the paradigm of integrating artificial intelligence in a wide variety of applications. Technological advancements to upsurge computing efficiency, access to large datasets, accurate and improvised algorithms, and a comprehensive investment from academia and industry has enabled a very powerful AI, that we witness today. The devastating effect of publications on 'Perceptron' by Rosenblatt led to the cessation of activity in connectionism for almost a decade. The revival of researchers in the Neural Network paradigm has made them an influential part of artificial intelligence[5]. Despite the immense progress, the hardware platform readily available today fails to cater to the needs of different application domains with varying intelligence levels (choice of algorithms), high demand for energy efficiency and faster execution.

There is a wide plethora of hardware that have evolved throughout history based on the application demands. The von Neuman based compute platforms including CPU (Central Processing Unit), GPU (Graphics Processing Unit), FPGA (Field Programmable Gate Array), and TPU (Tensor Processing Unit) fail, substantially, due to memory bottleneck. Data retrieval, transportation and storage leads to increased latency and energy consumption[20].

Table 2.1 encapsulates the comparison of spatial and temporal architectures to deploy AI workloads, however, it is not candid to compare which stands out as the best solution, since it is purely based on the application and various design constraints including time-to-market and cost of the individual chip. For example, edge-applications demand smaller chip area and low power consumption, whereas cloud-applications need to cater to the requirements of speed and flexibility[22].

The computational structure of both architectures are very similar, mainly consisting of a set of processing units. A distinguishing feature between both architectures is the way these units are controlled. Spatial architectures have internal control, whereas control is centralised in temporal architectures. CPU and GPU fall in the category of spatial architectures. CPUs consist of multiple ALUs to process multiple data in parallel. They adopt the SIMD (Single Instruction Multiple Data) execution model. On the other hand, GPUs have SIMT (Single Instruction Multiple Threads) in addition to SIMD, which gives them the true power to run regardless of Cache overhead. A primary reason of fast execution on GPUs is because they don't waste cycles waiting on resources.

The qualitative benchmarking of different hardware is shown in the Table 2.1 as surveyed by W. Dei and D. Berleant[6].

1. Compute Density: It is a correlation of which hardware uses how many processing units in comparison to others. As the traditional CPUs have sequential data processing with multiple cores, the compute density is low. However, parallel execution in GPUs is enabled via thousands of cores, making them compute dense.

2. Memory overhead: Data movements are reduced in FPGAs and ASICs to plummet energy consumption by employing reusable dataflow movements[7]. Data movement significantly consumes energy, in GPUs low-latency temporary storage architectures are used. Overall, the memory overhead is highest in CPUs.

3. Energy Efficiency: Edge devices have high demand for better energy efficiency because of their limited power envelope. Google's TPU for cloud and edge applications is moreover a special type of ASIC, therefore its energy efficiency is highly ranked than GPUs/FPGAs which come second and third in order respectively.

4. Upgradability: The biggest bottleneck for ASIC is undergoing a new upgrade after delivery. Flexibility is ranked highest for GPUs and then comes FPGAs, whereas CPU up-gradation is a bit of a challenge in itself.

5. Performance: In computing terms, performance is measured using FLOPS. A quadrillion FLOPS (petaflops) are used in ASICs and GPUs.The latest release by Google, TPU 3.0 provides 23.0 petaflops [11], whereas NVIDIA GeForce RTX 2080 Ti has only 13.4 Teraflops [8]. As critically acclaimed by [6] and [10], ASICs have the most FLOPS and performance wise stand out with respect to CPUs, GPUs and FPGAs.

6. Compatibility: FPGAs are bottlenecked by non-availability of special developing libraries. GPUs are the most compatible with large sets of ML frameworks in comparison to TPU (ASICs). Google's TPU might stand out in terms of energy efficiency and performance, nonetheless, they only support Tensorflow[9]. It is not even Tensorflow Lite but only models quantized to 8-bit integers (INT8), hence making them unsuitable for deep learning models.

Bottle-necked by synchronization and communication overhead, the neural computing is still plausible on von Neuman architecture, but not very efficient. On the other hand, FPGAs are capable of catering to the needs of neural computing, they are a promising surrogate target platform balancing between power hungry computing architectures and power-efficient and fixed function ASIC. The stroke of luck in FPGAs is their reconfigurable capabilities, that allows efficient mapping of CNN to meet latency, throughput and power requirements in areas from embedded systems to data centers[6]. However, the capabilities of FPGAs fail when there are millions of neurons and trillions of synapses to recreate the capabilities of our brain on a compute platform using Spiking neural networks (SNNs), which entails the fast-paced research and development of neuromorphic hardware. To be able to efficiently perform the neural network computing without any power overhead and compromise in throughput and performance, there is an immediate need to define and set an underlying architecture like neuromorphic chips[21].

Table 2.1: Comparison of qualitative benchmarking of different hardware platforms for AI

	CPU	GPU	TPU	FPGA
Compute Density	Low	High	High	Moderate
Memory Overhead	High	Low	Low	Low
Energy efficiency	Low	Good	High	Moderate
Architecture	Temporal	Temporal	Spatial	Spatial
Upgradability	High	High	Low	Moderate
Performance	Moderate	High	Very High	Low
Compatibility	Moderate	High	Low	Low

2.2 Neuromorphic Computing

The architecture of human brain makes it the most energy efficient and lowest latency system on earth[14]. Human brain is made up of dense neurons transmitting signals through synapses efficiently. Carver Mead in the late 1980's coined Neuromorphic Engineering as systems containing analog/digital circuits to mimic neurobiological elements[23]. A typical neuromorphic hardware encompasses any electrical device which mimics natural biological structures of the nervous system. The overarching goal is to impart cognitive abilities to a machine by implementing neurons in silicon. Today's methodology of implementing neural networks on the traditional von Neuman hardware fails inherently to provide low power and fault tolerant operation in comparison to the neuromorphic systems directly implemented in hardware[24]. The major drawback of von Neuman architecture is that it is built around the principle of transistors operating in saturation region with a very deterministic behavior, making them power hungry, primarily synchronous and vulnerable to physical damage[25]. On the other end of the spectrum lies the biological systems which have non deterministic operation with immense parallelism.

The theme of brain inspired approach is to utilise biological guidelines of neuromorphic computing for high speed design of large-scale biological nervous systems as BMI (brain machine interfaces)[26], or to address the limitations of CMOS digital computing in applications such as pattern recognition, computer vision, in general, learning at low power consumption[27]. Spiking neural networks are computational models quite different from their ancestor artificial neural networks, and hence understanding and appreciating how they work is important before diving into why a specialised hardware might be beneficial to run these spiking neural networks.

2.2.1 Spiking Neural Networks

Neuro-biology inspired Spiking Neural Network (SNN) are efficient to perform learning and classification, closely mimicking the human brain. The resemblance of DNNs is only marginal to the brain-like computation. The energy efficient and event-driven processing of brain is succinctly replicated in SNNs, making them a promising candidate for AI applications[12][13][15].

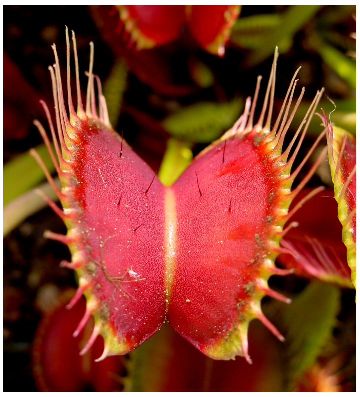


Figure 2.1: Illustration of Venus Flytrap showing trigger hairs to signal the contact of 1/10th of an insect in their trapping structure. The redundant trapping mechanism is closely related to how spiking neurons work. Image from [16]

- **Spiking Neuron** As much as the working of SNNs mimicks the biological nervous system, interestingly, it can be co-related to a Venus fly trap, a carnivorous plant as shown in Figure 2.1. The trapping mechanism is beautifully interwoven or timed to ensure a bug or insect is caught, just upon 1/10th of the contact of the insect with leaves, specifically contacting the sensitive hair on the inner surface of the leave. Atleast two-sensitive hair must be triggered to identify an insect and call for closure immediately upon trapping a fly.

  A neuron receives the incoming spikes from presynaptic neurons through synaptic connections in the form of membrane potential. These incoming spikes can possess either excitatory potential or inhibitory potential which solely decide whether the neuron is going to fire a spike or not. After integration of the various membrane potentials, once the threshold potential is hit, the neuron fires an output spike as shown in Figure 2.2.

- **Synapse** The neuronal information is transmitted from presynaptic neuron to postsynaptic neuron through a synapse. The standard communication protocol is facilitated through a train of spikes- consisting of a stream of binary numbers, where a '0' corresponds to spike off and '1' corresponds to spike on.

- **Synaptic Weight** Synaptic efficacy or weight definitely represents the impact that a transmitting neuron activity signal can have on a receiving neuron through synaptic connections. From the biological perspective, synaptic weight is the net propensity of the transmitting end neuron's action potential to release neurotransmitter, and the propensity of the neurotransmitter to open synaptic channels on the postsynaptic neuron.

  Synaptic weights determine what exactly is a neuron detecting. A strong weight value implies sensitivity of a neuron to a particular input neuron, while a low weight implies input of a neuron is comparatively not important. There are two types of synapses:
  Excitatory input: Increases neuron's membrane potential

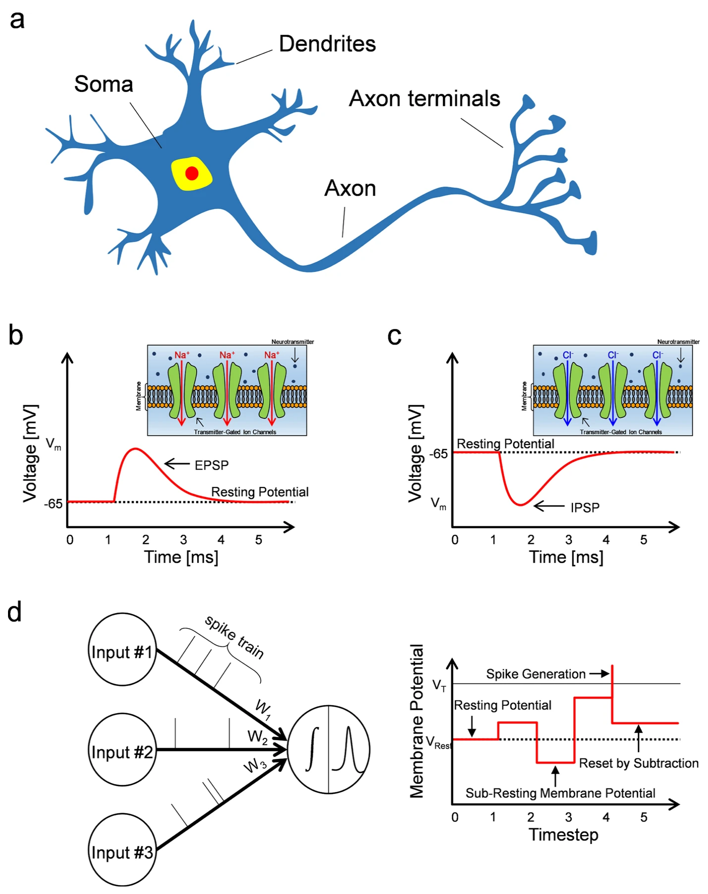


Figure 2.2: (a) Illustration of a biological neuron. Dendrites send pre-synaptic stimulus to the neuron, soma performs the integration of synapses. ((b),(c)) Action potentials release neurotransmitter in synaptic vesicle. EPSP(depolarisation) and IPSP(hyperpolerisation) characteristic through stimulation of ion channels are shown (d) Schematic representation of Spiking Neural Networks (SNNs).The integration of the membrane potential over incoming spikes to output a spike (shown in red) when the firing threshold is hit. Image from [29]

Inhibitory input: Decreases neuron's membrane potential

As depicted in Figure 2.2, the membrane potential of the neuron increases or decreases as a spike is injected to it. Variation of the membrane potential of a postsynaptic neuron is termed as 'Post-synaptic Potential(PSP)'. Positive or Excitatory Post-Synaptic Potential (EPSP) is the result of spike generated by excitatory synapse, whereas a negative or Inhibitory Post-Synaptic Potential (IPSP) is the result of spike generated by an inhibitory synapse.

- **Neuron models** There are a bunch of neuron models developed in the past over the decade and the analog neurons on the hardware are LIF (Leaky Integrate and Fire) neuron model.In a LIF neuron model, a leaky integrator fires the spike if membrane potential of the neuron hits a leaky threshold, after which it drops down to a refractory state to reset itself. Electrically, it can be envisioned as RC circuit[4]. LIF model is a popularly used spiking neuron model due to its low computational cost and simplicity. There are a couple other potentially stronger neuron models, although high on cost and they bring along extreme implementation plausibility. For example, the Izhikevich model.

2.2.2 Neuromorphic hardware architectures

Mimicking brain and scaling it down for different low power embedded applications to tap the benefits of parallel architecture in silicon is ac hived through a new paradigm of hardware called neuromorphic hardware. There are some architectures for neuromorphic hardware that evolved during the last decade by research institutes like IBM, Intel's INRC, Stanford etc that provide a good insight at the way of implementing spiking neural networks on these dedicated hardware and these systems are highly energy efficient. To be able to host real-time operation, realise low-power consumption and scalability, a radical departure from conventional design to brain-inspired architecture is taken.

1. **Loihi** : Intel's Loihi comprises 128 neuromorphic cores, each core realising the implementation of 1024 primitive spiking neural units that are a group of sets of trees constituting neurons. Loihi is built on Intel's 14nm process and it is the state-of-the-art fully integrated spiking neural network chip in silicon. It imparts most important qualities like programmable learning rules, dendritic compartments, synaptic delays etc. The chip consists of over 130,000 neurons that communicate with a thousand other neurons, the dedicated hardware is built to optimize SNNs and in turn support accelerated learning fulfilling the ultra low power requirements and high performance[59].

2. **TrueNorth** : IBM developed TrueNorth as a part of Defense Advanced Research Projects Agency (DARPA) SyNAPSE program[6], the largest neurosynaptic computer with 4096 neurosynaptic cores consisting of 1 million spiking neurons and 256 millions synapses interconnected in an event-driven routing infrastructure[55]. Each core out of 4096 neurosynaptic cores, brings together memory, processors and communication as Synapses, Neurons, and Axons respectively in IBM's 45-nm SOI process[55].

   As shown in the figure 2.1, the neurosynaptic core consists of buffers that receive inputs from the network, axons form the horizontal lines, dendrites form the vertical lines and neurons are represented by triangles. A synapse is the junction, represented by a dot, between axons and dendrites[55]. The output of each neuron

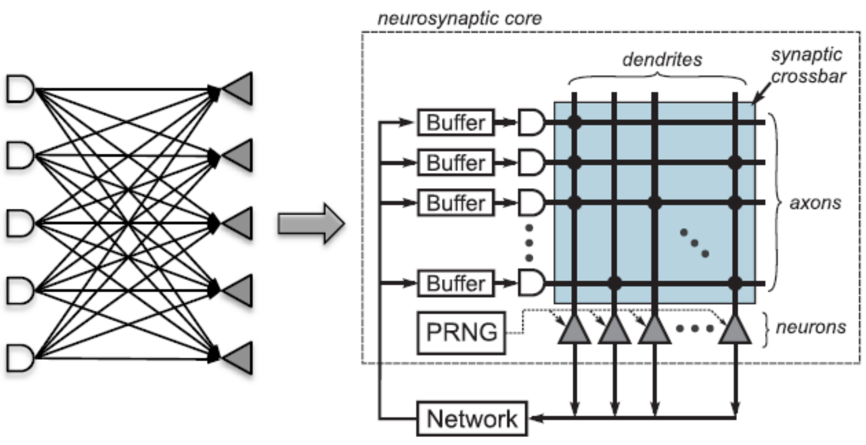


Figure 2.3: Bipartite graph of a neural network (left), with arbitrary connections between axons and neurons, and the corresponding logical representation of a TrueNorth core (right) depicting the input, neurons and synaptic junctions[55]

is connected to the input buffers. The following steps are the way the computation of neurosynaptic core proceeds.

- Input buffer stores the spikes arriving from the network.
- On the arrival of synchronization trigger, the current spikes are read from input buffers and distributed across the horizontal axon line.
- In case of joint formation at horizontal axon and vertical dendrite (synaptic connection), the spike from axon is delivered to the neuron through dendrite.
- The membrane potential updates in each neuron with every incoming spike. Upon integration of all spikes in a neuron, the leak value is subtracted from its membrane potential.
- Upon exceeding the threshold, a spike is generated and sent to the network.

3. **Neurogrid** : Neurogrid is the born at Stanford University's BrainLabs. With the help of a mixed-signal system, researchers were able to pull off a humongous and affordable simulator to provide computational neuroscientists with the capacity to perform brain simulations with millions of neurons and a billion synaptic connections. It has a 256x256 neruo-synaptic array fabricated in 180-nm CMOS which makes one neurocore. 16 such chips were used to build a board which was arranged in a tree-structure called Neurogrid.

4. **SpiNNaker** : SpiNNAker stands for Spiking Neural Network Architecture, a massive parallel computer which is a cost-effective and quite flexible simulator for neuroscientific experiments. Like any other neuro-synaptic chip, this one can also simulate billion neurons and trillion synapses in real-time. A SpiNNaker chip has 18 ARM968 processor nodes residing in synchronous islands, it has a custom designed globally asynchronous locally synchronous (GALS) system[54].

5. **DYNAP-SEL**: A novel mixed signal multi-core architecture for neuromorphic processors which amalgamates the perks of asynchronous digital logic for communication and dynamics of analog circuits for computation. There are two variations in the mixed signal chip i) Dynamic Neuromorphic Asynchronous Processors (DYNAPs) which was designed in 180nm CMOS process and ii) Dynamic Neuromorphic Asynchronous Processor with Self Learning abilities(Dynap-SEL), both were created at the University of Zurich in Switzerland. Both the chips have 4 different neural processing cores. Every core has neurons structured in 16x16 grid, the neuron model is Adaptive-Exponential Integrate and Fire (AdExp-IandF). There are 64 programmable synapses for each neuron.

2.3 Mapping SNNs on neuromorphic systems

Inspired by biology and supported by research in neuroscience, the hardware evolution has been seen through past years and it is evident that the neural networks can be mapped onto hardware with biological time plausibility [19]. The generic way of achieving simulation or emulation of the neural networks on von-Neuman based computing paradigm usually follows these steps: i) Compilation, ii) Resource allocation and iii) Run time mapping. There is sparse literature on how the spiking neural networks are deployed on the neuromorphic systems. In the past decade, there has been extraordinary research in accomplishing the implementation of brain-inspired spiking neural network on brain-inspired neuromorphic hardware.

2.3.1 Placement strategies

An onerous process of hand-designing and manually laying out the designed integrated circuits was metamorphosed into a set of software tools for designing electronic systems like IC(integrated circuits), ASIC(application specific integrated circuits), PCB(printed circuit boards) etc using electronic computer aided design (ECAD), also known as Electronic design automation(EDA)[17].

