Doctoral Thesis

Autonomous learning in neuromorphic systems for recognition of spatio-temporal spike patterns

Author(s):
Sheik, Sadique ul Ameen

Publication Date:
2013

Permanent Link:
https://doi.org/10.3929/ethz-a-010118718

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
Autonomous Learning in Neuromorphic Systems for Recognition of Spatio-Temporal Spike Patterns

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

SADIQUE UL AMEEN SHEIK

M.Sc. Physics, Indian Institute of Technology, Kharagpur

born on May 09th, 1985

citizen of Republic of India

accepted on the recommendation of

Prof. Dr. Rodney J. Douglas, examiner
Prof. Dr. Giacomo Indiveri, co-examiner
JProf. Dr. Elisabetta Chieca, co-examiner
Prof. Dr. Richard Hahnloser, co-examiner

2013
Acknowledgments

Several people have supported me during my PhD studies and even before that, and to them I would like to show my gratitude. I would like to thank my supervisors Giacomo Indiveri – for letting me work on what I really wanted to, Elisabetta Chicca – for making me work on what ever I needed to, and Rodney Douglas – for being my toughest critique and providing a strong impetus to better myself. I must also thank Martin Coath for being my pseudo-supervisor and introducing me to his thalamo-cortical model. Without them this work would not have existed.

Despite having lived away from home for approximately 7 years before I started my PhD, Zurich was the first time I stepped out of India. This was a journey full of excitement and anxiety. I cannot imagine how my years at INI would have been spent without my friends and colleagues who made me feel at home and supported me at every stage. I am greatly indebted to them; Daniel for showing me what it means to be a German, Mohammad for showing me what it is like to be an Iranian, Saber for showing me what it means to be an electrical engineer, and Emre for introducing me to the beautiful world of Python. I would like to thank Fabio for being my personal literature reviewer and encyclopedia and most importantly showing me what it means to be an Italian,

I would also like to thank my WingChun group; Roman for introducing me to it, Fabio for kicking my butt and Thomas for being the greatest teacher I could hope to have. My life in Zurich wouldn’t have been the same without this group.

Last but not least I would like to thank my parents who taught me how to think and reason at an early age, immensely valued education, and motivated me to take up research. And of course my “little” brother who has always put up with me and has always helped me in my mischiefs. Without them and their faith in me, this work would have been non-existent. I dedicate this work to them, my family.
Abstract

Spiking neuromorphic devices rely on the fact that spikes are an efficient mechanism for encoding and transmitting spatio-temporal properties of stimuli. Several neuromorphic sensors have been built recently that efficiently encode dynamic stimuli into spikes in real-time. But detecting stimuli based on the information embedded in such spikes in real-time is still an open problem. Typically, the problem of detection of stimuli based on spatio-temporal spike patterns is reduced to detection of coincident spiking activity in the spike pattern. The ordering of spikes and the precise Inter-Spike Intervals (ISIs) are often ignored thereby losing precious temporal information such as causality in the stimulus. These approaches also require very precise tuning of parameters to match the properties of the input spike patterns. In order to recognize spike patterns based on their temporal properties, I investigate alternate strategies for spike based neural processing. I propose a novel model, that exploits spike propagation delays and the temporal properties of Spike Timing Dependent Plasticity (STDP) to capture the spatio-temporal correlations in the spike pattern. This allows the model to go beyond detecting simply the coincidences in the stimuli. The model relies on a pool of propagation delays that enable neurons in the network to store precise inter-spike intervals in the target spike pattern. Through STDP the network converges to a connectivity that encompasses the ISIs corresponding to the target spike pattern. To ensure its robustness to imprecise neuromorphic devices, it is validated by implementing and testing it on a neuromorphic multi-chip system. The intrinsic transistor mismatch of neuromorphic devices is exploited to generate a wide range of propagation delays with point Leaky Integrate and Fire (LIF) neurons. By training the network on a parameterized set of synthetically generated spike patterns, it is demonstrated that the model can learn to differentiate between different spike patterns even if they share the same spatial properties but have different temporal properties. This work enables us to build spike-based neuromorphic systems that perform real-time learning and detection based on the spatio-temporal properties of the stimulus. The generality of this work to generic spike patterns enables us to perform challenging tasks such as multi-sensory fusion and to learn relevant spatio-temporal correlations across multiple sensory modalities.
Compensando

I sistemi neuromorfi si basano sul fatto che gli eventi sono un meccanismo efficiente per
codificare e trasmettere le proprietà spazio-temporali degli stimoli. Recentemente sono
stati costruiti molti sensori neuromorfi che codificano efficientemente stimoli dinamici in
eventi in tempo reale. Ciò nonostante, riconoscere in tempo reale gli stimoli basandosi
sull’informazione contenuta in tali eventi rimane ancora un problema aperto. Tipicamente
il problema del riconoscimento di stimoli basati sui modelli spazio-temporalità degli eventi
viene semplificato nel problema del riconoscimento di coincidenze da un flusso di eventi.
L’ordine degli eventi e gli ISI vengono spesso ignorati. Pertanto informazioni temporali
preziose, come la causalità dello stimolo, vengono perse. Questi approcci richiedono pre-
cise calibrazioni dei parametri per accoppiare le proprietà degli eventi in ingresso. Per
risconoscere i diversi tipi di flussi di eventi basandosi sulle proprietà temporali degli stessi,
ho indagato strategie alternative per il processo neurale di flussi di eventi. Propone
un modello nuovo, che sfrutta il ritardo di propagazione degli eventi e le proprietà tem-
porali dell’STDP per catturare correlazioni spazio temporali rilevanti. Questo permette al
modello di andare oltre al semplice riconoscimento di coincidenze negli stimoli. Il mod-
ello si basa su un gruppo di propagazione dei ritardi che consenta ai neuroni della rete di
memorizzare precisi intervalli temporali tra eventi (ISI) nel flusso di eventi da riconoscere.
Tramite STPD la rete converge ad una connettività che contiene gli ISI corrispondenti al
flusso di eventi da riconoscere. Per assicurare la robustezza del modello e quindi essere
in grado di utilizzare hardware impreciso neuromorfico, il modello stesso è stato validato
tramite un’implementazione su un sistema multi chip in hardware neuromorfo. La vari-
abilità intrinseca dei sistemi neuromorfi viene sfruttata per generare una vasta gamma di
ritardi di propagazione con neuroni LIF. Tramite un addestramento della rete, avvenuto
utilizzando stimoli sintetici generati dal computer, viene dimostrato che il modello può
imparare a differenziare tra diversi flussi di eventi anche se tali flussi presentano le stesse
proprieta’ spaziali ma differiscono in proprietà temporali. Questo lavoro ci permette di
costuire sistemi neuromorfi basati sugli eventi che apprendono in tempo reale e che dis-
criminano proprietà spazio temporali degli stimoli. La generalità di questo lavoro si
collocava nel contesto di computazione ad eventi. Questo nuovo paradigma computazionale
ci permette di svolgere compiti impegnativi come quello della fusione multi-sensoriale e
dell’apprendimento di correlazioni spazio temporali tra diverse modalità sensoriali.
Disclaimer

I hereby declare that the work in this thesis is that of the candidate alone, except where indicated in the text, and as described below.

Some of the sections of Chapter 3 have been adapted from the paper: S. Sheik, M. Pfeiffer, F. Stefanini, and G. Indiveri. “Spatio-temporal Spike Pattern Classification in Neuromorphic Systems”. In: Biomimetic and Biohybrid Systems. Springer, 2013, pp. 262–273.


The neuromorphic chips used in this work were designed by Prof. Dr. G. Indiveri and Dr. E. Chicca. The AER infrastructure (mapper, monitor & sequencing boards) was developed by D. Fasmacht.

The use of “we” in the thesis refers to the above-mentioned people in the relevant sections.
## Contents

1 **Introduction**  
1.1 Spiking Neural Networks (SNNs)  
1.2 Advent of neuromorphic engineering  
1.2.1 Imprecise devices  
1.3 Problem definition  
1.4 Thesis structure  

2 **Spike based spatio-temporal processing**  
2.1 Computational merits of precise spike timing  
2.1.1 Encoding  
2.1.2 Decoding  
2.2 Learning to detect spatio-temporal spike patterns  
2.2.1 Supervised learning  
2.2.2 Unsupervised learning  
2.3 Summary  

3 **STC**: An unsupervised learning model for detecting Spatio-Temporal Correlations in spike patterns  
3.1 Detecting coincidences with leaky integrate and fire neurons  
3.2 Unsupervised learning models rely on coincidence detection  
3.3 STC: An unsupervised learning model for detecting spatio-temporal correlations in spike patterns  
3.3.1 Working principles  
3.4 Summary  

4 **Implementation of STC model on imprecise neuromorphic hardware**  
4.1 Neuromorphic multi-chip system used in this work  
4.1.1 Multi-neuron chips
Appendices

A Stimulus Specific Adaptation

A.1 Stimulus-specific Adaptation
   A.1.1 Oddball Experiments
   A.1.2 Changing Stimulus Presentation Mode

A.2 Network model

A.3 Hardware Implementation
   A.3.1 Input

A.4 Results
   A.4.1 Oddball Experiments
   A.4.2 Presentation Mode Experiments

A.5 Discussion and Conclusion

B Hardware-Software ecosystem

B.1 Setting up the neuromorphic hardware
   B.1.1 AMDA board
   B.1.2 AEX board
   B.1.3 AER Mapper

B.2 Software for configuring neuromorphic hardware: pyNCS

B.3 Learning on IFSLWTA
Chapter 1

Introduction

The human brain consists of approximately 100 billion neurons, each of which is connected to other neurons through 10000 synapses on average. It constitutes a massively parallel information processing system. Neurons typically operate at a maximum of 100 Hz while CPUs in today’s digital computers carry out several hundred million machine level operations per second. Despite slower elementary components, the brain shows remarkable information processing capabilities. It is extremely efficient in performing complex computational tasks. Visual perception for example occurs within a 200 ms \([194]\), that is less than 20 processing steps considering that each neuron takes 10 ms per operation. Comparable operations with a traditional computing system requires orders of magnitude higher processing time. Brains are also extremely tolerant to damage. While computer programs tend to break on failure of a single component, the brain’s performance only shows slow and gradual degradation under partial damage. It can even recover from damage in several cases.

But perhaps the most remarkable is the brain’s ability to learn and reorganize itself from experience. This ability to learn about our environment has motivated the study of neural systems within the context of computation and lead to the field of neural networks. Computation with such networks was first formalized by McCulloch and Pitts \([128]\) who created a computational model for neural networks, called threshold logic, based on mathematics and algorithms. These were considered the first generation of neural networks. These networks could only produce digital output and the computation was done through step or threshold functions. Following this work, random network models for adaptive stimulus-response relations were proposed by Farley and Clark \([60]\). Further enhancements came through the work of Rosenblatt \([169]\), Widrow and Hoff \([209]\) and others. Even though these neurons could only produce digital outputs, they have been successfully applied in powerful artificial neural networks like multi-layer perceptrons \([169]\) and Hopfield nets \([87]\). The second generation of neural networks used continuous activation functions such as a sigmoid in place of step functions, making them suitable for analog input and
Neural networks, as used in artificial intelligence, have traditionally been viewed as simplified models of neural processing in the brain, even though the relation between these models and biological architectures is debated, as it is not clear to what degree artificial neural networks mirror brain function. Spiking Neural Networks (SNNs) are seen as the third generation of neural network models, increasing the level of realism in a neural simulation. While previous generations (now referred to as conventional) neural networks relied on neuronal and synaptic states for computation, SNNs include the concept of time into their operating model. In contrast to conventional neural networks where the activity of neurons is propagated continuously (or every “time step”), neurons in SNNs only fire a “spike” when the neuron’s “membrane potential” (a neuronal intrinsic state) reaches a specific value. This spike is then transmitted to other neurons influencing their membrane potential. The information in SNNs is therefore transmitted asynchronously through spikes at discrete points in time.

There have been several spiking neuron models proposed in the literature such as the Integrate and Fire (IF) neuron model [1], FitzHugh-Nagumo model [64], Morris-Lecar model [143] and Izhikevich neuron model [97]. The first scientific model of a spiking neuron that describe how “action potentials” or spikes are initialized and propagated was proposed by Hodgkin and Huxley [85]. This model described the ionic currents through ion channels through electric circuits comprising of resistors and capacitors and voltage sources. These models involve differential equations of varying complexity that describe the dynamics of the neuron’s internal variables such as the membrane potential and synaptic dynamics. With increasing complexity of the models, their analytical tractability is lost. Typically, numerical simulations are used in order to understand the dynamics of internal variables of individual and networks of neurons.

Most research on spiking neural networks is carried out by means of software simulations [26], but for larger networks such approaches are computationally demanding and need to be run on bulky and power-hungry workstations or distributed computing facilities. Neural network research stagnated after a publication by Minsky and Papert [138] in 1969. One of the issues pointed out by the authors was that computers were not sophisticated enough to effectively handle the long run-time required by large neural networks. Due to this neural network research slowed until computers achieved greater processing
The root cause of these limitations, then and now, is the computational substrate that is being used. Software and digital hardware simulations of brain processing represent the state variables of neurons and synapses in a discrete form and update them according to a numerical integration scheme. Conventional computers built on von Neumann architecture work with serial operations on sequential data, whereas neural networks are inherently massively parallel. Unlike von Neumann models of computations, artificial neural networks do not separate memory and processing. They operate by transmission of signals through a network of interconnected units, like those seen in biological neural systems. These differences prevent efficient simulations of large neural networks on conventional computers.

Neuromorphic engineering [129] is an alternative approach, based on full-custom implementation of spiking neural networks using analog-digital Very Large Scale Integration (VLSI) circuits. It is an inter-disciplinary discipline that takes inspiration from biology, physics, mathematics, computer science and engineering to develop artificial neural systems whose physical architecture and design principles are based on those of biological nervous systems. Neuromorphic engineering aims at understanding how the morphology of individual neurons, their networks, and overall architecture play a crucial role in the nervous system. How do they bring forth desirable computation? How do they influence the representation of information? How do they induce robustness and fault tolerance? How do they influence learning and development? These are some of the questions that lie at the core of neuromorphic engineering.

Neuromorphic systems, i.e. devices built with neuromorphic engineering, do not rely on conventional binary computations but instead exploit semiconductor physics to emulate neuronal processes. This was born from the observation that ion channels in biology follow the same dynamics as a transistor in sub-threshold [120]. They both involve drift and diffusion processes in their dynamics. This observation [129] led to the design of a new class of low-power devices designed to mimic the nervous system called neuromorphic devices. These devices included neuromorphic sensors across several modalities and multi-neuron chips.

The first few years of the advent of neuromorphic engineering saw development of several neuromorphic sensors. These sensors were based on realistic models of the early sensory processing stages. Several vision sensors were designed and developed in order to mimic the function of retinal processing [105, 117, 13, 108, 44, 115, 215, 112]. The Dynamic Vision Sensor (DVS) [115] is one of the most used spiking vision sensors of this day and has come a long way from early vision sensors in terms of robustness, pixel density and ease of use. Similarly several groups have built motion sensors to compute the direction of

The second issue pointed out by Minsky and Pappert was that single-layer neural networks were incapable of processing the exclusive-or circuit. Later advances in back-propagation algorithm effectively solved the exclusive-or problem [208].
motion or optical flow. Several artificial sensors have also been developed in the auditory domain. These include single or pairs of silicon cochlea and auditory processing chips for sound detection and localization. This was soon followed by hardware implementations of multi-neuron chips to emulate neurons and synapses. These systems were typically implemented as single VLSI devices (e.g. as silicon retinas, or silicon cochleas), that faithfully reproduced in real-time the bio-physics of the neural systems they modeled. While these devices were operating independently more interesting applications were envisioned through culmination of the capabilities of these devices into a single system. But it was preferable to do so using a modular approach where the devices could communicate among each other through spikes.

The advent of Address Event Representation (AER) facilitated the next step in the evolution of neuromorphic engineering and led to hybrid neuromorphic devices. AER allows representation of spike events with unique addresses assigned to their corresponding sources or destinations. Such events can then be transmitted over a common communication digital channel shared by a pool of neurons (source) and synapses (destination), eliminating the need of individual wires connecting spike sources to destinations. The definition of event (spike) based communication protocols based on the AER led to the development of a new generation of more complex multi-chip neuromorphic sensory systems. Large systems built following the neuromorphic approach involve multiple neuromorphic VLSI sensory and processing devices interfaced among each other to perform complex cognitive tasks. For example, they use spike-based sensors for producing sensory signals in the form of streams of spikes, and spiking multi-neuron chips for implementing state dependent neural processing and learning mechanisms to produce appropriate behavior. These spiking sensors and multi-neuron chips interact with each other by exchanging information embedded in spikes using AER to process information about their environment.

Neuromorphic engineering is a promising avenue for robotic applications because of low power consumption and real-time processing capabilities due to the sub-threshold transistor dynamics. Neuromorphic devices can also be used as a platform for neural network simulations. They can offer the modeller a real-time feedback during experimentation. Whilst high-level languages aid the development of neural models, running the simulations themselves can be time-consuming. This results in running simulations at a fraction of real time, even for small networks such of the order of a few hundred neurons and synapses with detailed dynamics. This is particularly an issue when the experiments necessitate long run-times (or require to interact with the real world). Being able to monitor the output in real time and thereby spot problems as they unfolded undoubtedly shortens the development process considerably.
1.2.1 Imprecise devices

Analog VLSI implementations of neuromorphic devices share several characteristics with their biological counterparts. They are massively parallel, asynchronous, consume little power and in most cases operate in real-time. But at the same time they are noisy, prone to parameter mismatch. They have a limited parameter resolution and are hard to configure. Because of packaging limitations of neuromorphic chips, the number of pins that can be used to set biases and configure parameters is limited. While a single chip can have thousands of neurons and synapses, the number of accessible pins is usually limited to a hundred or less. Even on devices equipped with on-chip bias generators [50] and calibration circuits [111], the overhead of such circuits for every single parameter on the chip dwarfs the actual neuromorphic circuitry. To avoid such overhead common bias voltages are used to control circuits of populations of neurons and synapses on neuromorphic chips. This ideally means a group of neurons and synapses share the exact same parameters and should be identical. However in practice, due to inexact size of transistors and process variations, fabricated analog circuits do not have identical properties. This inexactness in the fabrication of transistors is termed as transistor mismatch [158]. Because of transistor mismatch, despite identical circuits and transistor sizes in the design, neuron and synapse do not all behave the same but have mismatched parameters. Consequently, one cannot instantiate a large population of identical neurons. On the other hand, since biases are shared, one cannot tune the parameters of individual neurons and synapses without affecting the others. Thus, it is not possible to implement complex neural networks with exact parameters, especially when the number of individual parameters in the network is considerably large. Another issue related to the synapses implemented on neuromorphic chips is the limited reconfigurability of their weights. Usually synaptic weights can only be set with a limited precision, are bounded and in some cases may even be binary [139]. Because of the bounded nature of synapses, models like the Perceptron [169] or the Hopfield network [87] cannot be directly translated to neuromorphic implementations.

This brief overview should make it clear that there are practical limitations on the possible computational models that could be realized with neuromorphic hardware. On the other hand these limitations allow us to verify the biological plausibility and robustness of computational models. The availability of configurable analog hardware also establishes a helpful working practice: once a model has been piloted in the highly idealised setting of a software simulation, its robustness can be assessed in neuromorphic hardware, which, like the brain, consists of inhomogeneous, low-fidelity analog components. A model that is able to gracefully surmount these obstacles is a better candidate for describing what takes place in the brain than one which relies on finely-tuned parameters, a situation that can go undetected in software simulations. Therefore it is important to keep in mind the capabilities of the hardware while developing computational models.
Fig. 1.1: Raster plot of output spikes of the Silicon Cochlea [36] in response to an auditory stimulus (utterance of the numbers 1577391). The vertical axis shows different channels corresponding to different auditory frequencies. The horizontal axis shows time. Each star in the plot represents a “spike”. Spikes corresponding to each word (number) are approximately annotated and plotted with different colors. The signal embedded in these spikes is not static, and is spread across space (frequency space i.e. channels) and time. This spatio-temporal spike pattern represents the auditory stimulus presented to the cochlea.

1.3 Problem definition

Neuromorphic spike based systems are envisioned to perform autonomous cognitive tasks owing to their biological parallelism. This work aims at understanding how to build such autonomous systems that can interpret their dynamic surroundings and make independent decisions to generate appropriate behavior. Information about the surroundings acquired by neuromorphic sensors is embedded in their outputs – spikes. In real-world applications with neuromorphic devices, the spike patterns produced by various neuromorphic sensors are spread across space and time reflecting the spatio-temporal properties of the stimuli they are encoding. For example, Fig. 1.1 shows the spikes from an auditory sensor, the Silicon Cochlea [36] in response to a sound. The spike pattern is not static but is spread across frequency space (represented by different frequency channels) and time, much like the biological cochlea. Any moving object would elicit a similar spatio-temporal spike pattern in a spiking vision sensor. It is imperative, therefore, that a neuromorphic system is able to differentiate between spike patterns generated by the sensors to recognize its stimulus.

An autonomous system only has access to the inputs from its sensors and has no external supervision. It has to rely on statistical information embedded in spikes in order to form a meaningful representation of its surroundings and produce appropriate behavior. Therefore, we need to investigate and develop neuromorphic systems that can autonomously learn and recognize statistically prominent spatio-temporal pattern of events in order to make appropriate decisions. The work presented in this thesis is predicated on these ideas. It focuses on unsupervised learning for recognition of spatio-temporal spike patterns on a
1.4 Thesis structure

The rest of this thesis is divided into five chapters. We begin our investigation by exploring the role of precise spike timing in spike based systems in Chapter 2. We investigate the problem of classification and detection of spatio-temporal spike patterns and take a brief tour of some of the state-of-the-art models, both supervised and unsupervised that address this problem. We then point out in Chapter 3 that some of these models, especially the unsupervised models rely on coincidences to recognize and detect spike patterns. We argue that this feature is not sufficiently rich to differentiate between spike patterns generated by neuromorphic sensors. In this chapter we propose a novel neural network model called Spatio Temporal Correlator (STC) based on first principles in order to overcome the limitations of the aforementioned models. Chapter 4 describes the neuromorphic hardware system used to implement the STC model, and how its functionality was achieved. In Chapter 5 we describe the experiments carried out on the neuromorphic system to explore the functionality of the neuromorphic implementation of the STC model. Further simulation results are presented to demonstrate the computational capabilities of the model. In the final chapter (Chapter 6), we discuss various issues related to the computational aspects of the model and its influence on the present and future of neuromorphic engineering.
Chapter 2

Spike based spatio-temporal processing

Several experiments in various animal species have shown that behaviorally relevant information is encoded in spatio-temporal spike patterns, and used for decision making. Barn owls, for example, rely on auditory cues to capture the prey, by accurately determining the spatial location of the sound source. They do this by computing the temporal delay in the spikes originating at the two ears of the owl [35, 34]. Remarkably precise temporal codes have been found in bush crickets [160] in early auditory inter-neurons, even in the presence of strong natural background noise. Blowfly’s visual motion detection system utilizes the temporal order of visual activation to determine the direction of motion [23].

The timing of spikes does not necessarily have to encode only temporal features of the input signal, but can also carry information about other aspects. For example, in the human peripheral nervous system, Johansson et al. [100] found that the response of afferent projections from touch receptors in the fingertips carried significant information about pressure in the relative timing of the first spike. To utilize information encoded in spike times, temporal codes typically require a signal that indicates the onset of the stimulus. In the cortex there is evidence for an alternative model of temporal coding in which information is encoded by the phase relative to one or several background oscillations [186], rather than relative to an external input signal. As an example, hippocampal place cells in rodents encode the spatial location of the animal by their spike timing in relation to the phase of gamma band oscillation [156]. There have been studies that explore the role of temporal coding in cortical processing in the cortex [186], where the neurons are not necessarily directly driven by the spatio-temporal features of the external input but are driven by higher cognitive behaviors. For detailed review of the role of temporal coding in biological systems we recommend reading [189].

It should be clear from this brief account that biological systems employ multiple mechanisms and resources for efficiently processing and classifying spatio-temporal spike patterns. Is it advantageous for neural processing to encode information in spike timings? What are the computational implications of embedding information in the spike timing?
2.1 Computational merits of precise spike timing

Neurons propagate signals rapidly over large distances by generating action potentials also known as spikes. Sensory neurons, specialized to respond to specific sensory stimuli, fire single or sequences of these spikes in response to external stimuli, which are then transmitted to the central nervous system. The information encoded in these spikes is then decoded and processed to produce appropriate motor commands delivered through motor neurons. Could the precision of spike times be a useful medium for information transfer and function of the nervous system? In order to answer this question we first need to address a more fundamental question: How can the nervous system encode and decode information through spikes?

2.1.1 Encoding

There are several potential coding schemes that neurons could employ to encode information in their spike patterns [193]. Let us have a brief look at some of the coding schemes that are most often used in literature.

The Rate code is the most commonly used coding scheme in most modelling studies and experimental work. This follows historically from the first observations made by Adrian in studying sensory systems, who showed that firing rate recorded from sensory fibres increased with increasing stimulus intensity. This idea was adopted in artificial neural networks where each unit sends its activation level to all its targets as a continuous value, this value being a representation of the mean firing rates of biological neurons. A similar encoding is performed in the Count code. Count code encodes information in the number of neurons in a population that have fired within a certain time window. In fact, the original experimental findings of Adrian were reported as the number of spikes within a time window after the stimulus onset and so it is probably more accurate to associate those measurements as count code.

The Binary code on the other hand encodes information in the pattern of neurons that have fired within a time window. The neuronal activity in binary code is seen to be in one of the two states: active or inactive.

The Timing code relies on precise spike timing of each neuron to encode information. The amount of information that can be transmitted with this coding scheme is not only dependent on the number of neurons spiking but also the precision with which spike timing can be determined. The Rank order coding is a special case of the timing code. Rather than looking at precise timing of spikes, it relies on the order in which the neurons fire. There are other codes using synchrony to group neurons spread across different times or phases. Instead of looking at neuronal activity as in the binary code, the neurons could be grouped based on their synchronicity. The number of synchronous patterns observable in this case is a lot more than the capacity of binary coding scheme. Please see Fig.2 of [193]
for a description of such a coding scheme. The population temporal code [210] is a special case of timing code where the identity of individual neurons is irrelevant. The information is embedded the cumulative activity level of the population varying with time. A network with asymmetric connectivity and axonal delays has been shown to utilize population temporal code in order to form invariant representations of visual patterns [210].

**Capacity**

In order to assess the merits and demerits of various coding schemes, we quantify the information transfer capabilities of each of the above discussed coding schemes. We define encoding capacity as the total number of patterns that can be represented by a fixed number of neurons $N$ spiking within a certain amount of time $t$. To make the quantification easier each neuron is restricted to spike only once (if at all) within the time period $t$.

In the case of count code, the capacity is simply the total number of spikes possibly produced by $N$ neurons, which is a minimum of 0 and a maximum of $N$. That makes a total of $N + 1$ patterns that could be communicated by $N$ neurons using count code. Binary code embeds information in whether or not a neuron fires and is similar to representing 0 or 1 in binary. Just as $N$ binary digits can represent $2^N$ numbers, $N$ neurons can represent $2^N$ patterns with binary code. The timing code involves the timing of spikes to embed information. Let us assume our time has a resolution of 1 ms and our spike pattern is of $t ms$. A single neuron in this case can spike at $t$ different times and is similar to a single digit of a non-binary number system such as decimal ($t = 10$) or hexadecimal ($t = 16$). So analogous to $N$ digits of a base-$t$ number system, $N$ spiking neurons can represent $t^N$ patterns. Synchrony code is similar to temporal code with the difference that it is set in the context of background oscillations and so the temporal resolution with which a phase can be distinguished is lower than that with precise spike timings. Its capacity can therefore be defined as $t_\Phi^N$, where $(t_\Phi)$ is the number of distinct phases that can be distinguished.

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Capacity</th>
<th>Equivalent bits</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count code</td>
<td>$N+1$</td>
<td>$\log_2(N + 1) = 3.46$</td>
<td>Equivalent to rate code.</td>
</tr>
<tr>
<td>Binary code</td>
<td>$2^N$</td>
<td>$N = 10$</td>
<td></td>
</tr>
<tr>
<td>Timing code</td>
<td>$t^N$</td>
<td>$N \times \log_2(t) = 33$</td>
<td>In $t = 10 ms$ assuming time resolution of 1 ms.</td>
</tr>
<tr>
<td>Rank order code</td>
<td>$N!$</td>
<td>$\log_2(N!) = 21$</td>
<td></td>
</tr>
<tr>
<td>Synchrony code</td>
<td>$t_\Phi^N$</td>
<td>$N \times \log_2(t_\Phi) = 20$</td>
<td>Assuming possible phases $t_\Phi$ that can be distinguished in 10 ms is 3.</td>
</tr>
</tbody>
</table>

*Table 2.1: Information transmission capacity of $N$ neurons employing different coding schemes. For simplicity we limit the number of spikes per neuron to a maximum of one. The equivalent bits required to hold the same amount of information are shown for $N = 10$ neurons for comparison.*
within a given length of time. Rank order code relies on the order in which neurons spike. The total number of ways in which $N$ neurons can be arranged in a line in the order of their spike timings is $N!$. Table 2.1 compares information transfer capacity of each of these coding schemes.

As you can see, from this brief overview of different coding schemes, count coding is the most inefficient in encoding information with spikes among the coding schemes considered here. The number of patterns that one can represent using this encoding scheme with $N$ neurons is only $N + 1$ patterns. Temporal codes on the other hand show a vastly superior (30 more bits in the example shown in the table) information transfer capacity. The number of patterns one can represent with $N$ neurons within a time $t$ is $t^N$. That is of course because the information is contained both in space (identity of neuron) and time of individual spikes.

**Energy efficiency**

While, technically speaking, any information can be transferred through spikes using any of the above coding schemes, only some of those are practical and meaningful. For example, while one can represent a number using binary code, a more practical way to do the same would be to represent it through the firing rate of a neuron, or even better, through a single ISI of appropriate length in time. This will be more evident, if we wanted to represent timing information of a stimulus. We can either gather all the information about the input over a time window, make an image of the stimulus over time and convert this image to spikes using rate coding; or simply spread the spikes over time corresponding to the stimulus itself. While the first approach requires special mechanisms to sample the stimulus in time windows, the latter is a more natural way of encoding the temporal information.

Considering that spike generation is one of the most expensive processes in neuronal dynamics, it is reasonable to consider an efficient coding scheme that uses the least amount of spikes to transmit information. Here again, rate coding appears to be the least efficient given that the computation of rate requires a sufficient number of spikes to compute the mean. If the firing rates are uniform, then of course the mean firing rate can be simply computed from a single inter-spike-interval, making it more efficient. But if Poisson spike trains are considered then one requires a larger number of spikes observed over a sufficiently large interval of time to compute the mean firing rate of a single neuron. The same is true even if a population of neurons is considered for computation of the mean rate. We still require a sufficient amount of spikes from different neurons in the population to estimate the mean firing rate, although in this case you do not require a long time.

