Bayesian neuromorphic computing based on resistive memory

Thomas Dalgaty

Thomas DALGATY

Thèse dirigée par **Barbara DE SALVO** , Directeur de Recherche
et codirigée par **Elisa VIANELLO**
et **Jérôme CASAS**, Enseignant-Chercheur, Université de Tours

préparée au sein du **Laboratoire CEA/LETI**
dans **l'École Doctorale Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)**

Calcul neuromorphique Bayésien basé sur la mémoire résistive

Bayesian neuromorphic computing based on resistive memory

Thèse soutenue publiquement le **27 novembre 2020**,
devant le jury composé de :

Abstract

Artificial intelligence is a field that, historically, has benefited from the combination of ideas from across inter-disciplinary boundaries which have improved models of AI and the algorithms that operate on them. Biological nervous systems in particular have inspired various model topologies and algorithmic tricks that have led to leaps in performance. In contrast, as computing power and memory availability have increased relentlessly since the 1950's, models of artificial intelligence have largely failed to recognise the constraints imposed by, or incorporate the opportunities offered by, the underlying computing hardware. While this is not immediately apparent in the cloud computing setting; the mismatch between model, algorithm and hardware is the limiting factor that currently curtails the efficient application of locally-adaptive artificially intelligent systems at the edge. In this thesis, the interdisciplinary boundary between machine learning, emerging technologies and biological nervous systems will be explored with the objective of proposing a new, hardware-focussed, approach for the application of energy efficient and locally-adaptive edge neuromorphic computing systems. Resistive memories are a leading candidate as an enabling technology for AI to greatly reduce its energy requirements. This is largely owed to the efficient and parallelised implementation of the dot-product operation that pervades machine learning as well as its material-level compatibility with advanced CMOS processes. However, until now, the application of RRAM has been confined predominantly to implementations of gradient-based machine learning algorithms, namely backpropagation, to train RRAM-based multi-layer perceptron models. The fundamental properties of RRAM though, predominantly their conductance variability, are, on the contrary, not compatible with learning algorithms based on the descent of error gradients. This thesis recognises that, in contrast, the intrinsic properties of this technology can be harnessed through Bayesian approaches to machine learning where, like device conductance states, model parameters are described as random variables. RRAM-based implementations of Markov Chain Monte Carlo sampling algorithms are implemented and applied to the training of RRAM-based models. An RRAM-based computing hardware capable of supporting such models is also proposed. Inspired by the organisational principles of animal nervous systems, whereby memory and processing are distributed and arguably indistinguishable, this thesis proposes analogue circuit solutions for biological models of neurons and synapses and for a system-level architecture to interconnect such elements. Reflecting the role played by ion-channels embedded in biological neuronal membranes, these circuits co-localise memory and computation by incorporating resistive memory devices directly into the circuits themselves; determining model parameters and the interconnectivity between these elements locally. Relative to similar approaches, this obviates the need for volatile on-chip working memory and the use of analogue-to-digital conversion, both entailing significant energy demands. In recognition of the efficient solutions animals like insects have uncovered throughout the course of evolution, their nervous systems are used to guide the development of model architectures. Based on recent neurophysiological studies, models inspired by the cricket cercal system and the fruit fly motion detection system are proposed. To achieve an equivalent performance to the cercal system model, multi-layer perceptrons require between one and two orders of magnitude more memory elements; offering a means of scaling the proposed MCMC sampling algorithms to more complex tasks. It is also discussed how the 'small-worldness' of networks of neurons found in animal nervous systems can provide a solution to the spatial connectivity constraints inherent to the proposed RRAM-based computing fabric.

Key words: Bayesian, machine learning, resistive memory, neural networks, electronic circuits, bio-inspiration.