Placement, typically, deals with choosing location for each logic block in the technology-mapped netlist. Different components of the designed circuit are placed like standard cells, macro blocks, I/O pads (assigned specifically to pins around the periphery of the chip) etc. The complexity increases with a large circuit size. There is a close relation between EDA frameworks existed so far and the strategies employed to realise neural networks on the hardware. The placement techniques used in industry and academia in placing VLSI circuits with tens of millions of standard cells are as follows:

1. Stochastic approach
2. Partitioning approach
3. Analytical approach

Since the modern circuits are humongous in terms of module density, placers combine various optimization techniques with a hierarchical approach to efficiently place and route the net models.

2.3.2 Mapping on neuromorphic hardware

In a normal placement problem for VLSI circuits, the logic circuits have to be converted to a clique or net model before placing and routing them on the hardware. There is a paradigm shift while mapping neural networks on any hardware platform (i.e. CPU, GPU, or ASIC), the general flow will be discussed in Chapter 4.4 Section 4.1. The placement techniques can coincide, however the challenges posed by neuromorphic hardware and SNNs are very different than the usual circuits mapped. The strategies employed by state-of-the-art neuromorphic hardware platforms will be discussed below:

- DYNAPs

  DYNAPs is designed to have 4 crossbar arrays with 256 neurons per core in their underlying architecture. Partitioning based approach is to cluster the SNNs and the authors treat it as a bin-packing problem and employ greedy strategy to solve the mapping problem with crossbar utility maximization as their primary objective[52]. The evaluation of the approach is carried out through run-time complexity and resource utilization for applications with dataset covering feed-forward networks, convolutional neural networks and recurrent neural networks.

- SpiNNaker

  SpiNNaker is a massively parallel chip multiprocessor that realises large scale SNN simulations in real-time. PACMAN is implemented to map SNNs on SpiNNaker. The SpiNNaker chip is visualised as a connection of six neighbours, by assigning neurons to any processor arbitrarily and configuring routing tables to ensure the timely neural events[54]. Since, the entire system has about 18 cores with ARM processors, the primary objective is to optimize the routing resources to gain positive impact on network traffic. This is tangential to the research carried out in this work, as we deal with single core architecture with complex constraints.

- TrueNorth

  A digital neuromorphic chip, IBMs TrueNorth is built on 28nm technology with 4096 cores, accommodating 256 neurons per core with 1-bit SRAM for synapse weight storage. As proposed by authors in [55], TrueNorth uses corelet tool, an object-oriented language to map SNNs. The trained parameters are mapped using re-usable, composable hardware description function independent of any neuromorphic platform[56].

- Loihi

  Intel's digital neurormorphic research processor, Loihi is a 128 core, built on 14nm FinFET chip. Compiler developed in PythonAPI for Intel Loihi's NxSDK uses a greedy algorithm to map SNN onto multiple neuron cores[60]. The evaluation is carried out on realistic AI models and how communications cost in terms of energy and performance are optimised using the compiler built for Loihi.

The research effort by most of the above mentioned groups have been to address the mapping of spiking neural networks from a Network-on-Chip (NoC) perspective, addressing routing resources, inter-core spike traffic etc. The mention of mapping on a single crossbar is very sparse, only in [57], hence leaving placement under covered in [54][56][52]. The underlying architecture of the target platform and the complexity in the design dictates the complication of devising a placement technique to meet the constraints imposed by hardware within decent run-time. TrueNorth's architecture with 1:1 neuron and synapse leads to no segmentation in the crossbar array, hence no inferrable comparison is possible with target hardware. Intel does provide insights on time complexity of running different SNNs on Loihi, however there is no straightforward mention of the exact SNN architecture and the limited knowledge of their underlying hardware design leads to no conclusive comparison with target hardware.

2.3.3 Conclusion

The comprehensive review in this chapter has pointed the sparse literature on mapping complex SNNs on a specialised neuromorphic processor, even though mapping is a heavily researched topic. Different approaches used by industry and research institutes have a commonality in them, their underlying hardware is only comparable in certain degrees (example: design of the chip in terms of neurons, synapses and cores etc) and not directly in terms of the architecture. This poses a huge challenge in comparing the state-of-the-art, firstly. The sparse literature on mapping a complex variety of SNNs using a custom compiler or placement technique on a specialised neuromorphic hardware like target architecture is evident to show the complexity of the challenge this thesis is trying to address. Consequently, there exists no refined dataset on SNNs that can be experimented to qualitatively and quantitatively measure the performance of the mapper. By employing the existing placement algorithms for solving the problem posed by this thesis is not possible and needs further insights from literature. The limited knowledge of placement techniques for neuromorphic paradigm and the challenges imposed in the introduction chapter are in tandem which will be addressed and a solution will be proposed in methodology chapter.

Target architecture overview 3

3.1 Target hardware

The crucial part of this thesis is the hardware platform: neuromorphic crossbar architecture as shown in Figure 3.1. To be able to map the given neural network scheme, it is substantial to understand the hardware architecture. Hardware architecture is designed to accommodate large neural networks by intelligent resource sharing mechanism which impose constraints on the allocation of resources and availability of the processing elements (PEs). These challenges and hardware constraints are introduced in Section 4.2. This section will introduce hardware to understand the origin of constraints, the complex topologies of spiking neural networks and the impact of these challenges while mapping will be discussed in Chapter 5.

3.1.1 Neurosynaptic Array

The neurosynaptic array is the most important element of the spiking based neuromorphic hardware. Predominantly, the core comprises of M neurons and M^2 synapses. The dashed lines colored in pink in Figure 3.1 mark the boundary portraying the array.

The complex formation of synapses, neurons and the inner architecture of input resources, bring out the best in the hardware and forming a basis for fabrication of constraints which will be discussed in detail in Section 4.2.

3.1.1.1 Neuron

In determining the scheme of neural network that can be mapped onto the hardware, knowledge of activation function employed in the spiking neural networks, and a know-how of neuron model implemented in the hardware is necessary. This can decide the spectra of spiking neural network topologies that can be placed on the hardware. Nevertheless, it just doesn't limit to that, depending on the flexibility of the physical neuron model, there can be more complex networks mapped onto the hardware seamlessly.

3.1.1.2 Synapse

An important processing element of the neurosynaptic core is the SYNAPSE. The functioning of synapse is quite straightforward. Depending on the edges in the neural network that will be mapped onto the synapses on hardware, this enables/activates the synaptic operation to output the current upon receiving input current. Synapses store the weights of neural networks in the form a memory on hardware.

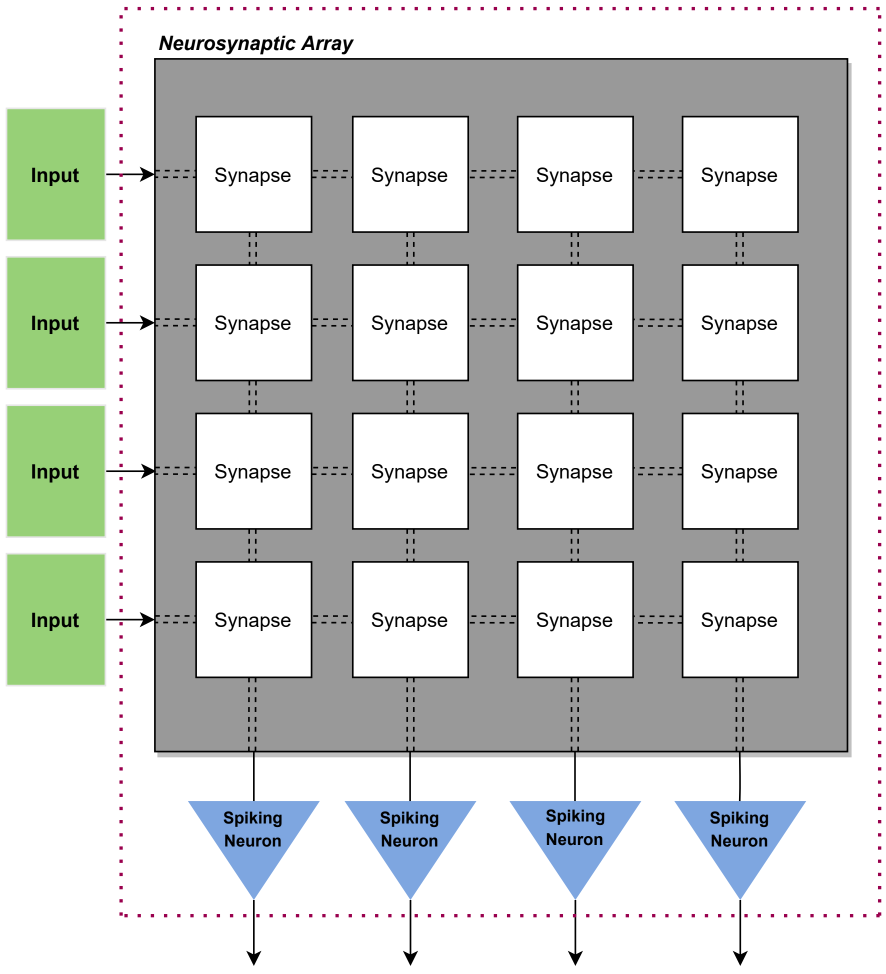


Figure 3.1: Illustration of the significant PEs on Spiking Neural Processor

3.1.2 Digital interface

This section is ommitted due to confidentiality purpose.

3.1.3 Architecture overview

The introduction to the target hardware architecture bridges the gap to understand the hardware constraints and the proposed methodology for mapping seamlessly. Having introduced the complex target hardware with thousands of processing elements (PEs), the next section will give a brief overview on surveyed topologies of spiking neural networks that needs to be mapped on to the target hardware to achieve the objective of this thesis work, and at an overarching level, configure the hardware to implement the neural networks for different applications.

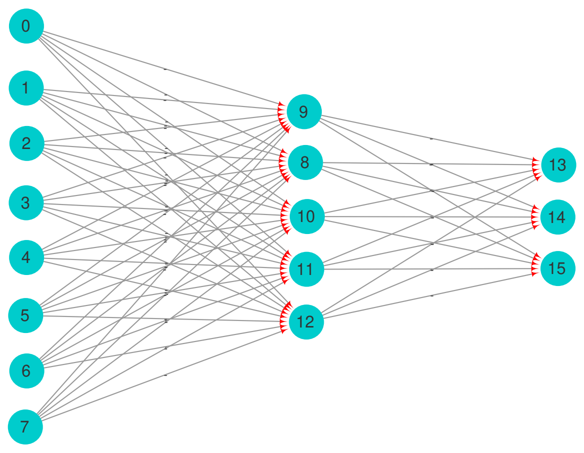


Figure 3.2: Feed Forward Network (Encoder style)

3.2 Spiking Neural Network Schemes

In this thesis work, there is a detailed investigation on which schemes of neural networks are possible to be mapped onto the neuromorphic hardware. Therefore, this section introduces background on different neural networks, topologically and functionally.

The elemental building blocks of any neural network are artificial neurons and synapses with weight values. Figuratively, the functioning of neural network is concretely defined by the signals paths and adjustable weights. Henceforth, throughout the thesis report, the terms nodes and edges will always refer to a neural network or its graphical representation, whereas neurons and synapses will be used in the context of neuromorphic hardware. The spiking architectures are reviewed in detail which form the basis for design of experiments in Section 5.

3.2.1 Feed Forward Networks

Feed forward networks are one of the simplest forms of neural networks with input and output layer with number of hidden layers in between. A notable characteristic of this scheme of network is the flow of signal which is in one-direction, without any loop-back.

Typically, a feed forward network has input, hidden and output layers with certain number of nodes that receive sum of product of input weight values and based on the activation function, the nodes decide to pass the output value to the next layer node or not. The feed-forward networks employ supervised learning, wherein the network is taught to which particular category does the pattern belong to. Once the category pairing is done for every pattern, you have achieved an epoch of learning.

Figure 3.2 is an over-simplification of an encoder style feed forward neural network with input layer with nodes (0-7), hidden layer with nodes (9-12) and finally, the output layer with nodes (13-15). The degree(in -degree and out -degree) of the hidden layer nodes are relatively higher than the input layer and output layer nodes.

The feed forward based deep neural networks are employed from cardiovascular engineering in clinical applications to cyber-forensics[33][36]. Notably, there are numerous NNs that employ feed forward architecture, like, Multi-layer Perceptrons (MLP), Deep Belief Networks (DBN), Convolutional Neural Networks (CNN) and Auto-encoders.

3.2.2 Autoencoders

As rightly explained by Hughes phenomenon or peaking phenomenon[34], the expected predictive nature of a classifier or regressor is linear with respect to the features, however due to curse of dimensionality, the predictive nature begins to decline progressively instead of gradual improvement with additional dimensions.

Nevertheless, pertaining to simple classifiers, Zollanvari et al[35], have shown analytically and empirically, that not just the size of the features but the discriminatory effect contribute towards the nature of predictive power. The vastness of high dimensional space leads to inefficacy in predictive performance, forming a predominant machine learning problem, if it is adding only noise instead of signal to the space. For unsupervised data analysis, this effect can be swamping, giving rise to techniques that employ dimensionality reduction using PCA(Principal Component Analysis) which leads to linear network generation. To address larger problem space, autoencoders are harnessed. Unlike PCA, autoencoders are powerful enough to model non-linear functions.

The simplest autoencoders are non-recurrent and feed forward in nature, essentially, comprising of input, output and hidden layers. From the discussion above, to achieve dimensionality reduction for non-linear activation functions, autoencoders are deployed for compressing the input data in the hidden layer, which later, represents the original input at the output layer. The hidden layer is also known as the "Bottleneck layer" as the number of nodes are constrained to limit amount of information flow through the network. Consequentially, letting the model to learn and retrieve the most important latent attributes of input data. The number of nodes in the hidden layer(s) accounts for the certainty that the model is not memorizing the input data.

3.2.3 Recurrent Neural Networks

As discussed in section 3.2.1, the signal moves in a single direction for feed forward networks, however, there is a possibility of feedback from one layer to previous layer in a recurrent neural network, as shown in Figure 3.4.

For a wide variety of application having different input-output scenarios, the network functions to accommodate the sequential inputs and outputs through time for applications like: Image captioning, document classification, video processing by frame etc.

Since, backpropogation through time is utilised for training RNNs, consequentially, introducing the vanishing gradient problem. This problem is over-amplified in RNNS due to the incurrence of time steps. For instance, a network trained for 1000 time steps, the gradient will vanish exponentially as it would in a Multi-layer perceptron (MLP) with depth of 1000. To combat the vanishing gradient issue, Felix Gers, Jürgen Schmidhuber and Fred Cummins architected forget gate (keep gates) in 1999 [37], leading to a advanced architecture of RNNs called as LSTM(Long-Short Term Memory) in

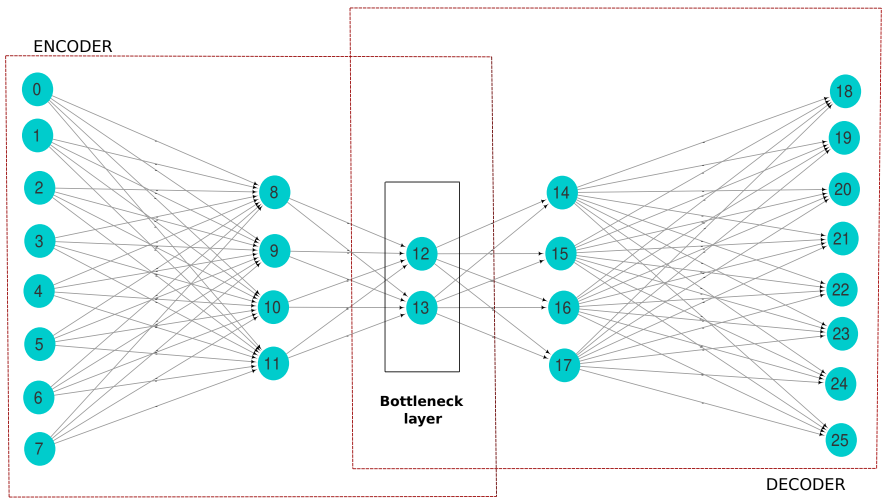


Figure 3.3: Undercomplete architetcure of Auto-encoders

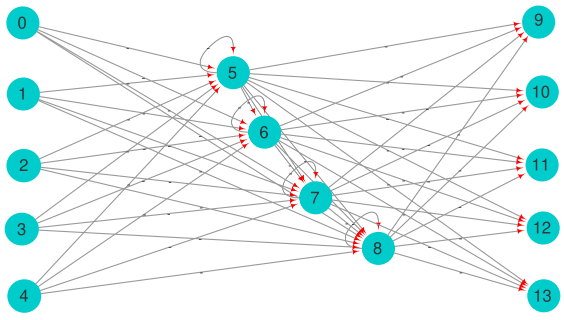


Figure 3.4: Recurrent Neural Networks with recurrence from layer n to layer (n-1) and self-loops

2000[37]. A much simplified variant named GRU (Gated Recurrent Unit) was proposed by Kyunghyun Cho et al[38].

Most significant application of recurrent neural networks are seen in auditory applications like speech recognition, text-to-speech recognition etc. It is also proven as the technology companies like Google employ LSTMs on Google Voice [40][41], Apple uses LSTMs for quicktype in iPhones for Siri[42][43], Amazon uses a bi-directional LSTM in Polly (voice behind Alexa)[46].

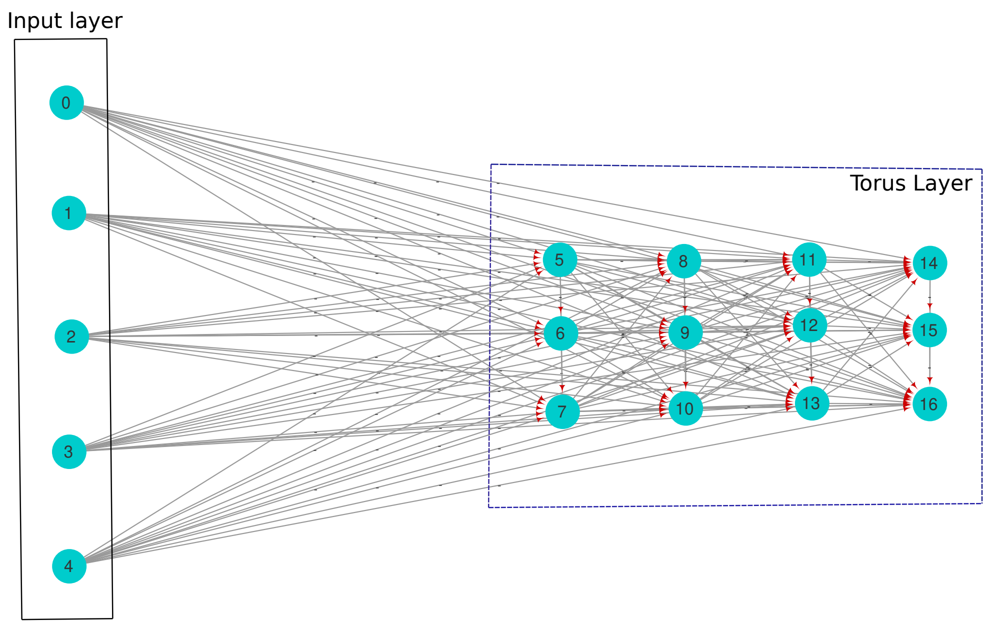


Figure 3.5: Self- Organizing Map/Kohonen Network architecture

3.2.4 Self-Organizing Maps

Self-Organizing Maps, or classical Kohonen networks are biologically inspired algorithm that mimic the mammalian cortices. Topologically ordered spatial representations of feature maps in various sensory areas of cortex are evident as acclaimed by H. Kaas and Rumbell [44][45].