Other encoding schemes such as binary code, rank order code or timing codes are more efficient than rate code. Since single spikes still carry information in them, fewer number of spikes can be employed to transfer information. Synchrony based coding schemes stand
somewhere in the middle of rate co ding and temporal coding in terms of energy efficiency.

2.1.2 Deco ding

It is not sufficient that information is efficiently encoded through spikes. Neurons should also be able to decode and operate on this encoded information to make decisions and perform actions. Let us evaluate which of the coding schemes discussed above are decodable and by what means.

The *Count code* is the easiest to decode using a simple IF with a threshold. The level of threshold can be adjusted so as to indicate that the number of input spikes have reached a certain number. The *Rate code* is a very commonly used coding scheme in modelling studies for its simplicity and ease of modelling with simple LIF neurons. By adjusting the synaptic weights of an LIF neuron, a *Perceptron* \[169\] can be built to classify input patterns. The Perceptron learning rule can also be well approximated with STDP \[110\] assuming the input spike trains are Poisson distributed. We will discuss this in detail in the next section. Decoding *Binary code* can be seen as time window based Perceptron classification task. *Synchrony based coding* is often used in modelling work although not as prominent as rate coding. Synchrony coding schemes employ coincident activity across a population of neurons to encode information. Detecting such coincidences are in fact a very natural property of LIF neurons. Depending on their leak and spiking threshold, neurons are capable of detecting coincidences over a range of time windows \[103\].

*Timing and rank order code* are not as much in use and are more difficult to decode. So while there are models that address decoding information from spikes employing temporal coding, these models are more complex. These coding schemes require some non-linear dynamics in the neuron model in addition to the spiking and refractory period of a simple IF neuron. Thorpe et al's work \[193\] is one such example where they use shunting inhibition for divisive gain control. Alternative models employ delay lines \[89\] to convert temporal code to binary code. Other works employ networks of recurrently connected neurons to form short term memory like in Liquid State Machines (LSMs) \[146\], or in *polychronous networks* \[96\]. LSMs was also used to classify signals encoded in *population temporal code* \[104\] where the input is a time varying signal provided by a single source.

While several spike coding schemes have been listed and distinguished here, they in fact belong to a continuum \[166\]. The count code for instance is defined by the total spikes within a time window. If the spikes are counted for each neuron in a population it is equivalent to rate code. If this counting is extended to a population, it is equivalent to population rate code. Narrower time windows lead to time varying rates and lead to time codes. If the same is done across a population of neurons then it is what is called population temporal code. A timing code that ignores the precise timing of spikes but only relies on the order of spikes leads to rank order coding. Synchrony coding on the other hand is temporal coding where the precision of spike timing is not entirely ignored but is coarsely
determined by its relation to phase of some global oscillations. As we have seen in the sections above timing code appears on one end of the spectrum and promises to be the most informative.

2.2 Learning to detect spatio-temporal spike patterns

Considering the fact that we live in a dynamic world and all the sensory input is in fact spread over space and time (even visual input), it is but natural to consider looking at temporal coding in greater detail. As discussed in the previous sections, temporal coding provides higher efficiency of information coding in addition to the natural means of encoding temporal information. If this is in fact the general coding scheme used by spikes at some stage in the sensory processing, it is imperative that an elegant decoding scheme exists for spatio-temporal spike patterns. The capability of a system to decode spike patterns rests on its ability to differentiate between various spike patterns that have different informations encoded in them. Such an ability to selectively respond to one specific spike pattern or a class of them is presumably imparted to neurons via a process of learning and adaptation. Several learning models have recently been proposed in order to address the problem of spatio-temporal spike pattern detection and classification.

2.2.1 Supervised learning

Learning from instructions is believed to be a fundamental property of our brain in order to learn skills and make behavioral decisions. This form of learning, where an instructional “teacher signal” governs the learning process, is called supervised learning. Several supervised learning rules have been proposed recently in the literature to address the issue of temporal coding with spikes. Let us look at some state-of-the-art models for supervised spike pattern classification.

2.2.1.1 The Tempotron

The tempotron [81] is a promising model of a neuron that learns spike-timing based decisions. The model consists of a LIF neuron driven by exponentially decaying synaptic currents generated by its $N$ synaptic afferents (see Eqn. 3.3). The tempotron is restricted to fire only once after which the membrane potential is smoothly reset to the resting potential by shunting all subsequent synaptic inputs. It is trained in a supervised manner to respond with a spike to a spike-pattern of the target class and remain silent otherwise. The tempotron learning rule used to train the tempotron updates the synaptic weights when ever the tempotron responds incorrectly. The synaptic weight changes are given by the following equation:
\[ \Delta w_i = \lambda \sum_{t_i < t_{\text{max}}} \epsilon(t_{\text{max}} - t_i) \]  

where \( \lambda \) specifies the maximum size of the synaptic update per input spike. \( \lambda > 0 \) for an incorrect response to the target spike patterns and \( \lambda < 0 \) for an incorrect response to non-target spike patterns. \( t_{\text{max}} \) denotes the time at which the post-synaptic potential \( u(t) \) reaches its maximal value. This learning rule effectively implements a gradient descent dynamics and the weights can be positive or negative.

In the classification task considered in the paper [81], \( p \) random spike patterns distributed into two classes were used, one of which was the target class. To investigate about which features the tempotron relies on to make its decisions, input patterns are generated with pair-wise synchronous spikes. Two different classes of spike patterns are generated by arbitrarily choosing pairs of neurons and assigning to one of the two (See Fig. 2.1a). After training the tempotron with these spike patterns it can be seen that the weights of synchronous afferents are both potentiated (or depressed) for the target spike pattern (See Fig. 2.1b). In contrast to the target stimuli, the weights of synchronous afferents of the non-target stimuli are anti-correlated, thereby ensuring that the tempotron does not respond to synchronous activity. This show that the tempotron relies on coincident spiking activity specific to the spike patterns of the target class while discarding the others in order to make its decisions.

The tempotron was also shown to classify spike latency patterns, where each of the \( N \) input afferents fired only once at random times within a time window of \( T = 500 \text{ ms} \). Here \( p \) input spike patterns were randomly assigned to the target or non-target class and were used to train the tempotron. It should be noted that the tempotron was shown to be
able to learn the classification task even when the load on the system $\alpha = p/N$ was above 2 (up to almost 3). This exceeds the capacity $\alpha = 2$ of a single-layer perceptron [138, 73, 28] reflecting the higher complexity of the computational performed by IF neurons across time.

2.2.1.2 ReSuMe

Remote Supervised Method (ReSuMe) [162] is a supervised learning model that allows learning transformations of input to output signals. The authors in [162] show that neurons can be trained to classify categories of input signals based on the temporal configuration of spikes. In fact, the Tempotron [81] has been shown to be quasi-equivalent to an application of ReSuMe [65] where the output signal is a single spike for the target class. This learning method represents a spiking analogy to the classical Widrow-Hoff algorithm proposed for rate-based neuron models [209]. The Widrow-Hoff learning rule differs from the Perceptron learning rule [169] in that the target response in Widrow-Hoff algorithm is a continuous variable whereas the Perceptron is a binary classifier.

The Widrow-Hoff rule defines the weight modification $\Delta w_{oi}$ for the $i$th synaptic input to neuron $o$ as

$$\Delta w_{oi} = \alpha (x_i y_d - x_i y_o)$$

(2.2)

where $\alpha$ is the learning rate, $y_d$ is the target response and $y_o$ is the actual response of the neuron $o$. This rule can seen as a compound of a Hebbian processes and an anti-Hebbain process. The Hebbian process relates to the first term representing the correlation between the target signal $y_d$ with input $X_i$. The anti-Hebbian process relates to the second negative term representing the correlation of the actual output signal $y_o$ with the input $x_i$. In the context of spiking neurons these correlation terms are interpreted as STDP and anti-STDP rules, which then results in the ReSuMe learning rule.

$$\frac{d}{dt} w_{oi}(t) = [S_d(t) - S_o(t)] \left[ a_d + \int_0^\infty a_{di}(s) S_i(t-s) ds \right]$$

(2.3)

where $S(t)$ represents a spike train as a sequence of impulses; $a_d$ is a non-correlative factor to adjust the average strength of the synaptic inputs so as to impose on a neuron a desired level of activity; $a_{di}$ defines the shape of the learning kernel used for the training process. Fig. 2.2 shows an illustration of the ReSuMe learning rule. The authors demonstrate that the learning rule can enable a neuron to learn to elicit precise sequences of spikes in response to the input and is largely independent of the neuron model since it operates on the spike timing.

Similar strategies have also been used in other works such as Spike Pattern Association Neuron (SPAN) [141] and Chronotron [66] for learning and transforming spike patterns. The authors in [141] use a batch mode incremental learning to adjust the weights instead
of online learning. The Widrow-Hoff rule is applied in a very straightforward manner by convolving the spike patterns with a kernel function which is then used to compute the error signals and adjust the weights with gradient descent. The Chronotron uses Victor-Purpura distance \[200\] to compute the error and measure a gradient descent, in contrast to SPAN whose error signal is more similar to van Rossum metric \[170\].

### 2.2.2 Unsupervised learning

The supervised learning algorithms that we have seen above rely on a teacher signal in order to define the identity a stimulus, “label”, and therefore to train the network accordingly. Unsupervised learning algorithms are a class of learning algorithms that do not rely on a teacher signal and therefore are not used for instructive learning and classification tasks. Since the examples given to the learner are unlabeled there is no teacher signal, and thus no explicit error or reward signal, to evaluate the potential solution. Instead unsupervised learning refers to the problem of trying to find hidden structure in unlabeled data. Below we investigate two state-of-the-art unsupervised learning algorithms related to spike patterns.

#### 2.2.2.1 Masquelier et al. 2008

Masquelier et al. \[127\] demonstrated in 2008 that STDP finds the start of a repeating patterns in continuous spike trains. An arbitrary fixed pattern spread across 1000 afferents was embedded at various times in randomly generated spike patterns of the same mean firing rate. The patterns were drawn from a Poisson process with a mean firing rate.
of approximately 64 Hz. These spikes were presented to a neuron with STDP synapses continuously for 450 s.

The STDP rule used by the authors in their work was an exponential update rule described by the equation

$$\Delta w = \begin{cases} 
    a^+ \cdot \exp((t_{\text{post}} - t_{\text{pre}})/\tau^+) & \text{if } t_{\text{pre}} < t_{\text{post}} \\
    -a^- \cdot \exp((t_{\text{pre}} - t_{\text{post}})/\tau^-) & \text{if } t_{\text{pre}} > t_{\text{post}}
\end{cases} \text{ LTP LTD} \quad (2.4)$$

where parameters $a^+$ and $\tau^+$ are the maximum potentiation weight change and the corresponding decay time constant. Similarly, $a^-$ and $\tau^-$ define the maximum depression weight change and decay time constant. The synaptic weight is bounded with a maximum and minimum weight.

The authors show that after learning, some of the synapses potentiate and some depress such that the post-synaptic neuron preferentially fires to the fixed spike pattern that was repeatedly presented during learning. They show that the neuron becomes selective to successive coincidences of the pattern. They also show that the neuron's response latency to the preferred pattern decreases with increasing learning time.

In subsequent work [126] the authors show, that multiple such neurons can be set to compete to perform spike pattern learning. They show that the population of neurons self-organize trying to cover different patterns or by coding different parts of each pattern. To maximize the coding efficiency they argue that the system should avoid having redundant neurons coding for virtually identical temporal parts of a pattern, but should ensure that the entire pattern is coded by avoiding having blind zones. They do this by appropriately adjusting the inhibition strength.

### 2.2.2.2 Spike-based Expectation Maximization (SEM)

A strategy similar to that used in Masquelier et al. 2009 [126] was also used in SEM model [154]. The authors show that STDP enabled spiking neurons to detect hidden causes of their input. They show that STDP is able to approximate a stochastic on-line Expectation-Maximization algorithm [52] for modeling the input data. The learning rule used in SEM is a weight dependent STDP rule as described by the following equation, where synaptic weight changes are triggered by the post synaptic neuron.

$$\Delta w = \begin{cases} 
    c \cdot \exp(-w) - 1 & \text{if } t_{\text{post}} - t_{\text{post}} < \sigma \\
    -1 & \text{if } t_{\text{post}} - t_{\text{pre}} > \sigma
\end{cases} \quad (2.5)$$

where $t_{\text{pre}}$ is the firing time of the last spike that occurred before the post synaptic spike $t_{\text{post}}$. $\sigma$ is a parameter of the learning rule that defines the time window within which the synapses can potentiate. Parameter $c$ defines the maximum potentiation weight. This learning rule is a simplification of the exponential STDP rule as is described by equation 2.4.
with the addition of weight dependence.

2.3 Summary

We evaluated different coding schemes for spike based information transfer. We observed that temporal coding, where the precise timing of spikes is used to embed information, had the most capacity of information transfer and was the most efficient encoding scheme. We discussed some of the state-of-the-art spike based learning models that rely on temporal properties of the stimulus i.e. spike timing in order to decode such spike trains. In particular, we evaluated some supervised learning models used for classification tasks, and some unsupervised learning models that aim to find the hidden structure in a continuous stream of incoming spike trains. In the following chapter we discuss the merits and demerits of unsupervised learning rules in the context of autonomous neuromorphic systems.
Chapter 3

STC: An unsupervised learning model for detecting Spatio-Temporal Correlations in spike patterns

Autonomous systems should be able to perform desired tasks in unstructured environments without continuous guidance. This is a key trait of animal or human intelligence. Such systems require the ability to gain information about the environment in order to work without intervention. An autonomous neuromorphic system therefore has to be able to gain information about its environment through its neuromorphic sensory inputs. Unsupervised learning studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns [46]. In contrast to supervised learning, there are no explicit target outputs or environmental evaluations associated with each input; rather the unsupervised learner brings to bear prior biases as to what aspects of the structure of the input should be captured in the output. The only things that unsupervised learning methods have to work with are the observed input patterns which are often assumed to be independent samples from an underlying unknown probability distribution, and some explicit or implicit a priori information as to what is important. So we can see that unsupervised learning directly relates to autonomy.

In this chapter we explore and understand unsupervised learning strategies in the context of spiking neurons and spatio-temporal spike patterns. In order to understand this we first study what kind of information can neurons perceive and learn. We choose to study the LIF neuron, which is one of the most commonly used neuron model for computation as is also a popular choice for neuromorphic implementations owing to its simplicity.
3.1 Detecting coincidences with leaky integrate and fire neurons

In order to understand how a neuron can differentiate between different stimuli encoded with spikes, we first need to understand the basic functioning of a neuron. We will discuss the most commonly used neuron model, the LIF point neuron model, to investigate its dynamics and study what features can be learned with this model in principle.

The dynamics of a simple LIF neuron are governed by a first-order differential equation. If we denote with $V(t)$ the neuron membrane voltage, the equation that expresses these dynamics is:

$$\frac{d}{dt}V(t) = -\frac{1}{\tau_m}V(t) + I(t)$$  \hspace{1cm} (3.1)

where $I(t)$ is the total pre-synaptic input current at any given time $t$ and $\tau_m$ is the membrane time constant. The pre-synaptic input current $I(t)$ can be expressed as:

$$I(t) = \frac{1}{\tau_s} \sum_{j=1}^{N} w_j \sum_{n} e^{-\frac{(t-t_n^j)}{\tau_s}} \theta(t-t_n^j)$$  \hspace{1cm} (3.2)

where $t_n^j$ is the spike arrival time of the $n$th spike at synapse $j$. The term $w_j$ denotes the synaptic weight of synapse $j$, with $1 \leq j \leq N$. The term $N$ represents the total number of synapses; $\tau_s$ represents the synapse time constant; and the function $\theta$ is the Heaviside step function. Substituting eq. (3.2) in eq. (3.1) and solving for $u(t)$ we get:

$$V(t) = \sum_{j=1}^{N} w_j \sum_{n} \epsilon(t-t_n^j) + \sum_{s} \eta(t-t_s).$$  \hspace{1cm} (3.3)

The first term of this equation describes the sum of all Excitatory Post-Synaptic Potential (EPSP) s produced by the input spikes at times $t_n^j$, while the second term accounts for the reset of the membrane potential after each of the output spikes at times $t_s$, including the effect of the neuron’s refractory period. The input EPSPs are described by the function $\epsilon(t)$, which is given by:

$$\epsilon(t) = w_j \cdot K \cdot \left[ e^{\frac{-t}{\tau_m}} - e^{\frac{-t}{\tau_s}} \right] \theta(t)$$  \hspace{1cm} (3.4)

where $K$ is a multiplicative factor that can be adjusted to normalize for the weight vector $\vec{w}$. This function $\epsilon(t)$ is the pre-synaptic spike kernel that reflects the effect of a single spike on the membrane potential $u(t)$. This methodology of applying kernels to spike trains to evaluate the membrane potential is described in the Spike Response Model (SRM) \[74, 75\]. The kernel functions described above are for LIF neurons, which is a special case of SRM.

Let us first consider a Poisson neuron that has a probability of spiking proportional
to its instantaneous input firing rate, which can be derived from the current membrane potential, given by the sum of all input EPSPs. In such a neuron model, the reset function $\eta(t)$ of eq. (3.3) does not affect the membrane potential ($\eta(t) = 0$) and the equation 3.3 depends only on its first term (the sum of all input EPSPs). With the input spike trains convolved with the EPSP kernel described by equation (3.4), since there is no membrane voltage rest, the membrane potential reaches a high value when ever the input spikes are synchronous. For constant mean firing rates of input spike trains the post-synaptic neuron’s membrane potential fluctuates around a mean value. It has a higher value than the mean when the instantaneous input firing rate is higher than the mean and lower in the opposite case. In these condition synchronous inputs increase the instantaneous input firing rate, and therefore increase the probability of a post-synaptic spike. This model can therefore be used directly to detect synchronicity [103].

Coherent spiking activity of a population of neurons is easily detectable with LIF neurons [75], although the analysis of the firing properties of LIF model is not as straight forward as in the Poisson case. This is because the value of the LIF spiking threshold has a strong effect the neuron’s firing properties. In [103] the authors show how the threshold and membrane time constant of a LIF neuron determine its ability to act as a coincidence detector. They demonstrate that even with constant mean input rate the mean output rate varies as a function of temporal structure of the input. They define “temporal structure” as the amount of spikes in the input spike trains that are phase locked to a periodic stimulus. They argue that a “small” membrane time constant ($\tau_m$ in eq. (3.1)) is essential for a coincidence detector. The small membrane time constant should be chosen such that the mean membrane potential $\bar{u}$, under uniform input spike rate, is just below the neuron’s firing threshold. Under these conditions, the neuron is sensitive to the fluctuations in the membrane potential, and can detect coincidences in its input spike trains in the same way that the Poisson neuron model described above does.

### 3.2 Unsupervised learning models rely on coincidence detection

So we can see that a LIF can detect coincidences in the input spike train. But, can such a neuron also capture temporal correlations beyond coincidences in the input spike pattern, like the ordering of spikes? We have seen in the previous chapter that some of the state-of-the-art unsupervised learning models are capable of learning and detecting randomly generated spike patterns. What is the key feature these models rely on to detect the stimuli presented in the form of spike trains? It is coincidences in the spike patterns. It is the coincidence-detection ability of the LIF model that allows single-neuron training schemes to learn to respond selectively to specific spatio-temporal spike train sequences [126, 81]. The synapses are selectively potentiated to identify the coincidences in the desired spike
Is coincidence detection a sufficient feature to be able to identify or classify spike patterns? To answer this question, let us consider a simple gesture recognition task that involves patterns of spike-trains. Take for example spikes produced by an event-based (spiking) vision sensor such as the DVS\textsuperscript{115}, in which every pixel responds only to temporal changes in light intensity in its visual field. When observing a gestures expressed as a visual stimulus moving from left-to-right and right-to-left, under ideal conditions, the left-ward movement of the visual stimulus would be equivalent to a temporal reversal of the right-ward movement. The spike patterns produced in response to the stimulus moving in one direction would then be equivalent to the time-reversed version of the patterns produced by the stimulus moving in the opposite direction.

We can transpose this example to a more generic case: let's consider five specific randomly generated spatio-temporal spike patterns and their time reversed versions, as in the case of the visual stimulus moving in two opposite directions. By using, for example, the learning strategy proposed in \textsuperscript{153}, we can train six neurons arranged in a Winner-take-all (WTA) circuit with an unsupervised STDP rule to recognize these five patterns (presenting repeatedly only the original non-reversed patterns), and the absence of the pattern. After training, five of the six neurons learn to fire selectively in response to one of the five input patterns, and a 6th neuron fires in response to the background activity, when none of the five patterns is present. The response of the neurons during presentation of their corresponding preferred spike patterns is shown in Fig. 3.1. Each spike pattern and the corresponding neuronal response is marked with a distinct color.

Figure 3.1 however shows also the response of the same set of neurons when they are exposed to the time reversed versions of the original spike patterns. As evident from these results the output neurons respond selectively not only to the trained spike patterns but also to their time reversed versions. This is because, even when the spike patterns are reversed, the key feature the learning mechanism captures is coincidences (or near-coincidences) in the spike pattern. The pattern of coincidences to which a neuron responds is encoded in the vector of weights, and thus such neurons cannot distinguish between patterns that share the same sets of coincident inputs. Similar results can be obtained using supervised learning algorithms, such as the “tempotron” model \textsuperscript{81}, which is designed to capture coincidences hidden in one class of patterns and not in the other.

In order to train neurons to respond to more complex temporal features in spike patterns it is necessary to introduce additional mechanisms that make the neuron sensitive to the causality of the input pattern, e.g., to responding differently to reversed patterns without relying solely on coincidences. Multi-compartmental models \textsuperscript{121, 205}, conductance-based neurons \textsuperscript{82}, and dendritic computation \textsuperscript{106, 205, 10, 163, 164} are all examples of strategies that allow to exhibit such properties. An alternative strategy, that does not require more elaborate neuron models and dedicated circuitry, is to exploit the dynamics of a
3.3 **STC**: An unsupervised learning model for detecting spatio-temporal correlations in spike patterns

We have seen some of the unsupervised learning strategies [127, 154] used for modeling spiking neural systems that can learn to detect spike patterns in section 2.2.2. It was shown that these models relied on coincidences to identify the incoming spike pattern and therefore failed in certain cases such as temporal reversal. We address this issue by developing a novel model for neuromorphic systems that can learn spatio-temporal features of a spike pattern and can overcome some of the limitations of the other theoretical models proposed earlier.

Spiking neural networks are typically characterized by their network topology (e.g. multi-layer, feed-forward, recurrent, etc.) and by their distributions of synaptic weights, while they seldom make use of temporal delays to carry out information processing tasks. However, temporal delays can provide an extra degree of complexity for solving computationally demanding problems, and can be used to implement faithful models of real neural network to obtain the needed sensitivity. In the following chapter such a network based model is proposed that can learn to detect and differentiate between complex spike patterns.
networks, as they account for the spike propagation delays that take place along the neuron’s axon. Indeed, axonal delays are often modeled to describe the temporal dynamics of biologically realistic spiking neural networks [59, 88, 171]. For example, it has been shown that transmission/conductance delays help enhance neural synchrony [54] and that axonal delays provide the anatomical and physiological basis for a neuronal map of inter-aural time differences in the nucleus laminaris of barn owls [35].

Some of the early models relating to auditory systems relied heavily on axonal and dendritic transmission delays. The Jeffress model [99] was the first to explain how auditory systems could register and analyze small differences in the arrival time of sounds at the two ears in order to estimate the direction of sound sources in the azimuthal plane. Licklider [116] used a similar idea in his duplex temporal autocorrelator model where he used a chain of neurons with axonal delays for pitch perception. The same can also be seen in a more complex model of auditory thalamo-cortical system described in [43]. The neuromorphic community exploited this idea to build several sensors. Sensors were built to be sensitive to specific temporal features of the stimuli. The DVS retina [114], for example, can detect temporal derivative of the light intensity of a pixel and output an ON or OFF event corresponding to an increase of decrease of intensity. Similarly several groups have built motion sensors to compute the direction of motion [192, 8] or optical flow [190]. These devices are therefore capable of detecting and processing temporal information in the sensory stimuli. They can be seen as specialized processing units acting on the raw sensory data to detect very specific features. Prior knowledge about the sensors such as the spatial proximity of the pixels and prior probabilities of the input stimuli allowed the development of these sensors tailored to detect these features. But in case of stimuli where their spatio-temporal features are unknown, there is need for a neuromorphic system that can actively learn the spatio-temporal features of the stimuli.

We take inspiration from earlier works that rely on delays for temporal processing. The neural network model we propose here - STC model - is derived from first principles and is based on a model of the auditory thalamo-cortical system described in [43]. The STC model comprises three populations of neurons, $A$, $B_1$, $B_2$ as shown in Fig. 3.2. The neurons within the populations $A$, $B_1$ and $B_2$ are spatially arranged on a line. These populations are stacked such that each column contains one neuron from each of these populations. The $A$ neurons provide spikes that represent the spatio-temporal signals which form the input to the network. Each $A$ neuron projects to $B_1$ and a $B_2$ neurons within the column in a one-to-one fashion via excitatory synapses. The $B_1$ neurons project onto $B_2$ neurons belonging to the neighboring columns. These projections introduce propagation delays $\Delta t$ akin to axonal delays and are mediated by excitatory plastic synapses. These plastic

\footnote{The use of word “column” is not to be confused with the “cortical columns” often used in neuroscience literature. It is merely meant to explain that these neurons are stacked vertically and have a spatial correspondence.}
Fig. 3.2: Neural network diagram. Circles represent neurons from populations A, B₁, B₂, and C. Input signals are produced by the spatially arranged neurons in population A. The spikes produced by neurons in A are projected onto both populations B₁ and B₂. Neurons in B₂ also receive delayed excitation over plastic synapses from B₁ neurons of neighboring units via delay neurons C. The output of the network is represented by the activity of population B₂.
synapses perform the key task of learning the temporal correlations in the time varying stimulus and enable $B_2$ neurons to learn coincidences in the incoming spikes. These $B_2$ neurons perform the role of readout population. Functionally this network consists of the following components: input neurons $A$ and $B_1$, projection delays $\Delta t$ and coincidence detection neurons $B_2$. $B_1$ neurons can be considered as a secondary input population because they only perform a simple one to one feed-forward computation of the input from $A$.

When neurons in population $A$ spike, they produce EPSPs in both $B$ populations. The synapses from $A$ to $B_1$ neurons have a slightly higher weight than those that project to the $B_2$ neurons, and is just sufficient for $B_1$ neurons to reach their spiking threshold on presentation of the stimulus. Therefore, in spite of being driven by the same activity from $A$, $B_1$ neurons spike where as $B_2$ neurons remain silent. When a stimulus, the exposure stimulus (ES), is presented repeatedly to the network, $B_1$ neurons fire stereotypically, following the input from $A$ neurons. This activity is delayed and reaches $B_2$ neurons at different times after stimulus onset. The correlation between this delayed activity and the membrane potential of $B_2$ neurons causes the plastic synapses to potentiate or depress. When a pre-synaptic spike reaches the post-synaptic neuron while its membrane is close to its firing threshold, the synapse is potentiated, otherwise the synapse is depressed.

Once the network has learned the ES, the output current of the potentiated STDP synapses coincident with the input from $A$ makes the $B_2$ neurons to fire. This causes the $B_2$ neurons to be active only when the right stimulus ES is presented and therefore, the activity of $B_2$ neurons can be used as an effective readout. A simple spike count from $B_2$ neurons is sufficient to discriminate the ES from any other stimuli.

### 3.3.1 Working principles

The working principle behind STC model is similar to that of a Reichardt detector [165]. A Reichardt detector is a simple circuit for detecting motion modeled after neural circuits found in the eyes of flies (See Fig. 3.3). The idea is that motion of a stimulus is encoded by two spatially separated detectors feeding to a comparator, one of which is fed through a slower channel than the other. When the transit time of the stimulus from the first to the second detector matches the delay associated with the first detector, both signals will reach the comparator at the same time and motion will be detected. When the stimulus is temporally reversed, i.e. moves in the opposite direction no motion will be detected because the two signals from the spatially separated detectors will arrive at different times.

This principle is what drives the $B_2$ neurons to detect spike patterns. Each $B_2$ neuron detects the temporal correlation between a subset of input stimulus coming from $A$. The subset observed by $B_2$ neurons is determined by the synaptic projections that have potentiated synapses after learning. The notion of Reichardt detector is extended further in the sense that each $B_2$ detector is responsive to specific temporal delays for more than two
Fig. 3.3: Reichardt detector to detect direction of motion between two points in space located at $i$ and $i+1$. It comprises of delay components incurring a delay of $\tau$ and multipliers $\times$. The multiplier components act as detectors of leftward and rightward motion, when the velocity of motion matches the time delay $\tau$ between points $i$ and $i+1$. In the original model \cite{165} the readout is computed with an additional component that computes the difference between the output of the two multipliers.

neurons (number of potentiated synapses per neuron). So while each $B_2$ neuron detects temporal features of a subsample of the stimulus, the population activity of $B_2$ neurons represents the similarity of the probe stimulus \textit{(PS)} to the learnt stimulus in terms of its spatio-temporal features.

While the initial network is all-to-all connected (in spirit), the learning enables this network to converge to a set of Reichardt detectors that are specifically tuned to different spatio-temporal sub-features of the stimulus. STDP based Hebbian learning enables $B_2$ neurons to learn the specific set of input stimuli that are synchronously activated. So the culmination of the ability to learn coincidences and the inclusion of slow components enables the network to learn and detect spatio-temporal spike patterns. The network model relies on the spatio-temporal correlations to learn and recognize spike patterns. We will use this principle to refer to this model in the future as Spatio Temporal Correlator (STC) model.

3.4 Summary

We have seen that \textit{LIF} neurons are capable of detecting coincidences in spike patterns. Learning strategies bank on this capability to train \textit{LIF} neurons to detect spike patterns and differentiate them from other randomly generated spike patterns. We saw that this
strategy has a limitation because it relies only on the synchrony of spikes and not on the temporal evolution of the spike patterns. We saw that such strategy confuse between spike patterns that share similar coincidences such as time reversed spike patterns. We then presented a novel neural network model, Spatio Temporal Correlator (STC) model, for learning the detection of spatio-temporal spike patterns based on the temporal properties of the stimulus. We pointed out the working principles upon which this network was developed. In the following chapter, we present how this network maps onto a neuromorphic implementation.

We validate the STC model with a neuromorphic implementation in order demonstrate its application to existent real-time analog neuromorphic multi-chip systems. Analog neuromorphic devices despite being extremely low-power and compact have their limitations. They are very imprecise because of inherent transistor mismatch and are difficult to configure. It is usually not trivial to map a neural network onto a neuromorphic system. It is important to understand what are the capabilities of the available neuromorphic hardware in order to understand how a neural network can be translated into a hardware neuromorphic implementation. So, before we move on to the neuromorphic implementation in the next chapter, we take a brief detour into neuromorphic engineering and examine the capabilities of the available neuromorphic hardware that was used to carry out the experiments.
Chapter 4

Implementation of STC model on imprecise neuromorphic hardware

Hardware implementations of neuromorphic systems [93] have been successfully used in the past to implement and characterize biophysically realistic models of the early sensory processing stages both for the visual domain [105, 117, 15, 108, 44, 113, 215, 112], and the auditory domain [206, 193, 172, 68, 174, 207, 36, 4]. This was soon followed by hardware implementations multi-neuron chips to emulate neurons and synapses [55]. These systems were typically implemented as single VLSI devices (e.g. as silicon retinas, or silicon cochleas), comprising hybrid analog/digital circuits that faithfully reproduced in real-time the bio-physics of the neural systems they modeled.