Résumé

L'intelligence artificielle (IA) est un domaine qui, historiquement, a grandement bénéficié de l'association d'idées interdisciplinaires, pour améliorer ses modèles ainsi que ses algorithmes. Les systèmes nerveux biologique ont ainsi été une source d'inspiration de valeur pour les topologies des modèles et des techniques algorithmiques, et ont été à l'origine de sauts en performances. Cependant, même si la puissance du calcul et la quantité de la mémoire ont augmenté sans cesse, les modèles de l'IA ont largement échoué à s'adapter aux opportunités et aux contraintes apportées par les technologies silicium. Ce constat est relativement masqué dans le cas du calcul dans le cloud, mais constitue le facteur limitant pour le calcul au niveau edge à faible puissance. Cette thèse explore la frontière interdisciplinaire entre l'apprentissage automatique, les technologies émergentes et les systèmes nerveux biologique, avec l'objectif de proposer une nouvelle approche de l'IA, focalisée sur le hardware, pour réaliser des systèmes calcul neuromorphiques à faible consommation et adaptifs localement au niveau edge. Les mémoires résistives jouent un rôle prometteur pour la réduction de la consommation d'énergie de l'IA. Elles permettent en effet de réaliser efficacement et de manière parallélisée l'opération de produit scalaire, extrêmement importante dans l'IA, et elles sont compatibles avec les technologies CMOS avancées. Jusqu'ici, l'utilisation de RRAM a été majoritairement confinée à des algorithmes basés sur les gradients, notamment la rétro-propagation, pour l'apprentissage de perceptrons multicouches. Cependant, les caractéristiques des RRAM, notamment la variabilité de leur conductance, ne sont pas compatibles avec les besoins de ces algorithmes d'apprentissage basé sur la descente de gradients. Ce travail de thèse propose à l'inverse d'utiliser ces caractéristiques fondamentales des RRAM comme un avantage pour la modélisation Bayésienne, où des paramètres des modèles sont, comme les états de conductance de RRAM, décrits par des variables aléatoires. Des implémentations ex-situ et in-situ de méthodes Monte Carlo par chaînes de Markov (MCMC) basées sur les RRAM sont proposées et appliquées à l'entrainement des modèles. De plus, un hardware de calcul basé sur les RRAM est aussi proposé, permettant de mettre en application ces modèles. Inspirées par les systèmes nerveux animaux, où les éléments du calcul et de la mémoire sont distribués et co-localisés, des solutions en circuit analogiques sont proposées pour modéliser des neurones et des synapses biologiques ainsi qu'une architecture au niveau du système pour les interconnecter. Reflétant le rôle joué par les canaux ioniques, noyé dans les membranes neuronales, les circuits incorporent la RRAM au sein des circuits et déterminent localement leurs paramètres et leur inter-connectivite. Contrairement aux autres approches, ce système ne nécessite ni mémoires vives statiques (SRAM), ni convertisseurs analogiques-numériques qui nécessitant une consommation énergétique élevée. Enfin, en reconnaissant les solutions efficaces qu'ils ont trouvé au cours de l'évolution, les systèmes nerveux des insectes, nous servent de guider le développement d'architectures de modèles d'IA. En s'appuyant sur les résultats récents d'études neurophysiologiques sur les insectes, deux modèles sont proposés : un inspiré par le système cercal du grillon et l'autre par le système de détection de mouvement de la mouche. Pour obtenir une performance égale au modèle cercal, nous observons que les modèles de perceptrons multicouches nécessitent entre une et deux fois plus des paramètres synaptiques – ce qui ouvre la possibilité d'appliquer le MCMC à base de RRAM à des tâches complexes. Les caractéristiques 'small-world' des réseaux des neurones biologiques sont également abordées : elles peuvent offrir une solution pour outrepasser les contraintes de connectivites spatiales liées à l'architecture proposée.

Mots clés: Bayésien, apprentissage automatique, mémoire résistive, réseaux de neurones, circuits électroniques, bio-inspiration.

Acknowledgements

This thesis was carried out in collaboration with various partner Universities and is composed of adapted versions of the papers and patents written with them in which I was the first author - those listed in the following page. Reflecting this, the thesis is written using ‘we’ instead of ‘I’ throughout. Specifically, I would like to thank Damien Querlioz at Université Paris-Saclay and his students Clement Turck and Kamel-Eddine Harabi for their collaboration in chapter 3, Giacomo Indiveri and Melika Payvand from the Institute of neuroinformatics, a joint laboratory between the University of Zurich and the ETH Zurich, for their collaboration in chapter 4 and to Jerome Casas, Claudio Lazzari, Teresita Insausti and Thomas Steinmann at Université de Tours as well as to John P. Miller from Montana state University for their collaboration in chapter 2. This is of course in addition to my many colleagues in CEA who have contributed to this work over the course of my PhD and are far too numerous to be listed here.