The spiking SOM architecture as shown in Figure 3.5 represents the input layer, which receives input from the external data set feeding into a bank of neurons in the first layer. The input data set is converted into a temporally rich spike series[45]. The connections from this input to the SOM layer is one-to-all in feed forward fashion. The SOM layer, more specifically, the torus layer implements the neighborhood function with All-to-all lateral synaptic connections. The three factors contributing towards the self-organization of feature maps are i) Temporal spike encoding of input dimensions, ii) neighborhood function, and iii) STDP learning rule.

3.2.5 Lateral Inhibitory Networks

Lateral inhibitory networks are a type of spiking neural networks with excitatory and inhibitory neurons. Inhibitory neurons have the ability to inhibit or decrease the action potential of the synapse to whichever neuron they are connected to[30][31]. Figure 3.6 shows the topological representation of lateral inhibitory networks, where the hidden

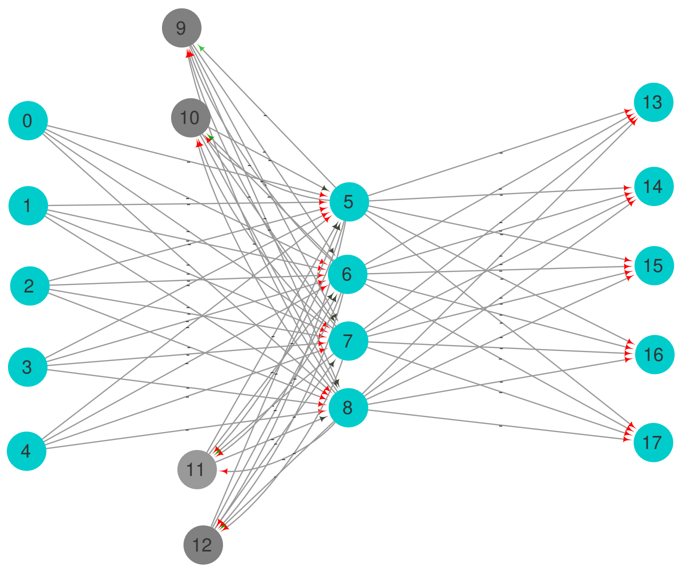


Figure 3.6: Layered full lateral Inhibition architecture in Lateral Inhibitory Network

layer neurons numbered(5-8) have a complimentary inhibitory neuron. The neuron in hidden layer fires an action potential, and the inhibitory neuron connected to it will fire and inhibit the rest of the neurons in the hidden layer from firing a spike.

3.2.6 Liquid State Machines

Reservoir Computing was proposed by two independent research groups. Echo state networks proposed by Jaeger in 2001 are rate-based approximation, in contrast, Liquid State Machines proposed by Maas et.al[12], are biologically inspired spiking neural networks. Liquid state machines use reservoir of untrained neurons. They are more tended towards unsupervised learning theme. It essentially has three layers, an input layer, a liquid layer and readout layer as shown in Figure 3.7.

Since LSM's are employed in audio classification, mostly binary classification, and at times multi-classification applications. Few examples of LSMs applications are Speech recognition, vision, music classification (Bach/Beethoven music). The number of neurons and the way they connect between each other determine the degree of complexity in liquid layer. The connections between neurons defines the dynamical process and the lateral recurrent connections defines the topology.

The need for sufficient liquid neurons is necessitated by the fact that there is conversion of input data to high dimensional expansion and readout layer fulfills the classification task. Ideally, the number of liquid neurons is 10x input neurons. Another important aspect is the sufficient recurrent connections so that the neural network maintains the information in a signal and it has the ability to separate different signals.

Primary neurons in liquid layer have the power of accepting feed forward signals, and auxiliary neurons have the ability to connect to readout layer neurons. Both, primary

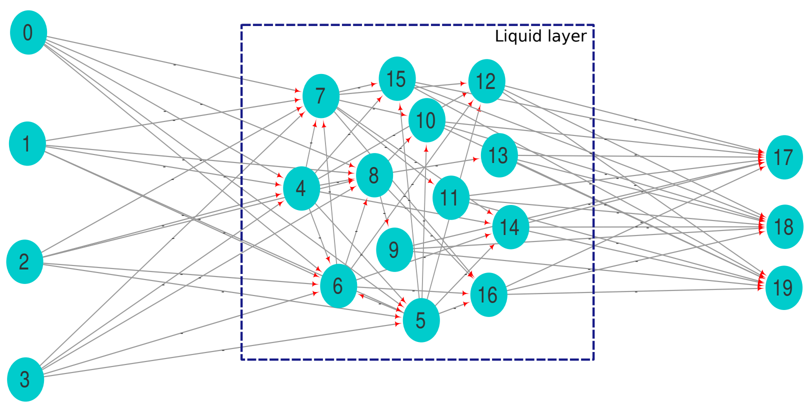


Figure 3.7: Liquid State architecture with randomly connected liquid layer

and auxiliary, neurons have the capability of connecting to each other in a random fashion. The number of synaptic connections from input layer to primary neurons in liquid layer can produce unique high-dimensional representation of the input data set, thus ensuring high accuracy.

3.2.7 Complete Hidden Layer

This scheme of neural architecture includes a complete hidden layer with input and output layer as shown in Figure 3.8. E. Frady and F. Sommer proposed a temporal neural coding scheme using networks with threshold neurons[49]. These neurons employ Hebbian type hetero-associative learning to store transitions of sequential neural activity patterns.

The hidden layer is essentially a Hopfield network or more closely in collation with bi-directional associative memory(BAM). The important applications of these network types is image segmentation or image restoration with/without enhancement which leads to two different modes of these networks. Firstly, the binary mode which employs Hopfield networks with no self-connections for image segmentation and secondly, continuous mode which allows self-connections mainly used in medical image processing or image restoration.

3.3 Conclusion

The underlying hardware was introduced in the first section of this chapter with emphasis on the significant resources that will be considered during design of the algorithm. The nature of the hardware and the careful design leads to many constraints that make mapping of neural networks not so straightforward. Detailed introduction to harwdare

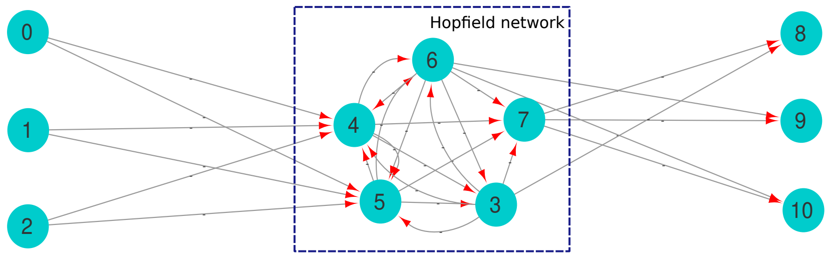


Figure 3.8: Complete Hidden Layer architecture

constraints will be discussed in section 4.2. Complex spiking neural network topologies were explored extensively. The behaviour and integration of the various SNN topologies on the neuromorphic hardware is sparsely explored in the literature. The proposed mapping algorithm will be evaluated on these aforementioned SNN topologies in detail in Chapter 5 to investigate the ability of the proposed mapper to map to the target hardware.

Part II
Proposed Methodology

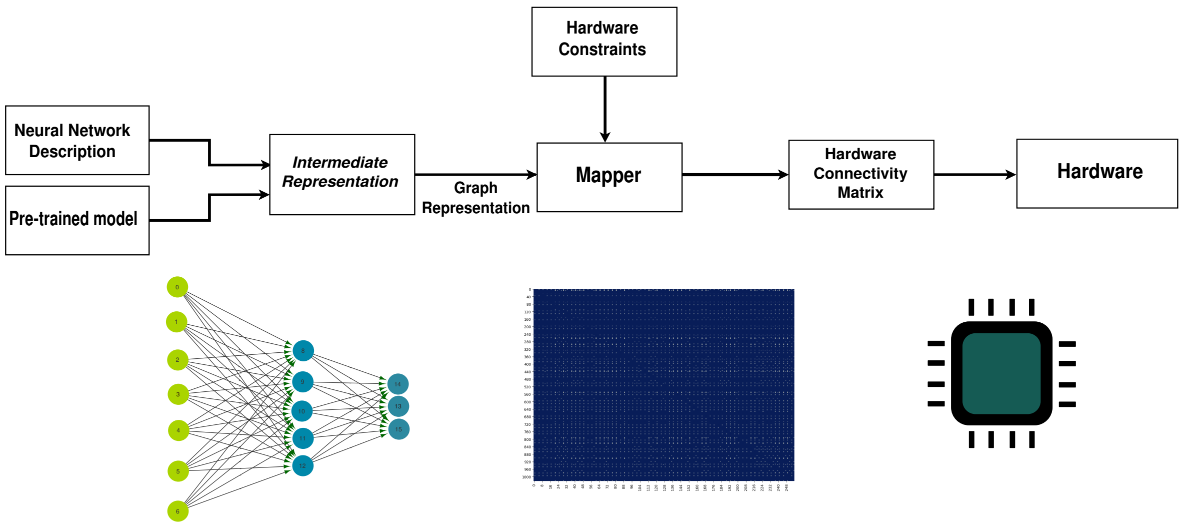
Possible Placement Matrix mapping method 4

The main goal of this thesis is to map different topologiess of SNNs on the target neuromorphic processor. Before building an algorithm for realising the mapping of the intermediate representation of a particular neural networks, it is crucial to know the underlying architecture of the neuromorphic hardware and the hardware connectivity constraints that form the basis of challenges for seamless mapping. The proposed methodology is divided into two sections: i) Hardware constraints due to the design of the target hardware and ii) the proposed mapping method.

4.1 Approach

The block diagram in Figure 4.1 depicts the approach and an overall flow taken in this thesis work. Every block in the diagram forms an important element towards achieving the end objective of mapping spiking neural networks onto the neuromorphic hardware, which will be discussed in Section 4.4.

The idea behind the mapping of neural networks onto a specific hardware which is inspired from biology is to reap the benefits in terms of processing power and speed when realising it on an embedded device. This thesis will address mapping of a pre-



Hardware Constraints
Neural Network Description
Pre-trained model
Intermediate Representation
Graph Representation
Mapper
Hardware Connectivity Matrix
Hardware

Figure 4.1: Block diagram illustrating the overall flow. Neural network description is input to the mapper, hardware constraint aware connectivity matrix is produced as output, which acts as an input to the configure the target hardware

trained model of spiking neural network rather than training the model on-chip itself. Therefore, a pre-trained model for applications like classification or detection etc are considered as the input to the entire system. The pre-trained model is essentially converted to an Intermediate Representation(IR) so as to ease the further process in obtaining a graphical representation. The IR is incorporated to be able to parse to any output format before giving it to the Mapper and in addition, to help visualize the neural network easily through standard network softwares like Cytoscape or YAML. The next block in the flow is the crux of this thesis work, the mapper. It performs the allocation of nodes and edges of a neural network onto the processing elements like physical neurons and synapses through input blocks on the specialised target hardware and mapped addresses of these PEs are produced with a connectivity weight matrix that can be used to configure the hardware.

The overarching goal of this hardware design is to accomplish running complex neural networks, and accommodate myriads of synapses and neurons on a single system. This complicates the software design to fulfill the constraints that transpire due to an intricate hardware architecture. Capturing and defining the hardware constraints is cardinal in designing and developing a matured mapping algorithm. Therefore, the 'Mapper' receives input from neural network and hardware. This block essentially converts the parsed graph representation of the neural network to fit onto neurons and synapses on the hardware, more definitively, in the form a hardware connectivity matrix. The mapped addresses of physical neurons, input resources, weights assigned to a synapse etc is fed in the configuration file, which can be later fed to the hardware to run a pre-trained model on the chip.

4.2 Hardware Constraints

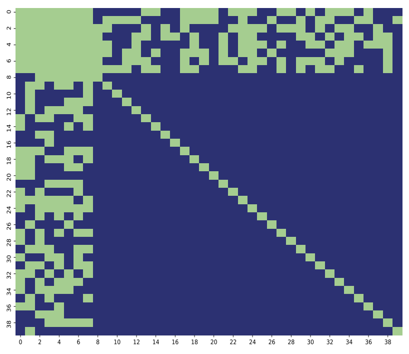
This section is omitted due to confidentiality purposes.

4.3 Challenges in mapping

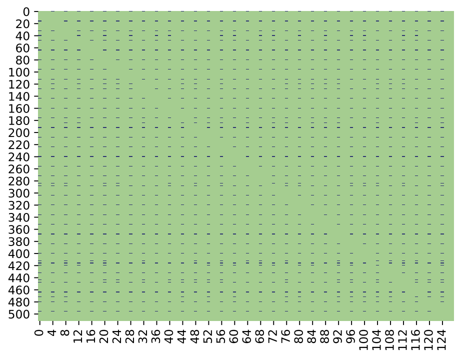
The different constraints introduced in the above section formulate a strong basis for architecting the algorithm and the design choices made. The computational complexity of the algorithm depends on how quickly the constraints are satisfied when the available hardware resources are exhausted for a huge neural network to be mapped.

At an interface level, the connectivity constraints dictate the feasibility of mapping a network with higher layer widths in the early stages of neural network. Careful consideration of all the hardware constraints based on the available hardware resources can be laborious as the network size grows.

Consider a colossal neural network, and a tiny hardware, but with millions of resource, it would take a normal human being about days to map these neural network onto the hardware by satisfying all the constraints imposed by the hardware. Hence, engendering the need of an automatic mapper. An intelligent mapping strategy or an algorithm is needed for achieving a valid and legal solution within decent time-frame.



(a)



(b)

Figure 4.2: (a) Weighted adjacency matrix which is obtained from the graph description, (b) Mapped connectivity matrix produced by proposed Algorithm respecting the hardware constraints

There can be multiple ways of performing resource allocation, a more static approach was taken in the early attempts of mapping which led to a fixed placement only on a particular corner of the hardware. This form of placement can introduce an over utilization of the same resource and the exploratory space is reduced if neurosynaptic delay or congestion etc is a major concern.

A formula based or mathematical approach to map the resources has been tried during the attempts of devising the algorithm. A mathematical approach led to invalidity in the placed neural network with overlap of resources and was applicable to single layer networks (effectively). Given a deep neural network, the input resources were exhausted quickly and hence, this mathematical approach is an ill-fit with different constraints and the resources available on target hardware.

The iterative research and tweak in design choices has led to the possibility placement technique which is insensitive to the layers or type of connections in a neural network to allocate resources and converge to a valid placement.

4.3.1 Weighted Adjacency matrix

Using the pre-trained model or synthetic neural network description, the IR produces a graph which is characterised using an adjacency list. Another possibility is using an adjacency matrix of the graph of neural network, which can directly co-relate to the placement of weights for different input and neurons on the hardware. This is visualised in Figure 4.2. An important point to be noted here is that a straight forward adjacency matrix may look like a viable option at first glance, however, it is not anywhere close to a valid placement solution.

An important observation to be made is the size of the matrix in each case. In Figure 4.2(a), the size is 38 x 38, however in Figure 4.2(b), the size is 128 x 128, which is more than enough to fit a network size of 8 x 32 Self-Organizing Map architecture on target hardware. It is clear that one is not same as the other and hence a weighted adjacency matrix cannot be directly put on hardware. This enforces the creation of a highly intricate placement technique.

4.4 Possibility Placement Technique

This section discusses the placement framework proposed to meet certain objectives which have been reached upon with iterative design choices in this thesis work. A placement technique is introduced to run an application consisting of neural networks on the neuromorphic target hardware. The mapping algorithm is designed with careful consideration of numerous complex constraints arising from hardware design to allocate the resources and arrive at a valid and legal placement solution.

Based on various hardware constraints and the format of intermediate representation of the neural network scheme as introduced in Section 4.1, the mapping algorithm is devised to meet the following objectives.
The primary goals of the placement technique proposed in this thesis work are:

1. Optimality of the solution
    - Obtaining valid mapping
    - Avoiding resource overlap
2. Valid placement within certain time-frame

4.4.1 Methodology flow

In this section, the possibility placement mapping method will be discussed in detail. The working flow as illustrated in Figure 4.3 is discussed below:

1. The pre-trained neural network is given as an input, which is converted into an intermediate representation. GraphML format is chosen to have interoperability and flexibility in converting to a graph representation. It is a standard and supported by a lot of graph libraries.
2. Obtained GraphML is graphically represented as an adjacency list which serves as an input to the mapper. The adjacency list makes sure that all the associated edges to a single node in the graph are placed at a stroke.
3. Depending on the target platform, hardware parameters are given as arbitrary inputs to the mapper.
4. Possible Placement matrix is generated using the hardware parameters which is essentially the search space of the mapper.
5. An empty container for physical neurons and input resources is created which holds the mapped address of hardware resources.
6. Depending on the input graph parameters and the global constraints, initial feasibility check is carried out. If this condition is met, then the detailed mapping will begin.

7. Breadth-first search traversal technique is employed while allocating the nodes onto physical neurons and edge weights on to synapses.

8. An initial random input resource is chosen from the available resources and an initial physical neuron is chosen randomly from the available resources.

9. Numerous checks will result in address updation of physical neurons and input resources adhering to the strict rules based on the constraints.

10. Depending on the placement stage, legality checks are carried out to ensure there is a non-overlapping solution produced.

11. After passing all the checks and if the end of adjacency list has been reached, the final connectivity matrix is generated using the mapped address of physical neurons and input resources. The compile-time generated mapped addresses and weight connectivity matrix can be infused into a configuration file which acts as an input to the hardware.

12. In addition to connectivity matrix, delay profiles and resource utilization estimates are obtained for further evaluation.

13. To obtain the spread of the network on the neurosynaptic array, congestion clusters are obtained in addition to noise points, which are co-relatable to congestion of resources on the hardware.

4.4.2 Re-Configurable strategy

This section is omitted due to confidentiality purposes.

4.4.3 ALAPIN Mapper

To find a feasible and valid solution for a given neural network adhering to the hardware imposed constraints, Alapin Mapper is proposed. It is an iterative constructive placement strategy with semi-stochastic approach to allocate resources. The proposed algorithm is unique because of its insensitivity to the SNN topology, its reconfigurability, and moreover solution to an overlap-free and valid placement within a decent time-frame.