The advent of AER [123, 49, 21] facilitated the next step in the evolution of neuromorphic devices. AER allows representation of spike events with unique addresses assigned to their corresponding sources or destinations. Such events can then be transmitted over a common communication digital channel shared by a pool of neurons (source) and synapses (destination), eliminating the need of individual wires connecting spike sources to destinations. The definition of such event (spike) based communication protocols based on the AER has led to the development of a new generation of more complex multi-chip neuromorphic sensory systems [176, 67, 40, 41, 119].

4.1 Neuromorphic multi-chip system used in this work

Here we describe the hardware and software infrastructure used in this work. The hardware setup consists of three custom analog VLSI multi-neuron chips [140] connected in an AER event loop via an AER mapper [61]. All these chips can be configured through dedicated USB interfaces through custom PCBs (AMDA boards). Spikes can be sent to these chips and output spikes monitored, through another custom PCB’s (AEX boards).
USB interface[62]. The hardware runs in real-time, facilitating the integration of other sensors like retinas and cochleae that emit spikes, and provides a real-time interface to the external world. For configuration of its parameters real-time communication a python based software ecosystem called pyNCS was developed during the course of the last four years.

4.1.1 Multi-neuron chips

The analog VLSI neuromorphic multi-neuron chips used here consist of an array (one dimensional arrangement) or a matrix (two dimensional arrangement) of neurons. They are implemented using low power analog circuits fabricated on silicon using VLSI technology[94]. These neurons, comprising of a soma and several synapses, are electronic circuits designed to exhibit realistic neuronal and synaptic dynamics and are based on the LIF neuron model. They have a set of configurable parameters to modify their behavior and an AER data stream for I/O of spikes. The multi-neuron chips[91,148] used in the experimental setup described here were designed by Indiveri and Chicca and fabricated using a standard 0.35 µm CMOS technology.

4.1.1.1 Neuron model

The multi-neuron chips in my hardware setup emulate adaptive integrate and fire neurons(Fig. 4.1) with constant leak[92]. The neurons accumulate synaptic current \( I_{syn} = I_{exc} - I_{inh} \) onto a capacitor representing the membrane voltage \( V \) and emit a spike when the membrane voltage reaches a threshold \( V_{th} \). Their membrane potentials are reset to their resting potential \( V_{rest} \) after every spike is produced. The sub-threshold dynamics of such a neuron can be described by the following equations:

\[
dV/dt = \frac{1}{C} \times (I_{exc} + I_{inj} + I_{fb} - (-I_{inh} + I_{lk} + I_{adap}) \times (1 - e^{-\frac{V}{U_T}}))
\]

\[
I_{fb} = I_{fb0} \times e^{\frac{\kappa \times (V - V_{th})}{U_T}}
\]

(4.1)

where \( V \) is the membrane potential, \( C \) is the capacitance of the soma, \( U_T \) is the thermal voltage (≈25 mV at room temperature), \( \kappa \) is the sub-threshold slope factor[120], \( I_{exc} \) and \( I_{inh} \) are the excitatory and inhibitory synaptic currents, \( I_{inj} \) is the injection current, \( I_{adap} \) is the adaptation current and \( I_{fb} \) is the feedback current that is activated when the membrane potential reaches the spiking threshold \( V_{th} \) that can be adjusted by a bias parameter. The membrane potential \( V \) never goes negative as can be seen from the eqn. 4.1 A refractory mechanism on the chip maintains \( V \) at the resting potential for a certain period of time \( \tau_{ref} \) that can be configured by a bias parameter.
Fig. 4.1: Schematic diagram of an integrate-and-fire neuron. The input current $I_{in}$ is integrated onto the neuron’s membrane capacitor $C_{mem}$ until the spiking threshold is reached. At that point the output signal $V_{spk}$ goes from zero to the power supply rail, signaling the occurrence of a spike. Then the membrane capacitor is reset to zero, and the input current starts to be integrated again. The “leak” module implements a current leak on the membrane. The “spiking threshold” module controls the voltage at which the neuron spikes. The “adaptation” module subtracts a firing rate dependent current from the input node. The amplitude of this current increases with each output spike and decreases exponentially with time. The “refractory period” module sets a maximum firing rate for the neuron. The “positive feedback” module is activated when the neuron begins to spike, and is used to reduce the transition period in which the inverters switch polarity, dramatically reducing power consumption. The circuit’s biases ($V_{lk}$, $V_{adap}$, $V_{alk}$, $V_{sf}$, and $V_{rf}$) are all subthreshold voltages that determine the neuron’s properties.
4.1.1.2 Excitatory/Inhibitory synapse model

Each neuron on these chips is equipped with a number of synapses of different types. The synapses on the chips used on our setup are classified into three types - Excitatory, Inhibitory and Excitatory Plastic. The circuits used for generating the temporal dynamics of excitatory and inhibitory synapse are based on the current-mode Differential-Pair Integrator (DPI) circuit proposed in [17]. These synapses produce Excitatory Post-Synaptic Currents (EPSCs) and Inhibitory Post-Synaptic Currents (IPSCs) respectively with realistic temporal dynamics in response to a presynaptic input spike. The excitatory(inhibitory) synapses produce an exponential time decaying positive(negative) current for every incoming spike. The dynamics of synaptic current [17] on the arrival of a spike is approximately described by the following equation:

$$I_{syn}(t) = \begin{cases} I_{gain} \frac{I_w}{\tau} e^{(t-t^-)/\tau} + I_{syn} e^{(t-t^-)/\tau} & \text{charge phase} \\ I_{syn} e^{(t-t^+)/\tau} & \text{discharge phase} \end{cases}$$

where $t^-$ and $t^+$ are the pulse onset and offset times corresponding to the $i$th spike. $\tau$ is the synaptic time constant and is inversely proportional to $\tau_I$: $\tau = C \tau_I / \kappa$. $I_w$ can be configured through a bias to appropriately set the synaptic weight $W_{syn}$. The inhibitory synapse is simply the complimentary circuit of the excitatory synapse with identical dynamics. The inhibitory synaptic current discharges the capacitor of the soma while the excitatory synaptic current charges it. A detailed mathematical analysis of the DPI synapse and its dynamics is described by Bartolozzi and Indiveri in [17, 39].

In addition to the above described temporal dynamics the synapses also include short-term depression. The depression rate $\Delta D$ is configurable using a bias parameter. The recovery dynamics of depression $D(t)$ of the synapse is described by the following equation:

$$\frac{dD}{dt} = M(1 - D^{\frac{\kappa}{\tau_d}})$$

where $M$ is related to the recovery time constant of synaptic depression $: \tau_d = 1/M$; $\kappa$ is the transistor’s sub-threshold slope factor. $D$ ranges from 0 to 1, where $D = 1$ for no synaptic depression. A detailed mathematical analysis of the weight dynamics due to synaptic depression at the circuit level can be found in [22].

4.1.1.3 Plastic synapse model

A modified version of the STDP rule is implemented in the excitatory plastic synapses included on the neuromorphic chips used in the setup [140, 78]. The synaptic update rule [24] adjusts the synaptic weight, or efficacy, $X$ upon arrival of a pre-synaptic spike, depending on the instantaneous membrane potential and the internal state of the post-synaptic neuron [70, 24]. The internal state, $C(t)$, is putatively identified with the post-
synaptic neuron calcium concentration, driven by the firing of the neuron \[184\].

The synaptic efficacy, \(X\), is altered according to rules given in Equation 4.4. If at the time of arrival of a pre-synaptic spike the post-synaptic membrane voltage, \(V(t)\), is above threshold \(\theta\) and its internal state \(C(t)\) is within the bounds \([\theta_{\text{up}}, \theta_{\text{up}}]\), then the synaptic efficacy is increased by an amount, \(a\). On the other hand if \(V(t)\) is below the learning threshold \(\theta\) and \(C(t)\) is within the bounds \([\theta_{\text{down}}, \theta_{\text{down}}]\), then the synaptic efficacy is decreased by an amount \(a\):

\[
X = X + a; \quad \text{if } V(t) > \theta \text{ and } \theta_{\text{up}} < C(t) < \theta_{\text{up}} \\
X = X - a; \quad \text{if } V(t) \leq \theta \text{ and } \theta_{\text{down}} < C(t) < \theta_{\text{down}}
\] (4.4)

\(X\) drifts to one of two stable states \(X_{\text{hi}}\) or \(X_{\text{lo}}\), depending on whether \(X(t)\) is above or below the threshold, \(\theta_X\):

\[
\frac{dX(t)}{dt} = \alpha; \quad \text{if } X(t) > \theta_X \\
\frac{dX(t)}{dt} = -\alpha; \quad \text{if } X(t) \leq \theta_X
\] (4.5)

The internal state, \(C(t)\), is driven by firing of the neuron and is governed by:

\[
\frac{dC(t)}{dt} = -\frac{C(t)}{\tau_C} + J_C \sum_i \delta(t-t_i)
\] (4.6)

where \(J_C\) is the amount of calcium contributed by a single spike. The synaptic current from the plastic synapses to the soma is mediated by NMDA gating. The synaptic current through these plastic synapses only have an effect on the post synaptic membrane potential \(V(t)\) if the membrane potential is above a threshold \(V_{\text{mda}}\). The threshold \(V_{\text{mda}}\) can be controlled through an external bias parameter. This NMDA gating is a useful mechanism to detect coincidences as we will see later in this chapter.

4.1.1.4 The IFSL-WTA Chip

The IFSL-WTA chip [91] comprises an array of 128 LIF neurons [92] and \(128 \times 32\) synaptic circuits. Each neuron in the chip is connected to 2 excitatory, 2 inhibitory and 28 excitatory plastic synapse circuits of the type described above.

This chip is also equipped with a synapse multiplexer, which enables us to redirect synaptic currents from neighboring rows onto a single neuron. This allows the use of more synapses per neuron, at the cost of using fewer neurons in total. Depending on the multiplexer configuration it is possible to achieve combinations of synapse and neuron numbers in binary steps. Practically this means the chip can be used in one of the following configurations: 128 neurons with 32 synapses each; 64 neurons with 64 synapses each; 32
neurons with 128 synapses each; ... 1 neuron with 4096 synapses.

The chip has hard-wired on-chip connectivity that enables it to perform soft winner-take-all operations [56]. This connectivity is not used in the experiments performed in this work but is described here for the sake of completeness. The first 124 neurons on the chip form the excitatory population and the remaining 4 the inhibitory population. These populations are recurrently connected. Each excitatory neuron is connected through excitatory synapses to itself and its 3 immediate neighbors on either side. The strengths of these connections are configurable through global biases. All the excitatory neurons project to the inhibitory population which projects back to each neuron in the excitatory population. The synaptic weights and time constants of these projections are also configurable with global biases.

4.1.1.5 IF2D-WTA-Chip

The IF2D-WTA chip comprises a two-dimensional grid of 32 × 64 linear integrate-and-fire neurons. Each neuron in the chip is connected to 3 synaptic circuits (2 excitatory, 1 inhibitory) of the type described in the previous sections. The chip also has hard-wired on-chip connectivity which was not used in the experiments described in this work. Similar to the IFSL-WTA, every neuron on the IF2D-WTA makes excitatory projections to all six of its neighboring neurons. Additionally each neuron projects to its second neighbours along the vertical axis. The synaptic weights of these connections are configurable through global biases. These local excitatory projections can be used to implement a single two dimensional or several one dimensional winner-take-all circuits in conjunction with a population of inhibitory neurons from a different chip.

4.1.2 Multi-chip setup

A multi-chip setup, illustrated in Fig. 4.2, was used for the experiments conducted in this thesis. The setup comprises of two IFSL-WTA chips (CHIP 1, CHIP3) and one IF2D-WTA chip (CHIP 2) interfaced to each other by means of AER communication infrastructure [62]. The neuromorphic chips are also interfaced to a workstation using dedicated PCBs - AMDa boards. These boards provide a programmable interface to set bias voltages of various pins on the chips using a microcontroller and several Digital to Analog Converters (DACs). The workstation is used to stimulate (via AEX boards [62]) the synapses on the chips with sequences of AEs (spike trains) and to monitor the activity of the neurons. An additional AER mapping device is used to map events from a source neuron to a destination synapse belonging to another neuron (either on the same chip, or on a different chip).
4.1.3 AER Communication

Within each neuromorphic AER based chip, each cell can be accessed through an AE with an n-bit address assigned to that particular cell. But in a multi-chip setup where there are multiple chips operating on an AER bus, an n-bit address might refer to multiple cells on different chips. To disambiguate this and to be able to specify which particular chip one want to access or monitor, some additional bits are added to the address space. These bits encode the channel on which a particular chip sends its output spikes or receives its input spikes. Since the setup we are using is equipped with 3 chips each with its own input and output AER activity, a minimum of 6 channels are required. In this particular implementation of the setup, 4 bits were used for channel information and 16 bits for address space to access the neurons and synapses on the chips.

While the routing of AEs within a chip is done through dedicated arbiters and decoders within the chip, external routers are required to arbitrate and filter AEs for the channel addresses. This is done through custom PCBs called AEX boards. These boards enable AER chips to communicate with other external devices, through one of three modes: 32 bit serial I/O ports, 16 bit parallel I/O ports or USB. Each of these boards in the setup is programmed to filter/stamp specific channels for input/output spikes to the chips. While all other ports operate asynchronously, the USB port time stamps the events before sending them to a PC.
4.1.4 AER Mapper

The AER mapper [61] allows the implementation of a wide range of neural network topologies including multi-layer networks and fully recurrent networks. Typically AEs are tagged with the address of the source neuron. The AER mapper uses this information to find the addresses of destination synapses from a lookup table. It then generates AEs with the synapse addresses found in the lookup table and puts back on an AER bus. In this way, arbitrary networks and connectivity patterns can be implemented. The topology of a desired network is realized by programming a look-up table with source-destination address pairs. The AER mapper utilizes it to route spikes from source neurons to destination synapses.

The AER mapper used in this work has the following features.

• 20 bit AER source address space.
• 20 bit AER destination address space.
• 6 bit spike transmission probability per connection.
• Maximum fanout = 512

4.1.5 Setup capabilities

The AER configuration of the setup described reserves 4 bits in its address space to identify the I/O channels of chips. This allows a maximum of 16 channels, corresponding to 8 neuromorphic chips (two channels per chip: one for input, one for output) to be included in the setup. This is sufficiently large to accommodate four chips in addition to the three chips used in the experiments described in this thesis. This could help in increasing the resources available in the setup in terms of the number of neurons and synapses available for implementing neural networks. Additional neuromorphic chips like the DVS [114] or the silicon cochlea [173] could be added to the setup in order to provide real-world AER inputs to the multi-neuron chips. With the setup installed with two IFSL-WTA chips and one IF2D-WTA chip, we have access to a total of 2 * 128 + 32 * 64 = 2304 neurons. The total number of synapses equipped with long term plasticity split across two chips is 2 * 128 * 28 = 7168. The mapper allows arbitrary connectivity with a maximum fan out of 512 for building neural networks. These numbers are sufficiently large for all the proof of concept experiments and demonstrations in this work.

4.1.6 Software infrastructure

While computational neuroscience models simulate neurons and synapses using parameters directly related to their biological characteristics (such as leak conductance, time constants, etc.), neuromorphic VLSI systems emulate them using circuits that can be configured by
setting bias voltages and currents. The biases in these circuits are often only indirectly related to the parameters of computational neuroscience models. More generally, the relationship between parameters in theoretical models, software simulations, and hardware emulations of spiking neural networks is highly non-linear, and no systematic methodology exists for establishing it automatically.

Current automated methods for mapping the [VLSI] circuits bias voltages to neural network type parameters are based on heuristics and result in ad-hoc custom made calibration routines. For example, in [27] the authors perform an exhaustive search of the parameter space to calibrate their hardware neural networks, using the simulator-independent description language “PyNN” [45]. This type of brute-force approach is possible because of the accelerated nature of the hardware used, but it becomes intractable for real-time hardware or for very large systems, due to the massive amount of data that must be measured and analyzed to carry out the calibration procedure. An alternative model-based approach is proposed in [150], where the authors fit data from experimental measurements with equations from transistors, circuit models, and computational models to map the bias voltages of [VLSI] spiking neuron circuits to the parameters of the corresponding software neural network. This approach does not require the extensive parameter space search techniques, but new models and mappings need to be formulated every time a new circuit or chip is used, making it’s application quite laborious.

A systematic and modular framework was developed for the tuning of parameters on multi-chip neuromorphic systems that combines and extends the approaches described above. On one side the modularity of the framework allows the definition of a wide range of generic (network, neural, synapse, circuit) models that can be used in the parameter translation routines; on the other side, the framework does not require detailed knowledge of the hardware/circuit properties, and can optimize the search and evaluate the effectiveness of the parameter translations by measuring experimentally the behavior of the hardware neural network. We implemented this framework using the Python programming language, and making strong use of its object-oriented features. Indeed, Python’s recent popularity in the neuroscience community [188], combined with its platform independence and the ease of extending it with other programming languages, makes it the natural choice for this framework.

The framework consists of two software modules: pyNCS and pyTune (see Fig. 4.3). The pyNCS tool-set allows the user to interface the hardware to a workstation, to access and modify the [VLSI] chip bias settings, and to define the functional circuit blocks of the hardware system as abstract software modules. The abstracted components represent computational neuroscience relevant entities (e.g. synapses, neurons, populations of neurons, etc.) which do not depend directly on the chip’s specific circuit details, and provide a framework that is independent of the hardware used. The pyTune tool-set allows users to define abstract high-level parameters of these computational neuroscience relevant enti-
Fig. 4.3: Neuromorphic system configuration framework. Low-level drivers interface custom chips to workstations (e.g., using USB connections). The pyNCS tool-set abstracts the chip specific characteristics, defines the setup, and the chip’s functional blocks (e.g., populations of neurons). The pyTune tool-set performs the calibration of high-level parameters and optimization of cost functions, using the optimization algorithms in the base sub-module.
ties, as functions of other high- or low-level parameters (such as circuit bias settings). This tool-set can then be used to automatically calibrate the properties of the corresponding hardware components (neurons, synapses, conductances, etc.), or to determine the optimal set of high- and low-level parameters that minimize arbitrary defined cost-functions. Using this framework, neuromorphic hardware systems can be automatically configured to reach a desired configuration or state, and parameters can be tuned to maintain the system in the optimal state.

4.1.6.1 The pyNCS tool-set

The pyNCS tool-set acts on neuromorphic chips interfaced to workstations. At the lowest level dedicated drivers are required to interface custom neuromorphic chips to computers. Although custom drivers must be developed for each specific hardware, they can be cast as Python modules and integrated as plug-ins in the pyNCS tool-set. Once the drivers are implemented, pyNCS creates an abstraction layer to simplify the configuration of the hardware and its integration with other software modules. The experimental setup is then defined using information provided by the designer on the circuit functional blocks, their configuration biases, and the chip’s analog and digital input/output channels. The setup, the circuits, and their biases are encapsulated into abstract components controllable via a Graphical User Interface (GUI) or an Application Programming Interface (API).

Experiments (equivalent to software simulation runs) can be defined, set-up, and carried out, using methods and commands analogous to those present in software modern neural simulators such as Brian [80] or PCSIM [157]. The topology of the neural-network can be described in terms of populations of neurons (Population) and inter-connections thereof (Mapping). The properties of the connections can be defined by the synapses used to make these connections (e.g. excitatory, inhibitory or plastic). pyNCS provides convenient methods to build such topologies on the neuromorphic setups. Listing 4.1 demonstrates how this works. pyNCS uses a server-client architecture, thereby allowing multi-client support, load sharing, and remote access to the multi-chip setups. Thanks to this server-client architecture multiple clients can control the hardware remotely, regardless of the operating system used.

```python
import pyNCS

# SETUP DEFINITION
setup = pyNCS.NeuroSetup('setuptype.xml', 'setupfile.xml')

# POPULATION DEFINITION
recurrent_neurons = pyNCS.Population('','', '')
recurrent_neurons.populate_by_number(setup, chipid='ifslwta', neurontype='excitatory', n=100)
input_neurons = pyNCS.Population('','', '')
input_neurons.populate_all(setup, chipid='dvs', neurontype='pixelon')
```

41
# CONFIGURATION
setup.chips['ifslwta'].load_parameters('defaultParameters')

recurrent_neurons.neuronblock.soma.parameters['leak'].v = 0.1 # volt

# NETWORK TOPOLOGY DEFINITION

recurrent_C = pyNCS.Connections(recurrent_neurons,
                                recurrent_neurons,
                                'learning',
                                'all2all')

input_C = pyNCS.Connections(input_neurons,
                             recurrent_neurons,
                             'excitatory',
                             'random_all2all',
                             {'p': 0.2})

# MONITORING TOOLS
outMon = pyNCS.SpikeMonitor(recurrent_neurons.soma)
inMon = pyNCS.SpikeMonitor(input_neurons.soma)

setup.import_monitors([outMon, inpMon])

# COMMUNICATE WITH SETUP (MONITOR/STIMULATING)
setup.run(duration=2000) # ms

# PLOT ACTIVITY
setup.monitors.raster_plot()

Listing 4.1: A code snippet showing how one can interact with the hardware with pyNCS. In the example code shown here, two populations are instantiated on two different chips (IFSL-WTA and the DVS), connections created between those populations, bias parameters set and finally the activity of the network monitored and plotted.

4.1.6.2 The pyTune tool-set

The pyTune tool-set is a Python module which automatically calibrates user defined high-level parameters, and optimizes user defined cost-functions. The parameters are defined using a dependency tree that specifies lower-level sub-parameters in a recursive hierarchical way. An example of such type of dependency tree is shown in Fig. 4.4. This hierarchical scheme allows the definition of arbitrarily complex parameters and related cost-functions. For example, synaptic efficacies in neural network models can be related to the bias voltages in neuromorphic chips which control the gain of synaptic circuits. In the DPI synapse [17], there are three bias voltages that simultaneously affect the synaptic gain in a non-linear fashion. Using the pyTune tool-set it is possible to automatically search the space of these bias voltages and set a desired synaptic efficacy by measuring the neuron’s response properties (e.g. mean output rate) from the chip.

The automated parameter search could be applied to more complex scenarios to optimize high-level parameters related to network properties. For example, the user can specify the mapping between low-level parameters and the gain of a winner-take-all network [214],

42
or the error of a learning algorithm [84]. Furthermore, the pyTune tool-set is not restricted to neuromorphic chip setups: it can be used in pure software simulation scenarios in which it is necessary to optimize cost-functions that involve complex or abstract parameters, as a function of direct or low-level model parameters.

In order to demonstrate how pyTune works we did the follow experiment. We created a population of 5 neurons and defined the Rate parameter as the mean firing rate of the population. Built-in functions of PyNCS are used to stimulate the neurons, monitor their output spiking activity and compute the population mean firing rate. In principle the population’s mean firing rate can depend on several parameters (e.g. injected currents, recurrent connectivity, external inputs, time-constants). In order to simply illustrate how pyTune handles the dependencies we define the parameter as dependent on two sub-parameters, the Injection current to the neurons and the Leak current of the membrane. These are in fact biases of the chip, i.e. voltages to the gate transistors generating the injection current (p-type transistor) and the leak current (n-type) [82]. To visualize the dependence of Rate on its two sub-parameters, we carry out a two-dimensional sweep across the parameter space. The points in the 3D plot of Fig. 4.3a represent the values of Rate, which lie on a non-linear surface because of the exponential relationship between
Fig. 4.5: Parameter tuning with pyTune. (a) A 3D wire-frame plot of the parameter space of a parameter $\text{Rate}$ in relation to two different parameters. Each dot on the wire-frame nodes is a measure of the $\text{Rate}$ value. The surface represents the non-linear dependence of the parameter to be optimized ($\text{Rate}$) on the two sub-parameters ($\text{Leak}$ and $\text{Injection}$). The sub-parameters are voltages on the gate of the n/p-transistor which controls the current of the leak/injection to the neuron circuit [92]. The blue line shows the algorithm’s path during the optimization process (the red star represents the final point at 48.3 Hz). The error-bars on each point show the standard deviation on the rate measurement computed over the population, mainly due to transistors mismatch. (b) The cost function is minimized to set the $\text{Rate}$ parameter to a desired value of 50 Hz. Error bars represent standard deviations. The final value of the population rate is 48.3 Hz, with a deviation from the desired value below the tolerance we passed to the algorithm. See text for details.

the biases and their respective currents. The default cost function is $(r - r_{\text{desired}})^2$, where $r$ is the value of $\text{Rate}$ measured from the hardware system and $r_{\text{desired}} = 50 \text{ Hz}$ is the target value. In this example, pyTune minimizes the cost function using an implementation of the Truncated Newton algorithm [144] provided by SciPy’s optimization module [101]. The blue path shown in Fig. 4.5a connects the points measured by pyTune while setting the $\text{Rate}$. Figure 4.5b shows the progress of the optimization algorithm (cost versus iteration step). After 10 iterations, the algorithm converges to a mean rate of 48.3 Hz, which is compatible with the target rate of 50 Hz and the tolerance of 3 Hz declared as argument of the algorithm. The code used to produce this data is available online (http://ncs.ethz.ch/publications/examples-iscas-2011).

4.2 Mapping the STC model onto the hardware system

We have had a brief overview of the available neuromorphic ecosystem system used in this work. We have discussed the types of neurons and synapses available to us. We will investigate how the STC model can be implemented on this hardware and weather or not
the hardware implementation has the desired computation capabilities.

The STC model described in the section 3.3 consists of three populations of neurons. Based on the number of neurons and plastic synapses available on the neuromorphic multi-chip system described above, we decided to implement the STC model with 32 input channels. The network topology of STC model shown in Fig. 3.2 was obtained by mapping the populations $B_1$ and $B_2$ onto CHIP-1 and CHIP-2 respectively (IFSL-WTA chips). The synapse multiplexers on CHIP-1 and CHIP-2 were configured to have 32 active neurons with 128 input synapses each, 112 of which are plastic. Each neuron in $B_1$ was connected to a neuron in $B_2$ using three plastic synapses. The use of multiple redundant synapses is helpful in reducing the effect of device mismatch [158] in the plastic synapse circuits by ensuring that at least a fraction of these synapses exhibit the desired dynamics.

The input population $A$ was simply realized through a PC generating spikes that were fed into the AER stream. The essential computational components required to implement the model described in Fig. 3.2 are transmission delays and coincidence learners. The realization of these functions with point neurons on the neuromorphic system is described in the following subsections.

4.2.1 Implementing delays with neuromorphic circuits of point neurons with no delays

While there have been sporadic attempts at implementing axonal delays in hardware spiking neural networks [57, 155, 205, 204], most VLSI neuromorphic setups do not support them, either at the single VLSI device level, or in multi-chip setups. Recent developments in the construction of VLSI spiking neural networks focus increasingly more on distributed, multi-chip setups [177, 185]. Much like the setup described above, these setups typically consist of several multi-neuron chips comprising hybrid analog/digital neuromorphic circuits, interfaced among each other using asynchronous event based digital communication modules. A common communication protocol used in these setups is the AER [48, 21]. In this representation spikes (AER produced by neurons) are routed from one chip to the other using a specified addressing schemes via custom digital boards, typically comprising one or more FPGAs [20, 61]. In principle, one could therefore exploit the digital domain used in the event-based communication across chips to emulate axonal delays. But this is not an optimal solution, as it requires additional dedicated hardware overhead. For example, in [175] axonal delays are implemented by accumulating address events in pulse packets, time-stamping them, and transmitting them to a dedicated digital network chip. Here the events are held, sorted, and buffered until a target delay is reached (after which they are sent to their target destination). While this approach is flexible and accurate, it requires specialized hardware for the computationally intensive real-time event sorting, and looses the efficient representation of time in the AER, where events are transmitted as they happen, and time represents itself.
Fig. 4.6: Histogram of delays of 1070 of the 2048 neurons on CHIP–3 due to device mismatch in the hardware. Delays range from 0.7 ms to 8.6 ms mean ≈ 2.5 ms. The remaining neurons either fire more than once or not at all.

An alternative approach, is to exploit both the digital domain used for the inter-chip communication and the analog one used with the silicon neurons and synapses inside the chips. This approach would not require explicit time-stamping of each event and so could reduce overhead costs. A common way of implementing temporal delays in electrical engineering is by using low-pass filters. We follow the same approach here and use the low-pass filtering properties of the analog synapse circuits present on the multi-neuron chips. We configure the synapse parameters such that the integration of a single pulse (the input spike) produces an output spike, after a set delay $\Delta t$. Given that the weights associated with the synapses are strong enough to produce a single output spike in the post-synaptic neuron, the time difference between the pre-synaptic spike and the post-synaptic spike is considered equivalent to propagation/transmission delay. Therefore, every projection in the model that requires a delay is passed through an additional neuron, referred to as a delay neuron. The delay neurons are labelled $C$ in Fig. 4.9. Neurons from CHIP–3 (IF2D-WTA chip) were used to implement the population $C$ implementing delayed transmission to $B_2$ neurons. By making use of inherent device mismatch present in the analog neuromorphic circuits, we implement a range of axonal delays, and exploit the communication digital infrastructure to (re)configure the placement of these delays in the neural network.

The transmission delay of a delay neuron is a function of synaptic strength, synaptic
Fig. 4.7: Delays implemented on hardware for projections $B_1 \to C \to B_2$ in ms. In case of no connection, it is marked as having 0 ms delay and is shown in blue. (a) A linear delay profile where delay increases with length of projection. The delays were chosen from the ones available in the hardware implementation and minor deviations from a pure linear profile can be seen. (b) A random delay profile where the delays were drawn for the available pool of delay neurons.

and neuronal time constants and firing threshold. Delays of the order of milliseconds can be achieved by setting a weak synaptic strength and a long time constant, to produce a long-lasting small output current. In this configuration, synaptic circuits are strongly affected by transistor mismatch \cite{158}. Therefore they produce a broad distribution of output currents, despite sharing global bias parameters. These output currents, integrated by the postsynaptic neurons, produce output spikes with a broad distribution of delays from the input spikes. Fig. 4.6 shows the distribution of delays measured across 1070 neurons on CHIP–3 (IF2D-WTA chip) for a specific configuration of its biases. The delays range from 0.7 ms to 8.6 ms with a mean of approximately 2.5 ms. Although the chip contains 2048 neurons, because of device mismatch, only a subset of these neurons can be used as delay neurons. The rest of the neurons either have too strong or too weak synaptic strength, causing the neurons to fire multiple or no spikes per incoming spike. Depending on the required delay, an appropriate delay neuron is selected and placed in the network.