List of publications and patents

First author publications

- *T. Dalgaty et al.*, In-situ learning using intrinsic memristor variability via Markov chain Monte Carlo sampling, **Nature electronics**, *2021*
- *T. Dalgaty et al.*, Bio-inspired architectures substantially reduce the memory requirements of neural networks, **Frontiers in neuroscience**, *2021*
- *T. Dalgaty et al.*, Ex-situ transfer of Bayesian neural networks to resistive memory based inference hardware, **Advanced intelligent systems**, *2021*
- *T. Dalgaty et al.*, Hybrid neuromorphic circuits exploiting non-conventional properties of RRAM for massively parallel plasticity mechanisms, **APL Materials**, *2019*
- *T. Dalgaty et al.*, Hybrid CMOS-RRAM Neurons with intrinsic plasticity, **ISCAS, Sapporo, Japan**, *2019*
- *T. Dalgaty et al.*, Insect-inspired neuromorphic computing, **Current opinions in insect science**, *2018*
- *T. Dalgaty et al.*, Insect-inspired elementary motion detection embracing resistive memory and spiking neural networks, **Living machines, Paris, France**, *2018*
- *T. Dalgaty et al.*, A Mosaic of neuromorphic memory for in-memory small-world neural networks, **In redaction**, *2021*

Subsidiary author publications

- *Y. Demirag et al.*, PCM-trace: Scalable Synaptic Eligibility Traces with Resistivity Drift of Phase-Change Materials, **ISCAS, Daegu, South Korea**, *2021*
- *M. Payvand et al.*, Analog weight updates with compliance current modulation of binary ReRAMs for on-chip learning, **ISCAS, Seville, Spain**, *2020*
- *E. Esmanhotto et al.*, High-Density 3D Monolithically Integrated Multiple 1T1R MultiLevel-Cell for Neural Networks, **IEDM, San Francisco, United States**, *2020*
- *F. Pebay-Peyroula et al.*, Entropy source characterisation in HfO2 RRAM for TRNG applications, **DTIS, Marrakesh, Morocco**, *2020*
- *D.R.B Ly et al.* Novel 1T2R1T RRAM-based TCAM for large scale pattern recognition, **IEDM, San Francisco, United States**, *2019*
- *E. Donati et al.*Processing ECG signals using reservoir computing on an event-based neuromorphic system, **BioCAS, Cleveland, United States**, *2018*
- *E. Vianello et al.* Metal-oxide resistive memory and phase change memory as artificial synapses in spiking neural networks, **ICECS, Bordeaux, France**, *2018*
- *D.R.B Ly et al.* Role of synaptic variability in spike-based neuromorphic circuits with unsupervised learning, **ISCAS, Florence, Italy**, *2018*

Book chapters

- Synaptic realizations based on memristive devices, Memristive Devices for Brain-Inspired Computing, **Woodhead Publishing**, *2020*

Patents

- Odd-even memory array architecture for MCMC sampling *2021*
- Temporal difference reinforcement learning algorithm using volatile conductance states *2020*
- Phase change memory based spike-time-dependent plasticity algorithm, *2020*
- Volatile memory based three-factor learning algorithm, *2020*
- Neuromorphic event-based routing architecture based on resistive memory, *2019*
- Resistive memory based Markov Chain Monte Carlo sampling *2019*
- Resistive memory based Bayesian neural network architecture and model transfer technique *2019*
- Hybrid CMOS-RRAM neuron circuit *2018*

Contents

Chapter 1

Introduction

The human species, homo-sapiens, emerged an estimated 300,000 years ago. As modern humans we share, in large part, the same genes as our ancestors and our bodies and brains remain relatively unchanged. We have little common ground however in terms of how we live our lives and how we see the world. We can attribute this difference, despite our similarities, to our innate drive and ability to invent tools and ideas that have incrementally impacted how we are able to interact with the world over these millennia. This encompasses pre-historic inventions such as stone daggers for hunting and techniques to start fires; those from antiquity like religion, philosophy and bronze; all along the winding road of inventions through printing, economics and 5G telecommunications networks that have led us to this modern age. This thesis is concerned with another invention which, at the time of writing, is anticipated to give rise to a further incremental step on this path - artificially intelligent systems. Artificial intelligence (AI) is a scientific and engineering field concerned with the development of computer models that reproduce certain intelligent behaviours observed in animals such as reasoning, planning or learning. Since AI models and computers are fundamentally inter-twined, the term ‘computer’ is not only reserved for the computing machines with which we are today familiar, but importantly extends to any engineerable system that harnesses physical principles for computation. These computer models, other than developing theories of psychology and cognition, can also be applied in an engineering sense to solve practical problems. For example, the replacement or augmentation of human labour in monotonous or dangerous tasks, in complex tasks like medical diagnosis as well as applications which are otherwise unsuitable for humans such as the ‘mining’ of, or pattern extraction from, large and high-dimensional datasets.