The randomness in the algorithm is introduced to create an exploratory space for different placement solutions possible. Having a fully random way of allocating the processing elements on the hardware can lead to an invalid placement and saturation of utilizable resources. Certain constraints like network mapping constraints and connectivity constraints have to be respected in order to have a functioning neural network when configured and run on the hardware. This necessitates to not randomize the allocation of resources completely and in turn, augmenting the complexity of resource

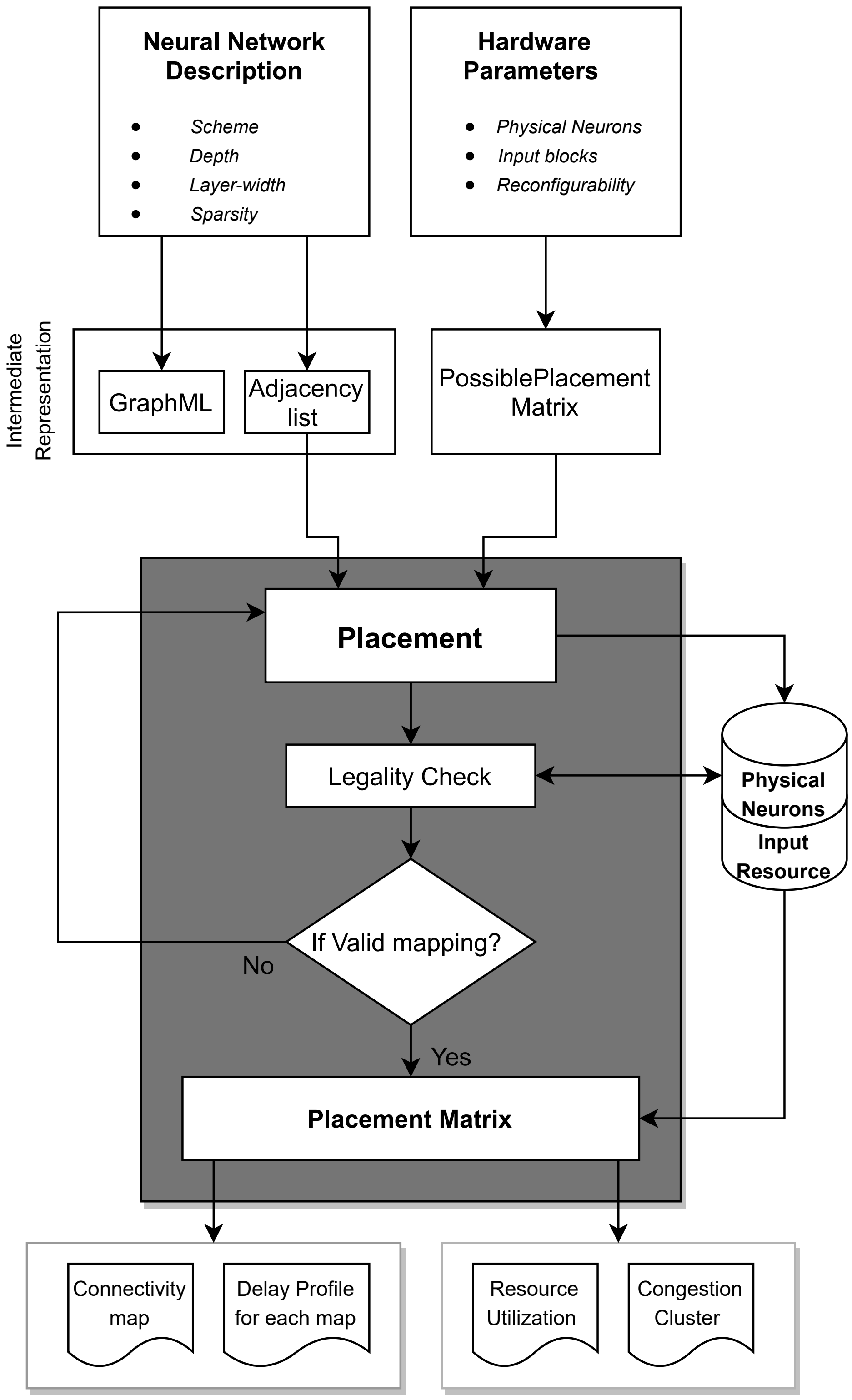


Figure 4.3: ALAPIN Mapper flow diagram

allocation and finding a valid solution quickly.

Algorithm 1: psblPlacement Algorithm

```
Input: Adjacency list, Hardware parameters, Node count[input layer of neural network]
Result: Hardware Connectivity matrix, Mapped address of PEs
1  psblPlcmt Matrix generation;
2  plcmtMatrix = [0];
3  $Mapped_{py\_neuron}$ = [];
4  $Mapped_{ip\_signal}$ = [];
5  while True do
6      for $Node_{source}$ in adjlist do
7          for $Node_{target}$ in adjlist do
8              if $Node_{target} \in Mapped_{py\_neuron}$ then
9                  Update address of py_neuron;
10                 if $Node_{source} \in Mapped_{py\_neuron}$ then
11                     if $Node_{source} \in Mapped_{ip\_signal}$ then
12                         Update address of ip_signal;
13                         Check legality (ip_signal, py_neuron);
14                         Search ip_signal in assigned input block;
15                     else
16                         Update address of ip_block;
17                         Search ip_signal in assigned input block;
18                     end
19                 else
20                     if $Node_{source} \in Mapped_{ip\_signal}$ then
21                         Update address of ip_signal;
22                     else
23                         Get random address of ip_signal;
24                     end
25                 end
26             else
27                 if $Node_{source} \in Mapped_{ip\_signal}$ then
28                     Update address of ip_signal;
29                     Get random address of py_neuron;
30                     if py_neuron is illegal then
31                         Search ip_signal in assigned input block;
32                         Update random address of py_neuron;
33                         Update address of ip_signal;
34                     end
35                 else
36                     if $Node_{source} \in Nodes_{Inputlayer}$ then
37                         Get random address of ip_sig ,py_neuron;
38                     else
39                         Allocate unique input block;
40                         Choose random ip_signal in assigned ip_blck;
41                         Get random address of py_neuron;
42                     end
43                 end
44             end
45         end
46         Check legality(ip_signal, py_neuron);
47         Update address of $Mapped_{py\_neuron}$;
48         Update address of $Mapped_{ip\_signal}$;
49         Populate plcmtMatrix with weight value;
50     end
51 end
```

Mapping procedure starts with first source node in adjacency list and maps it randomly on available input resources and available physical neurons. This process is carried out until a discovered node is hit in the neural network graphical representation. After which, depending on previously allocated resources, the same resources are re-allocated to the discovered nodes. When the hidden layer is reached, the checks take place to put them in either mapped py_neuron or mapped ip_signal category. Depending on if it has feed-forward or recurrent connections, the allocation of resources take place to either random resource or a previously allocated resource to meet the hardware constraints and the functionality of neural network is kept in check. As soon as the adjacent nodes of hidden layer are discovered, the rules change and the stringent local constraints needs to be met to converge to a solution quickly. The local constraints discussed in hardware constraints section (omitted due to confidentiality purposes) are transformed into satisfiability rules and employed in the algorithm as presented above.

4.5 Conclusion

The possibility placement method and the unique algorithm proposed in this chapter are in line with achieving the objectives of this thesis work. The algorithm's performance is dependent on numerous factors. Hardware constraints, network size and the resources available, concurrently dictate time-complexity of the algorithm. The nature of the target hardware (i.e. neuron model and synapse flexibility) influences the feasibility of mapping to a greater extent than just the global constraints.

Part III
Results

Experiments 5

This chapter narrates and discusses the experiments performed to evaluate the quality of proposed mapper and the feasibility of mapping different schemes of spiking neural networks on the target hardware. Section 5.1 gives an introduction to the goal of the experiments. Section 5.3 provides details on experimental set up, choice of parameters that provide insights into different aspects of mapper through neural network topologies and sizes. The experiments in Section 5.6 investigates the resource utilization demanded by each candidate SNN to evaluate the best fit on underlying neuromorphic hardware for a specific use-case in AI application.

5.1 Evaluation Objectives

The objective of this thesis is predominantly focused on creating a technique, more precisely, a mapping framework which maps the NN topology on the target neuromorphic platform. The experimental objectives are as follows:

1. **The feasibility of mapping a candidate SNN on target hardware.**

   In the light of this research question, the experiments are performed on chosen candidate SNN topologies which are concomitant to real-world applications.

2. **The time-to-solution of the mapper for surveyed spiking neural network topologies to qualitatively measure the performance of the proposed mapper**

   Major experiments are designed to evaluate the influence of different parameters of the neural network topologies on the solution time and therefore, performance of the proposed mapper.

3. **The resource utilization on the hardware prescribed by each candidate neural network architecture**

   Depending on the literature survey, experiments are designed to garner sufficient evidence on largest network sizes of the broad range of spiking neural networks that are a possible fit on the target hardware. A collateral advantage of performing the experiments with above mentioned objective is to report the amount of resources demanded by the SNN architectures which directly influences optimality in choosing the neural network for an application focused on low-area or high-performance etc.

5.2 Evaluation Methods

The proposed algorithm for mapping the SNN topologies onto a given configuration of the hardware needs to be evaluated. There are two ways of evaluating the proposed mapper.

1. Efficiency of mapper

2. Efficiency of mapping

5.2.1 Efficiency of proposed Mapper

To evaluate the performance of the mapper, a more detailed experimental analysis is carried out. Since, the goal of the thesis is to perform mapping of various spiking NN schemes on the target hardware platform, it is crucial to evaluate how sensitive the mapper is to a given neural network scheme. The following methods are used to evalaute the performance of the mapper.

- **Time-to-solution**
  The evaluation of the proposed methodology is carried out by investigating the computational complexity of the algorithm. Since the proposed mapper is semi-stochastic in nature, the time-to-solution for every map produced for the same network varies. Hence, the average time-to-solution for 5 iterations of the same neural network is measured to report time complexity of the algorithm.

- **Hardware Resource Utilization**
  Hardware resource utilised for each network scheme of different size is varying and the solution time's dependency on these hardware resources is evaluated. Different resources needed for different network schemes and how they are interdependent on each other and what restrictions they impose on the mapper is evaluated. The rationale behind this is to investigate a hardware resource space for candidate neural networks so as to indirectly estimate the number of cores needed in future to map a huge neural network on a multi-core architecture.

5.2.2 Efficiency of mapping

The proposed mapping technique is capable of producing 'n' mapping solutions for a given network. Due to the randomness, the value of n is quite large. An attempt has been made to create an exploratory space with 'n' placement solutions for the same network. The cost of each produced mapping is measured using the delay profiles and spread of the network. However, the optimization of cost function is not carried out in the present work.

5.2.2.1 Spread of the network

Spread of the network mapped on the hardware is a quantitative congestion metric. Higher congestion is not exemplar in usual cases with respect to VLSI or multi-core processor architecture. The approach taken in this thesis work is simple and abstract.

Density based clustering algorithm [48] is employed. A single value of epsilon value is set to measure the number of clusters based on the hardware connectivity matrix generated from the mapper. The noise points in the algorithm also provide information about the outliers.

Estimated number of clusters indicate the quantitative metric of congestion. Higher the number of clusters, lower the congestion score indicating a good spread of the network on the hardware, whereas, lower the number of clusters, lower the congestion score indicating high congestion and very low spread of the network on hardware.

5.2.2.2 Delay Profile

The path delay in the neuro-synaptic array is modelled to see what is the impact of each mapping for the same network in different run. Apparently, the delay increases or decreases depending on the way a network is mapped onto the hardware. Path delay is modelled as follows:
t_p: propagation delay to synapse
t_{sc}: Wirelength delay to physical neuron
t_{buffer}: buffer imposed delay
rf: Re-configurable factor
The delay inside the neuro-synaptic core is calculated as shown below:

$$T_d = t_p \times j + t_{sc} \times [(ip_b lck \times rf) - i] \tag{5.1}$$

where,
j = $mapped_{pscl\_neuron}$
i = $mapped_{ip\_signal}$

$$T_b = (j/32) \times t_{buffer} \tag{5.2}$$

$$TotalPathDelay = T_d + T_b \tag{5.3}$$

Since the value obtained using Equation 5.3 is large, a normalised value is calculated which is as shown below:

$$NormalisedDelay = TotalPathDelay \times 1e - 4 \tag{5.4}$$

5.3 Experimnetal Setup

The effectiveness of proposed mapping algorithm is illustrated through different experimental designs meeting the objectives proposed in section 5.1. In this thesis work, all the experiments are carried out on Intel i5-1035G1 4-core CPU machine with processor base frequency 1.00 GHz and 8GB memory.

All experiments are performed on the machine and only software implementation and verification is carried out in the present work. The hardware configuration used throughout the experiments are as per the data available for the target hardware.

Table 5.1: Candidate SNN architectures: The experiments performed in this thesis work involve candidates mentioned below, whose architecture is assumed to be spiking in nature. Note: *marked are not inherently spiking neural architectures and hence assumed to be one in this work.

Candidate	ID	Neural Network Architecture
1	FFN	Feed-Forward Network*
2	AE	Undercomplete Auto-encoder*
3	SOM	Self-Organising Maps
4	RNN	Recurrent Neural Network
5	LSM	Liquid State Machines
6	LIN	Lateral Inhibitory Network
7	CHL	Complete Hidden Layer

5.3.1 Experimental Dataset

Ideally, going by the primary theme of a challenging problem this thesis is trying to address, the end goal is to map a pre-trained neural model that accomplishes a classification / prediction application. Synthetic networks are generated which closely replicate the real-world neural networks employed for AI applications.

The synthetic networks for different spiking based applications are generated in Python covering all the candidate SNN architectures with sufficient flexibility to modify the network sizes. The neural network architectures are parameterizable on various parameters like network depth, layer-width, sparsity in the network, and few others which are architecture specific and will be explored in detail in further sections.

5.3.2 Parameter space

The experimental designs and the outcome of each experiment is dependent on the parameter space, which is meticulously designed and tuned based on the literature survey of candidate architectures as presented in Chapter 3 and Table 5.1. Since the experimental dataset was not readily available, this led to the generation of synthetic networks for all candidates through an automatic network generator tool, devised in Python exclusively to smoothen the experimental phase in this thesis. A collateral advantage is the flexibility to tune various parameters to investigate their influence on the hardware and on the proposed mapper.

A general notion while constructing a neural network architecture; especially the topology, excluding the nature of neurons, synapses and mathematical functions involved; circumscribes carving out intricate details as follows:

1. The problem itself, which usually correlates to the number of inputs and outputs needed.
2. Quality and quantity of available data

The most important parameters sweeped in the experiments performed on different candidate architectures are mainly:

- Depth: Total layers in a neural network
- Layer-width/Breadth: Number of neurons in a particular layer
- Sparsity: Number of connections between two layers in network

There are few non-trivial parameters pertaining to only some of the candidate architectures, which will be discussed in design section specific to those network schemes.

5.4 Mappability check

The proposed mapping technique manoeuvred the exploratory research conducted on different schemes of neural network architectures for feasibility check on the target hardware. The mapper already scrutinizes the process of mapping and convergence happens if and only if the hardware constraints are satisfied. However, a proof checker or an analytical checker is devised in this thesis work, which double checks the functionality of neural network before and after mapping on the hardware using the generated connectivity matrix with precise information about mapped address of PEs.

A visualization for the connectivity matrix produced from mapper is generated using Python. An example of the neural network against the connectivity matrix is illustrated in Figure 5.1. The network demonstrated in Figure 5.1(a) is a 3 layered RNN with various forms of recurrences. RNN is chosen as it covers the feed-forward, recurrence within and from other layer and self-loops. The hardware size is chosen arbitrary in this case to have clear resolution and to illustrate the behaviour of the mapper in case of tight constraints and the allocation procedure. The Figure 5.1(b) is a visualisation of the connectivity matrix from the mapper. It is a grid replicating the hardware for a size of 13x13. The green dots essentially indicate the synapses occupied, the horizontal rows relate to the mapping of input PEs and vertical columns are in direct co-relation with the total physical neurons occupied. This can be clearly verified from the RNN with total 7 nodes excluding the input layer in Figure 5.1(a).

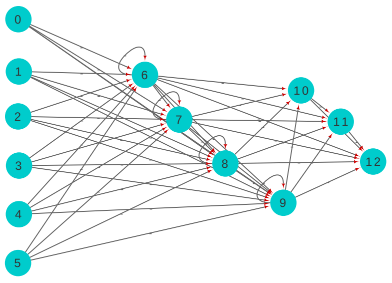
In a similar way, the networks generated for each of the candidate neural network architecture and an examination is carried out assuming the neurons in these NN are spiking architectures. The exploratory research led to the mappabiltiy results as shown in Table 5.1, which is divided into three categories.

1. **Possible**

   All the listed NN architectures and their variations are possible to be mapped onto the target hardware. The candidate neural networks are mainly assumed to spiking in nature, specifically for FFN, and AE for experimental purpose. The remainder of the architectures like SOMs, LSM, LIN and associative memory in case of CHL inherently have spiking behaviour.

2. **Potential**

   It is inherently difficult to realise the temporal dynamics of RNNs using spiking neurons as stated by Diehl et al[50]. Assuming the threshold neuron models on target hardware, the implementation of spiking RNN can be accomplished. However, Long Short Term Memory (LSTMs), the advanced versions of RNNs cannot



(a)

(b)

Figure 5.1: Mapped hardware connectivity matrix: (a) A recurrent neural network (RNN) with recurrences from layer n to n-1, recurrence within the layer and self-loops. (b) **(Omitted due to confidentiality purposes)** Visualization of the mapped results on a 13x13 hardware size to illustrate the occupation of PEs on hardware alongside the NN co-ordinates mapped onto respective PEs

be implemented in a straight-forward manner. These recurrent architectures involve non-linear tanh and sigmoid activation functions to realise store-and-release mechanism. Another recurrent variant is Gated Recurrent Unit (GRUs) involving forget and gate functions. The realisation of these non-linear activation functions in spike domain is challenging at the moment given the neuron model on the target hardware. Therefore, mappability of LSTMs and GRUs is potentially possible with certain upgrade in the hardware.

3. **Unexplored**

   Having based the argument of considering few of the ANNs for mappability check on the assumption of accomplishing ReLu (rectified linear units) activation functions through spiking neurons doesn't necessarily hold for more advanced ANNs like Convolutional Neural Networks. Foreign architectures like CNNs with convolutional kernels remain unexplored in this thesis work. Recent advancement in AI research and computer vision led to the introduction of Generative Adversarial Networks(GANs) for text-to-image description, generating very high-resolution images etc. GANs too remain unexplored in this thesis work.

After a preliminary investigation on which are the neural architectures mappable onto the target hardware, further investigation on the performance of the candidate spiking neural networks and the qualitative measure of the proposed placement algorithm will be discussed in next sections.

5.4.1 Architectural complexity measures of neural networks

A barrier in understanding the architectural complexity is the lack of a general definition of the connecting neural architecture. This section introduces a brief overview on the exploratory space of the neural networks being investigated to measure the performance of the mapper. Design space is categorised into feed forward networks and networks

Table 5.2: Mappability Results: Neural Network architectures summarised into three categories based on the exploration carried out for feasibility check on target hardware

Possible	Potential	Unexplored
FFN	LSTM	CNN
AE	GRU	GAN
SOM		
RNN		
LSM		
LIN		
CHL		

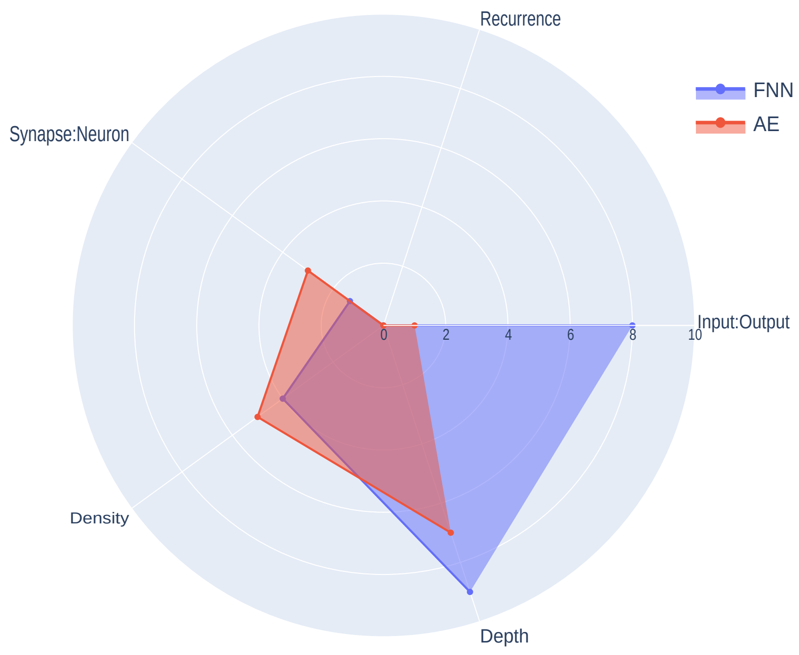


Figure 5.2: Architectural complexity measures of candidate networks without recurrence

with recurrence considering all of them as spiking architectures in nature showing the coverage on different aspects like:

1. Input:Output - It gives an impression on number of neurons in the input layer in comparison to output layer. Higher the ratio, more the number of inputs to outputs depending on the application. The ratio is 1 for AE, and CHL. It is very high for FFN and RNNs, whereas, the ratio is less than 1 for SOMs and other spiking schemes as seen from Figure 5.2 and 5.3.