The propagation delays (i.e. the time it takes for a spike to travel from $B_1$ to $B_2$ via $C$) implemented for the experiments in this chapter were chosen to be proportional to the columnar distance between pre-synaptic and post-synaptic neurons (see Fig. 4.7a). Such linear delay profile was chosen for analytical tractability of the functioning of the network. A connectivity of randomly distributed propagation delays was also used for a second set of experiments. The delay profile of this connectivity is shown in Fig. 4.7b.
4.2.2 Learning to detect coincidences with neuromorphic circuits

So the plastic projections from $B_1$ to $B_2$ are effectively replaced by two projections $B_1 \rightarrow C$ and $C \rightarrow B_2$. The projections $B_1 \rightarrow C$ are static while the projections $C \rightarrow B_2$ are plastic. The functional role of $B_2$ neurons is to learn to detect coincidences between delayed projections from $C$ and static input projections from $A$. This is realized through excitatory plastic excitatory projections from $C$ to $B_2$. These synapses are plastic and exhibit STDP, a spike based Hebbian learning rule \[33\]. The synaptic weight is incremented when the pre-synaptic spike arrives before the post-synaptic spike, and decremented if the order is reversed; hence this rule potentiates those synapses where an apparent cause-effect relationship predominates over time. A modified version of the STDP rule is implemented in analog VLSI on the plastic synapses of the IFSL-WTA chips used in this experiment \[140, 78\] (see section 4.1.1.3). The effect of different timings of pre-synaptic spikes, in comparison to the post-synaptic spike time, on the synaptic state is shown in Fig. 4.8. As the measurements show, the synapses that receive a spike within a time window of approximately $2.5\,\text{ms}$ before the post synaptic spike are potentiated in accordance with the STDP rule. This mechanism is used to learn coincidences between the spikes from $A$ and $C$. It should be noted that a burst of spikes from $A$ was used to drive the membrane potential $V(t)$ uniformly from its resting state to the firing threshold over a period of $5\,\text{ms}$. This was necessary because the learning rule relies on the post synaptic membrane potential and not the post synaptic spike timing as in the classical STDP rule. In line with this, the stimuli used in this chapter (see section 5.1.1) also consisted of bursts of spikes in place of individual spikes. Even if the post synaptic neuron did not produce a spike in response to such burst of spikes, if the membrane potential was driven above the learning threshold $\theta$ for some time, the synapses receiving a spike during that time would potentiate. Therefore it is possible to learn coincidences in the incoming spikes even if the post synaptic neuron remains silent. We exploit this behavior to ensure that $B_2$ neurons only fire when they have learnt a set of synapses that reflect coincidences in the input spikes.

In order to ensure that the $B_2$ neurons remain silent if there is no input spike from $A$, NMDA gating was activated. This mechanism shunts the synaptic current from a plastic synapse if the post synaptic membrane potential $V$ is less than a specified gating threshold. This gating threshold was chosen to be equal to the membrane learning threshold $\theta$ (see Sec. 4.1.1.3) for the hardware experiments. Since the learning rule heavily employs the post-synaptic membrane potential to determine synaptic updates, additional measures were taken to make sure that the membrane potential of $B_2$ neurons faithfully represents the input from $A$ and remains as close as possible to the activity of the $B_1$ neuron of the same column. This was done by an additional one-to-one non-plastic inhibitory projection from $B_1$ to $B_2$. The strong inhibitory current from these projections ensures that when ever a $B_1$ neuron fires, the corresponding $B_2$ neuron’s membrane potential is also reset.
Fig. 4.8: The measured effect of plasticity on the synaptic state of synapses on hardware. Several pre-synaptic spikes are sent at different times to different synapses (one spike per synapse) as shown in blue. The post synaptic neuron is also activated by a burst of spikes (also shown in blue) over non-plastic synapses to elicit a post-synaptic spike at time 16 ms (shown in red). While initially the synaptic weights of all the plastic synapses were at $X_{lo}$, some of the synapses potentiate to $X_{hi}$ as shown by the yellow bars. These potentiated synapses correspond to those that received input right before the post synaptic neuron fired.
This further guarantees that the NMDA gating is activated thereby preventing the neuron from firing simply based on the synaptic current elicited by spikes from $C$. The resultant network is shown in Fig. 4.9.

Fig. 4.9: Modified network model implemented on hardware. In addition to the projections shown in Fig. 3.2, inhibitory projections from $B_1$ to $B_2$ neurons were introduced in the hardware implementation of the network to ensure that the membrane potentials of $B_1$ and $B_2$ neurons behave similarly and faithfully represent input activity from $A$. The plastic synapses are NMDA gated.

4.3 Discussion

In this chapter we proposed a novel model STC for unsupervised learning of detection of spike patterns that can be implemented on neuromorphic hardware. We then proceeded to show how the STC model was actually realized with the available neuromorphic infrastructure.

4.3.1 Expansion and scalability of the neuromorphic system

The network architecture of the hardware system described in this section is what is technically called a shared bus architecture. In such an architecture, each neuromorphic device reads and writes using the same address bus. In order to differentiate and

---

1 All future references to $B_1$ to $B_2$ projections address delayed projections through $C$ unless explicitly mentioned as inhibitory.
uniquely identify different chips and neurons and synapses on each of them, unique addresses are assigned to each of them. While, increasing the address space would allow more neuromorphic devices to be placed in the AER loop, this approach stands the risk of flooding the AER bus with events coming from all the neuromorphic devices in the loop. This therefore limits the scalability of the setup architecture described in section 4.1. Another limitation is that of fanout of the mapper. The maximum number of fanout connections that can be realized with the AER mapper [61] is defined by the size (memory) of its lookup table and the total address space of the AER bus.

\[
\text{Fanout} = \frac{\text{Total memory}}{\text{Total number of addresses}}
\]

With increasing number of bits assigned for addressing neurons and synapses in the system, the memory assigned to each address in the lookup table goes down. Therefore to maintain flexibility in the system despite a large address space, or to have a large fanout we require large amounts of memory for the lookup table. An alternative architecture to circumvent these limitations is to adopt a local distributed mapping scheme. Such an architecture will require multiple AER mappers and optimized population mappings onto the hardware to minimize long distance spike transmissions. This is beyond the scope of this work. Such approaches are being explored and studied extensively by our colleagues [142] and by other projects involving large-scale systems like SpiNNaker [161], BrainScales [25] or Hier-IFAT [213].

4.3.2 Exploiting device mismatch for implementing propagation delays

In order to implement propagation delays a population of neurons was used whose time constants were proportional to the desired time delay. We were able to select delay neurons with different time constants by exploiting the mismatch effect in their analog circuit implementations [158]. The effect of this mismatch is shown in Fig. 4.6, where the range of delays exhibited by the neurons is plotted. Had there been no mismatch, this would not have been possible to achieve on a single chip with global/shared bias voltages. Such variability can also be found on other model parameters such as neuronal and synaptic time constants, injection currents and membrane firing thresholds. This opens up the possibility to explore network dynamics of populations with a distribution of parameter values, rather than using globally common values. While mismatch induced noise is often minimized in conventional analog VLSI design, here we try to use it to introduce variability in the populations of silicon neurons that can model or reproduce the inhomogeneities present in real neural systems.
Fig. 4.10: Dependence of effective delay on the inter-spike-interval. With decreasing [SI], the effective delay of a delay neuron decreases. The top line shows the neuron’s asymptotic delay value (4.82 ± 0.216 ms), i.e. the delay at very long [SI].

4.3.3 Reliability of delays

In section 4.2.1 we also saw that delay generation was dependent on EPSPs generated by synaptic currents with long time constants. In such cases, the amount of time it takes a pre-synaptic spike to generate a post-synaptic spike varies with the mean [SI] of pre-synaptic spikes. Fig. 4.10 shows how the delay generated by neurons decreases with decreasing [SI], i.e. increasing firing rate. This happens because of accumulation of residual synaptic currents after the spike generation of a delay neuron.

At sufficiently high firing rates, the delay mechanism breaks down if the gain is not exactly equal to one, which is almost always true. Synaptic depression could play an important role under these circumstances to modulate the mean firing rate. It could help suppress high frequency input spike trains and only transmit the onset of such activity.

4.4 Conclusions

A neuromorphic system, consisting of three imprecise real-time multi-neuron chips, was described in this chapter. These neuromorphic chips used in the setup were designed by Indiveri and Chicca. The AER infrastructure (mapper, AEX boards) was developed by Fasnacht. The software infrastructure developed during the course of my PhD for configuring and tuning the hardware system, pyNCS and pyTune, were described.

This system was used to realize the STC model proposed in Chapter 3 on hardware. The STC model relies on delays, which might arise from a number of processes including axonal
propagation and spike interaction via intermediate neurons. In the case of the hardware system used to implement the model there was no provision for the implementation of delays in the design. This problem was overcome by exploiting the inherent variation and mismatch \[158\] of the components in analog VLSI devices used in the setup. This opened up a new avenue for exploiting transistor mismatch to obtain heterogeneous parameters. Coincidence detectors that could learn the synchrony in their inputs were implemented using the learning circuits \[39\] on hardware.

The results presented in this chapter demonstrate, in hardware, how an implementation of STC network is able to learn and selectively respond to the dynamic spectro-temporal features of stimuli. The stimulus-driven modifications of the network connectivity result from the interaction between the stimulus itself and the spike-based plasticity (STDP) rule adopted for the delayed connections. After learning, the firing patterns of the neurons reflect the emergent connectivity. More of the neurons fire during a target stimulus presentation, and the network is tuned to the stimulus properties. As a result, differences in the response of the network induced by learning can be used to distinguish similar stimuli that have parametrically different dynamic properties, for example differences in the direction or speed of an FM sweep). This result as we saw in chapter 2 cannot be obtained from the state-of-the-art unsupervised learning models like Masquelier et al. \[127\] or Nessler et al. \[154\].

The experiments presented here show, in spiking hardware and in a sensory context, the emergence of feature sensitivity. It was also demonstrated how the network with a random profile of transmission delays can exhibit similar behaviour. We argued that the STC network with randomly distributed transmission delays could be used to learn and detect random uniformly distributed spike patterns. In the following chapter, we explore this idea further through phenomenological software simulations using different input stimuli, and learning rules.
Experimental results

In the last two chapters we introduced the STC model (Chapter 3) and its implementation on a neuromorphic hardware system (Chapter 4). Here we evaluate the computational aspects of the STC model through experiments on both the neuromorphic hardware and software simulations. The hardware experiments are carried out in order to confirm that the model is indeed functional and can be realized on neuromorphic systems. We demonstrate through a set of experiments on hardware that the network is able to learn to detect a class of spatio-temporal stimuli characterized by FM sweeps. We demonstrate that the network is capable of not only differentiating between time-reversed stimuli but also between different velocities of sweeps. Further experiments are conducted through software simulation of the network to demonstrate that the model is in fact capable of learning and detecting much more complex stimuli such as Poisson spike trains. We explore other learning rules through software simulations in order to investigate the network's capability to recognize repeating spike patterns embedded in a continuous stream of randomly generated spike patterns without any prior knowledge about the repeating spike pattern.

5.1 Hardware experiments

Neurons in primary auditory sensory areas are typically characterized in terms of tuning to particular spectro-temporal features. This is equivalent to 'spatio-temporal' in other modalities. Experiments in the auditory domain emphasize the importance of dynamic spectro-temporal patterns in the communication calls of mammals and birds. This has motivated the study of cortical sensitivity to Frequency Modulated (FM) sweeps \[11, 212\] and dynamic ripple noises \[107, 30, 53, 12\] as candidates for constituent features which are sufficiently simple to be parametrized. We chose to use similar stimuli for the experiments on hardware and therefore use the nomenclature of the auditory domain where spatial location corresponds to tonotopic (frequency of the auditory signal) location.

Experiments were carried out to investigate the emergence of feature sensitivity in
the hardware implementation of the STC model (Fig. 4.9) described earlier. The network connectivity was characterized after several presentation of the exposure stimulus (ES) and was compared to the predicted connectivity. Then the response of this network was evaluated with a set of probe stimulus (PS) that consisted of linear or forked frequency sweeps, as shown in Fig. 5.1, with different velocities. These results were used to determine the response curve after exposure. This was done first for the network with a linear delay profile, and subsequently with random delays (see Fig. 4.7).

5.1.1 Input stimuli used in the experiments

The population of neurons $A$ in the STC model are the source of input spikes to the network. In the context of auditory processing, the spikes elicited by this population output represent output of an artificial cochlea [36]. For experimental simplicity and as a proof of concept demonstration, these spikes were software generated and take the form of spectro-temporal patterns. These spikes represent cochlear spikes with high Q-factor i.e. each of the input neurons is tuned to a narrow range of frequencies.

Two classes of stimuli were used in these experiments. The first class is defined by linear FM sweeps (pure tones with increasing or decreasing frequency over time); the second class defined by forked FM sweeps i.e. stimuli with two tones whose frequencies diverge or converge over time. Examples of the spiking inputs, which were used as the ES and PS signals, are shown in Fig. 5.1. Fig. 5.1a illustrates a single linear frequency sweep, while Fig. 5.1b shows an example of the second class of stimuli (forked), with a diverging pattern of activity. Both types of stimuli were used with a range of FM sweep velocities. The velocity in each of these cases is defined as the number of channels activated per millisecond. For example Fig. 5.1a shows an upward sweep with the onset of activity in each channel separated by one millisecond, hence the velocity is $+1.0 \text{ ms}^{-1}$. Downward sweeps have negative velocities. In the case of the forked stimuli the velocity of the component in the upper half of the frequency axis is defined as its stimulus velocity. The stimuli with negative velocities are essentially time reversed versions of their corresponding stimuli of positive velocity. Such stimuli allow inspection of the network’s response properties based on a continuous temporal parameter (mean ISI), which is unique to each of these stimuli and can be used to define similarity between two different spike patterns. In case of random patterns, a whole class of diverse stimuli can be generated with the same mean ISI making the characterization of the network's response properties a more complicated task. Such stimuli are used later in this chapter, with software simulations, to characterize the STC network.

The differences in the spike patterns in each channel shown in Fig. 5.1 are systematically introduced to compensate for the device mismatch effects present in the neurons of population $B_1$. This is done to ensure that all the neurons in $B_1$ fire exactly once during the stimulus presentation [51]. For all experiments carried out the neurons were reset to their
Fig. 5.1: Spike patterns representing the two stimulus classes, linear and forked. Each dot represents a spike. The velocity of stimulus, i.e. the rate of sweep, is defined as the number of channels activated per millisecond. (a) Linear FM sweep at 1.0 ms$^{-1}$. (b) Forked FM sweep at −1.0 ms$^{-1}$ (defined by the upper component). (c) Linear FM sweep at 1.0 ms$^{-1}$. (d) Forked FM sweep at −1.0 ms$^{-1}$. The two stimuli on the lower half are time reversed versions of the stimuli on the top half of this plot, signified by their negative velocity.
resting state at the beginning of each trial. Similarly, the plastic synapses implementing the delayed projections were reset to their ‘low’ state \((X_{lo})\) i.e., effectively null synaptic efficacy. See section 4.1.1.3.) Input patterns were presented 30 times over a period of 3 seconds. The network ‘learns’ in a 3 second exposure phase during which it is repeatedly exposed to the same stimulus - the exposure stimulus ES.

5.1.2 Spike train distance metric for predicting the emergent connectivity

The STC network was described in section 3.3 and its hardware implementation is shown in Fig. 3.2. The learning dynamics of this network are deterministic and can be predicted given the delays in the connectivity and the input stimulus. While the input from A drives the membrane potential of \(B_2\) neurons (representing post synaptic activity), the delayed activity from \(B_1 \rightarrow C \rightarrow B_2\) drives the plastic synapses (presynaptic activity). Since both these activities are observable, a deterministic rule enables us to predict the final synaptic states after prolonged presentation of an ES. The STDP update rule is essentially Hebbian [33]. Therefore a measure of coincidences in the input spikes from different synapses of a neuron can predict whether a synapse is going to potentiate (Long Term Potentiation (LTP)) or depress (Long Term Depression (LTD)). The spike train metric called the van Rossum distance (VRD) [170] was used to measure the coincidences between two spike trains. Given two spike trains \(s_1\) and \(s_2\), VRD \(D(s_1,s_2)\) is defined as:

\[
D(s_1,s_2) = \sqrt{\int_{-\infty}^{\infty} [g * s_1 - g * s_2]^2 dt}
\]

where \(g = g(t; \tau_c)\) is a smoothing function (e.g., a decaying exponential) with time constant \(\tau_c\). Depending on the time constant \(\tau_c\), the distance metric interpolates between a coincidence detector (small time constant) and a rate difference counter (large time constant).

Given the decay rate set for the synaptic update rule (see Sec. 4.1.1.3) the time constant was fixed to \(\tau_c = 4 ms\). The smaller the distance between two spike trains, the larger is the number of coincidences between the two spike trains. So the van Rossum distance (VRD) between spikes from A and delayed spikes at \(B_2\) is small for synapses that should show an LTP. We use this measure to make predictions about the effective connectivity after learning and verify the functionality of the STC network. We define the distance matrix \(DM(i,j)\) as \(D(s_A(i),s_C(i,j))\). This is the van Rossum distance (VRD) between spikes of neuron A in the column \(i\) (\(s_A(i)\)), and the spikes of the neuron in population \(C\) projecting from \(B_1\) neuron in column \(j\) to \(B_2\) neuron in column \(i\) (\(s_C(i,j)\)). Since spikes produced by the A neurons drive the membrane potential of the \(B_2\) neurons, the distance matrix \(DM(i,j)\) is tightly linked to the final synaptic weight matrix of \(B_1 \rightarrow C \rightarrow B_2\) projections. Specifically, small VRD measures imply high probability of LTP in the corresponding
Fig. 5.2: (a) The distance, $D$, between spikes produced by neurons in population $A$ and spikes arriving at neurons in population $B_2$ for a network exposed to a linear frequency sweep at velocity of $1.0 \text{ ms}^{-1}$. (b) Corresponding synaptic weight matrix showing the number of potentiated synapses per projection from $B_1$ to $B_2$ neurons.

While the distance, $D$, acts as a coincidence measure between two spike trains, a time offset $\Delta t$ in one of the spike trains enables us to measure their temporal correlation at a time offset $\tau$ between the two. An optimal offset time $\Delta t$ corresponds to maximal correlation between the two spike trains. In the STC network, the delayed projections generate the time offset $\Delta t$ and the plastic synapses potentiate only if these correlations are above a certain threshold, thereby selecting projections that cause maximal coincidence at $B_2$.

Fig. 5.3 shows the potentiated synapses after exposing the network to three different velocities of FM sweeps. For a FM sweep of negative velocity (decreasing frequency with time) the synapses below the diagonal are potentiated (Fig. 5.3a) and above the diagonal otherwise (Figs. 5.3b, 5.3c). For $1.2 \text{ ms}^{-1}$ frequency sweep the mean distance of potentiated synapses from the diagonal is greater than that for $0.8 \text{ ms}^{-1}$ FM sweep. It can be observed that the distance of potentiated synapses from the diagonal is determined by the velocity of the presented frequency sweep. For different velocities of FM sweeps, the optimal time offset $\Delta t$ for maximal correlation between $s_A(i)$ and $s_C(i,j)$ is different; only
Fig. 5.3: Potentiated synapses, or the effective final connectivity from $B_1$ neurons to $B_2$ neurons, after exposing the network to linear Frequency Modulated (FM) sweeps at velocities (a) $-0.8 \, ms^{-1}$ (b) $0.8 \, ms^{-1}$ and (c) $1.2 \, ms^{-1}$.

those synapses for which the transmission delay $\Delta t$ causes maximal correlation between $s_A(i)$ and $s_C(i,j)$ potentiate on exposure to a given stimulus. As the delays in this network increase with increasing tonotopic distance, the larger the required $\Delta t$, the farther away from the diagonal the synapses are potentiated. If some or all of the neurons in $A$ were activated simultaneously (corresponding to $\infty \, ms^{-1}$) the optimal time offset $\Delta t$ for maximal correlation would be $0 \, ms$ or in other words this would enable the model to learn the purely spectral (or spatial) correlation in the input pattern. As the stimulus velocity decreases, the value of $\Delta t$ for maximal correlation increases. Since the value of $\Delta t$ between two neurons in the model depends on their distance (spatial) of separation, we observe a corresponding change in the pattern of connectivity for different velocities of the input stimulus.

When the network is exposed to a slightly more complex stimulus (the example I investigate is a forked frequency sweep as shown in Fig. 5.1b) the resulting connectivity is shown in Fig. 5.4. It should be noted that the connectivity is not just a linear combination of upward frequency sweep and downward frequency sweep, but also captures the correlation between the two branches of sweeps, as predicted by the corresponding distance matrix shown in Fig. 5.4a.

All of the emergent connectivity matrices in these experiments resulted from exposure to the ES in an unsupervised manner. As shown by Figs. 5.2,5.4, the post-exposure network connectivity always matches the area of minimum VRD in the distance matrix $DM$. The distance matrix $DM$ is effectively a measure of temporal correlations in the stimulus. From this we can conclude that the network implemented on hardware is capable of learning the spectro-temporal correlations in the stimulus. The resultant network can be described as a group of Reichardt detectors [165] (Fig. 3.3) that are specifically responsive to the temporal properties of the stimulus. As the resulting connectivity closely matches the distance matrix $DM$, the potentiated synapses should support reliable network responses to stimuli with spectro-temporal features that closely match those of the ES. This network
property was verified by measuring the FM sweep tuning curves, as described in the next section.

5.1.3 FM sweep tuning curves

After exposing the network to 30 instances of the ES over a period of 3 seconds, its response, i.e. the total number of spikes from $B_2$ neurons, was measured for different velocities of FM sweeps. Because the network relies on the statistical significance of the spatio-temporal correlations in its input patterns to learn its target ES without an explicit teacher signal to identify the ES, these statistics have to be maintained while testing the network's response properties. Without maintaining the statistics the network's connectivity and consequently its tuning properties would be altered while testing its response. In order to circumvent these complications, the synaptic plasticity was disabled after the presentation of ES while keeping all other network parameters unperturbed. This was also necessary in this experiment with hardware because the learning rate of the network is quite fast and presentation of any alternative stimulus would alter the synaptic states rather quickly. (See section 5.5 for further details.)

Two sets of experiments were carried out. In the first case, linear sweeps of different velocities were used to measure the FM sweep tuning curve after training to a linear FM sweep. In a second set of experiments forked sweeps of different velocities were used to measure the FM sweep tuning curve after training to a forked frequency sweep. These measurements were used to plot a tuning curve for the network. Fig. 5.5 shows two tuning curves after learning a linear and forked FM sweeps at $1.0 \text{ ms}^{-1}$. The tuning curves show a Gaussian-like profile which peaks at the velocity of ES. The width of the Gaussian is determined by the level of learning threshold, $\theta$ in Eqn. 4.4, which is equivalent to the
maximum $D(s_1,s_2)$ below which the plastic synapses are set to potentiate and depress otherwise. This width of the tuning curve can be interpreted as the network's resilience to variations in the time course of the stimulus. Qualitatively similar FM sweep tuning curves were obtained at different ES velocities for both linear and forked stimuli. This is shown in 5.5a and 5.5b, respectively. As can be seen from the figures, maximal response is seen along their diagonals implying maximum response to ES and lower response to any other.

5.1.4 Random transmission delays for $B_1 \rightarrow C \rightarrow B_2$ projections

The results described in the previous section were obtained with a linear delay profile as shown in Fig. 4.6, i.e. the projection delays were proportional to the (columnar) distance between the $B_1$ and $B_2$ neurons. While the experiments carried out here involve a small
number of neurons (32 columns), a larger network implementation with a linear delay profile would require a larger range of delays or a higher resolution of delays. Neither long delays of the order of tens of milliseconds, nor delays with high accuracy is practical for a neuromorphic implementation such as the one used for this work. It is also not biologically realistic to expect a perfectly arranged delay profile. It is more plausible that the delays are randomly distributed especially if the inputs are not spatially arranged in a linear fashion. What happens to the dynamics of the STC network under these conditions? Is a linear connectivity profile necessary for the network to learn the spectro-temporal features of the ES and selectively respond to them? When the STC network is exposed to ES synaptic plasticity potentiates connections that represent the spectro-temporal correlation in the ES as described earlier. As long as the transmission delays $\Delta t$ keep the distances $D(s_A(i), s_C(i,j))$ low, the corresponding synapses potentiate and learn to replicate the distance matrix $DM(i,j)$. Therefore the linearity of distribution of delays used earlier should not be crucial.

In order to explore the functioning of the STC network under biologically realistic conditions, it is initialized with a random delay distribution, as shown in Fig. 5.6a. The experiments conducted in the previous section were repeated with the random delay profile. The resulting network exhibits tuning properties similar to those observed for a linear delay profile. The connectivity matrix, after being exposed to forked frequency sweeps at $1.0\,ms^{-1}$ velocity is shown in Fig. 5.6c and the sweep tuning curve of this network is shown in Fig. 5.6d. Note that the tuning curves in Fig. 5.6d and Fig. 5.5d show a similar performance to that with linear delay profile. Similar results were also found for the simpler linear FM sweeps (data not shown). The network has a lower response level with random delay profile, which is a consequence of only two, not three, synapses used per projection. This reduction in number of synapses was necessary to have an all-to-all connectivity with the number of available plastic synapses per neuron. The lowering of activity also makes the tuning curve seem digital, as in the particular example shown here, although it is not always the case.

The case of random uniformly distributed spike patterns The above results show that the network with a random profile of transmission delays can learn and exhibit similar tuning properties as that of linearly arranged delays. It should be noted that random delay profiles imply that there is no structural necessity for the input neurons to be strictly tonotopically arranged. Any uniformly distributed spike pattern can be rearranged to appear as a linear sweep by sorting the neurons in the spike pattern by their spike timings. Therefore, without loss of generality, the input stimulus need not necessarily look well arranged as in Fig. 5.1 and could be shuffled to appear as random uniformly distributed patterns. By the same argument the network could also learn correlations in stimuli that are not so clearly ordered and could be any uniformly distributed spectro-temporal spike pattern.
Fig. 5.6: Connectivity and FM sweep tuning curve of a network with random transmission delays from $B_1$ neurons to $B_2$ neurons on exposure to forked FM sweeps of velocity $1.0 \text{ ms}^{-1}$. (a) Effective transmission delays from $B_1$ neurons to $B_2$ neurons (via $C$). (b) The van Rossum distance (VRD) between the spikes trains from input population $A$ and spike trains from population $B_1$ that reach neurons of population $B_2$. (c) The number of potentiated synapses per projection from a $B_1$ neuron to $B_2$ neuron after exposure to forked FM sweeps of velocity $1.0 \text{ ms}^{-1}$. (d) The measured tuning curve of the network after training.
pattern. The results demonstrated here focusses on parametrized stimuli similar to those used to characterize auditory responses and so random stimuli were not used to train the network here. We explore the dynamics of STC model on presentation of randomly generated stimuli with the help of software simulations.

5.2 Software simulations

In order to explore and quantify the computational capabilities of the STC network further experiments were carried out. These experiments required long run times and control on parameters such as network size and learning rule related parameters such as the slew rate $\alpha$. This was not feasible with the available neuromorphic system described in Chapter 4 because of limited resources and parameter reconfigurability. Therefore a software simulation platform was developed for a phenomenological study of the STC network. The simulation platform does not compute detailed dynamics of neurons or synapses in terms of membrane voltages and synaptic currents but simply operates on spikes. The neurons elicit a post-synaptic spike when ever a pre-synaptic spike is preceded by a minimum number of pre-synaptic spikes within a time window. The synaptic weights are updated based on the timing of the incoming spikes. The parameters for all these operations were chosen to approximate the dynamics of neurons and synapses in hardware experiments.

The simulations were performed based on the working principles of the STC model shown in Fig. 3.2. The platform used for the simulations works as follows. Input spike patterns are presented to $B_1$ and $B_2$ through $A$. The output of $B_1$ neurons is delayed by population $C$. The output of $C$ and input spike patterns converge on $B_2$. Coincidences between spikes from $C$ and input spike patterns are learnt at $B_2$. $B_2$ neurons spike when ever there are coincident spikes at potentiated synapses and input spikes at $B_2$. The response of the network is measured as the total number of spikes from $B_2$ neurons. It should be noted that for every input spike, the corresponding $B_1$ neuron produces an output spike (gain = 1), where as in the experiments with hardware a burst of spikes were sent as input to induce a spike at $B_1$. Bursts of spikes were used in hardware experiments to compensate for mismatch in the neuronal dynamics of populations $B_1$ and $B_2$ and is not consequential for software simulations.

We use a network size of the simulations to 32 columns as was the case with hardware unless specified otherwise. This helps us in cross validating the results from the software simulations to those obtained from hardware experiments and maintaining the same functional aspects. In the previous section it was shown that a random set of heterogeneous delays through $C$ was sufficient to learn different spike patterns on hardware. Throughout this chapter all the software simulations were carried out using random delays drawn from a normal distribution with mean delay of $2 ms$ with a minimum delay of $1 ms$. For consistency, a single instantiation of delays was used for all the experimental results that follow.
Fig. 5.7: Effective transmission delays for the $B_1 \rightarrow C \rightarrow B_2$ projections used for simulations. The delays were drawn from a normal distribution with a mean of 2 ms and a minimum of 1 ms.

(except when the network size is varied). Fig. 5.7 shows a color plot of the delays used here. The simulation results remained similar with different instantiations of these delays.

While the hardware implementation of STC model is limited in terms of the learning rules that can be used with the model, software simulations allow us to test the model with some other variations of the learning rules. The following three learning rules were used for the simulation results presented in this chapter.

**One Shot learning**

The hardware experiments were carried out by exposing the network to only the target stimulus, freezing the learning and testing the network’s response. The network learns within very few epochs ($< 10$) of stimulus presentation. Since there were no other spike patterns presented during learning to distract the learning process, in principle the spike pattern could have been learnt with a single epoch of target stimulus presentation. We will refer to this form of learning as one shot learning.

The reason for such an experimental protocol was the synaptic plasticity rule implemented on hardware (see Section 4.1.1.3). The synaptic efficacy $X$ is updated based on the membrane potential $V(t)$ at the time of an incoming spike. An increment or decrement of synaptic efficacy $X$ is made depending on whether $V(t)$ is above or below a certain threshold $\theta$ respectively. The synaptic weight then drifts to a high or a low state with a slew rate $\alpha$. This slew rate determines how long it takes for the synaptic efficacy to settle down to one of the two stable states. For the slew rates that can be realized on our neuro-morphic hardware, it takes a maximum of a second to settle. So once the synaptic efficacy $X$ is updated, a second update has to be done within a second from the first update if we want to see any dependence on the previous update. Otherwise, each update is made
Fig. 5.8: One shot learning with plastic synapses on hardware. Synaptic efficacy $X(t)$ is plotted in red. The synaptic high ($X_{hi}$) and low ($X_{lo}$) states, and the synaptic learning threshold $\theta_X$ are shown in black. The sub-figures on the top row show no synaptic transitions despite multiple updates $a$ with an ISI $\Delta t$. The slew rate $\alpha$ being too fast for the $\Delta t$, the synaptic efficacy $X$ settles to the stable state before the next update. The slew rate on hardware could not be altered to sufficiently small values to accommodate large ISIs of the order of hundreds of milliseconds. Two possible means to overcome this and induce a synaptic transition without altering the slew rate $\alpha$ are shown in the two bottom plots. In the bottom left, the ISI $\Delta t$ is lowered, thereby inducing a transition after 3 positive updates. This requires a high firing rate at the input where each spike induces a consistent synaptic update which is equivalent to making one large update. On the bottom right, the same transition was induced by increasing the update strength $a'$. Here the synaptic transition is one shot where the last seen spike determines the final state of the synapse.

independent of the other updates. Fig. 5.8 shows the two cases when a synaptic update is dependent or independent of the previous update.