The potential of the current generation of approaches in AI has been recognised both by governments and large technology corporations, which are jointly driving the rapid growth of its commercial application [1, 2]. However, lurking in the shadow of numerous grand success stories, there are fundamental drawbacks and limitations which, to support the anticipated growth, must be confronted. Principally there are environmental issues, concerning the spiralling quantities of energy demanded by state of the art AI models [3, 4] for incremental performance gains [5], the ‘cloud-centric’ application of AI instead of in distributed and resource constrained ‘edge’ environments [2] and ethical issues regarding the lack of model interpretability [6] and the absence of model uncertainty quantification in many cases [7, 8]. The progress in the field can be attributed predominantly to progress in underlying computing hardware, rather than a revolution from the algorithmic or modelling perspective, applied to ‘connectionist’, or ‘neural network’, AI models. This thesis explores how, in a similar spirit, an emerging technology called resistive memory can be married together with neural network models. It proposes a new, hardware-focused, approach that requires extremely low quantities of energy with respect to existing solutions. Additionally, in recognising the important role it has played in the development of the field via the tricks discovered by evolution, biological nervous systems will be used as a means of inspiration to guide the direction of the work.

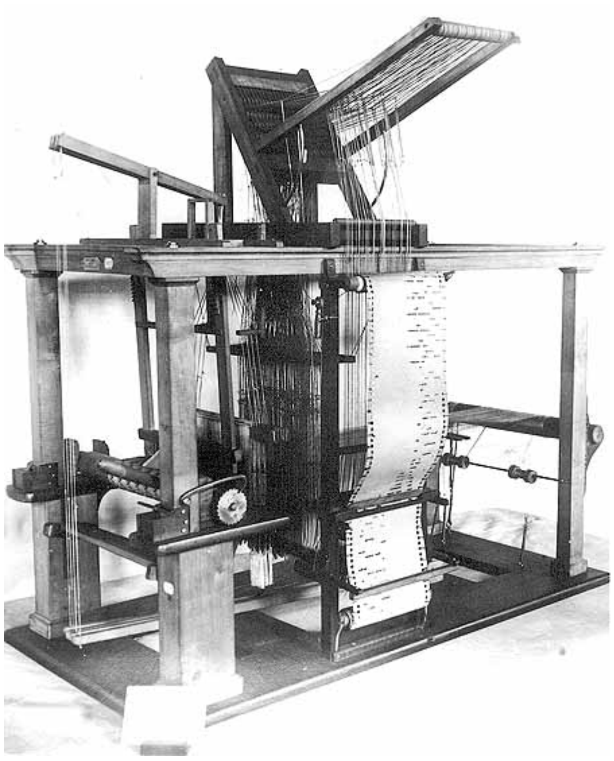
Before introducing the content of this thesis in more detail, it will first be instructive to briefly recount the history of artificial intelligence, the computer hardware that has been used as a substrate in AI and elaborate upon some of the predominant problems faced by field which this work aims to address, in order to better understand its place and contribution.

1.1 A brief history of artificial intelligence

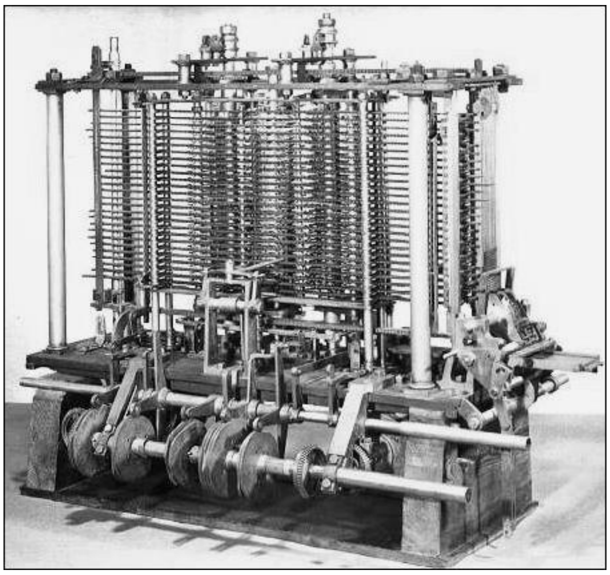
Although artificial intelligence would appear to be something new, being formally named in 1956 [9], the seeds that would go on to become the roots of the field were gradually sewn in the centuries before - during an AI ‘pre-history’.