2. Synapse:Neuron - This ratio gives a fair idea of the total synaptic edges in the

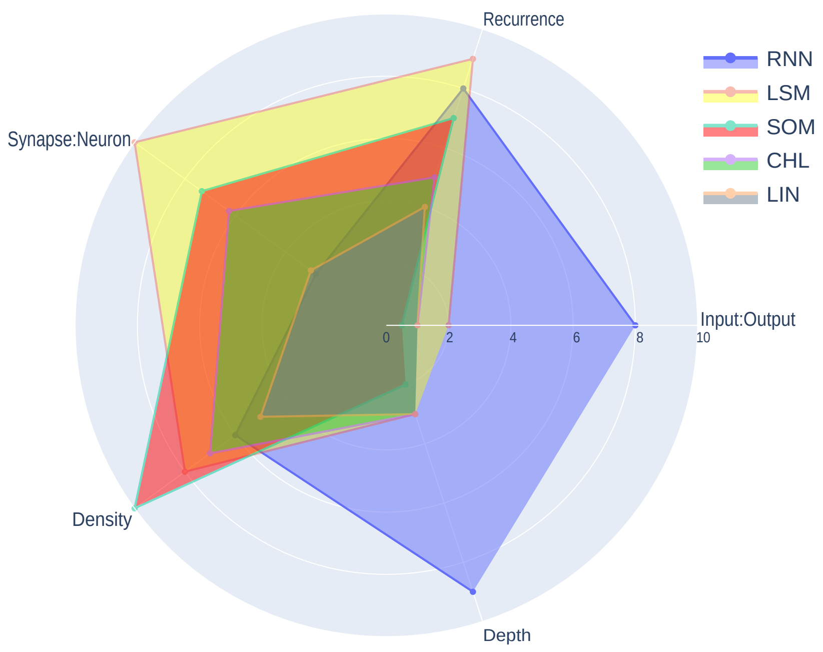


Figure 5.3: Architectural complexity measures of candidate networks with recurrence

architecture for given number of neurons. To estimate the values for all the candidates, more or less same neurons are considered, to see the resulting variation in synapse count. Topologies like SOMs, CHL and LSMs(highest) have high synapse to neuron ratio compared to other NNs as it evident from Figure 5.3. The impact of higher ratio of synapse:neuron on solution time is validated in further sections.

3. Recurrence - Another quantitative measure is the recurrence in the networks, it gives insights into complexity imposed on placement of different resources adhering to various connectivity constraints. Ideally, the recurrence value is clustered around 70-75% for RNNs as they have self-connections, recurrence within the layer and from their neighboring layers in addition to the recurrence depth. LSMs have highest recurrence value due the random nature of connections in the liquid layer. Self-organising maps do not have self-connections and only recurrence within the layer, thus an estimate of 50% is made. Lateral inhibitory networks have a reduced recurrence with less synapse to neuron ratio as seen in Figure 5.3. Therefore, a clear overlap is seen with the rest of the architectures and any experiments performed will be redundant.

4. Density - This relates to the kind of edges or synapses a neural network has as a default setting. The connections can be varied in number for experimental

Table 5.3: Parameter Space - FFN: Encoder style architecture of feed-forward networks designed with linear depth variation, input layer-width varying as 2^n, and conglomeration of all the layer width in the network is represented using (layer-width)LW-Difference

External Inputs	Network size	Depth	LW-Difference
2^n, n =1,2,3,..	Encoder style architecture	1,2,3,4,5,..	2^n, n=1,2,3..
8	8-4-2-1	4	1
32	32-16-8-4-2	5	2
128	128-64-32-16-8-4	6	4
256	256-128-64-32-16-8-4-2	7	4

purpose through a sparsity factor, however, topologies like SOMs have default full connections than the rest of the candidates as seen in Figure 5.3.

5. Depth - It is the number of layers involved in the neural network. SOMs, LSMs, CHL and LINs, usually have a two or layer depth, whereas, FFNs, AE and RNNs have deep networks.

5.5 Quality of Mapper

The quality of the proposed algorithm is evaluated through the time complexity for each of the candidate networks by performing a parameter sweep through various experiments. In this section, mainly the candidate SNN schemes will be experimented through the design as per literature survey and parameter sweep which is subjective to each candidate. This section will have a theme of experimental design of the candidate SNN, followed by results and discussion of the outcome.

5.5.1 Feed Forward Networks - Design

In this thesis work Encoder style FFN, which are realistically employed in numerous applications like classification and regression, are investigated. The general theme while designing the experiments in this thesis work has been to resemble the realistic models as closely as possible. By doing so, an unclouded projection of challenges in terms of mapping, hardware resources needed can be obtained.

We try to evaluate the execution performance of mapper for the encoder style architecture(FFN) by varying the depth parameter to garner the knowledge of how deep neural networks can be fit on a neuromorphic crossbar array. The number of nodes in a particular layer is referred to as layer-width/ breadth interchangeably in the course of experiments. Since, there are numerous layers involved in FFN, it is challenging to show the variation of LW(layer-width), hence, a compact representation is opted which is varied as 2^n. Typically, if we look at Table 5.3, starting with any external input, if the depth is known, the LW-difference can be achieved if we go decreasing it in 2^n. Another hint is the co-relation of output layer in the network to the LW-difference, they are in tandem.

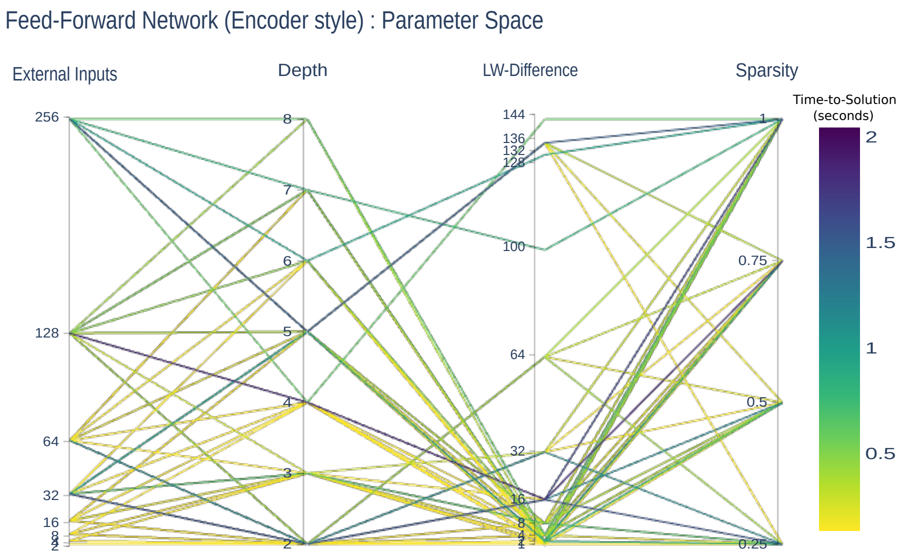


Figure 5.4: Parameter Space for Feed Forward Networks sweeped over different parallel coordinates to show the performance of mapper in terms of time-to-solution(seconds)

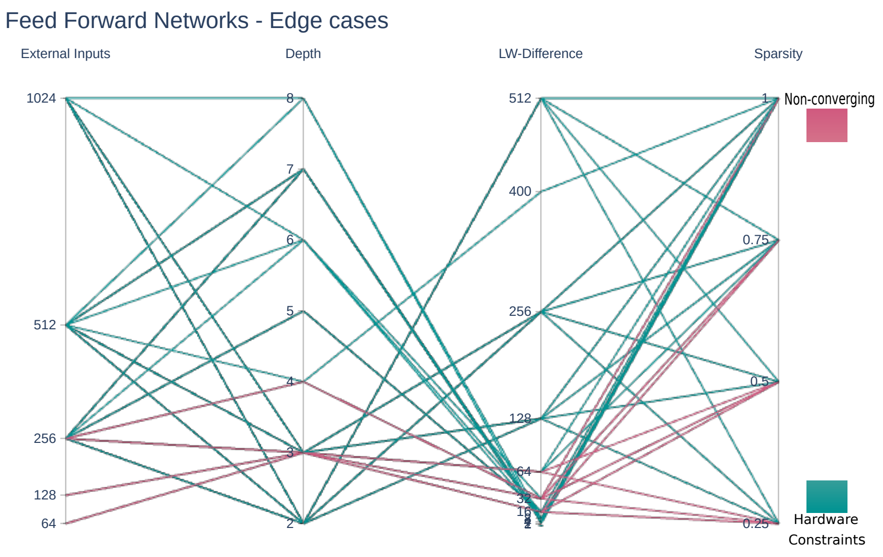


Figure 5.5: Parameter space for Feed Forward Networks showing different combinations that do not converge to a solution, either due to hardware constraints or limitations of the mapper

5.5.1.1 Results

The simulations of the designed synthetic FFN are carried to obtain a solution using the mapping algorithm. The hardware architecture size is kept constant throughout these experiments. The experimental data gathered are categorised into 4 parallel co-ordinates, external inputs, depth, LW-difference and sparsity as can be seen in the Parameter space in Figure 5.4. The colorbar indicates the increasing solution time. The reported time-to-solution for the highest possible network (256 external inputs, 5-deep and a LW-Diff of 128) to be mapped on hardware is about 2 seconds. This high solution time is the direct reflection of exhausted neurons and stringent connectivity constraints.

Every co-ordinate axis signifies the largest and smallest datapoint. For an encoder style feed forward network, depth of 8 is possible with usually as small layer width variations as possible. The cluster on LW-Difference in addition to the color shows that solution is possible in the range of 0.5 to 1 seconds. It is quite obvious that from any starting point on External inputs co-ordinate, with any depth in the network, the LW variation is tending towards a smaller number. Because, as the number increases, the convergence does not occur due to global constraints.

The contrasting color band is seen in Figure 5.4. Smaller network sizes with smaller LW variation and with increasing density (sparsity –>1) converge quickly, within milliseconds. Interestingly, when the external inputs are huge, (which is generally the case for realistic neural networks) i.e. higher depths in combination with higher external inputs, the performance of the mapper is pretty decent as the convergence occurs within 2 seconds. There is an additional skip in LW-variation considered to see the performance of the mapper in terms of meeting the global constraints. The network with ext inputs as 256, the first hidden layer is set to 64, giving rise to a LW-Difference of 144 or 136 etc. However, the convergence in these cases is still possible, only that the time-to-solution lies on the purple end of the spectrum.

Other side of the story is depicted in Figure 5.5, where the datapoints in the parameter space are shown when the convergence is not possible. The parallel co-ordinates remain the same, however it is a binary value 0 or 1 for the huge networks, mostly which are edge cases for target hardware size. The rose band signifies combination of co-ordinates for which the solution doesn't converge. The teal band portrays a larger dataset that are not possible to be mapped due to hardware constraints failing at a global level.

5.5.2 Auto-encoders - Design

An important variant of FFN is auto-encoder, mainly employed in image recovery or image enhancement applications. The most generic architecture i.e. undercomplete autoencoders are chosen to see their performance on the mapping algorithm and how they fit on the hardware with a special architectural complexity of input:output as 1. In a realistic application, the undercomplete architectures employed have a code size, that is the bottleneck layer, with lesser number of neurons in comparison to rest the encoder/decoder layers.

The evaluation is carried out by varying the depth and layer-width of the auto-

Table 5.4: Parameter Space -AE: Undercomplete architecture of auto-encoders are designed, depth of the AEs is generally odd-numbered, input layer-width varying as 2^n, and the code size is varying as 2^n, and sparsity lies between 0 and 1.

External Inputs	Network	Depth	Code size	Sparsity
2^n, n =1,2,3,..	Undercomplete architecture	3/5/7/9..	2^n, n=1,2,3..	0 to 1
32	32-16-8-16-32	5	8	1
64	64-32-16-8-16-8-32-64	7	8	1
128	128-64-32-16-8-16-32-64-128	9	8	1
256	256-128-64-32-64-128-256	7	32	1

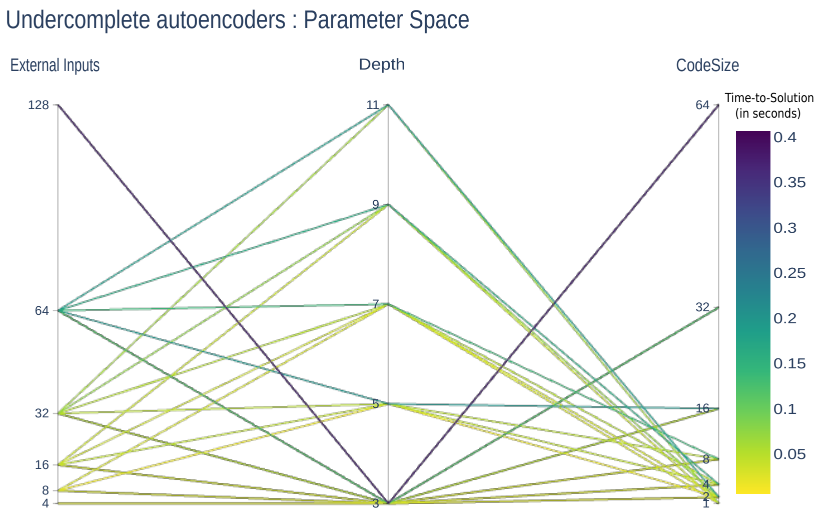


Figure 5.6: Parameter Space for Undercomplete Autoencoders sweeped over different parallel co-ordinates to show the performance of the mapper in terms of time-to-solution(seconds)

encoders, keeping full connections in the network (sparsity =1). The depth is varied in odd numbered fashion as it is the nature of undercomplete architecture of AE, having a converging and a diverging trend. Therefore, the depth variation is 3, 5, 7, or 9 depending on what is the max fit on the target hardware. The variation of layer-width is again central to the fact how neural networks are architected in realistic application, in the fashion of 2^n, with n = 1,2,3,4,5...

5.5.2.1 Results

Figure 5.6 depicts the time-to-solution for the different network sizes of an auto-encoder. It is clearly observable that, as external inputs increase, the solution time increases. The maximum data-points on each axis is corresponding to the largest network feasible on target hardware of size $M \times K$ and its co-relation with the highest time taken to converge to a solution.

Table 5.5: Parameter Space -SOM: Spiking architecture of self-Organizing Maps/Kohonen networks are designed, input and torus layer are considered making depth as 2, input layer-width varying as 2^n, the layer-width is varying as 2^n, and sparsity lies between 0 and 1.

External Inputs	Network	Depth	Breadth	Sparsity
2^n, n =1,2,3,..		2	2^n, n=1,2,3..	0 to 1
8	8 - 8x8	2	64	1/0.75/0.5/0.25
8	8 - 16x16	2	256	1/0.75/0.5/0.25
32	32 - 8x16	2	128	1/0.75/0.5/0.25
64	64 - 8x8	2	64	1/0.75/0.5/0.25

The entire parameter space shows the tried networks that have a convergence value of 1. There are few edge cases which do not converge to a solution either due to limited hardware resources or failure to meet constraints imposed by the neurosynaptic crossbar array. An observation of the dipping values or a cluster formed with low depth values, and low codesize indicates that as we go higher on these co-ordinate axis, the feasibility of a solution is bleak. Having deep AE with high external inputs on hardware is constrained by the available physical neurons. Even though other input resources and synapses are abundantly available, the shortage of neurons resists the implementation of huge networks.

Another observation is the difference in time-to-solution for the largest networks for feed-forward networks and auto-encoders is about 80% less. This can inferred from Figure 5.2, where the difference between FFN and AE is clearly visible in terms of input:output ratio of FFN almost 9x AE. The primary reason for not having bigger networks in comparison to FFN can be best reasoned out as due to the input:output ratio of AE as 1, which makes the design of bigger networks extremely challenging with exhaustion of physical neurons quickly.

5.5.3 Self-Organising Maps - Design

Self Organizing Maps are an efficient way of performing unsupervised learning by forming ordered mapping, a projection of a set of given data into 2-dimensional grid. SOM's are heavily employed in speech recognition applications. The external inputs are arranged to increase in 2^n fashion. Keeping in mind the grid formation in the torus layer of the self-organizing map, the hidden layer or the SOM layer is constructed to be a grid. The depth is kept constant throughout all experiments, assuming an input layer and SOM/Torus layer as the hidden layer in the network. Sparsity is varied between 25% to 100% i.e.(less/sparse connections to dense connections). Table 5.5 gives a complete overview of the design and knobs varied in this experimental setup. The connections from input to torus layer are all-to-all and the connections in torus layer are one-to-all for all the neurons forming an inhibitory layer of connections within the grid.