For consistency with the one shot learning implemented in hardware, the one shot learning in simulations is based on a deterministic synaptic transition depending on the membrane potential $V$ and at the time of incoming spike at a synapse. In practice, since the simulator does not compute a membrane potential, it potentiates a synapse if the incoming spike occurs within a time window $2 \times \Delta t_p = 2 \text{ms}$ of an input spike. The length of this time window was chosen to be comparable to that on hardware. It is not exactly identical to the time window shown in Fig. 4.8 which is measured from a single neuron, but is an average estimate over the population. It should be noted that there is no depression of synapses here and all the plastic synapses are initially at $X_{lo}$ and only the appropriate synapses are potentiated. Fig. 5.23a shows the effective learning kernel for the synaptic update described here. This rule will be used to reproduce and cross validate the results demonstrated with the hardware implementation.
5.2.1 Unsupervised learning and detection of random spike patterns

We inspect the learning capabilities of the STC network here with respect to the complexity of the spike patterns ES. We start by using linear FM sweeps to validate the simulation platform and to ensure that the simulation platform presents a faithful representation of the hardware system used previously. We then increase the complexity of the stimuli by training and testing the network with random uniformly distributed spike patterns. These stimuli are essentially spatially shuffled versions of linear FM sweeps. Following this we increase the complexity of the stimuli further by introducing Poisson spike patterns as ES. These experiments will demonstrate that the network is indeed capable of learning to differentiate between spike patterns across all these complexity levels.

**Linear sweeping spike patterns** In order to cross validate the simulations we first use linear FM sweeps of spikes described in section 5.1.1 and try to reproduce the results. Fig. 5.9 shows the result of a simulation after training the network to an ES, a linear sweep of velocity $v = 1 \text{ms}^{-1}$ (Fig. 5.9a). One shot learning was used for this simulation. Consistent with the results shown in Fig. 5.3, the potentiated synapses for a positive velocity are above the diagonal. The tuning curve in Fig. 5.9c measured from the network after learning is also qualitatively comparable to that measured on hardware (Fig. 5.5c). The response activity level is determined by the minimum number of coincident spikes required at the input of $B_2$ neurons to elicit a spike and was tuned to match the activity measured on hardware (it was set to a minimum of 2 spikes from $C$).

**Random uniformly distributed spike patterns** As was argued in the previous chapter the above results have larger implications than simply showing the ability of the network to learn and detect linear sweeps. The delay profile and connectivity used in the network as shown in Fig. 5.7 is completely independent of the topology of the 32 neurons of $B_1$ and $B_2$. Any uniformly distributed spike pattern where each neuron spikes once can be sorted by their spike timing resulting in a linear sweep similar to that in Fig. 5.9a. Consequently, we hypothesized that the STC network can learn and detect any uniformly distributed spike patterns. While this hypothesis was not tested on hardware, we proved it here with our software platform.

Similar to the experiment carried out with linear sweeping stimulus we generated a random uniformly distributed spike pattern where the ISIs across the input population is the same as that used in linear sweeps with velocity $1 \text{ms}^{-1}$. The effective mean firing rate of the input is $31.5 \text{Hz}$ and lasts for $31 \text{ms}$. Fig. 5.10a shows one such realization of a spike pattern that was used as ES to train the network. The resultant connectivity after one shot learning is shown in Fig. 5.10b. The response of this network is then tested by

---

1Similar results were also obtained for negative velocities where the potentiated synapses were below the diagonal (data not shown).
Fig. 5.9: Simulation results for linear input after one shot learning. (a) Linear sweep ES with velocity $1 \text{ ms}^{-1}$ used for training the network. (b) Effective $B_1 \rightarrow C \rightarrow B_2$ connectivity after learning. Red pixels show potentiated synapses. The pattern of potentiated synapses appears to be noisy because of the fact that the delays in the network are not uniform but are randomly drawn from a distribution. (c) Response of network to various linear sweep PS of different velocities after learning is shown in blue. The network shows maximal response to linear sweep with a velocity equal to that of ES.

presenting various randomly generated uniform spike patterns PS with the same mean rate or ISI as that of ES. This response is compared to the network’s response to the target ES in Fig. 5.10c. The network responds maximally to ES and shows minimal response to other randomly generated uniform spike patterns despite having the same mean population firing rate. This result shows that the network can in fact learn and detect uniformly distributed spike patterns. Since this experimental protocol used here is qualitatively identical to that used for the hardware experiments, these results show that hardware implementation of the network is also capable of learning and detecting randomly generated uniform spike patterns.

Poisson spike patterns. We have seen that the STC model with random delays is capable of learning and detecting random uniformly distributed spike patterns. Let us now test whether this network can learn to detect more complex stimuli like Poisson spike patterns drawn from an exponential distribution. A random $31.5 \text{ Hz}$ Poisson spike pattern of length $50\text{ ms}$ was used to train the network. The effective connectivity after learning is shown in Fig. 5.11. The measured response of the network to 200 different randomly generated Poisson spike patterns of the same length is shown in Fig. 5.11c. Here again we see that the network responds maximally to the target spike pattern ES and is capable of detecting this spike pattern within a large sample drawn from the same distribution.

The above results are obtained from simulations that follow identical experimental conditions.
Fig. 5.10: Simulation results for uniform input after one shot learning. (a) Uniformly distributed random 31.5 Hz spike pattern ES with mean ISI 1 ms used for training the network. (b) Effective $B_1 \rightarrow C \rightarrow B_2$ connectivity after learning. Red pixels show potentiated synapses. (c) Response of network to 200 randomly generated uniformly distributed spike patterns PS is plotted in blue. The network shows maximal response to the ES marked by the brown horizontal bar.

Fig. 5.11: Simulation results for Poisson input after one shot learning. (a) Randomly generated 31.5 Hz Poisson spike pattern ES of 50 ms used for training the network. (b) Effective $B_1 \rightarrow C \rightarrow B_2$ connectivity after learning. Red pixels show potentiated synapses. (c) Response of network to 200 randomly generated Poisson spike patterns PS. The network shows maximal response to the ES marked by the brown horizontal bar.
Fig. 5.12: Response of network to time jittered versions of ES. The mean firing rate of the input spike pattern was 31.25 Hz. The response of the network decreases with increasing perturbations to the original ES. The standard deviations are measured from 50 realizations of jittered spike patterns for each value of jitter.

protocol and mechanisms as those done on hardware. This includes coincidence detection, delays and network architecture. Therefore based on the hardware experimental results with random delays and the simulation results demonstrated here, it can be argued that the hardware implementation demonstrated in the previous chapter will also be capable of learning and detecting randomly generated uniform and Poisson spike patterns.

5.3 Robustness vs Precision

The experimental results shown above demonstrate that the STC network is indeed capable of identifying ES among several other spike patterns randomly drawn from the same distribution. In order to quantify how the response properties vary with the stimulus properties (as was done with linear and forked sweeps) we carried out the following experiment. We measured the response of the network to time jittered versions of the target stimulus ES with varying levels of perturbations. The test spike patterns PS were generated by altering individual spike timings of ES by a duration drawn from a normal distribution. This allows us to quantify the precision of network’s response by systematically measuring its tuning properties against a range a values for a stimulus parameter. Fig. 5.12 shows the response of the network to an ES, a Poisson spike pattern of mean firing rate 31.5 Hz. The network shows maximum response to the unperturbed spike pattern and shows gradual decrease in response to increasing perturbations of the spike pattern ES.

The response of the network drops significantly as soon as the amount of jitter gets close to the time window of coincidence detection $2 \Delta t_p = 2 ms$. This is expected because
the coincidence time window defines the sensitivity of the network to coincidences. If the coincidence window is narrow, the network will be very sensitive to spike timing jitters and if it is very wide (with reference to the mean ISI) the coincidence detection capabilities deteriorate. This dependence of the network’s tuning properties on the coincidence window is plotted in Fig. 5.13 for three different $\Delta t_p$ values. For $\Delta t_p = 0.5 \text{ ms}$ the network’s response is lowered even for time jitters of less than $1 \text{ ms}$. This sensitivity to jitter decreases with $\Delta t_p = 1 \text{ ms}$. For $\Delta t_p = 2 \text{ ms}$, the network’s sensitivity to spike jitter shows a significant decrease in comparison to the other two and the network responds with a higher (but still lower than ES) activity even for large jitter values. Therefore $\Delta t_p$ plays a crucial role in determining the sensitivity of the network to temporal variations of the spike pattern. A very precise response might not be desirable in order to account for temporal variations in the spike timings and to keep the network robust to noise. On the other hand a very wide coincidence window means that the network is not very selective to the temporal properties of the desired spike pattern. Therefore an appropriate coincidence window has to be chose depending on the temporal properties of the stimulus such as mean firing rate and noise in order to obtain an appropriate level of robustness and precision.
5.4 Scaling properties of the network

We saw that the network’s tuning properties were dependent on the coincidence detection window $\Delta t_p$. The tuning curves in Fig. 5.13 show the mean and standard deviation of the response to jittered versions of the target spike pattern $ES$. The response of the network therefore gives an estimate of the proximity of the spike pattern to the target spike pattern given the smooth profile of the mean response. But the reliability of this estimate is given by the standard deviation shown in the curves. If the standard deviation is large in comparison to the mean value, we cannot assess the stimulus properties based on the response of the network. The standard deviation of the normalized tuning curves such as in Fig. 5.13b give us an estimate of how reliable the response of the network is for estimating the temporal properties of a spike pattern.

So far we kept a one-to-one correspondence between the input neurons $A$ and $B_2$ neurons since both populations had equal number of neurons. To test the scaling properties of the network we require a strategy to vary the number of input neurons $A$ and readout neurons $B_2$ independently. A simple methodology we adopt here is as follows; if the number of readout neurons $B_2$ is less than that in the input $A$, only a subset of input neurons project to the readout neurons; if the number of readout neurons $B_2$ is more than that in the input population $A$, neurons in $A$ make multiple projections to $B_2$ neurons. In both cases we ensure that each readout neuron $B_2$ receives an input from one (and only one) of the neurons in $A$. The number of neurons in population $B_1$ always remains the same as that of population $A$, since their spiking activity is identical in the simulations. The rest of the network functions exactly as it did before.

5.4.1 Scaling the input population size

How does the network’s response scale with the size of input population? Does the network’s response maintain its reliability with increasing number of neurons in the input population? We address these questions here by performing the following measurements.

We first tested whether the network retains its ability to differentiate and detect the $ES$ among other $PS$. Fig. 5.14 shows the results of a network with 128 input neurons $A$ and 32 output neurons. Fig. 5.14c shows the response of the network to 200 randomly generated Poisson spike trains $PS$ in blue and its response to $ES$ (Fig. 5.14a) is shown by the brown horizontal line. The network’s response is clearly higher for the target spike pattern $ES$ in comparison to others. It is therefore able to detect the target spike pattern $ES$.

We also measured the tuning properties of networks with different input sizes $N_{inp}$ with increasing magnitude of jitter to the $ES$. Fig. 5.15a shows a plot of mean response of the network with increasing jitter, for three different network sizes. The standard deviations are measured across 50 instantiations of jittered $ES$. The differences in the response levels are due to different instantiations of $ES$ and delays. The normalized responses in Fig. 5.15b
Fig. 5.14: Simulation results for Poisson input from 128 neurons after one shot learning and 32 readout neurons. (a) Randomly generated 31.5 Hz Poisson spike pattern ES of 50 ms used for training the network. (b) Effective $B_1 \rightarrow C \rightarrow B_2$ connectivity after learning. Red pixels show potentiated synapses. (c) Response of network to 200 randomly generated Poisson spike patterns PS. The network shows maximal response to the ES marked by the brown horizontal bar.

are also comparable although not identical due to the same reasons.

In order to assess the scaling properties of reliability of the network’s response we study how the standard deviation varies with increasing size of the network. We train networks of different input population sizes with ES spike trains of mean firing rate of 31.5 Hz keeping the output population size constant at 32 neurons. Their response to 200 instances of ES jittered by a mean value 1 ms equal to the coincidence window $\Delta t_p$ is measured. The mean responses of these networks is plotted against input dimension in Fig. 5.16a. The differences in the mean values arise as a result of different instantiation of ES and delays in the network. The general trend in the response properties remains consistent. The corresponding standard deviations normalized to the mean responses are shown in Fig. 5.16b. The plot remains horizontal, showing that the network’s response retains its reliability with increasing input population size and consequently with increasing input complexity.

5.4.2 Scaling the output population size

The results shown in the last subsection show that the network’s response reliability scales with increasing size of input population. But it does not improve the reliability. How can the network’s response reliability be increased? Remember that the readout neurons $B_2$ individually observe a subset of the temporal correlations of the input and the readout is an accumulated evidence of the temporal correlations. Therefore it logically follows that more the number of readout neurons, the better the evidence accumulation and consequent

---

3The magnitude of jitter was chosen such that it affects the coincidence detection properties of readout neurons but the spike pattern does not get altered to an extent where its similarity to $ES$ is completely lost.
Fig. 5.15: Tuning properties of the STC network to ES of 31.5 Hz and spike timing jitter with a mean of 1 ms. Tuning properties are plotted for three different sizes $N_{\text{inp}}$ of input populations $A$. The standard deviation was calculated across 50 measurements. (a) Population mean firing rate of the network’s response to different levels of jitter. (b) Tuning properties with maximum response normalized to 1 for comparison. No drastic differences are observed in the mean response properties with increasing number of input neurons $N_{\text{inp}}$.

Fig. 5.16: Scaling properties of the STC network response with varying input population size. (a) Mean frequency response of network to ES spikes jittered by a mean of 1.0 ms with different number of input neurons neurons $A$. The mean response remains similar with increasing number of input neurons. (b) The standard deviation (normalized to the mean response) of the network’s response to 200 ES spike trains jittered by a mean value of 1.0 ms is plotted across number of input neurons $B_2$. The standard deviation $\sigma$ also does not vary with increasing dimensionality of input population. This shows that the network retains its capabilities even after scaling it to higher input dimensions.
higher reliability. Here we put this hypothesis to test. We vary the number of readout neurons and measure the tuning properties to the jittered ES as we did before.

Fig. 5.17 shows the tuning curves of the network to three different readout population sizes. The mean response properties of the network remain largely unaltered by a change in size of the readout population. Although there is a slight difference in the mean response frequency, the normalized response plot in Fig. 5.17b shows that the sensitivity i.e. the slope of the curves remain the same. This shows that the mean response properties are governed by the coincidence window only and not by the size of the readout population. On close observation it can be noticed that what depends on the size of the network is the standard deviations of the tuning curves. This can be seen from Fig. 5.18a where the standard deviation of the response to ES with a jitter of 1 ms (equal to the coincidence window $\Delta t_p$) is plotted against different population sizes of $B_2$. The same data is shown in Fig. 5.18b where the standard deviation normalized to the mean response is plotted against network size on logarithmic axis. These plots clearly show a lowering of standard deviation with increasing population size. Similar measurements were obtained for several different instantiations of network delays and ES spike patterns. These results demonstrate that the network’s capability to identify ES and its ability to differentiate between jittered versions of ES scales with the size of output population $B_2$. The scaling might be following a power law but we need to do an extensive set of experiments in order to confirm this and
Fig. 5.18: Scaling properties of the STC network response. (a) Mean frequency response of network to ES spikes jittered by a mean of 1.0 ms with different number of readout neurons $B_2$. The mean response remains similar with increasing number of readout neurons, but the standard deviation $\sigma$ decreases. (b) The standard deviation (normalized to the mean response) of the network’s response to 200 ES spike trains jittered by a mean value of 1.0 ms is plotted across number of readout neurons $B_2$. The standard deviation $\sigma$ decreases with increasing number of readout neurons. This shows that the reliability of the STC network’s response increases with an increase in the number of readout neurons.

is left for future investigation.

These results show that the network’s capabilities scale with the complexity of the input spike patterns i.e. with increasing number of input neurons. They also demonstrate that the network’s response reliability increases with an increase in the size of readout population $B_2$.

### 5.5 Recognition amidst distractors

In all the experiments presented so far, both on hardware and in simulations, the only spike pattern the network was being exposed to was the target stimulus that was to be learnt. The spike patterns ES were being presented repeatedly in quick succession in case of hardware experiments and the simulations involved a single presentation of ES. In both cases there were no other distractor spike patterns during the learning process. This is not a issue for an isolated system where the input and learning are supervised and the input is modulated to ensure that the network is only exposed to the target stimulus. But for a real-time autonomous system, the network should be capable of learning statistically significant stimulus features and differentiate it from other randomly appearing stimuli i.e. distractors with features that are statistically less significant. It should be able to recognize
the most often seen spike patterns as objects of interest. To test the potential of the current system to operate autonomously we expose it to stimuli embedded in distractors.

We use a spike pattern of $50\,\text{ms}$ randomly generated from a Poisson distribution with mean firing rate of $62.4\,\text{Hz}$ as $\text{ES}$. This spike pattern is inserted at various times into randomly generated distractor spike trains as shown in Fig. 5.19. The distractor spike trains are drawn from the same distribution as $\text{ES}$ and have the same mean firing rate. The mean distractor length between two subsequent presentations of $\text{ES}$ was $50\,\text{ms}$. The final spike trains included $50$ interspersed epochs of $\text{ES}$. We now use this cumulative spike pattern to test whether the network is able to learn and recognize $\text{ES}$ in an unsupervised manner. This experimental protocol is similar to that used by others for unsupervised learning [127, 154].

The response of the network after training with $\text{ES}$ inserted in distractor spike trains is shown in Fig. 5.20b. The network responds with high spiking activity even for some non-$\text{ES}$ spike patterns. This can be explained by the resultant connectivity after learning shown in Fig. 5.20a. All the synapses are potentiated, thereby making the network insensitive to any differences in the temporal properties of the spike patterns. This outcome is attributed to the one shot learning protocol. Since every input spike induces a synaptic transition, and there is no distinction between a distractor and the target spike pattern $\text{ES}$, the network learns the correlations in every spike pattern presented to it. In addition, the lack of LTD of synaptic weights results in continually potentiating synapses in the connectivity matrix.

5.5.1 Slow learning

The hardware experiments resulted in one shot learning as discussed earlier in section 5.2. The effect of such learning can be seen in the experimental results shown in Fig. 5.20. This is due to the fact that the learning rule (see Section 4.1.1.3) makes deterministic transitions with binary synaptic weights. Of course this is only true when the synaptic weights settle

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure5.19}
\caption{Target spike pattern $\text{ES}$ (red) embedded in distractor spike trains (blue).}
\end{figure}

\footnote{Until now, given that all synaptic weights were initialized with their low state ($X_{\text{lo}}$), and the learning was one shot, there was no need for LTD.}
Fig. 5.20: Simulation results for Poisson input merged with distractors after one shot learning. (a) Effective $B_1 \rightarrow C \rightarrow B_2$ connectivity after learning. Red pixels show potentiated synapses. (b) Response of network to 200 randomly generated Poisson spike patterns $PS$ is shown in blue. The network’s response to $PS$ is marked by the brown horizontal line.

Fig. 5.21: Effect of decreasing the slew rate $\alpha$ to $\alpha'$ on the synaptic learning dynamics. When the slew rate $\alpha$ is high, the synaptic state $X$ always settles back to its initial state. On lowering the slew rate to $\alpha'$, the synaptic efficacy makes a successful transition from $X_{lo}$ to $X_{hi}$ for the same set of input spikes as shown on the right.

to one of the two($X_{lo}/X_{hi}$) states. If the slew rate $\alpha$ of learning is very slow ($\alpha \approx 0$) and the synaptic weight update $a$ is small ($a \ll X_{hi} - X_{lo}$), the weight of the synapses is effectively continuous (bounded by $X_{lo}$ and $X_{hi}$). The effect of lowering the slew rate $\alpha$ is shown in Fig. 5.21.

The low slew rate $\alpha$ enables learning dynamics of a bistable synapse to be driven in a similar fashion to that of plastic synapses with continuous analog values. This is obvious given that the dynamics of the synaptic efficacy are driven largely by the updates rather than the slew rate. Taking this idea to its extreme limit, in the simulations that are to follow, slow learning refers to slew rate $\alpha = 0$. Fig. 5.23b shows the effective learning kernel for the synaptic update for slow learning. In the simulations, the magnitude of synaptic updates are $a_+ = 0.1 \times (X_{hi} - X_{lo})$ and $a_- = -0.05 \times (X_{hi} - X_{lo})$. The time
window for potentiation $2 \times t_p = 2 \text{ms}$ and the time window for depression is $2 \times t_d = 10 \text{ms}$. These parameters were chosen in accordance with the theoretical analysis in [76] where the authors analyzed the relationship between parameters for potentiation and depression for learning with STDP. The general rule of thumb is $a_+ \times t_p < a_- \times t_d$ which ensures that synapses with negative or random correlations depress whereas those with consistent correlations potentiate [127].

We now carry out the same experiment as above but use slow learning in place of one shot learning. After learning, a binary version of synaptic weights is used to test the response of the network; 0 ($X_{lo}$) for weights less than 0 and 1 ($X_{hi}$) for weights above 0. This is done to keep the effective resolution of the network consistent with that of the hardware implementation, and mimic the property of synapses being bounded. Fig. 5.22 shows the results of this simulation. The resultant connectivity (Fig. 5.22a) is much sparser in comparison to that after one shot learning (Fig. 5.20a). The network now responds selectively to the ES and remains almost silent to other randomly generated spike patterns or distractors as can be seen from Fig. 5.22b.

5.5.2 Stochastic learning

Neuromorphic implementations do not often include plastic synapses with continuous weights and long slew rates like the one we proposed in slow learning. This is due to limitations on components used to implement circuits on hardware such minimum currents values and capacitor sizes. Therefore it is not trivial to realize slow learning as we
Fig. 5.23: Effective STDP kernel for the three learning rules used in the simulations. The horizontal axis shows the time difference between a spike from $C$ at the plastic synapse $t_{syn}$ and an input spike of the presented spike pattern from $A$, $t_{inp}$. The vertical axis in (a) and (b) represents the magnitude of synaptic update $a$. (a) For one shot learning the magnitude of synaptic update $a = X_{hi} - X_{lo}$. It only involves potentiation of synapses and no depression. (b) Slow learning involves both potentiation and depression of synaptic weights. The vertical axis shows the magnitude of synaptic efficacy update $a$. The magnitude of updates here are typically much smaller than $X_{hi} - X_{lo}$. (c) Stochastic learning involves only binary synaptic weights and the synaptic state transitions are done probabilistically. The probability of these transitions depend on $t_{syn} - t_{inp}$. A positive value on the vertical axis shows the probability for potentiation and a negative value shows probability for depression.

The probability of synaptic update is determined by the timing of incoming spikes from $C$ ($t_{syn}$) with respect to input spikes from $A$ ($t_{inp}$). Fig. 5.23 shows the probability for a synaptic update. In simulations, the magnitude of synaptic update is $a = \pm (X_{hi} - X_{lo})$, with a probability of potentiation $P_{ltp} = 0.005$ and a probability of depression $P_{ld} = 0.01$. The time window for potentiation is $2 \times t_p = 2 ms$ and the time window for depression is $2 \times t_d = 10 ms$. The time windows are chosen to be identical to those used in slow learning. The probabilities were chosen to be very low to ensure that the synaptic state transitions are only triggered by statistically significant correlations in the input spikes.

The experiment where ES is embedded in distractor spike patterns is repeated with the stochastic learning rule. Unlike slow learning, the synaptic weights here are always binary. Fig. 5.24 shows the results of this simulation. Because of very low transition probabilities used for simulations, the number of presentations of ES was increased from 50 to 500.
5.5.3 Response to time reversed spike patterns

The results demonstrated in this section are comparable to those of [127] where the authors follow an identical training protocol to train a neuron with STDP to selectively fire to a specific spike pattern. The synaptic learning rule used in their work utilized graded

\[ B_1 \rightarrow C \rightarrow B_2 \]

connectivity after learning. Red pixels show potentiated synapses. The response of network to 200 randomly generated Poisson spike patterns PS is shown in blue. The network’s response to ES is marked by the brown horizontal bar.

The resultant connectivity (Fig. 5.24a) is much sparser in comparison to that after one shot learning (Fig. 5.24a). The network with stochastic learning also shows a higher response to the ES in comparison to randomly generated spike patterns or distractors as can be seen from Fig. 5.24b. The reliability of the response in this case is not as good as with slow learning. The parameters for stochastic learning were not optimized and it might be possible to find a better configuration through an extensive parameter search. An alternative might be to scale up the size of the readout population, which as we have seen earlier enhances the reliability of the network response. The objective of this exercise was not to find the optimal set of parameters but was merely to explore alternative learning strategies. In this context the results demonstrated here show that a stochastic learning rule induces some degree of selectivity to ES embedded in distractor stimuli.

It should be noted that the results from slow and stochastic learning not trivial since there was no explicit mechanism to identify the target stimulus that was to be learnt. The statistical significance of the spike pattern ES drove the network to a connectivity that makes the network sensitive to this specific spike pattern.

Fig. 5.24: Simulation results for Poisson spike patterns merged with distractors after stochastic learning. (a) Effective $B_1 \rightarrow C \rightarrow B_2$ connectivity after learning. Red pixels show potentiated synapses. (b) Response of network to 200 randomly generated Poisson spike patterns PS is shown in blue. The network’s response to ES is marked by the brown horizontal bar.
Fig. 5.25: Response of STC network to the ES (black) it was trained with and its time reversed version (grey) after one shot learning. The middle subplot shows the response of the network to ES in red and to the time reversed version in yellow. The bottom subplot shows the mean firing rate of the population’s response to each of the stimuli. The figure shows clear response selectivity of the network towards ES in comparison to its time reversed version.

synapses much like those used in slow learning. In section 3.2 we argued that single neuron based learning systems such as [127] are unable to distinguish between the target spike pattern and its time reversed version because they rely on coincidences. The STC network on the other hand is capable of differentiating between time reversed spike patterns by virtue of its transmission delays. In section 5.1 we showed that hardware implementation of the STC network is able to distinguish between time reversed frequency sweeps. The tuning curves in Fig. 5.5 show that after training the network to a frequency sweep ES with a positive velocity \( v \), it can differentiate between ES and its time reversed version i.e. frequency sweep with velocity \( -v \).

Similar results can also be demonstrated with more complex stimuli such as Poisson spike trains. Fig. 5.25 shows the response of the network after one shot learning of a randomly generated poisson spike pattern ES and the response of the network to its time reversed version. The network shows a clear selectivity in its response to ES in comparison to its time reversed version.

The network’s response properties were also verified after training with the modified learning rules described in this section. It’s response to the time reversed version of ES
(shown in Fig. 5.19) was measured after oneshot, slow and stochastic learning. The network responded with a single spike after one shot learning (without distractors). It responded with 5 spikes with stochastic learning and 4 spikes with slow learning. These response levels are lower than that of the ES and comparable to network’s response to other randomly generated spike patterns PS. These measurements demonstrate that the network is indeed able to differentiate between time reversed spike patterns, even with slow or stochastic learning. This shows that the temporal selectivity for network’s response is a network property and is not restricted to the fine details of the learning rule.

5.6 Multisensory Integration

Multisensory integration is a ubiquitous neurological phenomenon by which organisms integrate complex environmental stimuli. Convergence of inputs from different sensory modalities onto individual neurons is a widely observed phenomenon occurring throughout the brain (superior colliculus (SC), cerebral cortex, thalamus, hypothalamus, basal ganglia, hippocampus, inferior colliculus, reticular formation, cerebellum, primary sensory nuclei) at many phylogenetic levels (primates, cats, rodents, birds, reptiles, amphibians, fish, invertebrates) and appears to represent a basic neural mechanism by which an organism integrates complex environmental stimuli [134]. Regardless of the principles by which different multisensory cells in the brain deal with their various inputs, it seems likely that the results of their combined activity will have profound influences on perception as well as on behavior.

The convergence of modalities are most evident in the SC. This brain area is intimately involved in attending to, localizing and orienting to sensory stimuli. The response of SC cells to a given sensory stimulus is influenced by the presence or absence of other sensory cues [132]. Meredith and Stein in a series of experiments [132, 134, 131] investigated the effect of multisensory inputs on cells in SC and reported the following observations. By pooling sensory inputs, most of these cells amplify the effect of subtle environmental cues in certain conditions, whereas in others, responses to normally effective stimuli can be blocked. The enhancement of response most often reflects a multiplicative rather than a summative change in activity. Potential for response amplification is greatest when responses evoked by individual stimuli is weakest. Temporal disparity among combinations of different sensory stimuli was also shown to be a critical factor influencing the integration of multisensory stimuli by SC neurons [131]. Maximal enhancement depended most on overlapping the periods of peak activity evoked by unimodal stimuli, rather than on matching either the stimulus onset or response latencies [131].

Since discharge patterns often depend on the physical parameters of the stimulus, changing stimulus parameters change the resultant interaction [131]. Although the effects of intra modal interactions as well as some multimodal interactions have been classified using such terms as inhibition, occlusion, summation and facilitation, this nomenclature was
avoided by Meredith and Stein [134]. They argue that manipulation of physical properties of the same stimuli produces interactions within the same cell that can fall into all of these categories and inhibition, occlusion, summation and facilitation appear to reflect levels of multisensory interactions along the same continuum.

It is unlikely that random interactions among modalities could provide the nervous system with meaningful information about the external world. Rather, for any given cell or population of cells, these interactions must follow specific rules that are predicated on the physical properties of the external stimuli and their relationships to one another [134]. The STC model can provide a possible mechanism to capture these relationships between various external stimuli. As we have seen previously, the STC model relies on the spatio-temporal correlations between the spike patterns to respond to the incoming stimuli. Let us examine whether such an approach can enable neurons to integrate information from various stimuli and respond accordingly. More specifically, here we were focus on the observation that multimodal stimulus presentations elicit a multiplicative enhancement in response of cells in SC. This observation was done by comparing the response of SC cells on presentation of their preferred unimodal stimulus and multimodal stimulus. We will do a similar comparison by observing the response properties of individual neurons in readout population $B_2$.

### 5.6.1 Simulation results

In order to examine whether the STC model can perform stimulus cue integration and learn relationships between different stimuli, we train the network to two different stimuli that are temporally coincident. We then test the response of its readout neurons $B_2$ to each of these stimuli individually and when they are presented together. We expect to see an enhanced response in some of these readout neurons on presentation of both stimuli together similar to that seen in SC neurons.