1.1.1 Pre-history

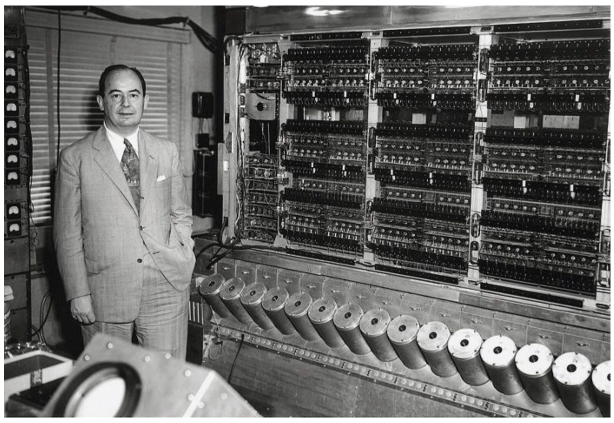
The link between artificial intelligence and computer science is unshakeable - if intelligence were to be implemented artificially, on what other substrate besides a ‘computer’ could it be realised? The first system bearing the hallmarks of computer would appear in 1804 in France - the Jacquard loom [10]. The loom was a fabric weaving machine, programmable via hole punched paper cards which determined routines to be executed with the apparatus (Fig. 1.1a). Some decades later, in 1832, Charles Babbage who was an admirer of the Jacquard loom, proposed a mechanical computer he called the Analytical engine (Fig. 1.1b), featuring an arithmetic logic unit, memory and a means of realising conditional flows and loops [11] - although it would never be built. George Boole would go on to formalise the mathematics of the logical manipulation of symbols (which had been evolving from the time of the ancient Greek philosopher Aristotle) in 1854 [12], before Alan Turing would take such concepts of logic and combine them with Babbage’s ideas and propose the first digital computer system during the second world in 1936 [13]. The ideas of Turing would then be taken forward by the likes of the Hungarian scientist John von Neumann [14] who led the effort to manufacture the first binary stored-program Electronic Discrete Variable Automatic Computer (EDVAC) (Fig. 1.1c) architecture which remains widespread today. Largely it has been the manipulation of models, residing in the memory of the computer, by mathematical algorithms, which execute in the computers arithmetic centres, of these ‘von Neumann’ computers that has been the principal means of implementing and applying AI ever since.



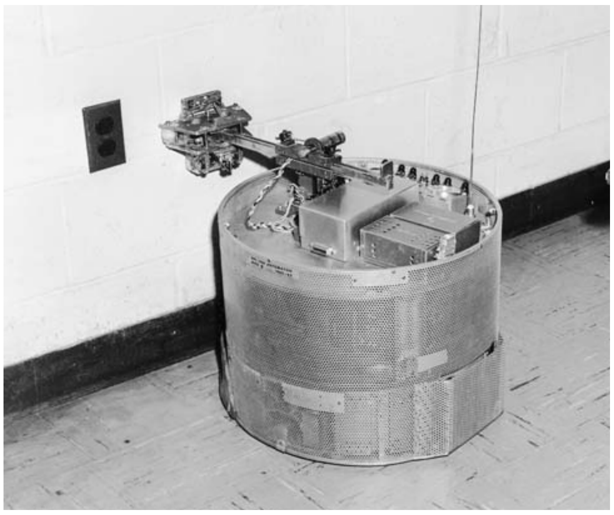
(a)



(b)



(c)



(d)

Figure 1.1: **Early computer systems.** (a) A photo of a Jaquard loom. The hole punch card can be seen as a vertically running white sheet on the front face of the apparatus. (b) A model of the analytical engine proposed, but never built, by Charles Babbage. (c) Jon von Neumann stands next to a portion of the Electronic Discrete Variable Automatic Computer (EDVAC). (d) The beast robot consisting of large metal drum which, inside, contained a small network of transistors that controlled its movement.