5.5.3.1 Results

The density of the connections from input layer to SOM layer is sweeped from 0.25 to 1. In this experiment, full connections are assumed in the torus layer without any

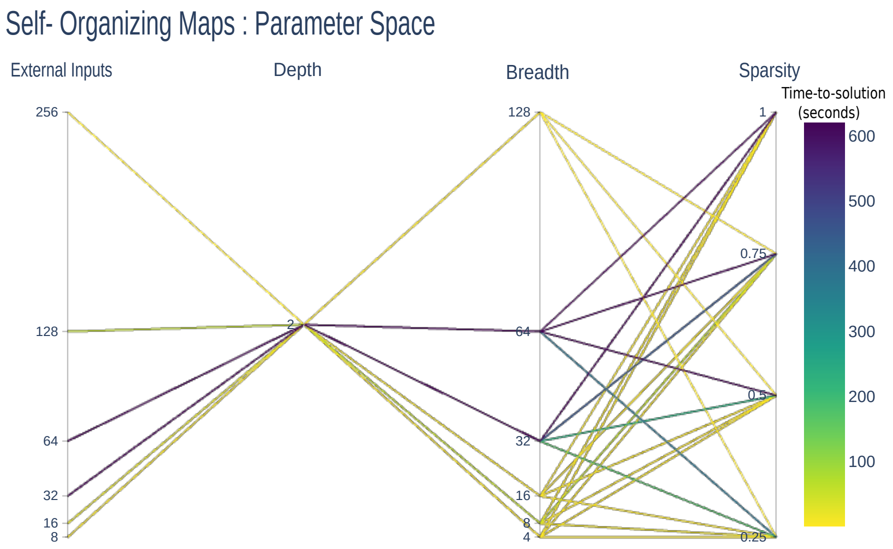


Figure 5.7: Parameter space for Self-Organizing Maps with four co-ordinate axis, keeping depth constant. The performance is measured in execution time to find a valid placement with different combinations

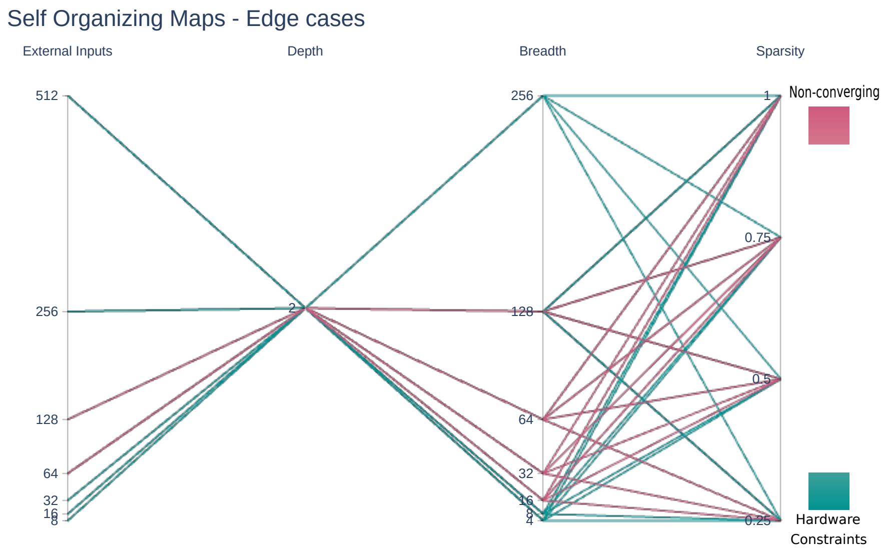


Figure 5.8: Parameter space for Self-Organizing Maps showing the combinations which couldn't converge to a valid placement solution

variation in sparsity in the torus layer. Figure 5.7 shows the parameter space for SOMs when convergence is 1 i.e. a solution is feasible on the target hardware. All the possible combinations tried to obtain a placement solution is illustrated using different axis co-ordinates.

The contrasting band shows that with smaller input size and highest value of breadth, the solution is possible and it converges within 10 seconds. However, as the external inputs are increased, the constraints on the mapper in terms of finding an input block for lateral connections especially influences the mapping time heavily, which goes upto 500-600 seconds.

Another observation is the color band seen in varying sparsity. The trend is seen on how the sparsity influences the solution time even with the large networks like 64- 8x8 with sparsity 0.25 and 0.5 with just an increase in 1000 edge connections. The sparsity variation projects which networks don't stand a chance to even converge to a solution, can essentially have a valid placement on the target hardware with limited connections (i.e less density).

There are networks which do not converge to a solution, and Figure 5.8 shows networks which fail to converge on the target hardware. These network sizes impose great difficulty on mapper in terms of finding a solution. The teal band illustrates the failed solution due to limited hardware resources, and the rose band shows the limitation of the random mapper. Networks with 128 input neurons, with a breadth size of 16, 32 should be possible, but there is no convergence with the proposed mapper. However, smaller external inputs with smaller and higher breadth sizes fail due to hardware constraints imposed by synaptic connectivity constraints in the target hardware.

5.5.4 Liquid State Machines Design

Liquid state machines are SNNs with unsupervised learning theme. Depending on the application, there are two types of LSMs designed and discussed further. As the LSMs are mostly employed in audio classification or speech recognition etc, need very less neurons at the output, typically 2 to 4[53]. The most generic LSMs are assumed and evaluated for reporting the performance of the mapper. In case of generic LSMs, the inputs are varied in arithmetic progression and the liquid breadth is 10x the external inputs. To naturally reach the mark of 250 neurons overall, the external inputs are varied in arithmetic progression. Since there is no readout layer present in the generic architecture of LSMs, the output breadth is assumed to have neurons ranging from 2 to 8 depending on the application.

5.5.4.1 Results

Since, the hidden layer in these networks is usually of paramount interest and hence, their breadth is varied in accordance with the given external inputs and plotted as different co-ordinate axis in Figure 5.9. Only generic LSM's are plotted in this section. The largest possible network size with about 250 neurons, has high convergence time recorded as 160 seconds. A clear contrast in the color bands in the last three co-ordinate axis is notable. This signifies that higher liquid breadth dictates the solution time than the sparsity and output neurons. For smaller inputs, the liquid breadth is also in the

Table 5.6: Parameter Space-LSM: Two variants of liquid state machines are designed, input and output breadth are either power series or arithmetic progression depending on architecture, liquid breath is 10x inputs, depth varies depending on generic architecture, and liquid sparsity lies between 0 and 1.

External Inputs	Liquid breadth	Liquid sparsity	Readout breadth	Output breadth	Architecture
2^n, n =1,2,3,..	10x Ext inputs	0 to 1	2^n, n=1,2,3..	2/4/8	LSM+MLP
1, 13, 25, 37...	10x Ext inputs	0 to 1	-	2/4/8	Generic LSM

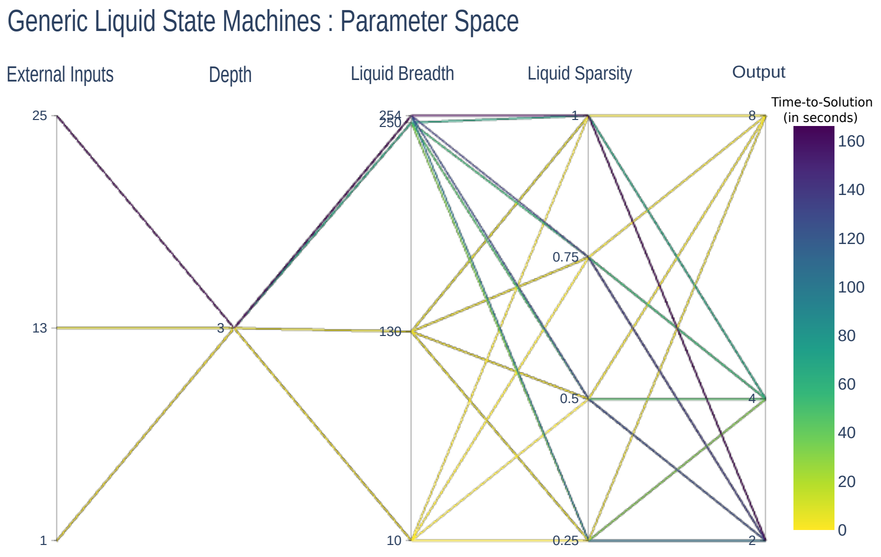


Figure 5.9: Parameter space for Liquid State Machines with five co-ordinate axis, keeping depth constant throughout. The performance is measured in terms of execution time to find a valid placement with different combinations

middle range of neurons (about 130) and sparsity variation doesn't have much effect on the solution time. A typical case of 254 neurons was experimented keeping a split of 64 and 190 nodes in the liquid layer, and it is reflected in terms of increase in the solution time to about 160 seconds from about 80 seconds. This split of having 64, with allocating nodes to mostly all neurons available on the hardware is the reason why there is 2x increase in the solution time in comparison to liquid breadth of 250.

5.5.5 Complete Hidden Layer - Design

Complete hidden layer is essentially a spiking variant of RNNs. Hidden layer is either a Hopfield network or a bidirectional associative memory mainly used in image enhancement applications.

Table 5.7: Parameter Space -CHL: Hopfield networks in binary mode and continous mode are considered for design of hidden layer. Architecturally, the cue inputs and retrieved outputs are generally of same length, Hopfield breadth is varied, Cue:Hopfield and Hopfield:Retrieved ratio are kept same, and have only two variations.

Cue inputs	Hopfield breadth	C:H ratio	Output	Architecture
2^n, n =1,2,3,..	2x Cue inputs	1	2^n, n=1,2,3..	Hopfield in binary mode
2^n, n =1,2,3,..	2x Cue inputs	0.5	2^n, n=1,2,3..	Hopfield in binary mode

The experiments are designed for CHL using Hopfield networks as the hidden layer in binary mode, i.e without self-connections. The input and output neurons are considered to be same. The input and output neuron values are varied in 2^n fashion. Depth is kept constant instead of any variations to stick to general guidelines of the network architecture defined for CHL. A sweep is done for C:H ratio, i.e. ratio of input nodes to hidden layer nodes and hidden layer nodes to output nodes is varied as 0.5 or 1 to see the affect of reduced nodes and connections in the network.

5.5.5.1 Results

The nature of architecture for CHL heavily restricts the number of combinations possible to be tried. Figure 5.10 shows the results of different networks for CHL and the time taken by the proposed mapper to produce a placement solution. A parameter sweep for hopfield breadth varying as 2x the cue inputs leaves us with not having network sizes of 64-128-64 or 128-256-128 possible to be mapped because of limited number of available physical neurons. This limitation leads to maximum of 32 on input and output co-ordinate.

The resulting solution time for the largest network size possible on target hardware is about 845 seconds, so far the highest time-to-solution among the rest of the candidate networks being evaluated. For all the small networks, irrespective of the C:H ratio (ratio of neurons from input to hidden layer and from hidden to output layer) with about few thousand synapses, the solution time is from milli-seconds to few seconds. Contrastingly, when the number of synapses increases in large networks, in addition to a tight ratio of input to hidden layer nodes i.e. from any number of input nodes to hidden layer with 64 nodes will certainly have a greater impact on the mapper to find a valid solution as the input connectivity constraints become overwhelming in combination with finding the valid synaptic resources on crossbar array. This explains the dramatic rise in solution time.

5.5.6 Recurrent Neural Network - Design

Recurrence is observed in all the spiking architectures like LSMs, SOMs and CHL etc. These architectures have recurrence within the layer. The experiments designed for RNNs are particularly to explore the effect of recurrence from layer n to n-1. Similar to FFNs, the depth is an important variable parameter in this experiment, this also necessitates the inception of LW-difference as breadth cannot be generalised to all

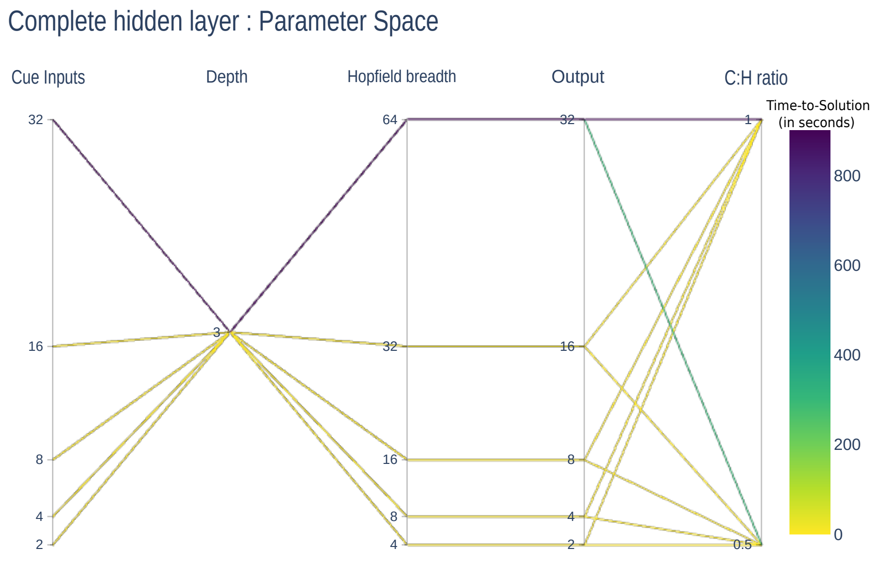


Figure 5.10: Parameter space for Complete Hidden Layer with five co-ordinate axis, keeping depth constant throughout. The performance is measured in execution time to find a valid placement with different combinations

Table 5.8: Parameter Space-RNN: Recurrence from layer n to n-1 is considered for design of RNNs. Recurrence sparsity is varied with depth, LW-difference and external inputs.

Ext inputs	LW-Diff	Recurrence Sparsity	Depth	Architecture
2^n, n =1,2,3,..	2^n fashion	0 to 1	3,4,5..	RNN with recurrence from layer n to n-1

layers. The recurrence occurring from one layer to another layer is controlled through a recurrence sparsity parameter.

5.5.6.1 Results

Figure 5.11 shows that having recurrence from layer n to n-1 impacts the depth of the network possible to be mapped on the hardware to just 5 layers with less LW difference. As the external inputs and depth in the network increases, the solution time increases because it becomes difficult to allocate input resources and recurrent connections in the same line of synapse. The solution time is in bound with about 23 seconds due to the fact that the network sizes possible with this kind of architectures are small and the worst hardware constraints and exhaustion of resources have not reached to have a very high solution time for valid placement.

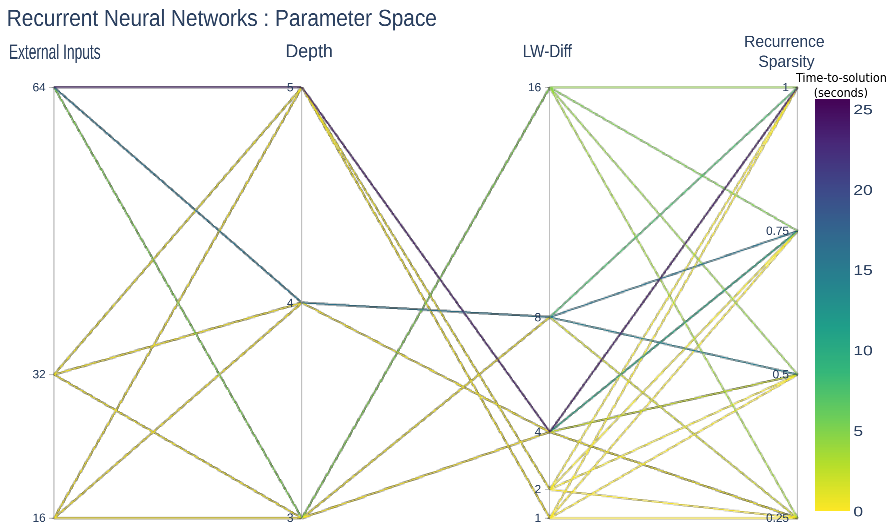


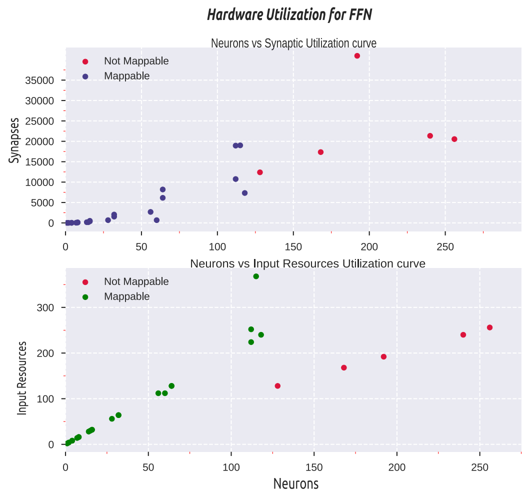
Figure 5.11: Parameter space for Recurrent Neural Networks with four co-ordinate axis. The performance is measured in execution time to find a valid placement with different combinations by varying recurrence sparsity, depth and external inputs

5.6 Estimate of Resource Utilization

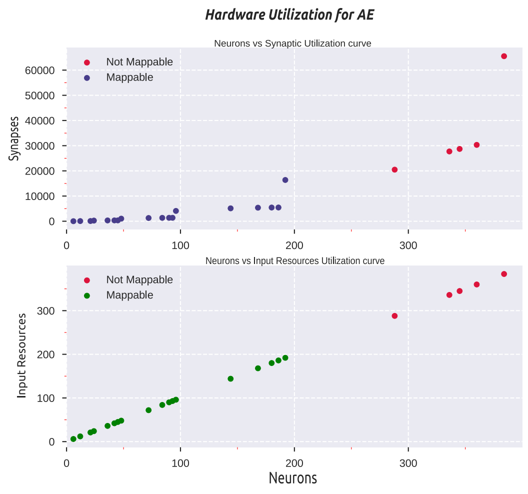
This section discusses the hardware resource utilization for the candidate networks to provide an evidence of the largest network sizes that can fit on target with available hardware resources. These experiments give an insight into the resources demanded by each of these candidate SNN architectures that can influence the choice of a particular neural network to run on the target hardware. This section also gives an overview on time taken to converge to a placement solution for variation of each individual hardware resource using the proposed algorithm. A trend of synaptic utility for networks with certain number of neurons, and varying the specific parameters is explored.

5.6.1 Feed Forward Network

The experiments designed as shown in Table 5.3 are used to garner the results of the variation of one hardware resource with another. By varying the FFN depth-wise and variation of sparsity in the network can be seen on Figure 5.12(a). With increase in the network size, as a result of adding layers and increasing the layer-width, leads to variation in the number of neurons that can be allocated on the target hardware. With increase or decrease in the number of connections, the synapse utilization varies on the hardware. The first window in the Figure 5.12(a) shows the neuron vs synapse utilization, and the extent to which there is a mapping solution possible. The red circles are the combinations which are not mappable because of the constraints at the input resources having a direct impact on the FNN's first layer nodes, leading to never



(a)



(b)

Figure 5.12: Hardware Utilisation: (a) utilisation of synapses and input resources against neurons for FFN are shown for combinations which are mappable and not mappable on the hardware (b) utilization of synapses and input resources against neurons for AE are shown.

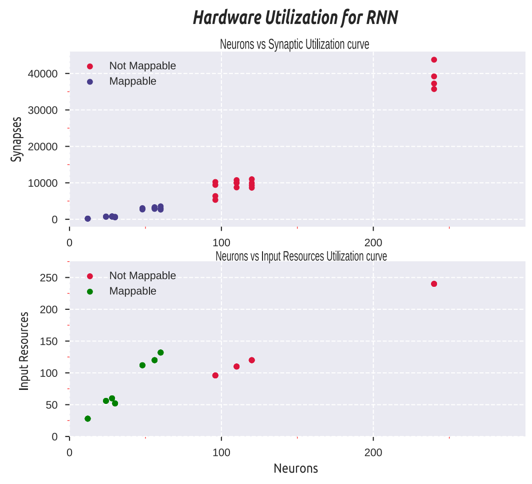
reach a count of full neuron utilisation on target hardware.

With increase in neurons, the synapse utilization also increases, the value where it stops, around 128 neurons, is the point after which the network sizes are not mappable as the global constraints do not satisfy. The second window in Figure 5.12(a) provides insights about the variation of input resources with increase in network size for feed forward encoder style architecture. The variation is linear and it is observable from the way the encoder style FFN are architected.