**Stimuli** An STC network with 32 input neurons was initialized with random propagation delays introduced through population $C$. This network was then trained with two spike patterns presented together. The two Poisson spike patterns, spread across 50 ms, were randomly generated from an exponential distribution spread across 16 neurons each. Fig. [5.26] shows one such pair of spike patterns. These two spike patterns represent stimuli from two different modalities. They are used as target stimuli to train the network with one shot learning protocol as described previously.

**One shot learning** The response of this network after training was tested by presenting random spike patterns. The resulting activity for 200 such random spike patterns is shown in Fig. [5.27]. The horizontal lines in the figure show its response to unimodal and multimodal presentation of target spike patterns. Upon presentation of the two target spike patterns individually shown in green and red in Fig. [5.26] the network responded
with 36 and 32 spikes respectively. This is the unimodal response of the network. Upon presentation of the two spike patterns simultaneously, we see a considerable increase in the response of the network, 103 spikes. This is more than the sum of its response to unimodal presentations, $36 + 32 = 68$.

Nine $B_2$ neurons responded to unimodal stimuli and showed no enhancement on multimodal stimulus presentation. Two $B_2$ neurons respond only to multimodal stimulus presentation. Ten $B_2$ neurons showed enhanced response (25% to 100%) to stimulus 1 on presentation of multimodal stimulus. Eleven $B_2$ neurons showed enhanced response (33% to 200%) to stimulus 2 on presentation of multimodal stimulus.

**Slow learning**  Similar response enhancement was also observed after *slow learning* (See Fig. 5.29a). Its response to unimodal stimulus presentation was 13 and 16 spikes for each spike pattern. Its multimodal response to the two spike patterns presented simultaneously was 69 spikes. 9 $B_2$ neurons responded to unimodal stimuli and showed no enhancement on multimodal stimulus presentation. 11 $B_2$ neurons responded only to multimodal stimulus presentation. 6 $B_2$ neurons showed enhanced response (50% to 200%) to stimulus 1 on presentation of multimodal stimulus. 5 $B_2$ neurons showed enhanced response (50% to 300%) to stimulus 2 on presentation of multimodal stimulus.

**Stochastic learning**  *Stochastic learning* also resulted in similar response properties although with a net lowering of activity (see Fig. 5.29b). Its response to unimodal stimulus
**Fig. 5.27:** Response of STC network to multimodal stimuli after one shot learning. The horizontal lines show its response to unimodal and multimodal presentation of target spike patterns shown in Fig. 5.26. As a control measure, its response to 200 random spike patterns \(PS\) of the same mean firing rate as that of the multimodal stimulus is shown in blue.

**Fig. 5.28:** The effective connectivity from \(B_1\) to \(B_2\) neurons (delayed through \(C\)) after learning spike patterns shown in Fig. 5.26. The network was trained using one shot learning. Potentiated projections are shown in red.
Fig. 5.29: Response of STC network to multimodal stimuli after (a) slow learning, (b) stochastic learning. The horizontal lines show its response to unimodal and multimodal presentation of target spike patterns shown in Fig. 5.26. In both cases the network shows a substantial enhancement of response during multimodal stimulus presentation. Their response to 200 random spike patterns $PS$ of the same mean firing rate as that of the multimodal stimulus is shown in blue.

presentation was 7 and 4 spikes for each spike pattern. Its multimodal response to the two spike patterns presented simultaneously was 31 spikes. Seven $B_2$ neurons showed response to unimodal stimulus presentation and no enhancement on multimodal stimulus presentation. Eleven neurons showed response only to multimodal stimulus presentation. One $B_2$ neuron showed enhanced response (50%) to stimulus 1 on presentation of multimodal stimulus. One $B_2$ neuron showed enhanced response (100%) to stimulus 2 on presentation of multimodal stimulus.

The lowering of net response, both to unimodal and multimodal stimulus presentations after slow and stochastic learning (in comparison to one shot learning) can be explained by the sparseness of the resultant connectivity at the point of testing. Fig. 5.30 shows the networks effective connectivity after learning in both cases. In comparison to the resultant connectivity after one shot learning as shown in Fig. 5.28, the number of effectively potentiated synapses in both cases is far fewer and can account for the lowering of response of the network.

5.7 Discussion and Conclusion

We explored the capabilities of the STC model through hardware and software implementations. We demonstrated that the network is capable of learning not only simple FM sweeps but also complex spike patterns such as uniformly distributed random spike patterns and Poisson spike patterns. The hardware experiments stress the robustness of the
model and its success despite the imprecision of the neuromorphic devices used in this work. Software simulation results proved that this network scales linearly with increasing complexity of the stimulus in terms of the input population size. The reliability of the response was shown to improve by increasing the size of readout population. The network fails to learn and recognize target spike patterns when they are embedded in distractors. This failure was attributed to the learning rule originally used in the model because of constrains imposed by the available hardware implementation. To verify this hypothesis, other learning rules were explored. Slow learning rules such as those with continuous weight values and stochastic synaptic transitions can enable the STC network to learn and recognize statistically significant spike patterns despite distractions. Last but not least, this simple neural architecture can even explain neuro-physiological data which form the basis of the current understanding of multimodal fusion in biological system.

The software simulations demonstrate that the hardware could have potentially been used for more complex stimuli such as Poisson spike trains. The neuromorphic implementation of STC model described here will be used with real-world stimuli. Further work is needed to validated the network against noise in the stimulus, and limitations in the system’s state variables (e.g. due to device mismatch, limited resolution and signal-to-noise ratio limitations, bounded synaptic weights, power constraints, etc.)

The experiments carried on hardware demonstrate the robustness of the network to imprecise parameters. The network is oblivious to the exact delays provided by the delay neurons as long as they provide offer a distribution of delays. As we have seen from the simulations with random delays, such connectivity allows the network to learn rich spatio-temporal features of the spike patterns and assists in differentiating the ES from other
random spike patterns. An important aspect of the network’s robustness can also be seen from the fact that it relies on simple binary synapses to make its decisions and does not require a high resolution. This is unique to this network in comparison to other state-of-the-art models such as Masquelier et. al. [127], SEM [154] and ReSuMe [162]. A reason for such robustness arises from the fact that coincidence detectors in the STC network only observe a small subset of the entire input (represented by the potentiated synapses) to make their decisions in contrast to other models and therefore do not require detailed synaptic weights.

One of the distinguishing features of the STC model and the results demonstrated here is the lack of supervision in the learning process. This is trivially done with one shot learning as is the case with the hardware implementation. This has a lot more significant role when the stimuli are embedded in distractors. Software simulations demonstrated that the current hardware with one shot learning is not ideal for this type of unsupervised learning when it comes to accumulating statistically relevant information over long periods of stimulus presentation. We also saw two alternate learning rules that allows the network to learn such statistically significant spike patterns which are embedded in distractors.

The lack of supervision is particularly attractive to neuromorphic engineering because of the possibility for autonomous real-time computation. The requirement for a teacher signal often demands constant attention by an agent and also has to operate in real-time. Such real-time supervision in real-world uncontrolled surroundings is very difficult to provide. The use of unsupervised learning simplifies this by alleviating the need for constant supervision. While, it might be that higher cognitive tasks require instructions, at least basic tasks such as feature extraction can be carried out with unsupervised learning.

Multi-sensory fusion is a challenging task for robotic systems involving several sensors and even more so when they span multiple modalities such as vision [105, 117, 15, 108, 44, 115, 215, 112], audition [206, 195, 172, 68, 174, 207, 36, 4] and touch. This is a hard problem because each of these modalities have completely different features. While in vision, the features of stimuli are classically described by spatial features like color, intensity and edges, in audition this is done with Spectro-Temporal Receptive Fields (STRFs). These features are very different, while one operates on spatial features, the other operates on temporal features. Therefore algorithms have to be tailored to accommodate all these differently modalities in order to take an appropriate decision. This is the same also with neuromorphic systems. However in neuromorphic systems we know that all the sensors share the same alphabets - spikes. Therefore an algorithm that can capitalize on this commonality can be a good candidate for multi-sensory fusion. The STC model relies solely on input spike patterns for recognition. It does not rely on any features specific to a given modality. Therefore in a neuromorphic robotic system involving multiple sensory modalities the spike streams of all the sensors can be merged to form one spike stream.

As shown in section 5.6, the STC network without any modifications integrates mul-
timodal stimuli. The results showed qualitative similarities with neuro-physiological data such as enhancement of response of unimodal cells on presentation of multimodal stimuli. This was a direct result of the working principles of the model, where the readout neurons rely on temporal correlations between different neurons. These neurons could be encoding features across different modalities. The network finds features that are a mixture of several modalities that are most relevant to the task at hand. This idea of mixed feature selection can also be seen in decision making models like attractor concretion for formation of context representation $^{[167]}$. The task of information merger across various modalities is trivially done here without relying on explicit feature detection of individual sensory modalities. Such multimodal integration capabilities are crucial for spike based neuromorphic systems that involve multiple sensors.

5.7.1 Limitations

**Hardware bugs** Neuromorphic devices are inspired by the robustness of biological neural systems not only to heterogeneity of parameters but also to failure or cell death. This robustness is usually attributed to the large number of neurons and the massively parallel processing that comes with them. While neuromorphic devices such as the ones used here are massively parallel in that they have a large number of neuron and synapse circuits, they are not entirely immune to device/circuit failures. While the analog circuitry within these neuromorphic chips is parallel and fault tolerant, the digital circuitry associated with the AER communication $^{[123, 49, 21]}$ is not. The AER infrastructure used in this work $^{[62]}$ uses a digital four phase hand-shaking protocol and events are communicated in sequential fashion. This hand-shaking protocol involves a request and an acknowledge signal. The commutation is put on a hold until there is an acknowledge signal. If a device fails to send an acknowledge signal due to some device failure then the complete AER communication system fails. This is a critical issue in the hardware system that was described in Chapter 4. The IF2D-WTA chip has an internal bug that prevents some of the synapses to return an acknowledge signal. So if one of these synapses is addressed the entire AER communication system gets stuck. The digital communication infrastructure used for AER communication is prone to data corruption, causing a rare corrupt event to address these faulty synapses. These issues prevent long runs of experiments for statistical data measurements. This problem was one of the reasons for supplementing the hardware experiments with software simulations.

**Long vs Short time scales** Sensory stimuli, and in particular auditory stimuli, contain both short and long range temporal correlations. The techniques described in this work primarily address correlations over relatively short time scales, i.e. those in a range from the order of synaptic or membrane time constants, up to those represented by the propagation of excitation to adjacent regions. In case of the network proposed in this chapter, the range
of propagation delays implemented in the network defines the range of stimulus velocities (not the duration) that could be learnt. Temporal correlations over a longer time scale could be addressed using many levels of processing between widely separated layers, as is observed in the mammalian auditory system. Alternatively it could be tackled with working memory and neuromorphic implementations of state machine based approaches \[149, 148\]. It could also be done by exploiting other non-linear processes such as short term plasticity \[125\] which are slower and span several hundreds of milliseconds. In appendix A we investigated the role of short term plasticity in temporal processing at longer time scales of the order of a few hundred milliseconds with the neuromorphic hardware system used in this chapter.

**Static stimuli and high firing rates** The STC model, in its current form, assumes the stimulus to be dynamic and does not consider static stimuli where frequency channels are persistently active. The learning rule used \[24\] dictates high probability of LTP in the presence of highly active inputs. As a result, if such persistent stimuli were to be presented to this network a large number of synapses would potentiate to reflect the temporal correlation across different \(\Delta t\) values. However transient responses, in particular onset responses, are found at all levels of the auditory system \[63\]. This could be modelled with strong synaptic depression \[187\] at the input synapses of \(B_1\) and \(B_2\) or as post synaptic adaptation of neurons in \(A\), thus eliminating the problems caused by persistent input stimuli.

**Multiple spike patterns** Throughout this work, the network’s interest was only linked to one single spike pattern ES that was learnt and detected by the network. In a real-world scenario, there are usually multiple objects of interest, i.e. multiple spike patterns as ES. In the SEM model \[154\] and Masquelier et. al.’s work on spike pattern detection \[126\], a winner-take-all mechanism was introduced in the readout population to enable different neurons learn different features. A similar strategy could be adopted in the STC model by introducing competition among readout population \(B_2\). This could enable a subset of readout neurons to tune themselves to one of the many patterns that are statistically significant. This is a subject of exploration for future work.

**Absence of response depression in multi-sensory experiments** The experimental measurements from the SC show not only response enhancement but also response depression during multimodal stimulus presentation \[134\]. While the simulation results in section 5.6 show response enhancement properties in some of the \(B_2\) neurons, none of the cells show response depression. In \[133\] the authors explain the response enhancement and depression through the existence of receptive fields with excitatory centers and suppressive surrounds. These suppressive surrounds enable response depression during presentation of multimodal stimuli. In the simulations shown above, there was no such provision for cen-
ter surround receptive fields. All the inputs were excitatory. Introduction of suppressive surround inputs in the STC model might introduce depression of response to multimodal stimulus presentations. This possibility has not been explored in this work and is left for future studies.

5.7.2 Summary

The results demonstrate the STC network can be successfully realized on a neuromorphic hardware and can be used to learn to detect complex spatio-temporal spike patterns in a completely unsupervised manner. The network does this by capturing the spatio-temporal features of the spike patterns. It relies on precise spike timing of the spikes as opposed to simply the coincidences in the spike patterns. This enables the network to learn to differentiate between spike patterns that share the same set of coincidences but have different spike timings. The network scales linearly with increasing complexity of the input spike patterns. The response reliability increases with increasing size of the readout population. This allows for interesting spike based computations such as multi-sensory fusion shown here. The current hardware does not allow learning of spike patterns that are embedded in noise. Alternative learning strategies were proposed to overcome this problem. It was shown that slow learning strategies that can retain the synaptic state over a period of time (longer than the length of the stimulus) can enable the STC to learn to recognize spike patterns embedded in background activity.
Discussion and Conclusion

An important problem in neuromorphic engineering is that of mapping neural computational models and their parameters onto neuromorphic hardware. This is typically done by calibrating the hardware and tuning parameters [150, 183, 152] on hardware. Such calibration procedures also involve complex techniques to compensate for intrinsic mismatch [151]. This is very challenging and sometimes even impossible because of restrictions on the parameters available on the chips. To solve this problem, one of the solutions adopted by modelers working with neuromorphic multi-neuron systems is to adopt population coding. Mean field theory based approaches like attractor networks [71, 37, 32, 77] and Finite State Machines (FSMs) [147] have been used in the past and continue to be quite popular. The Neural Engineering Framework (NEF) [58] has also gained popularity within the neuromorphic community in the recent past owing to its population coding strategy. While these population coding and rate coding approaches can overcome the problems of parameter mapping and parameter variability, they do not exploit the full potential of spike based neuromorphic devices. The timing of spikes is often ignored. There have been some attempts at exploring coincidence and synchrony based computation [202, 38] in neuromorphic systems. However, in those examples the timing of spikes have a limited role and do not exploit their full potential for computation. There have also been attempts at building elaborate neuromorphic devices with dendritic compartments [205], to enable dendritic computation for processing spatio-temporal spike patterns, this approach requires elaborate circuitry and might lead to inefficient utilization of silicon area on VLSI chips. Other approaches involve FPGA based implementations of neural networks [203] that include mechanisms for delays. This approach requires large amounts of memory with increasing size and complexity of networks and its underlying neural mechanisms.

An efficient neuromorphic building block that can utilize the full potential of spike based computation would lay a strong foundation for building intelligent neuromorphic systems. Several such systems could be used as parallel and sequential filters to perform complex tasks that go beyond simple spike pattern recognition. An important contribution
of this work was the development of such a building block. It shows how spike timing based neuromorphic systems can be built with minimal complexity and configuration.

In particular this work concludes that such a building block can be built simply from a pool of heterogeneous neurons projecting onto coincidence detectors. A pool of heterogeneous point neurons disperse the input stimuli in space and time, into a unique pattern that is specific to the spatio-temporal properties of the stimulus and the architecture of the network. The temporal dispersion results from the heterogeneous propagation delays inherent to the architecture and parameters of individual components in the network. A population of coincidence detectors can then be trained to monitor the activity of such a network to detect arbitrary spike patterns. The selectivity of the coincidence detectors to respond to a specific spike pattern emerges as a result of repeated presentations of the stimulus in an unsupervised manner. The resultant synaptic states of input stimuli are predictable and are dependent on the effective delays of the feed forward network. We employed this idea to implement the STC model described in Section 3.3 on analog VLSI based neuromorphic hardware (Section 5.1) by exploiting inherent transistor mismatch [158]. The STC model allows online learning and real-time recognition of repeating spatio-temporal spike patterns in the afferent stimulus by embedding the spatio-temporal information in the propagation delays of the network. The plasticity of the system allows it to self-organize based on stimulus properties and mismatch within the system. While the precise parameters of heterogeneous neurons causing delays help predict the resultant synaptic states, they are not required for the functioning of the network. This is a key feature that can enable us to build large scale, low power, spike based neuromorphic systems that do not require explicit configuration of every single component in the system but can mould themselves to the physical properties of the substrate.

6.1 Working principles

Reichardt detector The functionality of STC model after training can be described as that of a ensemble of Reichardt detectors [165] each tuned to respond to specific subsections of the spike pattern. The readout neurons $B_2$ act as Reichardt detectors in conjunction with the delayed projections that were learnt, to ensure convergence of spikes on presentation of the target spike pattern. As was discussed in the previous section this can also be seen as formation of a polychronous group that is activated on the presentation of the target spike pattern. A key aspect of the model is the ability to learn the right set of projections that lead to such convergence of activity at the readout neurons. (The analogous concept in the spatial domain is that of associative memory formation [87, 88, 102].)

Non-specific STDP rule The STC model implemented in chapter 4 utilizes a synaptic update rule that is based on calcium concentration and membrane potential at the onset
of presynaptic spikes (see 4.1.1.3 [24]). This learning rule was operated in a regime where it approximates to the classical exponential STDP rule [19]. The choice of learning rule was purely for practical reasons owing to the fact that the available neuromorphic chips only had this learning rule implemented on their plastic synapses. The true requirement of the learning rule is simply that it should enable the readout neurons to learn patterns of coincident spikes, since that is the underlying functionality of $B_2$ neurons. Therefore any online spike-based learning rule that can fulfill this requirement can be utilized to implement this model on neuromorphic systems. In [76] the authors analyze various weight independent and dependent STDP rules that are commonly used in literature and provides an overarching picture of the inherent algorithm. They show that a linear IF neuron with pairwise STDP performs a spectral analysis on the temporal cross-correlograms resulting in principle component analysis. In other words, the neuron learns to detect consistent coincidences in its afferent inputs. Fig. 6.1 shows several possible spike based learning kernels simply based on pairs of pre and post synaptic spike times. Evidence of such diverse learning rules has been found in vivo [3, 69]. Most of these learning rules can endow a neuron with coincidence detection capabilities. Except for the bottom left plot, all the other types of STDP kernels can be used with the STC model without hampering its performance. The bottom left plot in Fig. 6.1 results in depression of synapses that are activated simultaneously and so the neuron performs the opposite of what we require for coincidence detection.

The non-specificity of the learning rule has important implications to neuromorphic hardware implementations. It provides immense robustness to learning parameters of the synapses. This is especially interesting when it comes to emergent technologies like the memristors [178] which show a heterogeneous mixture of weight update functions very similar to those shown in Fig. 6.1. One of the major criticisms against memristors so far has been that they are unreliable and very heterogeneous. But such heterogeneity can also be found in the nervous system [3, 69]. This work provides an apt methodology to utilize such heterogeneous systems and gives us an understanding of how those systems can self-organize towards a common goal.

**Exploiting mismatch**  
Device mismatch can be minimized using standard electrical engineering approaches and appropriate analog VLSI design techniques. But this leads to very large transistor sizes and layout designs, which can significantly reduce the number of neurons and synapses that can be integrated onto a single chip. Rather than attempting to reduce mismatch using brute-force engineering approaches, neuromorphic approaches should try to exploit the adaptation mechanisms and learning strategies that they seek to model and implement in hardware. For example this has already been a successful strategy in neuromorphic vision sensors that employ adaptation (adaptive photoreceptor circuits) in each individual pixel rather than the single/global auto-gain mechanisms used
in standard imagers [51, 90, 83, 117]. Hardware neural networks can also employ population coding approaches and make use of redundancy, exploiting the large number of parallel elements present in these devices [151]. Homeostatic mechanisms [16] and plasticity are also very effective mechanisms for compensating the effects of device mismatch [31, 14]. The use of these strategies would allow designers to implement large arrays of compact, redundant and possibly plastic synapses that can carry out robust computation even if they are affected by mismatch. In approaches as the NEF [58], mismatch can provide a heterogeneous population of neurons with different tuning curves that can then be used for encoding different variables. In this thesis, mismatch was used as a source of heterogeneity to generate a range of propagation delays. Mismatch can then be used as a feature, rather than being something to try to minimize.

**Dendritic computation**  The source of delayed activity is irrelevant to the functioning of the system. In fact, during the inception of model, these delays were envisioned to be axonal or dendritic delays. In the context of dendritic delays being used in the model, the functioning of STC model can be seen as dendritic computation [121]. In [205] a VLSI implementation of neurons with active dendrites was used to demonstrate spatio temporal processing of spike patterns. The delays through dendritic arbors can enable convergence of non-synchronous presynaptic spikes. This idea was exploited even as early as 1948 by
Jeffress [99] to explains how auditory systems can register and analyze small differences in the arrival time of sounds at the two ears in order to estimate the direction of sound sources in the azimuthal plane. But it is important to note that dendritic delays are limited to a few milliseconds (10 ms at best) and whereas other non-linearities through network dynamics even with point neurons can account for much longer propagation delays. The implementation of \texttt{STC} model with neuronal propagation delays as opposed to dendritic delays opens up several other possibilities and adds to its computational abilities as we discuss above.

A spike based neuromorphic system with an implementation of \texttt{STC} model was demonstrated in chapter 4. It was shown in Chapter 5 that the \texttt{STC} model can learn to recognize complex spatio-temporal spike patterns. This model differs from other state-of-the-art models like the Tempotron [81], the SEM [154], STDP based models like [127], recurrent network based models like LSM [146, 122] and polychronous networks [96] in several aspects. In this final chapter we will discuss these computational aspects, possible extensions and its implications to spike based processing in neuromorphic engineering.

### 6.2 Key differences from single neuron models

Several state-of-the-art models for detection and classification of spike patterns rely on a single neuron to detect spike patterns. We have investigated some of these models in Chapter 2 and in Chapter 3 concluded that these models largely rely on coincidences in the spike patterns to identify and discriminate them. In most cases, especially when dealing with randomly generated spike patterns, coincidences can be a very handy criterion to differentiate between them. As is shown in [127, 154], such a strategy of relying on coincidences can be used to successfully recognize a specific spike pattern embedded in a stream of random spike patterns. Part of the reason behind the success of this approach is the fact that randomly generated spike patterns are drawn from a uniform distribution. The probability that a randomly drawn spike pattern shares the same coincidences as that of the target spike pattern is very small. But as we have seen from the simple gesture example with time reversed spike patterns in Sec. 3.2, in real-world scenarios spike patterns often do have similar synchronicity but differ in their temporal ordering.

The \texttt{STC} model takes an alternative approach [99, 116] to this problem by relying on ISIs to detect spike patterns. The ISIs in the spike pattern are transformed by the delay neurons $C$ and the appropriate ISIs of the target patterns are learnt through saving the correct set of connections from the delay neurons. It must be noted that eventually the \texttt{STC} model also relies on coincidence detection at $B_2$ population. But these coincidence detector neurons are not presented the actual spike pattern but the dispersed spiking activity from the delay neurons. This dispersion through delay neurons breaks the synchronicity when the target spike patterns are time reversed and therefore the coincidences caused by the
actual spike pattern are not the same as those when the time reversed spike pattern is presented.

A second difference between the STC model in comparison to the others [81, 127, 154] mentioned above is in the precision of synaptic weights of plastic synapses. The STC model uses synapses [24, 139] with binary weights, i.e. the final synaptic weight always drifts to one of two values, a high weight and a low weight. The synaptic weights used in other works like [81, 127, 154] use continuous weight values both during and after learning to detect spike patterns. The precision of these synaptic weights plays a crucial role especially during learning in these models. This is because the inputs through the plastic synapses drive the post-synaptic neuron. Even small changes in their weights influence the spike timing of the post-synaptic neuron and consequently affect the weight changes through STDP. In the STC model, on the other hand, the readout neurons $B_2$ receive inputs from both plastic and non-plastic synapses and therefore the learning is relatively more stable.

The use of binary synapses have a crucial part to play when it comes to neuromorphic implementations. Maintaining precise weights on analog circuits is challenging because of leakage currents in the silicon substrate. It is much more feasible to store synaptic weights in a binary form by maintaining only two distinct voltages for the synaptic weights to drift towards [92].

Another key factor that limits implementation of those models [127, 154] for online learning is that they have unstable dynamics owing to the fact that they are driven by the input through plastic synapses. If the synaptic weights are too low, the neuron does not fire and therefore it ceases to learn through STDP. On the other hand, if the synaptic weights are too high, the neuron fires at a high frequency thereby driving all the synaptic weights to a high value making the neuronal response non-selective to the input stimuli. There is a very small range of STDP parameters that depend on the firing rate of the input and frequency of target stimulus presentation that will ensure these neurons learn to detect the target spike patterns and have stable weights.

It is usually very difficult to find the correct set of parameters to ensure that the synaptic weights remain stable and learn to detect the desired spike patterns. An important mechanism to overcome this difficulty is to include homeostasis [2] in the neuronal dynamics. The homeostatic mechanism ensures that neurons maintain an average post-synaptic firing activity and helps regulate synaptic weights to a mean value suitable for learning. Such homeostatic mechanism operates at a very long time scales of the order of minutes or hours. Realizing such long time constants is challenging to implement on analog neuromorphic hardware.

SEM [154] uses competition among several spike detectors with global inhibition as an alternative to intrinsic homeostasis to maintain the activity in the network. The STC model does not require homeostasis (but having homeostasis has its advantages as we will discuss later in this chapter). This is because each of the readout neurons $B_2$ only
observes a small number of synapses and their coincidence detection is limited to a very small number of afferents. The small number of plastic synapses in addition to the fixed input from static synapses ensure that the neurons remain sufficiently activated.

The tempotron [81] does not require explicit homeostasis mechanism because the supervised learning rule acts as a homeostatic mechanism by regulating the weights at every epoch to ensure that the neuron fires appropriately (i.e., spikes on presentation of target stimulus and remains silent otherwise). Moreover, the neuron is allowed to spike only once during the presentation of the spike pattern, thereby eliminating the need for any homeostasis. The same cannot be done in unsupervised learning models like Masquelier et al. [127], SEM [154] and STC without an intrinsic homeostatic mechanism.

In all these models, even if one could accurately determine the right set of parameters required to operate and learn the spike patterns, translation of these parameters into heterogeneous hardware is yet another barrier. Accurate calibration routines need to be employed in order to ensure that the desired set of parameters are precisely translated to hardware. This is even more of an issue with neuromorphic hardware where populations of mismatched synapses are controlled by global parameters, because individual calibration of neurons and synapses is not feasible. The STC model on the other hand is easier to configure and does not require each individual neuron to be perfectly tuned. Because the readout in the STC model is based on population activity, as long as a subpopulation of readout neurons act as coincidence detectors, the model can still detect spike patterns.

6.3 Key differences from recurrent networks

There are several parallels between the network architecture of STC model and other state-of-the-art network based models such as polychronous networks [96] and liquid state machines [146, 122]. Let us have a look at how STC model compares to those.

Polychronous networks

Izhikevich presented a minimal spiking neural network that can exhibit reproducible time locked but not synchronous firing patterns with millisecond precision, defined as polychronization [96]. His network consists of spiking neurons recurrently connected with axonal delays and STDP. The interplay between STDP and delays in the network leads to self-organized groups of neurons that generate patterns of stereotypical polychronous activity. The principle working component of this network is neurons that are activated by other presynaptic neurons which spike at different times and propagate their spikes with specific delays causing coincident convergence of spikes at the postsynaptic neuron. The principle behind this is identical to that of a Reichardt detector [165]. The same is the case with the readout neurons $B_2$ in STC model.
The difference lies in the fact that in polychronous networks, polychronous groups (group of neurons generating patterns of stereotypical spiking activity) emerge as a result of spiking activity and structure of the network. On presentation of a spike pattern, a small number of these polychronous groups are activated reflecting the presentation of the spike pattern. Since these groups could have formed within any subpopulation of the network, in order to detect whether or not a pattern has been presented, these groups have to be identified first. Furthermore, a single neuron could participate in multiple polychronous groups. So monitoring a single neuron within the group is not sufficient to detect the activation of the group. This requires an additional pattern detection mechanism and poses a circular problem in terms of spike pattern detection and is therefore not a suitable solution to spike pattern detection.

After repeated external stimulation with spike patterns, multiple polychronous groups are activated on presentation of the stimulus. But not all groups corresponding to a pattern are activated when the network is stimulated. Only a small percentage of these groups are randomly activated based on the internal oscillations in the network. This makes the problem of pattern recognition even harder because we now require a readout mechanism that can monitor multiple groups in order to make a decision. The lack of an explicit or simple readout mechanism makes it hard to use polychronous networks as pattern detectors. The STC model on the other hand provides a dedicated set of readout neurons whose output could be readout with a simple spike count.

Liquid State Machine

Liquid State Machines (LSMs) \cite{146, 122} were proposed as a framework for real-time computing without stable states based on perturbations. A snapshot of perturbations caused in a network of recurrently connected neurons called liquid neurons by temporal inputs are readout by a set of readout neurons. While the connections from the liquid neurons to the readout neurons are plastic and are learnt depending on the specific filter operation they need to perform on the input, the liquid neurons are completely untrained. The authors argue that LSMs can demonstrate universal computation power under idealized conditions. The two necessary and sufficient conditions for such powerful computation are a separation property and an approximation property \cite{122}. The separation property implies that the liquid neurons should be able to produce two distinct trajectories of neuronal dynamics on presentation of two different patterns that need to be distinguished. The approximation property means that the readout neurons should be capable of distinguishing and transforming different internal states of the liquid into given target outputs.

The temporal dynamics of synapses employed in LSMs can in general be described as facilitation and depression of synapses with time constants of the order of hundreds of milliseconds (30 ms to 300 ms). Implementation of such dynamics in neuromorphic devices require additional circuitry \cite{17} involving several transistors and capacitors in addition to

102
the simple synaptic circuits for creation of an EPSP. In addition, the nonlinearity of these dynamics requires a physical synapse circuit dedicated to each synapse in the model and cannot be shared to function as multiple virtual synapses. Increasing complexity and size of the network demands more resources in terms of silicon area on the chip. So most available neuromorphic multi-neuron chips do not implement these dynamics rendering their utilizations for temporal information storage (as is done in LSM) non-optimal.