Parallel to the development of the ideas that lead to modern digital computers, several important mathematical theories would also be developed during these centuries that would go on to lay the foundations of AI. In 1805, one year after the invention of the Jacquard loom and also in France, the mathematician

Adrien-Marie Legendre published a paper on the least-squares regression method [15] which remains, two centuries later, one of the central concepts of subsets of approaches to AI based on the minimisation of an error metric of a model given some data. Later, an iterative method for solving such problems would be proposed by Augustine-Louis Cauchy in 1847 called gradient-descent [16]. At the time of writing, gradient-descent is arguably the most important algorithm in modern day artificial intelligence and, in a broader sense, the field of mathematical optimisation.

Remarkably, some decades even before Adrien-Marie, an English reverend and mathematician named Thomas Bayes wrote down a novel interpretation on probability theory [17]. What would come to be known as Bayes' rule described how, instead of calculating probabilities based on frequency of past events, a posterior probability could be updated through the product of a prior belief on this probability and the likelihood of a set of observations. Bayes actually did not publish this work during his lifetime, and his manuscripts were encountered by chance after his death in 1761 while his family were sorting through his possessions. Had they not been discovered, the French mathematician Laplace would also independently arrive at the same set of conclusions as Bayes half a century later in 1820 [18]. These techniques set the foundations for a statistical approach to artificial intelligence called Bayesian inference whereby a posterior probability is iteratively updated as new evidence becomes available [17]. A Russian mathematician Andrei Markov, also working in the field of probability theory one and a half centuries after Thomas Bayes, published several works describing what have become known as Markov chains [19]. Markov chains describe a random and 'memoryless' process whereby new states of an arbitrary system results from random permutations to its current state, offering a way to model systems with apparently stochastic characteristics. Markov chains and Bayes' theorem would eventually be married together in several works in the follow century, beginning with an article from Nicholas Metropolis and Stanislav Ulam on the Monte Carlo method [20, 21], and later expanded upon by Wilfried Keith Hastings [22], that set the groundwork for 'Markov chain Monte Carlo' (MCMC) sampling algorithms. Crucially, Markov chain Monte Carlo would allow for Bayesian inference to applied to problems where, in the majority of cases, conjugate likelihood and prior distribution pairs did exist and did not permit an analytical solution to the posterior [23]. At the time of writing, MCMC is considered to be of the most important and influential computer algorithms ever proposed [24], with a marked impact upon countless scientific and commercial fields - artificial intelligence among them.

In the early 20th century, prompted by experimental results such as the measurement of action potentials propagating down the giant axon of the squid to trigger an escape response [25], a group of scientists who called themselves Cyberneticians, would begin to build 'bottom-up' models of animal nervous systems [26]. An early contribution to the field was made by Americans Warren McCulloch and Walter Pitts in 1943 when they proposed a model of a neuron that was activated as a function of a weighted sum of its input [27]. This would be followed with another model proposed by Alan Hodgkin and Andrew Huxley, based on their own experiments they conducted with the squid giant axon, that expanded on the ideas of the McCulloch-Pitts model by incorporating dynamical properties of neuronal membranes and the 'spiking' activation behaviour observed in the squid axon [28]. They would go onto receive the Nobel prize for their work in 1963. At the same time, a Canadian scientist named Donald Hebb proposed that neurons, described by such models, could organise themselves via correlative rules based on their co-activation as a means of adaptation and learning in the nervous system - an idea that would come to be called Hebbian learning [29]. Successive works on the visual systems of *Drosophila* [30] and cats [31], would go onto recognise that networks of these neuronal elements were were capable of computing certain functions defined by their topology - respectively motion and edge detection. An early Cybernetic system, brought to life by a hard-wired network of neuronal like transistors, was the 'beast' robot - assembled by scientists at John Hopkins University [32]. The beast, a big metal can on wheels (Fig. 1.1d), was equipped with ultrasonic sensors on it's exterior which were connected to a small network of analogue logic circuits - culminating in no more than some dozens of transistors. The outputs of these gates controlled the wheels of the robot and, it is said, it would wander aimlessly around the corridors of the University avoiding obstacles and turning corners. When the beast was low on battery it would find, and plug itself into, an electrical socket. Then, when fully charged and ready to wander anew, it would un-plug itself and once again be on its way. The behaviour of the system has since drawn comparisons with the feeding and survival behaviours of simple uni-cellular organisms such as Paramecium [33] as well as multi-cellular organisms equipped with simple nervous systems such as the *C. Elegans* worm [34].