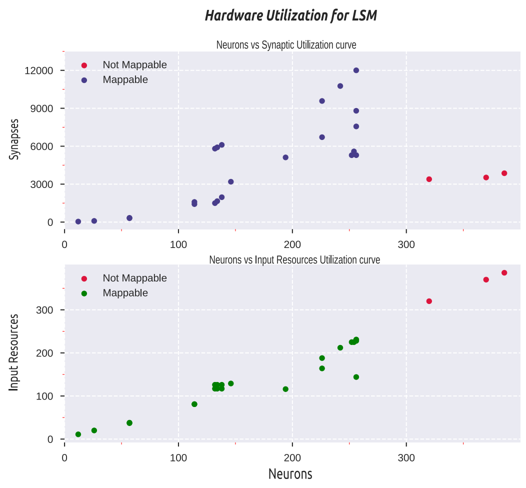
5.6.2 Autoencoders

The under-complete architecture of autoencoders have a special characteristics as it can be seen in the second window of Figure 5.12(b), where exact number of input resources are utilised as that of the neurons, making it perfectly linear. This can also be explained from Figure 5.2, where the input-to-output ratio is 1 for auto-encoders. The variation of parameters as designed in Table 5.4 leads to specific maximum values that are possible to be mapped on the hardware. The not-mappable cluster on both the windows in Figure 5.12(b), is an estimate of the hardware resources, whereas the mappable data-points are obtained by post-processing on the obtained connectivity/placement matrix from the algorithm.

Approximately 200 neurons can be mapped with same amount of input resource utilization and approximately 20K synapses on the hardware. The number of neurons possible are certainly more than FFN, however, less input resources are exhausted in comparison to FFN which is also just 40% of the total input resources available on the target hardware. The network sizes with 256 or 300 neurons are not very huge, however, the available neurons on hardware are exhausted and hence, not very large



(a)



(b)

Figure 5.13: Hardware Utilisation: (a) utilisation of synapses and input resources against neurons for RNN are shown for combinations which are mappable and not mappable on the hardware (b) utilization of synapses and input resources against neurons for LSM are shown.

auto-encoders are possible to be mapped even though the input resources and synapses are ample enough.

5.6.3 Recurrent Neural Network

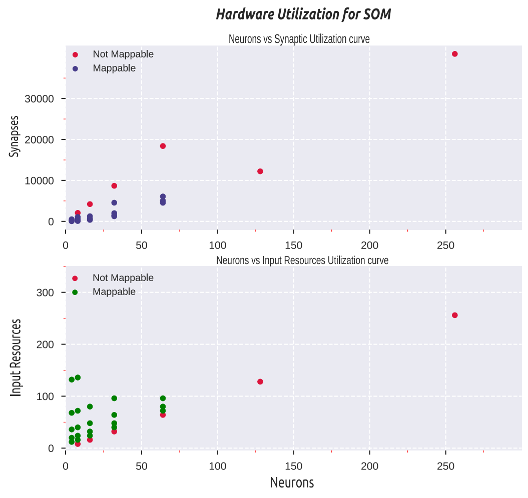
The recurrence sparsity is varied alongside the depth in these experiments. Due to the recurrence in the networks, which also is affected by the connectivity constraints defined in Section 4.2, the possible network sizes are small as it can be witnessed from lesser number of resources utilized i.e. only about 15% of input resources, 25% of neurons and roughly 14% of total synapses available on the target hardware.

The input resource utilization is not in tandem with variation of neurons, as it can be seen from second window in Figure 5.13(a). This can be explained by the high recurrence value in the RNNs which leads to saturation of input resources even if single feedback lines are originating from neurons.

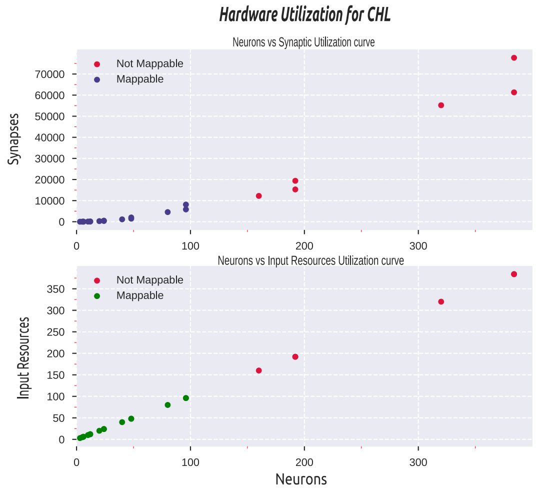
5.6.4 Liquid State Machines

For LSMs, mainly the breadth of the liquid layer is varied in combination with sparsity, keeping depth as constant value as explained in Table 5.6(b). The mappability and the resources occupied on the hardware is shown on both the windows of Figure 5.13.

The number of neurons possible for these candidate architectures are nearly 256, full utilization, which leads to a synaptic utilization of roughly 12K and about 200-250 input resources. If there is a network with 250 neurons, with any number of connections, keeping in check with the total synapses and input resources, the mapping solution can be obtained for LSMs.



(a)



(b)

Figure 5.14: Hardware Utilisation: (a) utilisation of synapses and input resources against neurons for SOM are shown for combinations which are mappable and not mappable on the hardware (b) utilisation of synapses and input resources against neurons for CHL are shown for combinations which are mappable and not mappable on the hardware

A linear variation of input resources is observed with respect to the neurons on the target hardware. The amount of synaptic resources utilized for a certain number of neurons is greater in case of LSMs than the rest of the candidates so far making them better for applications where high utilization can be achieved with less number of cores. The overall resources used for decently big network is higher for LSMs than the rest, approximately 100% neurons, 30% input resources and 20% synapses.

5.6.5 Self-Organizing Maps

The breadth of the torus layer and the external inputs are varied in this experiment alongside sparsity. The bar like lines seen in Figure 5.14(b) is due to the variation of breadth for same external input, and the corresponding change in solution time is observed due to exhaustion of available input resources and increased connectivity constraints. The number of neurons utilised is very less(30%) and the occupied synapse count is less(15%) with about input resources(18%) less in count as well. The not mappable dots overlapping the mappable one's arise due to the higher external inputs and lower breadth not possible to be fit on the target hardware due to lack of input resources.

5.6.6 Complete Hidden Layer

Hopfield networks have a circular fashion of neuron arrangement with recurrences within the layer, forming a complete graph structure. Breadth variation is carried together with variation of sparsity in the network. Figure 5.14(b) shows that CHLs use less synapses even when there are full connections in the entire network, making them

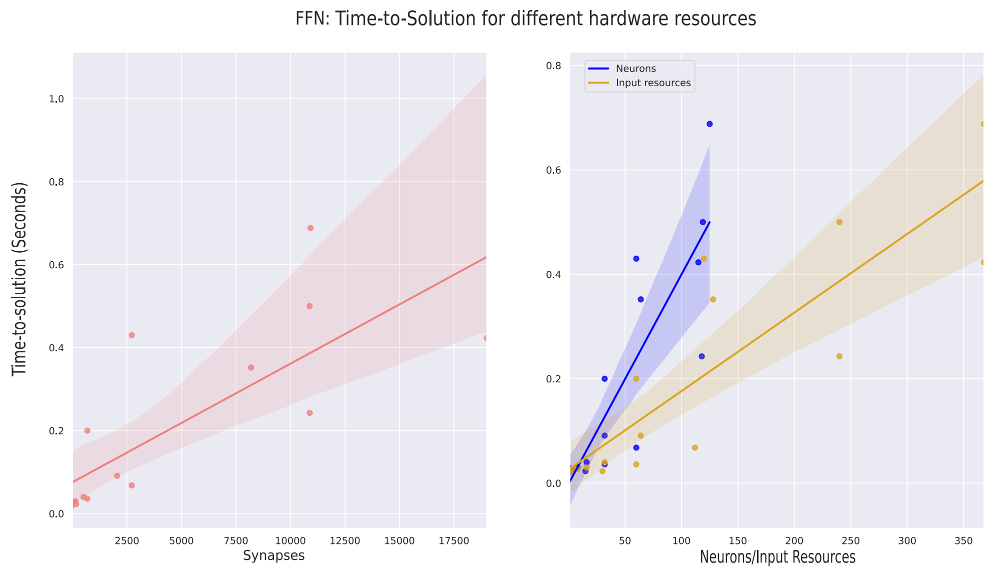


Figure 5.15: FFN: Variation of average confidence of time-to-solution for different hardware resources

much worse candidates in terms of resource utilization. The neuron utilisation is 39% and input resource utilization is about 9% slightly varying than SOMs as they have CHLs have different input:output characteristics.

5.6.7 Time-to-solution vs Hardware resources

The proposed mapper is also evaluated by the time it takes to allocate resources and produce a valid placement against the hardware utilization in the plots below. There are three different types of resources, neurons, synapses and input resources, as per the target architecture. The relation between variation of neurons to input resources and synapses for different candidate SNNs, and the impact on the solution time is reported in this section.

The time-to-solution in measured in seconds on the y-axis and the hardware resources are presented on the x-axis of the plot in Figure 5.15. Since the number of synapses are in 10's of thousands, and therefore, the single x-axis is not suitable for representing all the hardware resources.

The resources varied on x-axis in Figure 5.15 is representative of the different network sizes experimented. The depth variation and breadth variation leads to increasing nature of resources. When there is a time-variation for the same resource, it can be inferred as the variation of sparsity in the network from less connections to full connections. For FFNs, the blue line indicates the variation of time with neurons. It is observed that the number of input resources needed for a particular network size will be at-least two times the number of neurons. The synapses occupied reach upto 19K on the hardware for FFNs and the convergence time for such a huge network is just about 0.75 seconds which means even when network size increases, the solution time doesn't

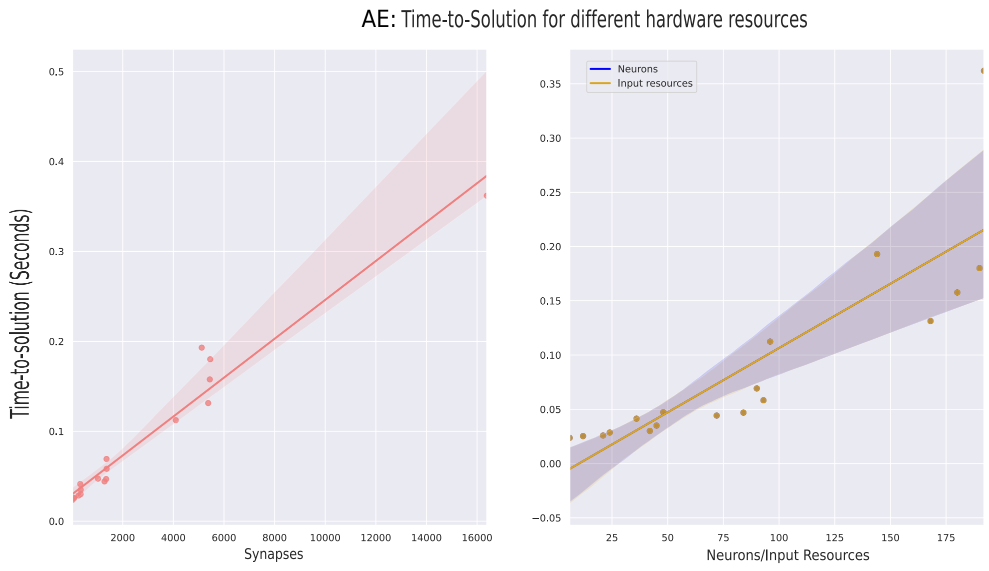


Figure 5.16: AE: Variation of average confidence of time-to-solution for different hardware resources

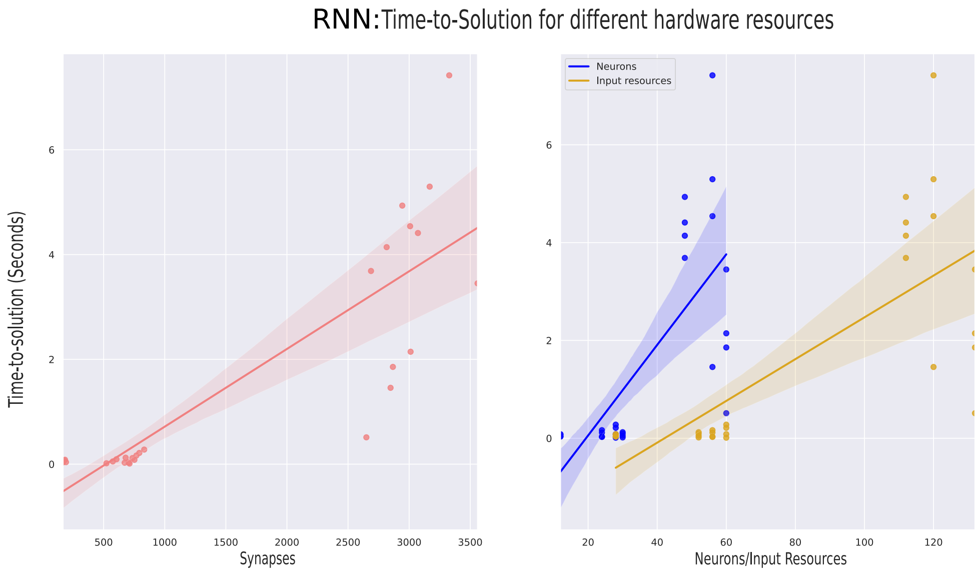


Figure 5.17: RNN: Variation of average confidence of time-to-solution for different hardware resources

overshoot to hours or days for simpler networks with feed forward signal because not all connectivity constraints come into picture and the input resources are not saturated.

Unlike FFN, a significant distinction is seen for AE in the second window of Figure 5.16. The overlap of neurons and input resources is due to the nature of converging and

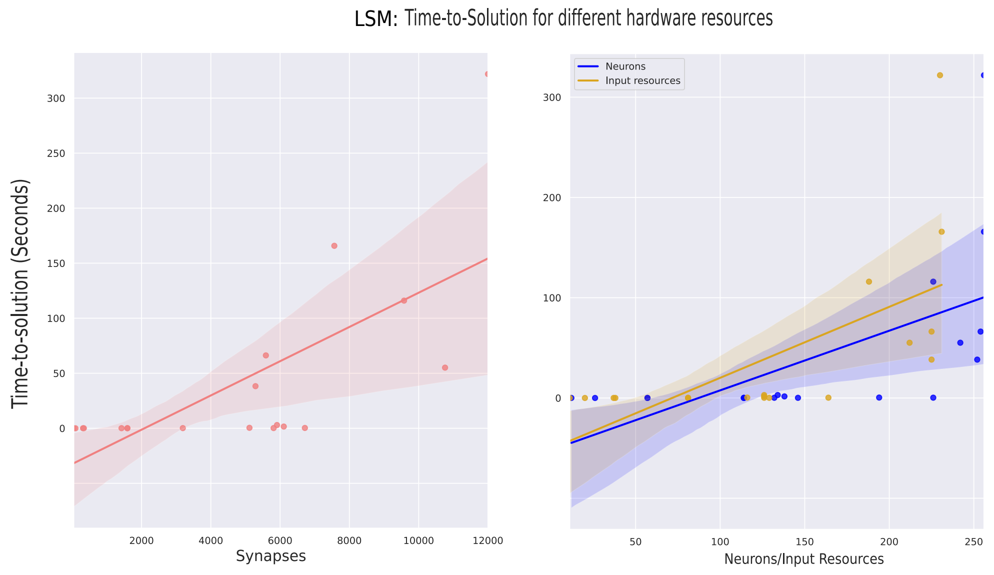


Figure 5.18: LSM: Variation of average confidence of time-to-solution for different hardware resources

diverging style of undercomplete auto-encoders. The maximum number of resources that can fit on the target hardware take about half the time taken by FFNs. For small network sizes of smaller depths, the resources utilised are very few and the time to find a valid placement is an average value of milli-seconds iterated over 5 runs. A confidence interval of 95% is plotted with a regression fit that shows the variation of the time within the translucent bands. Auto-encoders better occupy the input-output resources than FFN.

In case of RNNs, only recurrences from layer n to n-1 is considered excluding self-connections and recurrence within the layer. The network sizes are varied sweeping over the depth, and accordingly varying breadth as presented in Table 5.8. The first window in Figure 5.17 shows the maximum synapses that are utilised for the largest RNN possible on the target hardware. It also shows the time-to-solution is in milli-seconds for 500-1000 synapses, and there is sudden increase in solution time as the synapses increases, the neurons and input resources also increase. However, the explanation for this would be the recurrences in the network introduces tighter input connectivity constraints and finding suitable synapses for allocation, making it longer to find a valid placement solution.

In liquid state machines, the neuron utilization goes upto 256 when depth is kept constant, and the breadth of the liquid layer in combination with I:H and H:I is varied. For 256 neurons, about 12000 synapses are utilised. Since the connections in liquid layer are random and there is high recurrence among the layer itself, the time taken to find a valid mapping solution is also high in case of LSMs. With maximum resources utilised on hardware, the solution time is about 300 seconds with readout layer. The number of neurons in liquid layer is 10x that of the input layer, this explains the advancement

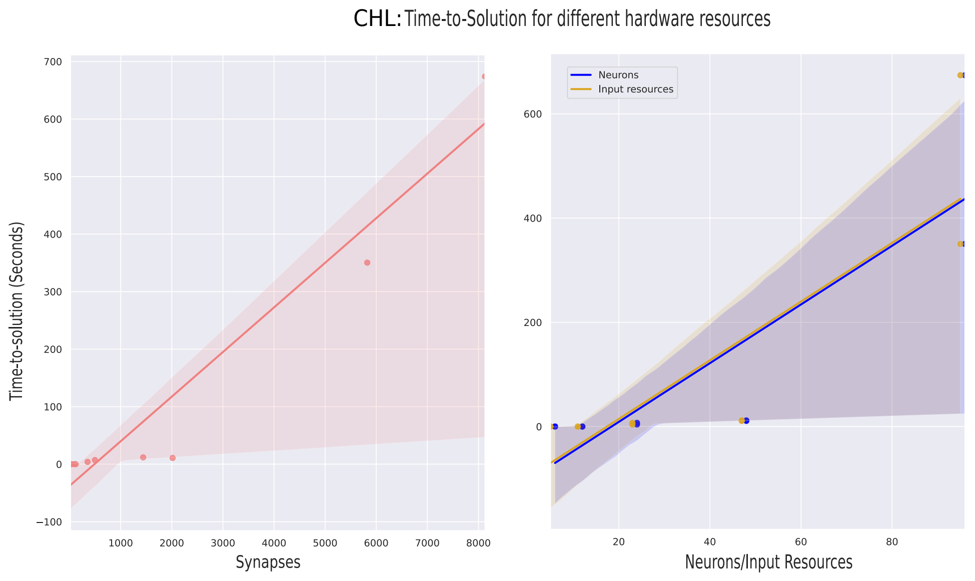


Figure 5.19: CHL: Variation of average confidence of time-to-solution for different hardware resources

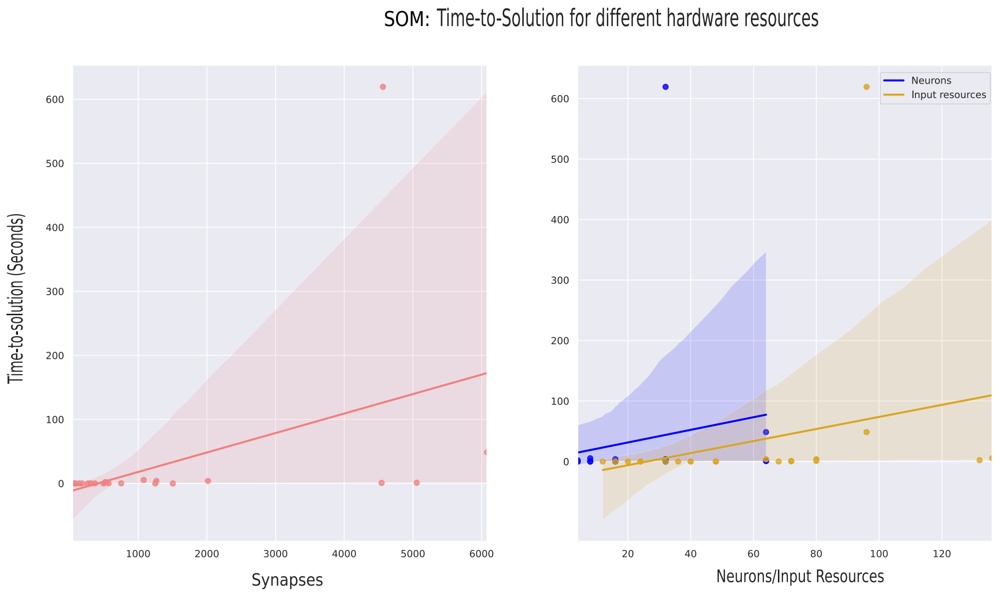


Figure 5.20: SOM: Variation of average confidence of time-to-solution for different hardware resources

in neuron curve than the input resources in second window of Figure 5.18.