Moreover, tuning the parameters of recurrent networks is often heuristic and one needs to choose the right parameters for recurrent connections according to the task at hand. This is usually done on a trial and error basis as it is difficult to predict the behavior of the network. The mean firing rate of the network activity and its dynamics depend nonlinearly on the mean firing rates of the input stimuli because of the gain through recurrent connections. So the weights and connectivity of the recurrent network have to be carefully tuned according to the mean firing rates of the input stimuli. To ensure that the network activity doesn’t explode and still has some kind of fading memory properties, relies on fine tuning of parameters and is heavily dependent on the mean firing rates of the input. Limited parameters available in neuromorphic hardware makes it an even more daunting task to configure recurrent networks on hardware systems. On the other hand, in case of delays neurons of STC model it is a straightforward task of choosing the appropriate set of delays corresponding to the temporal correlations that are desired to be learnt. This makes implementation of STC model on neuromorphic systems effortless in comparison to LSM.

The readout neurons in LSM are typically trained on a snapshot of the liquid that represents the state of the liquid after time \( t \) of the presentation of the input. This is done in a supervised manner \([13]\) in order to train the weights of synapses such that the readout neurons respond with the desired output. In a real-time system, obtaining a snapshot for training requires a temporally precise training routine. Such a routine is required in order to activate the learning mechanism at the perfect time after the presentation of the stimulus and to send the correct teacher signal in conjunction with the input. Doing so in real-time with neuromorphic systems is non-trivial and requires elaborate training routines, prior knowledge of the stimulus being presented and their labels. The readout neurons in STC model on the other hand employ a unsupervised learning strategy and do not require any additional teacher signal, a substantial advantage for building autonomous systems.

6.4 Outlook

The STC model learns to recognize spatio-temporal spike patterns perfectly well under controlled input conditions like the ones shown in section 5.1.1. It can learn to detect arbitrary spike patterns as long as the sub features can be captured with the delays available.
within the network. But in a real-world application the spike patterns generated by spiking sensors are very complex and vary in terms of firing rates and length of stimuli and temporal correlations. Below we discuss some of the limitations of the model in handling such stimuli and possible upgrades to the model to tackle those limitations.

### Input normalization

The inputs used in this thesis to characterize the learning and detection capabilities of the STC model (see Fig. 5.1) consisted of one spike burst per neuron within the full length of the stimulus. Each of this spike bursts caused a $B_1$ neuron to fire exactly once. We also saw in the software simulations that the model can learn when the $B_1$ neurons fire multiple spikes within a poisson distributed spike pattern (see Fig. 5.11a).

This is an effective methodology as long as the temporal correlations are meaningful and relevant to the stimulus. If the stimulus consists of long periods of sustained activity at high firing rates, the temporal correlations might not be as crucial as the spatial correlations. This is especially the case in static spatial stimuli where the stimulus properties can be encoded with firing rates. For such stimuli spatial correlations are sufficient to encode the relevant properties of the stimulus. In such cases, the STC model will learn the spatial correlations in the input pattern and detect the stimulus, after which it will not be able to differentiate between spatio-temporal stimuli that share the same spatial characteristics. This is not really an issue in practice because it is very unlikely that the same synaptic afferents carry information with both rate code and temporal code.

It is conceivable that the stimulus contains long bursts of spikes where transients like onset/offset times are the relevant features of the stimulus. In such cases, the STC model in its current form will confuse the persistence of input spikes for long periods of time to rate coding, because the $B_1$ neurons activated by input will fire continuously. Therefore a mechanism that is crucial for such tasks is an onset/offset detection. It was shown in [187] that strong synaptic depression can enable LIF neurons to perform robust onset detection. The same can also be an active mechanism to regulate high firing rate in the input spike trains so that $B_2$ population which act as coincidence detectors do not get triggered simply because of high input firing rates from one of its afferents.

### Delay generation

The spike patterns corresponding to a stimuli may vary in their duration from tens to hundreds of milliseconds. The STC model should be able learn the correlations in the stimuli according to the length of its corresponding spike patterns. The observable time window for $B_2$ neurons, the coincidence detectors in the model, is defined by their membrane time constant and the delays of $B_1 \rightarrow C \rightarrow B_2$ projections. The smaller the range of these delays is, the smaller the calculable temporal correlations are. It is therefore important to
understand the necessary range of delays for a given set of stimuli. For the stimuli used in this thesis, these delay ranges are defined by the velocity of sweeps. Note that it is not the total length of the stimulus that defines the necessary delays but the length of sub-features that each coincidence detector $B_2$ observes and detects. In the examples used in the thesis, each $B_2$ post training was only observing three to four neighboring neurons as can be seen from the resultant weight matrices (see Fig. 5.3). Since the delay between input activation of neighboring neurons for stimulus sweeps is only of the order of a few milliseconds (1 to 4 ms), delays in that range are sufficient to learn and detect stimuli that last up to $4 \times 32$ ms. While this would cover considerably long spike patterns, it would be advantageous to have longer time delays. Longer delays would accommodate learning and detection of longer spatio-temporal spike patterns. It will also allow $B_2$ neurons to learn temporal correlations in the stimulus that are temporally separated for longer periods of time. Therefore it is worth investigating how longer delays can be realized in $B_1 \rightarrow C \rightarrow B_2$ projections.

Axonal or dendritic propagation delays beyond 30 ms are of course not biologically prevalent [191]. But those are not the only mechanisms that cause delays. Even in this work, delays were not generated by dendrites or axons but were a consequence of slow synaptic and membrane dynamics (see Sec. 4.2.1 [180]). An additional neuron $C$ was introduced between $B_1 \rightarrow B_2$ projections to incorporate transmission delay. The range of delays obtained there was simply a result of transistor mismatch [158], where some $C$ neurons caused a shorter delay and others longer delays. This idea can be taken further by introducing a sequence of neurons like a syn-fire chain [5]. Fig. 6.2 demonstrates this idea, with a chain of neurons forming a syn-fire chain and projecting to $B_2$ neurons. The delays accumulate in this connectivity with growing distance, greatly extending the range of delays available to the network.

So in the fashion described above, each $B_1$ neuron can form a syn-fire chain [5], the neurons of which project to $B_2$ neurons. From a neuromorphic engineering perspective,
you can achieve a longer range of delays with the same number of neurons as we did in Chapter 4. This strategy of generating delays through a chain of neurons was also used by Licklider in his duplex temporal autocorrelator model for pitch perception of auditory stimulus. Adopting the same strategy for implementing the STC model will greatly increase the capacity of the hardware implementation in terms of the length of patterns that can be learnt and recognized.

In fact, a neat arrangement of syn-fire chains is not necessary for the model to learn and detect spatio-temporal spike patterns. Let us consider a simple two layer feed forward network with random connectivity as shown in Fig. 6.3. Such a network of feed forward connections still induces transmission delays of the order of synaptic time constants. The activation time of neurons in population C will still have some temporal spread based on the spatio-temporal properties of the input. So projections from C to B₂ can be learnt just like we did in chapter 4. Such feed-forward randomly connected networks are shown to have very interesting computational properties like mixed selectivity with respect to spatial pattern classification. The STC model can gracefully adopt this structure into its network in place of population C to take advantage of mixed selectivity.

Within this framework of randomly connected feed forward networks for delayed activity propagation, a similar idea to that of syn-fire chains can be applied. Multiple layers of feed forward structures can be employed for delay generation. The divergence and convergence of activity in such feed forward networks can cause neurons sensitive to specific features (random features as a result of random connectivity) of the spatio-temporal spike patterns, that could be utilized by B₂ neurons to learn temporal correlations. The sparsity and strength of the random projections will dictate the capacity of activity propagation and information retention in the network.
LSM approach for delay neurons

The STC model is also based on computation of temporal information without stable states. It fits into the LSM framework if the delay neurons $C$ are seen as liquid neurons. Even though they delay neurons in the STC model are not recurrently connected as in the liquid neurons, both these populations perform the same job of separating the input space in their corresponding networks thereby endowing separation property to the network. In both models, a diversity of parameters of the liquid or delay neurons enhances their separation properties. The readout neurons in STC model are similar to those of LSM in that they both perform the approximation property. While both STC model and LSM work on the same computational principles, the implementation of these networks is different. The temporal properties of the stimuli are preserved through propagation delays in STC model where as they are preserved through the temporal properties of short term dynamics of the synapses in LSM. Additionally LSMs employ recurrent networks to enhance the memory of the network whereas STC model as described in Section 3.3 employs simple feed forward connections. The approach used in LSM for the liquid could be adopted in STC model without any change to the rest of the network by replacing the delay neurons $C$ with a liquid. But this brings forth some limitations like more complex synapses and parameter tuning as was discussed earlier.

Patterns embedded in noise

The learning strategy adopted in the STC model is that of unsupervised learning i.e. there is no external teacher signal. In the case of hardware experiments (see Section 5.1) the network was first exposed to the desired spike pattern that is to be learnt and then the tuning curves were measured with presentation of different stimuli. During training phase, the only stimulus the network is exposed to is the target stimulus. The number of stimulus presentations required to learn the stimulus features is also very small in this particular hardware implementation. The experimental protocol is described in detail in section 5.1.1. This is not an ideal unsupervised training protocol. Typically in unsupervised learning, the target pattern is embedded in noise or other patterns \[127, 154\]. The target pattern is identified by the fact that it is the pattern that has occurred repeatedly during the presentation while other spike patterns were not, making it statistically significant. Unsupervised learning rules rely on capturing the statistically significant stimulus cues in order to detect the target stimuli. This capability was demonstrated through software simulations of STC (see Section 5.5) but could not be demonstrated with the available hardware system. In the hardware implementation if the learning is always active, the network will always learn the correlations presented in the last presentation and will not rely on statistically significance of the stimulus cues. This happens because the network learns too quickly and overwrites its previous learnt memory. This is because of the
fact that the learning rule used for the hardware implementation uses bi-stable synapses. After every presentation of the stimulus, the synaptic weight drift to either a high or a low value. So unless the patterns are presented very quickly in succession, there is no “evidence accumulation” since the weight perturbations are not accumulated. In other words, the learning rate of the system is very high and consequently so is the rate of forgetting.

So clearly if we would like to train the network and learn statistically significant information, we need a slow learning rate. This can be achieved by using a very small drift velocity ($\alpha$ in eqn. 4.5) of synaptic weight such that weight changes over several stimuli can be accumulated. This was demonstrated in the software simulations. However, with the current analog VLSI synapse implementation, the lowest possible velocity is limited by the transistor leakage current, resulting in effective weight retention for a maximum of one or two seconds. (In fact this is partly the reason why synaptic weights on hardware were chosen to be bi-stable in the first place.)

One solution to be able to learn slowly on neuromorphic hardware and alleviate the problem of overwriting of memory of the network is to have multi-stable synapses as opposed to bistable synapses without altering the learning rule. This will impart some amount of memory retention capability into the network. The more stable states the synapses can accommodate, the slower the effective learning rate is. In software simulations for example, weights are stored with an $N$ bit (32 or 64 bits) resolution, thereby enabling slow learning. The simulations by Masquelier et al. [127], use continuous weights bounded by a maximum and minimum value, to do unsupervised learning of spike patterns embedded in noise (see 2.2.2.1 for a short description of how their model works). The “continuity” of weights is defined by the number of bits used for the weight variable. However, for a reasonably slow learning rate with multi-stable synapses, one would require a large number of stable states of the order of few tens of bits. This requires considerable addition of real-estate and digital circuitry to the synaptic circuits.

An alternative approach to learning statistically significant synaptic transitions is to use stochastic synapses which are still bi-stable. Stochastic synapses make synaptic weight transitions probabilistically based on the properties of the stimulus. An LTP or LTD is triggered based on the timing of the pre-synaptic or post synaptic spike timings, where the transition probability is equivalent to the weight increments in exponential STDP rule (see Sec. 2.4 for exponential STDP rule). It is important here to note that, while the learning rule implemented on hardware used in this thesis (see Sec. 4.1.1.3 [24]) is often referred to as a stochastic learning rule, it is stochastic only when the spike timing of neurons is considered to be draw from a distribution. The learning rule is typically operated with Poisson spike trains and the stochasticity is a result of the stochastic nature of the spike trains and membrane potential. The synaptic transitions are not stochastic given a specific state of the pre and post synaptic spike times and membrane potentials. What we require here is truly stochastic transitions in the synaptic weights. Recent work with memristor
based synapses is a possible avenue for such stochasticity \[95\]. It has also been argued (by Kwabena Boahen in several of his talks) that as the transistors shrink with shrinking CMOS technology, the current through a transistor itself becomes stochastic, since only a few electrons can pass through its narrow channel. This could also be a possible mechanism to realize stochastic synapses in CMOS.

**Learning multiple patterns**

This thesis demonstrates how a single pattern can be learned with the STC model. In real-world applications, stimuli might contain multiple recurring patterns. Therefore it is desirable that a neural system is capable of learning a set of features that the sensory inputs might contain. Nessler et al. \[153\] demonstrate that a population of neurons competing with each other can learn different hidden causes in their stimuli. The same strategy is also employed by Masquelier et al. \[126\] where different time windows of a stimulus are learnt by a population of neurons competing with each other. In other words, where multiple stimulus patterns are repeatedly presented embedded in background activity, each neuron in the population learns one of the many patterns that were repeatedly presented. This way, each neuron in the population learns to detect a different pattern, and the population as a whole learns to detect the entire set of stimuli.

A similar idea can be adopted in the STC model by introducing competition among the $B_2$ neurons. Since the number of sub-features that the network needs to remember increases with the number of patterns repeatedly presented to the network, the size of $B_2$ population should also be scaled up. Note that this does not require any increase in the number of neurons in population $C$. The same population can be used to project onto any number of $B_2$ neurons. With a larger number of $B_2$ neurons, the network can learn a richer set of temporal features present in the stimuli and can learn to recognize multiple patterns.

**Readout**

The STC model identifies the target stimulus by spiking activity of $B_2$ neurons. The total number of spikes from $B_2$ neurons represents the feature proximity of the observed stimuli to the target stimuli. In the experiments conducted in this thesis, this count is performed manually. Instead this could have been done with an additional LIF neuron that gets its inputs from $B_2$ neuron and fires when the number of spikes from $B_2$ neurons firing exceeds a reasonable threshold. Let us call this neuron a recognizer neuron. This approach was taken in \[109\], where the readout neuron receives time delayed input from multiple pattern detectors tuned to respond to different time windows of a target spike pattern. While this is sufficient to recognize and detect a single stimulus pattern, to recognize and detect multiple patterns we need multiple recognizer neurons. An additional population
Fig. 6.4: Feed forward projections from $B_2$ neurons of the model described in Section 3.3 to a population of recognition neurons $R$.

$R$ with competition can be introduced for such multiple pattern recognition, that receives inputs from $B_2$ neurons as shown in Fig. 6.4.

The STC network is based on unsupervised learning principles and so the size of recognizer population $R$ is given by the number of stimulus patterns that are required to be recognized. Since $B_2$ neurons are already tuned to temporal properties of a subset of neurons, it is sufficient for the $R$ neurons to rely on the number of spikes regardless of the precise temporal properties of the spikes coming from $B_2$. It is important to note here that the connectivity from $B_2$ neurons to the recognizer neurons still needs to be learnt in order to separate the various spatio-temporal features of different stimuli that are to be recognized. This is a relatively easy task because these neurons only have to perform a spatial classification task as opposed to a spatio-temporal classification.

**Feedback connections**

The readout neurons $B_2$ in the STC model (see Fig. 3.2) are driven by input from $A$ and delayed input from $B_1$ neurons in a purely feed-forward fashion. The activity of any one readout neuron is independent of the activity of other readout neurons in the current model. Similarly, the activity of delay neurons $C$ is driven in a feed-forward manner by the input from $B_1$. Let us investigate the role of feedback projections in such a network analogous to feedback from higher cortical areas in the nervous system.

For simplicity let’s introduce feedback connections from readout neurons $B_2$ via delay neurons $C$ similar to the feed forward projections from $B_1$ to readout neurons. Since the activity of the readout neurons is phase locked with the corresponding input activity from
A, their activation is similar to that of $B_1$ neurons for the target stimuli (readout neurons remain silent for non-target stimuli). The difference between the activation of $B_1$ and readout neurons is the following: $B_1$ neurons get activated when the corresponding input neurons $A$ are activated; readout neurons get activated when a pool of input neurons $A$ are activated in a specific temporal order leading to coincidence activity at the presynaptic end of the readout neurons. Therefore the activity of readout neurons is a richer source of evidence of stimuli than the activity of $B_1$ neurons. So if the readout neurons learn coincidences based on the activity of other readout neurons, a given readout neuron would only fire if the previous readout neurons have fired with the learnt spike timings. This argument can be repeatedly applied to activation of other readout neurons till the activation of the first readout neurons (which are only activated by the stimuli). In such a network, the activation of a single readout neuron (last activated in time) is sufficient to detect the target spike pattern instead of observing the population activity.

Such feedback projections have also been exploited by Hoerzer et al. [86], in the context of generic microcircuits for different computational specializations. Their work was an extension of the original LSMs [122] where they train multiple readout populations to perform different tasks. They demonstrate that some complex tasks can be learnt on the basis of previously learnt simpler tasks and therefore depend on the activity of these more primitive readout populations. They use feedback connections from readout neurons back to the liquid neurons of LSM to perform these complex computational tasks.

This idea of recurrent connections is similar to that described in *Timing-Based Plasticity and Prediction* by Dayan and Abbott (see section 8.3 in [47]). They explore the effect of time-dependent synaptic plasticity in a network of neurons with recurrent connections. When such a network is trained with repeated presentation of a spatio-temporal stimulus, they show a backward shift in tuning properties of the neurons after training. They interpret this as predictive response to the stimulus. This happens because the neurons are activated by strengthened recurrent projections from other neurons in the network which were previously activated by the stimulus much like the activation of the readout neurons as discussed above. Extending this idea of recurrent feedback connections to the STC model will greatly extend its theoretical computational capabilities [86].

The research activities in this thesis have produced very interesting results by demonstrating a practical strategy for implementing autonomous systems with imprecise neuromorphic systems for spatio-temporal spike pattern processing. The results obtained here open new avenues in the field of neuromorphic engineering with more focus on spike-based computation and less on the practical problems of configuring the systems. Further exploration of this work if successful can lead to novel computational paradigms and architectures that are non von Neumann. The ideas explored here can be extended to emerging technologies like memristors [178], nano-devices or yet to be discovered technologies that share commonalities with the nervous system and show a potential for neuromorphic engi-
neering. They can lead us towards building real-time low-power self-configurable systems that play a crucial role in embedded systems, medical implants, brain-machine interfaces and cognitive machines assisting us in our daily chores.
Appendices
Appendix A

Stimulus Specific Adaptation

Biological neurons communicate sensory information efficiently by reducing the resources consumed for the transmission of frequently-occurring stimuli, and preserving them for the transmission of rare stimuli that surface against a background of common events. The enhanced response to rare stimuli or events is important, because they are likely to signal changes in the environment relevant to behavior (e.g., the arrival of a predator). Evidence for such a filtering mechanism appears in the form of Stimulus-Specific Adaptation (SSA), that is, a decline in spiking in response to a repeating stimulus, which recovers following the presentation of a different stimulus [198]. Neurons that exhibit Stimulus Specific Adaptation (SSA) in response to tone sequences have been located at various stages along the auditory pathway, specifically, in auditory cortex in cats [198, 197] and rats [18], auditory thalamus in mice [7, 9], and inferior colliculus in rats [159, 124].

This chapter demonstrates the development of a computational model of SSA and its implementation in neuromorphic analog VLSI. It compares the output of the hardware model to findings presented in the physiological literature [18, 159]. Two types of experiments were performed. The first were “oddball” experiments: sequences of tones (i.e., encoded input patterns) were presented, in which one tone was rare and other was common. A heightened response to the rarer tones was sought. The effect of separating tones in frequency, making the rare tone rarer, or changing the presentation rate was investigated. A second series of experiments presented tones at many frequencies but varied their order, so that the local history preceding any given tone was dominated either by the same tone, a similar tone or random tones. The results obtained are broadly consistent with those published in the physiological literature [159].

The remainder of this chapter is divided as follows. Section A.1 describes some of the stimulus configurations used in previous SSA experiments and the results that were obtained. Section A.2 introduces the architecture of the neural network (without reference to the hardware) and discusses how it accounts for the physiological results. Section A.3 describes how the hardware was configured, both in terms of implementing the network
Fig. A.1: An illustrative example demonstrating how the SI value is computed for a short oddball sequence in which $p_{dev} = 0.2$. The number in each circle is a spike count. (A) The first oddball sequence, in which $A$ is deviant, and from which we obtain $d_A = 6, s_B = 13$. (B) The second oddball sequence, in which $B$ is deviant, and from which we obtain $d_B = 6, s_A = 17$. From these four quantities, it follows that $SI = \frac{3}{13} \approx 0.23$. This SI value exceeds zero and thus indicates $SSA$.

model and encoding the stimuli as input. Section A.4 reviews the results obtained from the hardware and comments on them in light of biological findings. Section A.5 states the main conclusions.

A.1 Stimulus-specific Adaptation

A.1.1 Oddball Experiments

The stimulus-specificity of adaptation is generally measured by comparing the spike count elicited from a neuron in response to a rare stimulus to that elicited by a common one. Some examples of SSA measures include the SSA index (SI) [198], used in this work, and the novelty response index [159]. To compute the SI, two tones ($A$ and $B$) are presented in two configurations: in the first, tone $A$ is the deviant (appearing with probability $p_{dev} \leq 0.5$) and tone $B$ is the standard; in the second, the two tone probabilities are swapped, so that $B$ is the deviant and $A$ is the standard. Spike counts in response to each tone are recorded, and then

$$SI \triangleq \frac{d_A - s_A + d_B - s_B}{d_A + s_A + d_B + s_B}, \quad (A.1)$$

where $s_x$ and $d_x$, $x \in \{A, B\}$, denote the average spike count recorded for tone $x$ when presented as standard and deviant, respectively [198]. The effect of rarity upon the response is tested for both $A$ and $B$ to ensure that any simple preference of the neuron for either tone is cancelled. The denominator is included to normalise the index onto the interval $[-1,1]$. Stimulus-specificity is revealed when the SI values measured over a population, or for a single neuron over repeated trials, are positive on a significant number of occasions (according, e.g., to a signed rank test [130]). A short example oddball sequence is shown.

\footnote{Refs. [18] and [124] refer to the SI as ‘sAI’ and ‘NSSI’, respectively.}
Fig. A.2: Stimulus presentation modes: block, sequential and random. These abbreviated examples show 4 repetitions of 4 frequencies, instead of 10 × 10. These presentation modes are described in section A.1.2 and [159].

in Fig. A.1 actual oddball experiments employ several hundred tones (e.g., 800; [18]). SSA experiments typically vary basic parameters of an oddball sequence, such as the frequency difference between the tones (\(\Delta f\), octaves\(^2\)), the probability of a deviant (\(p_{\text{dev}}\)) and the interval between successive tones (\(\Delta t\), seconds), and investigate the effect upon the SI, i.e., the degree of SSA, as a dependent variable. Frequency separation and deviant probability appear broadly to have the same effect upon SSA, regardless of the brain area in which it is found [198, 197, 124, 7, 18]; namely,

- increasing \(\Delta f\) increases SI (with SI = 0 for \(\Delta f = 0\));
- increasing \(p_{\text{dev}}\) decreases SI (with SI = 0 for \(p_{\text{dev}} = 0.5\)).

Our first requirement of the hardware model is that it capture these two trends, given their ubiquity. To these we may add the third, weaker principle:

- increasing \(\Delta t\) decreases SI.

Two studies strongly support this conclusion [198, 9] (see also [159]). The effect is apparent, though weak, in one study [124], and absent (or at least obscured) in another [7].

A.1.2 Changing Stimulus Presentation Mode

For SSA neurons to respond with fewer spikes to more common stimuli requires a memory of past stimuli. Evidence from multiple studies suggests that this memory extends over multiple time scales [197], with the principal contribution to SSA arising from a time

\(^2\)Some studies [198, 197], instead of measuring \(\Delta f\) in octaves, make use of the normalised frequency difference, defined as \(|F_A - F_B|/\sqrt{F_A F_B}\), where \(F_x\) is the frequency of tone \(x \in \{A, B\}\) in Hertz.
scale that corresponds to the course of just a few tones [197] [18]. For instance, if the probabilities of tones A and B are switched during an oddball sequence, then after a few tones, an SSA neuron shows an enhanced response to the new deviant, as demonstrated using the switching-blocks paradigm [197].

Another form of SSA experiment, proposed in [159], presents an equal number of repetitions of many tone frequencies but varies the order of their presentation. Rather than comparing the mean responses to different tone frequencies, the overall mean response to all tones in the sequence is computed. In [159], ten tones are presented ten times (10 × 10 = 100 tones) in three presentations modes: block, sequential and random. These are illustrated in Fig. A.2. In block mode, tones are presented in blocks of identical frequency, in ascending order of frequency. In sequential mode, the protocol steps through the frequencies in ascending order, and then repeats this sequence. In random mode, the tones are presented in a random order.

SSA is demonstrated by this experiment in a more indirect manner and rests on the “short-term memory” idea outlined at the start of the section. In block mode, most tones are preceded by tones of the same frequency, and so the overall mean response is low. In sequential mode, tones are typically preceded by neighbouring tones, and so the slight difference between a tone and its predecessor provides some release from adaptation. Furthermore, when a sequence restarts, the jump from the highest frequency back to the lowest leads to a large response, which is incorporated into the mean. Consequently, the mean response is greater for the sequential mode than for the block mode. Finally, a random order of presentation ensures that tones are unlikely to be preceded by tones of the same frequency, or even neighbouring frequencies. The average response for the random mode thus exceeds that for the block and sequential modes. This pattern of responses is observed for biological SSA neurons but is absent in neurons that do not exhibit SSA [159].

A.2 Network model

The SSA model I present in this chapter consists of a single layer of Poisson input units (L1) and two layers of integrate-and-fire neurons (L2 and L3), with each layer connected to the next via dynamic synapses. A schematic illustration of the model is provided in Fig. A.3. The purpose of this section is to describe how the network fulfills requirements set out in Section A.1. A more detailed description of the network configuration used in the hardware experiments, including the numbers of neurons and synapses, is provided in Section A.3.

The units in layer L1 of Fig. A.3 represent tonotopically-ordered (i.e., frequency-ordered) input neurons. The frequency of a tone is encoded in the firing rates of the population. Overlapping receptive fields in L1 imply that a single tone activates a block of inputs.
Fig. A.3: Schematic diagram of the SSA model, depicting various network states during its operation (the numbers of units and synapses shown are for illustration only). 
A. The first time tone A is presented, a range of three units is activated in L1. As the L1–L2 synapses are not depressed, the signal is immediately conveyed to L3 via L2. B. After a few presentations of tone A, a subset of the L1–L2 synapses is depressed, and the units in L2 and L3 are silent. Thus if the standard tone is A, the average response to A will be low. C. The first time tone B is presented, it activates a partially distinct set of synapses, which means that some response is elicited in L3 (the deviant response). The lower the overlap between A and B (i.e., ↑△f), the greater the deviant response.

The task of subsequent layers is to adapt to blocks that are activated repeatedly. An essential feature of the model is that the synapses between layers L1 and L2 are depressing and spread out over a small range. Depression means that when spikes are communicated across a synapse in quick succession, the size of each resulting EPSP in the target membrane potential decays rapidly; and during the time between spikes, it recovers slowly (on the order of ∼100 ms; see [196]). Most models of synaptic depression, including the one used in this work, rest on the notion that one or more chemical substances responsible for generating EPSPs are gradually depleted by the arrival of spikes, and are remanufactured or recycled at a slower rate thereafter [196]. Thus a repetitive stimulus, which generates spikes in the same region of L1 each time, will quickly drain the synaptic resources needed to reproduce the spikes in the equivalent region of L2, and eventually activity in L2 will reduce or cease altogether. Fig. A.3B depicts the state of the network after multiple presentations of a standard ‘A’. The synapses that project to layer L2 from the units that encode ‘A’ in layer L1 are depressed, and consequently there is no longer any activity in layer L3.

The implications for the transmission of oddball sequences is immediately clear: the average response to standard tones will be diminished due to the effects of synaptic depression, whilst the average response to deviants (which are associated with synapses that
have had time to recover) will be relatively higher. For instance, Fig. A.3C illustrates the response of the network to a deviant input ‘B’ after many standard ‘A’s’. Resources are still available at the synapses that only code for ‘B’ (not ‘A’), which provides a route for a signal to propagate from L1 to L3. The fact that regions of activation overlap more for smaller $\Delta f$ implies that larger $\Delta f$ will be associated with greater deviant responses, satisfying the $\Delta f$ requirement stated in Section A.1.1.

The instantaneous state of the entire array of synapses may be regarded as a form of memory that encodes the statistics of recent stimuli. It is evident that the time course of synaptic recovery, $\tau_r$, must be on the same order of magnitude as the stimulus rate: if $\tau_r$ is too short, then the memory recovers too quickly to extend over the time period needed to distinguish standards from deviants; if $\tau_r$ is too long, then the model is unresponsive, as even the mean time between deviants is too short to permit recovery, with the result that the whole synaptic array remains in a depleted state. As it happens, the recovery time constants employed in models of synaptic dynamics are on a similar order of magnitude (e.g., 450ms, 800ms [196]) to the tone rates used in SSA experiments (125–1000ms). This relationship may be viewed as fortuitous, or as supporting the hypothesis of a direct link between synaptic dynamics and SSA.

The synapses that connect L2 to L3 are excitatory, do not depress, and exhibit broad, many-to-one (‘constructive’ [137]) convergence. The purpose of units in the final layer is to integrate activity in L2. The width and shape of a receptive field of a unit in L3 is determined by how broadly it receives inputs from L2 and how those inputs are weighted. The units in L2 adapt, but only for the units in L3 is this adaptation stimulus-specific, in the sense that they respond preferentially to deviant stimuli. SSA models based on convergent depressing synapses are examined from a theoretical perspective in [135].

### A.3 Hardware Implementation

The results presented in this chapter were obtained using the hardware setup described in section 4.1. While the setup consists of three chips, only two of them were used for the experiments carried out in this chapter. We implemented a network based on the model described in Section A.2. The implementation of various layers of the network is described below.

**Input Layer** L1 comprises 240 Poisson input units. The spike trains from these units are generated in software on a PC workstation and fed to synapses on the chip by means of the AER protocol. The population is divided into 15 groups of 16, each of which is tuned to respond to a particular range of tone frequencies, as measured on an octave scale. The firing rate $r_i$ of neurons in group $i \in \{0, \ldots, 14\}$ when a tone at frequency $f_c$ (octaves) is
presented is given by
\[ r_i = r_{\text{max}} \exp \left( \frac{(f_i - f_c)^2}{-2\sigma^2} \right), \tag{A.2} \]
\[ f_i \equiv ai - b, \]
where \( \sigma = 0.2124 \) and \( r_{\text{max}} = 100 \). Group \( i \) fires at the maximum firing rate \( r_{\text{max}} \) when the tone is presented at frequency \( f_i \) (octaves). The parameters \( a \) and \( b \) control the frequency range spanned by the input neurons. In the oddball experiments, \( a = \frac{1}{8}, b = \frac{7}{8} \), so that the inputs span a range of 1.75 octaves, with the centre of the input range nominally tuned to zero octaves. In experiments where presentation mode was varied, \( a = \frac{3}{28}, b = 0 \), so that the inputs span a range of 1.5 octaves and the lowest edge of the input range is tuned to zero octaves. The parameter \( \sigma \) controls the bandwidth of the tuning curves: \( \sigma = 0.2124 \) implies a bandwidth (measured between the points where \( r_i = \frac{1}{2} r_{\text{max}} \)) of 0.5 octaves.