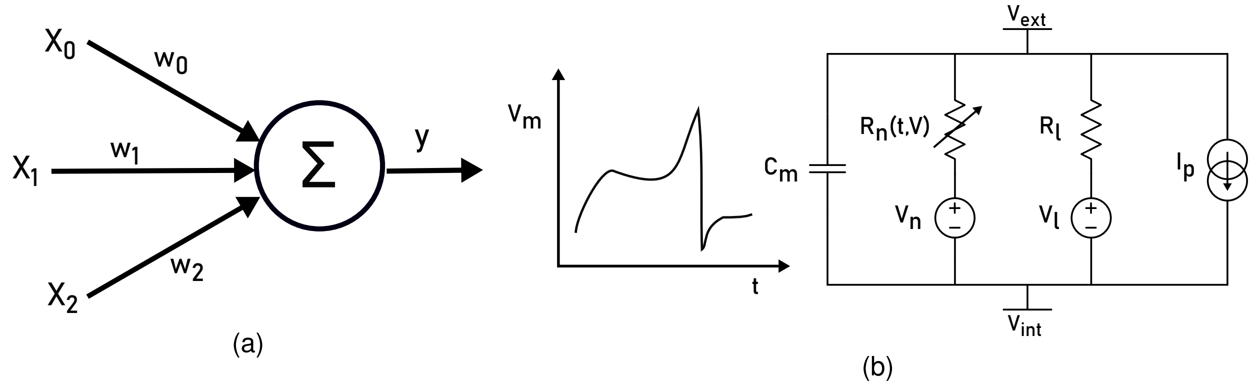


Figure 1.2: **Types of neuron models.** (a) The McCulloch-Pitts neuron model whereby the sum of the multiplication of the elements of an input vector **X** and a synaptic weight vector **w** is output by the neuron. (b An electrical circuit model for a neuron proposed by Hodgkin and Huxley that describes how parameters, such as the neuron membrane voltage V_c, evolves in time.)

1.1.2 1950's, 60's and 70's

The first general purpose model inspired by biological neural networks was the perceptron [35], proposed in 1958 by Frank Rosenblatt. The perceptron is composed of layers (i.e. a multi-layer perceptron) of McCulloch-Pitts neurons. Each layer is networked by a so-called fully-connected matrix of synaptic edges, weights or parameters. Perceptrons allowed neurons to be leveraged as single linear units, in that they define a linear hyper-plane through their input feature space, which, through their combination in a network permitted non-linear functions to be implemented. An intuitive example of the utility of such networks is the three neuron, six synapse network which implements the non-linear XOR logic function [35] - something that cannot be solved with a single linear neuron alone. The parameters of early perceptrons were determined by optimisation techniques such as the Widrow-Hoff delta learning rule [36]. This was a first example of a supervised 'machine learning' algorithm whereby the model would learn, not in biologically-plausible fashion as Hebb had envisaged [29], but rather using iterative mathematical operations implemented in the arithmetic centres of von Neumann computers. In their rule, the difference between a 'teacher' signal and the actual output of the neuron units would be multiplied by the activation of the input neurons and a learning rate constant to provide the updates that were required to applied on the intermediate synaptic weight matrix. However, the delta rule could only be used to parameterise synapses in a two layer perceptron. A technique called automatic differentiation, proposed in 1970 by a Finnish master student called Seppo Linnainmaa [37], would facilitate, although not until one decade later, the 'training' of multi-layer perceptrons. In the meantime, another type machine learning approach, inspired by Darwinian theories of evolution, was developed and applied to the optimisation of models, such as the multi-layer perceptron, called genetic and evolutionary algorithms [38]. Such algorithms operate by mixing together properties of a previous 'generation' of candidate models, selecting the best according to some fitness metric and then 'breeding' together the parameters of the best, or the 'fittest', models to produce a new generation of models, and so on.