Similar to AE, CHL also have same cue patterns and retrieved patterns at the input and output respectively. This explains the overlap of neuron and input resource lines. The recurrence in the Hopfield network in the hidden layer is the reason for rise in synapses for only a rise of 50% neurons on hardware, leading to very high time-to-solution which is about 680 seconds. CHLs don't occupy lot of input-output resources for huge networks in comparison to rest of the candidates, however, with the increase in recurrence, they form amongst the candidates with very high solution time.

For SOMs, generally, the solution time is in the bottom band within few seconds. However, as we have full recurrences within the torus layer for SOM, leading to stricter synaptic connectivity constraints and input global constraints. The co-ordination of input resources for recurrent connections causes high solution time like 619 seconds for larger networks on the target hardware indicating a tangential mapping approach for special topologies like SOMs than the stochastic approach taken in this thesis work.

The experiments have shown that with rise in architectural complexity of networks and strict hardware constraints that become significant when resources are on the verge of saturation, the algorithm has exponential worst case complexity. The random assignment of an initial resource could result in an incomplete solution class and there is no confidence of finding a solution when it comes to a stochastic approach limiting their performance.

5.7 Efficiency of Mapping

The proposed algorithm performs placement of different graph attributes on the given target neuromorphic hardware considering different hardware constraints and strictly adhering to these connectivity rules to provide a valid mapping in the form of hardware connectivity matrix and mapped addresses of the different resources.

The placement technique does not consider any cost function optimization into account while performing mapping. The algorithm is designed to produce a random mapping given a network complying to the size of the hardware. This presents an avenue of briefly investigating the different mapping solutions produced by the mapper and infer the results in terms of low delay on neurosynaptic core or quantitatively measure the congestion cluster through spread of the network. Different mapping solutions are plotted as heatmaps showing the weight matrix or connectivity matrix on the target hardware. A vast variation of placements can be observed in Figure 5.21.

Different placement solutions obtained for the same network is quantitatively measured by the total path delay inside the core as per Equation 5.2.2.2. The more closer the network towards the topright corner of the hardware array, higher is the delay. If the network is clustered in the bottomleft corner of the hardware array implies least delay as it froms the ff corner on the chip. The effects of different mappings can lead to various non-idealities in the hardware. It is not yet fully known what is the effect of increased delay on the inference accuracy on target hardware or the effect of congestion of resources in one cluster or corner in terms of non-linearity or cross-talk and which of this outweighs the other. Therefore, this needs further insights which is not addressed by this thesis work.

Figure 5.21: Illustration of the different placement solution for the same network (generic LSM of network size 6-48-2) on the hardware array size of MxK **(The images are omitted due to confidentiality purposes)**

6 Conclusion and Future Work

6.1 Conclusion

With the advent of new architectural era of neuromorphic computing, we are marching towards ultra low power systems to be infused in embedded devices. On the other hand AI models are becoming more powerful, tinier and energy efficient. This thesis work tries to bridge the hardware-software gap by deploying these neural networks onto a custom neuromorphic architecture.

A review of the literature revealed that the mapping algorithms employed have an objective of improvement of certain cost function with lesser focus on the mapping of neural networks onto low level hardware primitives. The underlying architecture dictates the strategy and placement of neural network onto various hardware resources. The literature in terms of mapping heavily diverse variety of spiking topologies is quite sparse.

This thesis work focuses on setting up a mapping pipeline to deploy various candidates of spiking neural networks onto the target neuromorphic architecture. Predominantly, the main objective has been to devise a placement algorithm strictly adhering to the hardware constraints in order to provide a valid and non-overlapping mapping solution in the form of hardware connectivity matrix, mapped addresses of different PEs(processing elements) on the hardware.

In academia and research, the placement algorithms are evaluated using the time-complexity. The proposed ALAPIN mapper is also evaluated using the time-to-solution for chosen candidates of SNNs by performing a parameter sweep on depth, layer-width, sparsity, recurrence sparsity etc. After a detailed evaluation, it is observed that the proposed algorithm can accomplish mapping for all chosen candidates seamlessly. The resource allocation and placement is achievable within milli-seconds for simpler networks like feed-forward and auto-encoders, however, with recurrence in the network, limited resources and overwhelming hardware constraints, the average solution time rises to few minutes (about 600 seconds for SOMs and CHLs).

Hardware resources occupied by each candidate SNN for different network sizes have shown that physical neurons are the major bottleneck causing strict constraints for the target hardware in order to accommodate bigger networks. The synapses and input resources are never utilised to their maximum. Additionally, a comprehensive experimentation on synthetic SNN-based topologies with vast architectural complexity has shown that Liquid State Machines(LSMs) tend towards utilising most of the resources combined for the largest network possible on hardware in comparison to any other candidate SNN making them most suitable for spiking based application on the target architecture employed in this work.

6.2 Future work

This section narrates the future directions researchers can take to advance exploration in the mapping of neural networks on neuromorphic architecture or embedded hardware architectures. The below sections are divided into two subsections, the first one talks about the immediate steps one can take specific to this thesis and the next section is an opinion of the author on alternative future directions to realise neural networks on embedded devices.

6.2.1 Natural Extensions

- Since this thesis has shown the maximum network sizes for different schemes of spiking neural networks, an immediate extension is to optimize these networks depending on the network sizes, and to map on the multi-core or multi-chip hardware. Due to the inception of multi-core system, the interconnect can introduce different non-idealities like latency. Therefore, it is crucial to account for these delays in the mapper to plummet the communications costs.

- As the experiments have already shown that the time-to-solution increases as the network size grows. This fact will become even more significant when the realistic workloads are employed to map on a multi-core hardware. With the advent of optimization of neural networks, the run-time of the algorithm can grow exponentially, therefore a solution friendly algorithm with low run-time is needed. To enhance from the present work, a flat approach of mapping needs to be employed. Unlike a hierarchical approach, in a flat placer, thousands of nodes will be allocated on the hardware resources in a single run.

- Results of analysing the mapping trend for each network has shown that even with full recurrence in the network, the synapse utilization cannot be 100%. Having a mechanism to utilize the synapses of external signal occupied input PEs through a spike-time aware mapping strategy can be monumental. To achieve this, a run-time compilation approach must be taken.

6.2.2 Optimization Avenues

- Research has shown that neural networks don't perform at their best when they are densely connected, rather may give optimum results when they are sparsely connected. This fact is backed by biology as well. From an algorithmic viewpoint, there is enough research on-going as to how the dense neural networks can be pruned or made sparse for weights and activation functions. A placement approach insensitive to structured and scattered sparsification will be needed to seamlessly accommodate the energy efficient and sparse neural networks on the hardware.

- The spiking neurons have a specific activation functionality, which limits deploying more complex network architectures with extremely complex activation functions which are non-linear in nature and involve lot of MAC operations. There is on-going research to emulate LSTM/GRUs in the neuromorphic hardware by a small tweak in the threshold variation. It will be exceptionally interesting to see how a mapper can handle this instead of increasing the hardware cost by additional resources. A solution to these architectures can open up avenues for a wider variety of complex neural architectures to deploy on hardware.

Bibliography

[1] L. Liu et al., "Computing Systems for Autonomous Driving: State of the Art and Challenges," in IEEE Internet of Things Journal, vol. 8, no. 8, pp. 6469-6486, 15 April15, 2021, doi: 10.1109/JIOT.2020.3043716.

[2] John C. Dvorak: Intel's 80170 chip has the theoretical intelligence of a cockroach in PC Magazine Volume 9 Number 10 (May 1990), p. 77,[1], retrieved May 16, 2021

[3] Gschwind, Michael; Hofstee, H. Peter; Flachs, Brian; Hopkins, Martin; Watanabe, Yukio; Yamazaki, Takeshi (2006). "Synergistic Processing in Cell's Multicore Architecture". IEEE Micro. 26 (2): 10–24. doi:10.1109/MM.2006.41

[4] Rhodios, Apollonios. (2007). The Argonautika : Expanded Edition. University of California Press. p. 355. ISBN 978-0-520-93439-9. OCLC 811491744

[5] McCorduck 2004, pp. 104–107, Crevier 1993, pp. 102–105, Russell and Norvig 2003, p. 22

[6] W. Dai and D. Berleant, "Benchmarking Contemporary Deep Learning Hardware and Frameworks: A Survey of Qualitative Metrics," 2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI), 2019, pp. 148-155, doi: 10.1109/CogMI48466.2019.00029.

[7] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient Processingof Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE. 2017.

[8] "Graphics Reinvented: NVIDIA GeForce RTX 2080 Ti Graphics Card," NVIDIA. NVIDIA COMPANY.

[9] "Google reveals more details about its second-gen TPU AI chips," techcrunch. [Online]. Available: https://www.theinquirer.net/inquirer/news/3023202/google-reveals-more-details-about-its-second-gen-tpu-ai-chips.

[10] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," Proc. IEEE, vol. 96, no. 5, pp. 879–899, 2008

[11] "Google announces a new generation for its TPU machine learning hardware," techcrunch. [Online]. Available: https://techcrunch.com/2018/05/08/google-announces-a-new-generation-for-its-tpu-machine-learning-hardware/

[12] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. Neural Netw. 10, 1659–1671.

[13] Ponulak, F., and Kasinski, A. (2011). Introduction to spiking neural networks: Information processing, learning and applications. Acta Neurobiol. Exp. 71, 409–433.

[14] https://www.nwo.nl/en/cases/braincomputer

[15] Grüning, A., and Bohte, S. M. (2014). “Spiking neural networks: Principles and challenges,” in European Symposium on Artificial Neural Networks (ESANN), Computational Intelligence and Machine Learning (Bruges: ESANN).

[16] https://en.wikipedia.org/wiki/Venus_flytrap

[17] https://en.wikipedia.org/wiki/Electronic-design-automation

[18] F. Akopyan et al., ”TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 10, pp. 1537-1557, Oct. 2015, doi: 10.1109/TCAD.2015.2474396.

[19] G. C. Qiao et al., ”A Neuromorphic-Hardware Oriented Bio-Plausible Online-Learning Spiking Neural Network Model,” in IEEE Access, vol. 7, pp. 71730-71740, 2019, doi: 10.1109/ACCESS.2019.2919163.

[20] Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. ACM Comput. Surv. 51, 3, Article 56 (July 2018), 39 pages. doi:10.1145/3186332

[21] Z. Yu, A. M. Abdulghani, A. Zahid, H. Heidari, M. A. Imran and Q. H. Abbasi, ”An Overview of Neuromorphic Computing for Artificial Intelligence Enabled Hardware-Based Hopfield Neural Network,” in IEEE Access, vol. 8, pp. 67085-67099, 2020, doi: 10.1109/ACCESS.2020.2985839

[22] M. Capra et.al, An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks

[23] C. Mead, “Neuromorphic electronic systems,” Proc. IEEE, vol. 78, no. 10, pp. 1629–1636, Oct. 1990.

[24] R. A. Nawrocki, R. M. Voyles and S. E. Shaheen, ”A Mini Review of Neuromorphic Architectures and Implementations,” in IEEE Transactions on Electron Devices, vol. 63, no. 10, pp. 3819-3829, Oct. 2016, doi: 10.1109/TED.2016.2598413.

[25] M. Sharad, C. Augustine, G. Panagopoulos, and K. Roy.“Proposal for neuromorphic hardware using spin devices, arXiv 2012, 1206.3227

[26] F. Corradi, D. Bontrager, and G. Indiveri, “Toward neuromorphic intelligent brain-machine interfaces: An event-based neural recording and processing system,” in Proc. IEEE Biomed. Circuits Syst.Conf. (BioCAS), Oct. 2014, pp. 584–587

[27] K. Boahen, “Neuromorphic microchips,” Sci. Amer., vol. 292, no. 5,pp. 56–63, 2005

[28] Gurney. Kevin, ”An Introduction to Neural Networks” Taylor and Francis, Inc., USA, 1997, doi: 10.5555/523781

[29] Hwang, S., Chang, J., Oh, MH. et al. Impact of the Sub-Resting Membrane Potential on Accurate Inference in Spiking Neural Networks. Sci Rep 10, 3515 (2020). doi: 10.1038/s41598-020-60572-8

[30] Yang G., Qiao J., Li W., Chai W. (2013) The Effect of Lateral Inhibitory Connections in Spatial Architecture Neural Network. In: Guo C., Hou ZG., Zeng Z. (eds) Advances in Neural Networks – ISNN 2013. ISNN 2013. Lecture Notes in Computer Science, vol 7951. Springer, Berlin, Heidelberg. doi: 10.1007/978-3-642-39065-4-38

[31] Meir, E., von Dassow, G., Munro, E., Odell, G.M.: Robustness, Flexibility, and the Role of Lateral Inhibition in the Neurogenic Network. Current Biology 12(10), 778–786 (2002)

[32] Burkitt, Anthony. (2006). A Review of the Integrate-and-fire Neuron Model: I. Homogeneous Synaptic Input. Biological cybernetics. 95. 1-19, doi: 10.1007/s00422-006-0068-6.

[33] Dabiri, Y., Van der Velden, A., Sack, K.L. et al. Application of feed forward and recurrent neural networks in simulation of left ventricular mechanics. Sci Rep 10, 22298 (2020), doi: 10.1038/s41598-020-79191-4.

[34] G. Hughes, "On the mean accuracy of statistical pattern recognizers," in IEEE Transactions on Information Theory, vol. 14, no. 1, pp. 55-63, January 1968, doi: 10.1109/TIT.1968.1054102.

[35] Zollanvari, A., James, A.P. and Sameni, R. A Theoretical Analysis of the Peaking Phenomenon in Classification. J Classif 37, 421–434 (2020). doi: 10.1007/s00357-019-09327-3.

[36] Mohammad, Rami. (2018). A Neural Network based Digital Forensics Classification. 1-7. 10.1109/AICCSA.2018.8612868.

[37] Gers, F.A. (1999). "Learning to forget: Continual prediction with LSTM". 9th International Conference on Artificial Neural Networks: ICANN '99. 1999. pp. 850–855. doi:10.1049/cp:19991218. ISBN 0-85296-721-7.

[38] Cho, Kyunghyun; van Merrienboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". arXiv:1406.1078

[39] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276. S2CID 1915014.

[40] Beaufays, Françoise (August 11, 2015). "The neural networks behind Google Voice transcription". Research Blog. Retrieved 2017-06-27.

[41] Sak, Haşim; Senior, Andrew; Rao, Kanishka; Beaufays, Françoise; Schalkwyk, Johan (September 24, 2015). "Google voice search: faster and more accurate". Research Blog. Retrieved 2017-06-27

[42] Efrati, Amir (June 13, 2016). "Apple's Machines Can Learn Too". The Information. Retrieved 2017-06-27

[43] Ranger, Steve (June 14, 2016). "iPhone, AI and big data: Here's how Apple plans to protect your privacy — ZDNet". ZDNet. Retrieved 2017-06-27

[44] J. H. Kaas and K. C. Catania, "How do features of sensory representations develop?", BioEssays, vol. 24, no. 4, pp. 334-343, 2002

[45] T. Rumbell, S. L. Denham and T. Wennekers, "A Spiking Self-Organizing Map Combining STDP, Oscillations, and Continuous Learning," in IEEE Transactions on Neural Networks and Learning Systems, vol. 25, no. 5, pp. 894-907, May 2014, doi: 10.1109/TNNLS.2013.2283140.

[46] Vogels, Werner (30 November 2016). "Bringing the Magic of Amazon AI and Alexa to Apps on AWS. – All Things Distributed", Retrieved 2017-06-27

[47] J. V. Arthur et al., "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," The 2012 International Joint Conference on Neural Networks (IJCNN), 2012, pp. 1-8, doi: 10.1109/IJCNN.2012.6252637

[48] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise". In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96). AAAI Press, 226–231.

[49] E. Frady, F. Sommer, "Robust computation with rhythmic spike patterns", Proc Natl Acad Sci USA volume 116, issue 36, P18050-18059 2019 doi: 10.1073/pnas.1902653116

[50] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in Rebooting Computing (ICRC), IEEE International Conference on, 2016.

[51] Sepp Hochreiter, Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276. S2CID 1915014.

[52] S.Moradi et al., A Scalable Multicore Architecture WithHeterogeneous Memory Structures for DynamicNeuromorphic Asynchronous Processors (DYNAPs), IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, VOL. 12, NO. 1, FEBRUARY 2018.

[53] Ju. Han, Xu. Jian-Xin and Vandongen, Antonius. (2010). Classification of musical styles using liquid state machines. Proceedings of the International Joint Conference on Neural Networks. 1 - 7. 10.1109/IJCNN.2010.5596470.

[54] M. M. Khan et al., "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor," 2008 IEEE International Joint Conference on Neural Networks

(IEEE World Congress on Computational Intelligence), 2008, pp. 2849-2856, doi: 10.1109/IJCNN.2008.4634199.

[55] F. Akopyan et al., "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 10, pp. 1537-1557, Oct. 2015, doi: 10.1109/TCAD.2015.2474396.

[56] A. Amir et al., "Cognitive computing programming paradigm: A Corelet Language for composing networks of neurosynaptic cores," The 2013 International Joint Conference on Neural Networks (IJCNN), 2013, pp. 1-10, doi: 10.1109/IJCNN.2013.6707078.

[57] W. Wen et al., "An EDA framework for large scale hybrid neuromorphic computing systems," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015, pp. 1-6, doi: 10.1145/2744769.2744795.

[58] A. Das, Y. Wu, K. Huynh, F. Dell'Anna, F. Catthoor and S. Schaafsma, "Mapping of local and global synapses on spiking neuromorphic hardware," 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE), 2018, pp. 1217-1222, doi: 10.23919/DATE.2018.8342201.

[59] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinyaet al., "Loihi: A neuromorphic manycore processor with on-chip learning,"IEEE Micro,vol. 38, no. 1, pp. 82–99, 2018.

[60] Chit-Kwan Lin, Andreas Wild, Gautham N. Chinya, Tsung-Han Lin, Mike Davies, and Hong Wang. 2018. Mapping spiking neural networks onto a manycore neuromorphic architecture. SIGPLAN Not. 53, 4 (April 2018), 78–89. doi: 10.1145/3296979.3192371