**Adaptation Layer**  
L2 comprises 15 linear constant-leak, integrate-and-fire neurons on Chip 1 (ISSL-WTA). Each neuron \( i \in \{0, \ldots, 14\} \) receives 16 depressing synapses—one from each member of group \( i \) in L1.

**Integration Layer**  
L3 is implemented on Chip 2 (IF2D-WTA) as 15 linear, constant-leak, integrate-and-fire neurons. Each neuron in L3 receives 10 inputs, via non-depressing synapses, from neurons chosen uniformly at random from those in layer L2. The fairly dense random connectivity ensures that:

1. Each neuron typically responds over the entire frequency range.
2. The neurons in layer L3 vary in their response to the same stimulus, due to their respective patterns of connectivity.

This is supported by the fact that a population does not respond uniformly to an oddball sequence; rather, the SI values collected across neurons follow a distribution (e.g., Fig. 2c in [198]). Furthermore, the frequency response area of a neuron, though confined to some range, is often somewhat uneven within this range (see Fig. 1c in [198]; Fig. 1B,C in [124]).

**A.3.1 Input**

**Oddball Experiments**

Oddball sequences were presented to the hardware and SI values were recorded in every unit of L3. The SI values were computed from two oddball sequences consisting of 400 ‘tones’ (i.e., input spike sequences): with A and B being deviant in the first and second sequence, respectively. In all conditions, the tone duration was 200 ms.
In the first set of experiments, $p_{dev}$ and $\Delta f$ were varied in six combinations, which are listed in the two left-hand columns of Table A.1. The tones were presented at an interonset interval of $\Delta t = 1$ second. All these parameters were set to match those given in an experimental study [18]. Each condition was repeated five times.

In a second set of experiments, $\Delta t$ was varied, assuming the values $0.125$, $0.25$, $0.5$ and $1$ seconds, which are standard for physiological experiments [198, 7, 124, 159]. The probability of a deviant was fixed at $p_{dev} = 0.1$, and the frequency separation was fixed at $\Delta f = 0.5$ octaves. In these experiments the tone duration was 75 ms. Each condition was presented a single time.

### Presentation Mode Experiments

The stimulus presentation mode experiments used ten tone frequencies, which were spaced evenly over the 1.5-octave input range. For each mode—block, sequential and random—each tone frequency was presented ten times (see Section A.1.2). Tones were presented with a duration of 75 ms and at a rate of 4 Hz ($\Delta t = 0.25$ seconds), consistent with [159]. Output spikes were measured in layer L3.

Following [159], block, sequential and random sequences were presented over a range of “intensities”. In the physiological experiments, intensity corresponded to the physical sound level (in dB) of a tone played via a loudspeaker. This raised the question of how sound intensity should be encoded in the input to the hardware. Noting that, on a simplified view, nonlinear mechanisms in the cochlea establish a linear relationship between firing rate in auditory nerve fibres and sound level (at least, within a certain operational range) [113, 211], I scale the overall input rate to the model ($r_{max}$) to represent input intensity level.

Neurons that did not exhibit SSA but were otherwise similar to their adapting counterparts were modelled in hardware by disabling the depression between layers L1 and L2, and maintaining the same configuration in all other respects.

### Table A.1: SI values from physiological data [18] and hardware model. For a description of each column see Section A.4.1.

<table>
<thead>
<tr>
<th>$p_{dev}$</th>
<th>$\Delta f$</th>
<th>$SI_{hw}$</th>
<th>$# p &lt; 0.05$</th>
<th>$SI_{bio}$</th>
<th>$SI_{bio}/SI_{hw}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>0.0239</td>
<td>15</td>
<td>0.089</td>
<td>3.7238</td>
</tr>
<tr>
<td>0.3</td>
<td>0.5</td>
<td>0.0112</td>
<td>12</td>
<td>0.041</td>
<td>3.6607</td>
</tr>
<tr>
<td>0.1</td>
<td>0.25</td>
<td>0.0115</td>
<td>11</td>
<td>0.055</td>
<td>4.7826</td>
</tr>
<tr>
<td>0.3</td>
<td>0.25</td>
<td>0.0058</td>
<td>10</td>
<td>0.019</td>
<td>3.2759</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>−0.0007</td>
<td>1</td>
<td>−0.001</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>−0.0006</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.4 Results

A.4.1 Oddball Experiments

Varying $p_{\text{dev}}$ and $\Delta f$

The mean SI values measured in L3, taken over all units and all trials for the various conditions, are tabulated in Table A.1 under the heading $\text{SI}_{\text{hw}}$. To the right of each SI value, under the heading $#p < 0.05$, is written the number of neurons in L3 (of 15) for which SI was significantly greater than zero (five trials; one-sided Wilcoxon sign test [130], $p < 0.05$). For comparison, under the heading $\text{SI}_{\text{bio}}$ are listed median SI values based on the responses of a population of neurons in rat auditory cortex to the same stimuli [18]. Whilst the SI values are not the same, they do preserve the same ordering—in keeping with the principles set out in Section A.1.1 (i.e., rank correlation coefficient [130], $\rho = 1$). Furthermore, the two columns of figures are approximately directly proportional, as the ratios in the final column demonstrate (Pearson product moment correlation coefficient [130], $r = 0.9734$).

The SI value was significant for the conditions in which there was a true deviant (i.e., $\Delta f > 0$; $p_{\text{dev}} \neq 0.5$) for most of the L3 units. In the conditions where there was no deviant, the SI value was not significantly greater than zero (except for one measurement, presumably attributable to chance). Histograms of the SI values collected over all L3 units and all five trials are provided in Fig. A.4.

Varying $\Delta t$

In the second set of experiments, the time between onsets ($\Delta t$) was manipulated, and $p_{\text{dev}}$ and $\Delta f$ were constants. The mean SI values measured in layer L3 in response to each sequence are plotted in Fig. A.6 as markers. The error bars show the sample standard error of the mean (s.e.m.) derived from the responses of the 15 neurons. Note that these responses are correlated, so the s.e.m. is an underestimate. The SI value monotonically decreases with $\Delta t$, which is consistent with the majority of SSA studies (see discussion in Section A.1.1).

A.4.2 Presentation Mode Experiments

In the stimulus presentation mode experiments, frequencies were presented at a particular intensity (i.e., value of $r_{\text{max}}$) in blocks, in sequences and at random. (See sections A.1.2 and A.3.1) Each curve in Fig. A.7A plots the mean responses of SSA neurons in layer L3 to a particular presentation mode. The block, sequential and random configurations elicit the least, middle and greatest mean responses, respectively. This ordering is apparent at all intensities and is consistent with the physiological data [159], where the curves have a similar appearance (see Fig. A.8).
Fig. A.4: Histograms of SI values measured in L3. Each is constructed from 75 data points (15 neurons × 5 trials). The black markers on the abscissae indicate the grand mean SI. A normal distribution is superimposed on the histograms for conditions in which the SI significantly exceeds zero. These measurements are comparable to physiological findings [18] (See Fig. A.5).
Fig. A.5: Neuronal adaptation in the awake rat A1 (adapted from Fig. 2 of [18]). Distribution of SIs for different stimulus conditions (indicated on the right column; $p_{dev}/\delta f$) for single units (gray bars) and multiunits (white bars), with the last one (0.5/0.5) being the control condition. Median SIs are indicated as dashed lines with the actual value printed next to it ($** p < 0.01$).
Fig. A.6: Mean SI value in layer L3 as a function of the time between tone onsets, $\Delta t$. The error bars denote the standard error of the mean. For an analogous figure, showing that SI decreases with $\Delta t$, see Fig. 4a in [198].

Fig. A.7: Mean firing rate in layer L3 as a function of stimulus intensity. The curves/marker types correspond to presentation modes (see legend above). Error bars relate s.e.m. with respect to the 15-neuron population. A. SSA neurons respond most vigorously to frequencies presented in a random order and least vigorously to frequencies presented in blocks. B. Non-adapting neurons do not respond preferentially to any sequence configuration. (The markers corresponding to the three configurations coincide.) Note that the scale of the ordinate in (B) differs to that of (A). Compare to Fig. A.8 (Fig. 4 in [159]).
Fig. A.8: Effect of stimulus presentation mode on rate-level functions and frequency response areas (taken from [159]). Rate-level functions of a novelty neuron (A) and an IC neuron that did not exhibit SSA (non-habituating) (B) measured using three different stimulus presentation paradigms. In both plots, red lines indicate randomized stimulus presentation; green lines indicate sequential presentation; blue lines indicate block presentation.

The experiment was also repeated for non-adapting neurons (Fig. A.7B). Under these circumstances, the synapses between layers L1 and L2 do not depress, with the consequence that the mean firing rates in L2 and L3 are an order of magnitude higher in all presentation modes. Moreover, unlike SSA neurons, mean firing rates are not affected by the order in which the tones are presented. Both of these observations are true of biological neurons [159]: the mean firing rates of SSA and non-SSA neurons in inferior colliculus differ by a similar factor, and the latter show no sensitivity to presentation mode (see Fig. A.8).

A.5 Discussion and Conclusion

The short term synaptic depression on the hardware is similar to that observed in the biological nervous system [125, 199]. Markram and Tsodyks proposed a mathematical description of this plasticity rule [196]. It was used to model SSA in a feed-forward neural network. Experimental measurements on a hardware implementation of the network qualitatively reproduce biological measurements in [18, 159]. It was shown that increasing the separation between the inputs ($\Delta f$) and decreasing the probability of a deviant both increased the SI. Increasing the time between the onsets of inputs ($\Delta t$) led to a decrease in SI. A comparison of the SI values obtained in hardware with those obtained in rat cortex [18] revealed that the two were directly proportional, suggesting that it was only a difference in overall sensitivity to deviants which distinguished the two.

The results demonstrate that a feed-forward network of neurons equipped with short-term depression can show enhanced response to rare events within a stream of repetitive stimulus. The non-linearity induced by synaptic depression enables the neurons to do so. It should be noted that the effect of this non-linearity extends for a few hundred milliseconds which enables the neurons to adapt to stimuli even at a presentation frequency less than
A second set of experiments measured how the mean spike rate in response to a tone depended on the tones that immediately preceded it. The likelihood of certain stimulus histories prior to a tone was controlled indirectly by varying the stimulus presentation mode. These modes were: block (likely preceded by the same frequency), sequential (likely preceded by a neighbouring frequency) and random (unlikely to be preceded by similar frequencies). The overall spike count was greatest when the tones were ordered randomly, and least when the tones were presented in blocks. This matches the findings of a similar experiment carried out in the rat [159].

These measurements show that synaptic depression leads to a high pass filtering mechanism in the incoming stimulus. They demonstrate that synaptic depression can be a very effective mechanism in enhancing novelty in the input stimulus and can be used to model SSA.

Buonomano [29] showed that such synaptic depression can play a crucial role in temporal information processing. He showed that the activity of a network of neurons equipped with short term depression can be used to differences between input patterns that have the same spatial properties but different temporal properties. This is a direct consequence of the nonlinear properties induced into the neuronal dynamics because of synaptic depression. Synaptic depression also provides an additional mechanism that operates on a longer time scale than the typical synaptic time constant, thereby allowing temporal information processing over longer time scales.

We have seen in the previous chapter that the STC model does not cope well with high firing rates in the input spike patterns. Synaptic depression can highlight the novel features of the stimuli and might lead to enhancement of the quality of features learnt by $B_2$ neurons. Such synaptic depression can be introduced into all synaptic projections in the network. The projection that will benefit the most will be $B_1 \rightarrow C \rightarrow B_2$ since that is the projection that acts as memory element. So, the sparser the information is that needs to be retained within the network, the more efficient and cleaner the projection can maintain this information. This possibility has not been explored and is left as a hypothesis for future work.

We have also seen in the previous chapters that neuromorphic hardware has inherent parameter mismatch and therefore heterogeneity. Together these properties (nonlinearity, heterogeneity, long time constants) can be used to construct Definite Memory Machines (DMMs) a special case of FSMs [145]. The current output of a DMM depends only on a finite number of previous inputs in addition to the current input. In [145] the authors show that any arbitrary DMM that relies on a small number of previous states can be realized by training a feed forward network of neurons with heterogeneous dynamic synapses. The hardware used in this work with its dynamic synapses and inherent heterogeneity could be used to train DMMs. Such DMMs could act as filters on the incoming spikes, for
example, to enhance relevant information in the spike trains. Such a mechanism could be used to suppress redundant information and enhance transients in the input stimuli. It could be incorporated into the STC model described in Section 3.3 to enhance the quality of spatio-temporal features presented to the network. Since the STC model relies on the spatio-temporal correlations of the target stimuli, sparsification of input stimuli could result in a clear set of spatio-temporal correlations, that are characteristic of the target stimuli.
Appendix B

Hardware-Software ecosystem

The work described in this thesis involves use of a complex neuromorphic multi-chip system and custom software infrastructure for its configuration. This chapter lists out the experimental details related to the use of this hardware-software ecosystem.

B.1 Setting up the neuromorphic hardware

In order to use the neuromorphic chips described in this thesis, we need two PCBs (per chip), that allow AER communication (AEX board) and setting of bias voltages (AMDA board). In addition, if we wish to implement neural networks with complex mappings, that go beyond a single layer feed forward network, we will require a mapper.

In order to find all these components at INI please refer to the resources page on the NCS website http://ncs.ethz.ch/internal/resources, where the location and availability of all these hardware resources is maintained.

B.1.1 AMDA board

The AMDA boards are assigned unique IDs that are used to identify them. Once you have acquired an AMDA board, update the resources page with the corresponding ID and location.

In case you are unable to find a board and want to build your own PCB the following resources should be useful for you.

- PCB design: https://svn.ini.uzh.ch/repos/hw/pcbs/AMDA/AMDA_002/
- Atmel firmware source: https://svn.ini.uzh.ch/repos/hw/pcbs/AMDA/Software/Atmel/Firmware3/
- Software documentation: https://svn.ini.uzh.ch/repos/hw/pcbs/AMDA/Software/Docs/
From the above mentioned resources you should be able to acquire an **AMDA** board. This board can then be used to bias neuromorphic chip mounted on it. The biases can be configured by interfacing the **AMDA** board to a Linux PC. You will need to install the following software and drivers on this PC.

- **AMDA** board driver: [https://svn.ini.uzh.ch/repos/hw/pcbs/AMDA/Software/udevamda/](https://svn.ini.uzh.ch/repos/hw/pcbs/AMDA/Software/udevamda/)
- **pyAMDA** python interface: [https://svn.ini.uzh.ch/repos/ncs/ini/code/python/pyAMDA/](https://svn.ini.uzh.ch/repos/ncs/ini/code/python/pyAMDA/)

Once you have all the necessary software installed, you can mount your neuromorphic chip (with its daughter board) onto the **AMDA** board, and connect to the PC through the USB cable. You will find a device file associated to the **AMDA** board /dev/udevamdaXXX where XXX is the board ID. This is the device file that is used by the software to communicate with the **AMDA** board. See Listing B.1 for a demonstration of **pyAMDA** being used to set and read biases from an **AMDA** board.

```
from pyAMDA import AmdaBoard

# board: **AMDA** board ID
# host: hostname of the PC to which the **AMDA** board is connected
ab = AmdaBoard(board='201', host='localhost')

# Set bias on channel 20 to a value of 0.5 using a DAC
ab.setchannel(channel=20, norm=0.5, typ='DAC')

# Get value of DAC on channel 20
bv = ab.getchannel(channel=20, typ='DAC')
```

**Listing B.1:** A code snippet showing how one can interact with the **AMDA** board using **pyAMDA**. In the example code shown here, a bias value is set on a DAC and its value is read out.

In order to set specific biases on a chip mounted on an **AMDA** board, you need to know which channel corresponds to which bias. This information can be found in the documentation of the chip’s daughter board that was mounted on the **AMDA** board. Typically this information is translated into an XML based description of the chip called the NHML description. This description of chips from Institute of Neuroinformatics (INI) can be found at [https://github.com/inincs/NHML/tree/master/INI](https://github.com/inincs/NHML/tree/master/INI).

In case you run into problems, the following page might provide useful information about some of the bugs and quirks related to the hardware and software [http://ncs.ethz.ch/internal/experiments-with-ncs-vlsi-chips](http://ncs.ethz.ch/internal/experiments-with-ncs-vlsi-chips)
B.1.2 AEX board

You should also acquire an AEX board in case you want to have AER communication for your neuromorphic setup. Refer to the resources page to locate an available board. In case you are unable to locate an unused board and decide to build a new board from scratch the below resources will be helpful.

- PCB design: [https://svn.ini.uzh.ch/repos/ncs/ini/code/AEX/PCB/](https://svn.ini.uzh.ch/repos/ncs/ini/code/AEX/PCB/)
- Instructions to program AEX board: [http://ncs.ethz.ch/internal/experiments-with-ncs-vlsi-chips/introduction#section-9](http://ncs.ethz.ch/internal/experiments-with-ncs-vlsi-chips/introduction#section-9)

After programming the firmware on the AEX board or acquiring a pre-installed AEX board you can connect it to a PC over a USB interface. It is preferable to use the same PC for connecting the AMDA and AEX boards. In order to send and receive spikes through this AEX board you will need to install the following software on this PC.

- pyAEX python interface: [https://svn.ini.uzh.ch/repos/ncs/ini/code/python/pyAex/](https://svn.ini.uzh.ch/repos/ncs/ini/code/python/pyAex/)

Once the drivers are installed, on connecting the AEX boards to the PC, you should find a device file /dev/aerfx2_0. This file then allows us to send and receive AEs from the AEX board. On pairing an AEX board with an AMDA board through the Rx/Tx parallel ports, you will be able to interact with a neuromorphic chip through AEs. See List B.2 for a demonstration of how the program xio can be used to send and receive AEs to the AEX board. See Listing B.2 for a working example.

```
$ ./stim/xgen.py  35678  -1 | ./x86_64/xio-bin-dx MONSEQ 1222333 /dev/aerfx2_0
```

**Listing B.2:** A code snippet demonstrating the use of xio to send and receive spikes from an AEX board. The program xgen.py generates 35678 AEs distributed over 3 seconds with incrementing addresses for AEs. (-1 denotes no upper limit for the address.) These AEs are used as inputs by the xio-bin-dx program to send spikes to AEX board /dev/aerfx2_0 and monitor events for 1.22333 seconds after all the input AEs have been sequenced.
If you are using multiple AEX board in your neuromorphic system, they can all be serially connected in a loop over the serial ports through SATA connectors. Atleast one of these AEX board needs to be connected to the PC, which will then allows us to infuse AEs into the loop from an external source.

## B.1.3 AER Mapper

There are a limited number of AER mappers with the NCS group. You should find a list of available mappers also on the ncs resources page ([http://ncs.ethz.ch/internal/resources](http://ncs.ethz.ch/internal/resources)) under ExCol PCI Mappers section. As always you can always build one yourself from scratch from the following resources.

- Acquire a PC different from the one you used to connect AMDA and AEX boards. This PC’s mother board requires a PCI slot to plug in the mapper.

Once you have a mapper, you can introduce it into your neuromorphic system by inserting it in the AER serial loop using the SATA connectors. This mapper takes pairs of input/output addresses corresponding to the desired AER mapping scheme. Such mapping can be installed onto the mapper by logging into the PC hosting the mapper and running the mappinglib. See Listing B.3 for an example of how to use the mapper in conjunction with mappinglib.

```
# Login to the mapper machine
$ ssh root@mapper

# Go to the mappinglib directory
$ cd mappinglib

# Clear the mapping table
$ ./clearallmappings

# Set a mapping from a file
$ ./setmapping mappingtable.txt

# Write current mapping table
$ ./getallmappings > mappingtable.txt
```

**Listing B.3:** A basic working example of mappinglib to set and read mappings on the AER mapper board.
B.2 Software for configuring neuromorphic hardware: pyNCS

**Software installation:** The experiments conducted and described here in this thesis involved use of a python library developed in the group called pyNCS [https://github.com/inincs/pyNCS](https://github.com/inincs/pyNCS). Install the software and drivers as described in the instruction pages on pyNCS webpage [http://inincs.github.io/pyNCS/general/installation.html](http://inincs.github.io/pyNCS/general/installation.html). This software package employs all the software you have installed above as drivers in order to communicate with the chips and serves as a single point of contact with your neuromorphic system.

**Setup files:** In order to initialize your new neuromorphic setup in pyNCS, you require two XML files that describe your setup, its components and their corresponding drivers. A detailed explanation of how these files should be written for your own setup can be found at [http://inincs.github.io/pyNCS/general/installation.html#setup-files](http://inincs.github.io/pyNCS/general/installation.html#setup-files).

**NHML Chip description files:** In addition to the setup files you will also require Neuromorphic Hardware Markup Language (NHML) description of your neuromorphic chips [http://inincs.github.io/NHML/](http://inincs.github.io/NHML/). You will find NHML files for some of the existent neuromorphic chips hosted on this git repository [https://github.com/inincs/NHML](https://github.com/inincs/NHML). If you just developed a new neuromorphic chip, you can write a new NHML file for your new chip by modifying the NHML description of an existent chip description.

**Step-by-step tutorial:** Following this you are technically equipped to use your neuromorphic system using pyNCS. The documentation of pyNCS includes a step-by-step tutorial [http://inincs.github.io/pyNCS/general/tutorial.html#step-by-step-tutorial](http://inincs.github.io/pyNCS/general/tutorial.html#step-by-step-tutorial) that you can follow to get started with using the library. The tutorial employs a basic example of how to build a neural network with a population of 50 neurons that are excited by spikes form a vision sensor (DVS). It shows how such a network can be initialized, stimulated and its activity monitored. See Listing B.4 for a quick glance of pyNCS functionalities.

```python
import pyNCS

# SETUP DEFINITION
setup = pyNCS.NeuroSetup('setuptype.xml','setupfile.xml')

# POPULATION DEFINITION
recurrent_neurons = pyNCS.Population('','')
recurrent_neurons.populate_by_number(setup, chipid='ifs1wta', neurontype='excitatory', n=100)
input_neurons = pyNCS.Population('','')
input_neurons.populate_all(setup, chipid='dvs', neurontype='pixelon')

# CONFIGURATION
```
Listing B.4: A code snippet showing how one can interact with the hardware with pyNCS. In the example code shown here, two populations are instantiated on two different chips (IFSL-WTA and the DVS), connections created between those populations, bias parameters set and finally the activity of the network monitored and plotted.

**STC model:** The python code, used to implement the STC model and perform experiments in Sec. 5.1 can be found at [https://svn.ini.uzh.ch/repos/ncs/SCANDLE/experiments/AuditoryModel/](https://svn.ini.uzh.ch/repos/ncs/SCANDLE/experiments/AuditoryModel/) This code can serve as a rather complex template of pyNCS coding, if you want to do a complicated experiment with a multi-chip neuromorphic hardware system.

**B.3 Learning on IFSLWTA**

One of the crucial aspect of this thesis is the spike based online learning. This learning was realized using the learning rule implemented on the IFSLWTA chip.

The learning rule implemented on the IFSLWTA is described in section 4.1.1.3. Its parameters, used for the experimental results shown in Sec. 5.1 were chosen such that a quick succession of stimulus presentations at a rate of $10 \text{ Hz}$ can induce a synaptic potentiation within the first two or three stimulus presentations. The actual bias parameters used for the plastic synapses for the experiments described in this thesis are listed in table B.1.
<table>
<thead>
<tr>
<th>Bias name</th>
<th>V (volts)</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>synaerhi</td>
<td>2.0</td>
<td>$X_{hi}$</td>
</tr>
<tr>
<td>synaerlo</td>
<td>0.1</td>
<td>$X_{lo}$</td>
</tr>
<tr>
<td>psynaerlk</td>
<td>2.99</td>
<td>$\tau$</td>
</tr>
<tr>
<td>synaerth</td>
<td>0.6</td>
<td>$\theta$</td>
</tr>
<tr>
<td>nwtA</td>
<td>0.5</td>
<td>$a$</td>
</tr>
<tr>
<td>psynaertau</td>
<td>2.84</td>
<td>$\tau_{syn}$</td>
</tr>
</tbody>
</table>

Table B.1: Biases for learning synapses for IFSLWTA chip.

<table>
<thead>
<tr>
<th>Bias name</th>
<th>V (volts)</th>
<th>Thresholds</th>
</tr>
</thead>
<tbody>
<tr>
<td>thk1</td>
<td>3.3</td>
<td>Both LTP and LTD</td>
</tr>
<tr>
<td>thk2</td>
<td>2.2</td>
<td>Only LTP</td>
</tr>
<tr>
<td>thk3</td>
<td>3.3</td>
<td>None</td>
</tr>
</tbody>
</table>

Table B.2: Stop learning parameters

**Calcium variable bug** The weight of the synapse is maintained over a capacitor and is designed to drift to one of the two stable states $X_{hi}$ and $X_{lo}$. The potentiation and depression of synapses is gated by a calcium variable $C(t)$ that represents the firing rate of the soma. Due to a quirk in this circuit, the voltage representing this variable drifts away from its resting state when the soma ceases to produce spikes for a few seconds. This intern disables the learning of plastic synapses. In order to reset this variable and enable learning in plastic synapses, on the IFSLWTA chip, it is necessary to make the neurons fire shortly before the experiment is carried out. This was done by injecting current to the soma just before the ES is presented to the network.

**Stop learning** The original learning rule and therefore the circuits also include a *stop learning* mechanism [24]. This mechanism enables synapses to cease from learning if the neuron’s firing rate is too high or too low. This is a very useful mechanism when operating in rate mode since it helps modulate the learning process and hence prevents neurons from operating at unnatural firing rates. But since the work presented in this thesis is not targeted towards firing rate based learning, the stop learning mechanism was disabled. See table B.2 for the bias values used in order to enable learning without any stop learning constraints.

A handy script that one can use in order to work with the IFSLWTA chip was developed by me and Fabio and can be found at [https://svn.ini.uzh.ch/repos/ncs/SCANDLE/experiments/AuditoryModel/getwij.py](https://svn.ini.uzh.ch/repos/ncs/SCANDLE/experiments/AuditoryModel/getwij.py) By initializing the WeightMartix one can use several of its handy methods for enabling and disabling learning and to read and write the synaptic states. All other scripts related to the hardware experiments with the STC model can also be found in the same folder.
Bibliography


Acronyms

AE  Address Event
AER  Address Event Representation
API  Application Programming Interface
DAC  Digital to Analog Converter
DMM  Definite Memory Machine
DPI  Differential-Pair Integrator
DVS  Dynamic Vision Sensor
EPSC  Excitatory Post-Synaptic Current
EPSP  Excitatory Post-Synaptic Potential
ES  exposure stimulus
FM  Frequency Modulated
FSM  Finite State Machine
GUI  Graphical User Interface
IF  Integrate and Fire
INI  Institute of Neuroinformatics
IPSC  Inhibitory Post-Synaptic Current
ISI  Inter-Spike Interval
LIF  Leaky Integrate and Fire
LSM  Liquid State Machine
LTD  Long Term Depression
LTP  Long Term Potentiation
NEF  Neural Engineering Framework
NHML Neuromorphic Hardware Markup Language
PCB Printed Circuit Board
PS  probe stimulus
ReSuMe  Remote Supervised Method
SC  superior colliculus
SEM  Spike-based Expectation Maximization
SNN  Spiking Neural Network
SRM  Spike Response Model
SSA  Stimulus Specific Adaptation
STC  Spatio Temporal Correlator
STDP  Spike Timing Dependent Plasticity
STRF  Spectro-Temporal Receptive Field
VLSI  Very Large Scale Integration
VRD  van Rossum distance
WTA  Winner-take-all
Curriculum Vitae

Sadique Sheik

Institute of Neuroinformatics,
Winterthurerstrasse 190,
8049 Zurich
+41 (0)44 635 3044
sadique@ini.ethz.ch
sheiksadique.github.com

Date of birth: 9th May 1985
Nationality: Indian

Academic Experience

PhD in Physics and Neuroinformatics <2009--2013>
Institute of Neuroinformatics, ETH Zurich
Supervised by Rodney J. Douglas, Giacomo Indiveri, Elisabetta Chicca,

Indian Institute of Technology - Kharagpur

Professional Experience

IBM India Pvt. Ltd. <2007-2009>
Associate System Engineer - SAP Basis Consultant

Projects

PhD

- SCANDLE European FP7 project, Biologically inspired computational modeling and implementation on custom neuromorphic chips for auditory processing.
  - Stimulus Specific Adaptation
  - Bistable and multi-stable perception
  - Auditory Thalamo-cortical model.
- Spike based Learning and classification of temporal sequences.
  - Liquid state machines.
  - STDP, Tempotron like learning rules.
- Developing software tools for interfacing neuromorphic multi-neuron chip setups.
- Automatic tuning and configuration of neuromorphic chips.

Undergraduate

- Microscopic Polarization Imaging: Imaging of Biological cells and Tissues (Semester project; Advanced Experiments in Physics with Prof. Partha Roy Chaudhuri, IIT Kharagpur)
- Nonlinear Dynamics and Boundary Conditions of Excited Double Pendulum (Summer project with Prof. Krishna Kumar, IIT Kharagpur)
- Design and Development of Web Based Text Editor (SRIC sponsored project at CET, IIT Kharagpur)
IBM India Pvt. Ltd

SAP Basis Consultancy deals with installation and maintenance of software that integrates all the jobs and data of an organization, in order to get the optimum efficiency. It also deals with system security. Some of my key contributions are listed below.

- Implementation of SAP R/3 4.7 Data Recovery Server.
- On-site SAP Basis support and maintenance for a major petro-chemical firm (HPL)

Technical Skills

Software
- Programming languages: C/C++, Java, Matlab, Python.
- Web design: HTML, JavaScript, XML, XSLT, JSP.
- GUI design: Java Swing, Python GTK, Enthought Traits+Chaco.
- SAP Basis and Security.

Hardware
- Surface mount design (SMD).
- PCB design – Eagle

Workshops

- Making sense of sounds, Plymouth, UK; February 2012.
- Learning and plasticity workshop, Marseille, France; November 2011.
- Telluride Neuromorphic Engineering Workshop 2012, Telluride, CO, USA; July 2012.
- SCANDLE workshop on modeling hearing and listening, Delmenhorst, Germany; May 2010.
- Probabilistic Inference for Motor Control, AMARSI workshop by Technische Universitat Graz. in Zurich; January, 2011.
- Introduction to Reservoir Computing, AMARSI workshop by Reservoir Lab, Ghent University, in Winterthur; January, 2011.
- Workshop on Ornithopters (by Mr. Nathan Chronister), Khhitij 2007, IIT Kharagpur.

Publications

- Exploiting Device Mismatch in Neuromorphic VLSI Systems to Implement Axonal Delays. (Sadique Sheik, Elisabetta Chicca, Giacomo Indiveri) International Joint Conference on Neural Networks (IJCNN), 2012.
If a *detail* is what makes or breaks your system, it’s probably not *just a detail*!