As an alternative to the 'black-box' optimisation of perceptrons, another prominent early direction of exploration was into models of symbolic reasoning. A first, extremely famous, example of such a model was the chess playing program developed by American electrical engineer Claude Shannon - the same Shannon who would later make even greater contributions to information theory - in the year 1950 [39]. Shannon's program searched a 'tree' of possible moves from a current game state using a search algorithm called minimax. After searching the tree to a certain depth, or a number of moves in the future, the algorithm 'backs-up' possible eventual outcomes to the top of tree and permits a decision to be taken regarding the optimal move (Fig. 1.3b). Some years later Arthur Samuel would augment Shannon's approach with a technique called 'rote' learning (this time using the game of draughts) where the algorithm would back-up and assign to nodes a measure of value based on the time to reward [40]. This concept of assigning value based on a temporal difference would set a foundation for a great number of algorithms that would be proposed in later decades. Another tree-based model, the goal-tree, was developed by James Slagle in 1963 [41], which took an initially complex integration problem and proceeded to use a set of pre-defined heuristic transformations to expand, therein constructing a tree, and simplify the original problem into a final form, at the terminal tree leaf nodes, that could be easily solved with another set of simple mathematical rules. Goal-trees were later applied to a set of problems known as 'block worlds', whereby the computer model was tasked with arranging scattered blocks on a table into a specific, user defined, arrangement - a non-trivial task requiring the computer model to reason and plan. In this case, based on a limited set of pre-defined logical rules and block manipulation functions, the computer would proceed to heuristically build a goal-tree and define a sequence of block manipulations required to obtain the required arrangement [42]. One feature of goal trees was that, after performing a set of actions, the computer model could be queried to explain it's reasoning by backing-up the tree it has just built - offering an interpretation of its actions. A related AI model, based on logical relations between 'symbols', were semantic networks, often called knowledge graphs, applied by Ross Quillan in 1966 [43] as a knowledge inference engine. Semantic networks are composed of nodes representing entities, and edges between the nodes explaining their relation or interaction and set the basis for the 'knowledge graphs' that underpin search engines like Google. For example, a directed edge in a semantic network 'is a' could connect the node 'cat' to the node 'animal'. In 1970, Patrick Winston would propose 'arch', or 'one-shot', learning whereby, an algorithm constructs a semantic network by inter-connecting symbols through specific relations consistent with examples presented to the model - building a comprehensive description of an object based on very few observations, or very few shots. An arch, for example, can be characterised by a horizontal block that is related to two vertical blocks below in that they support it, and those two vertical blocks were related to each other in that they do not touch each other (Fig. 1.3c) [44]. Unlike the delta learning algorithm used to train perceptrons, which required many training examples to produce incremental changes to model parameters, arch learning is capable of constructing and augmenting a number of relations between nodes from single examples. Additionally, like goal-trees, the resulting semantic graph also benefits from a means of interpretability. The culmination of these, and many other related ideas from this epoch that have not been discussed (such as constraint propagation [45, 46]), would ultimately lead to what were marketed as expert systems [47] in the 1980's. These systems incorporated knowledge into a hand-crafted logical model using control flow statements like if, then and else. Despite their perceived lack of elegance from AI practitioners, expert systems would ultimately mark the first time that an AI system was massively commercialised and applied to real-world problems.

Many of these approaches to machine learning, such as the application of the Widrow-Hoff delta rule to the perceptron, can be labelled as 'supervised' algorithms. Modern day machine learning is often split into three such categories - supervised, unsupervised and reinforcement learning. Ideas pertaining to the latter two were also proposed during these decades. Unsupervised learning algorithms, those without a teaching signal or labelled data, can trace their roots back to a paper in 1967 by Thomas Cover proposing the nearest neighbour algorithm [48]. Unsupervised learning algorithms, instead of learning to produce a desired output, looked for underlying structure in a set of data and use mathematical constructs, like euclidean distance from a common centroid or a parametric probability distribution, to group together similar data-points. (Fig. 1.3d). A few years earlier, in 1964, a Scottish computer scientist called Donald Michie, described the first reinforcement learning approach with his 'tabular' approach to noughts and crosses (tic-tac-toe) [49]. Perhaps for want of a computer, Michie demonstrated his idea using a set of 304 drawers, each containing coloured beads, and required a human operator to correctly move the beads between the drawers based on a small set of rules. Each drawer corresponded to one of the 304 possible game states of noughts and crosses and the colour of beads in each drawer corresponded to the possible actions taken from that state. For each step of the game, a bead was randomly sampled from the drawer of the current state by the operator and a nought or cross was placed accordingly. If the actions that were taken during a round of the game resulted in a loss, Michie's model would be 'punished' whereby the all of the drawn beads that were played in that losing game were removed from their drawers. In the event of winning a game, the model was 'rewarded' by placing three extra beads of the colours that were drawn in the respective drawers. This had the effect that, with an increasing number of iterations, the model would have a higher probability of choosing actions in game states it had already visited that would lead to an eventual victory.

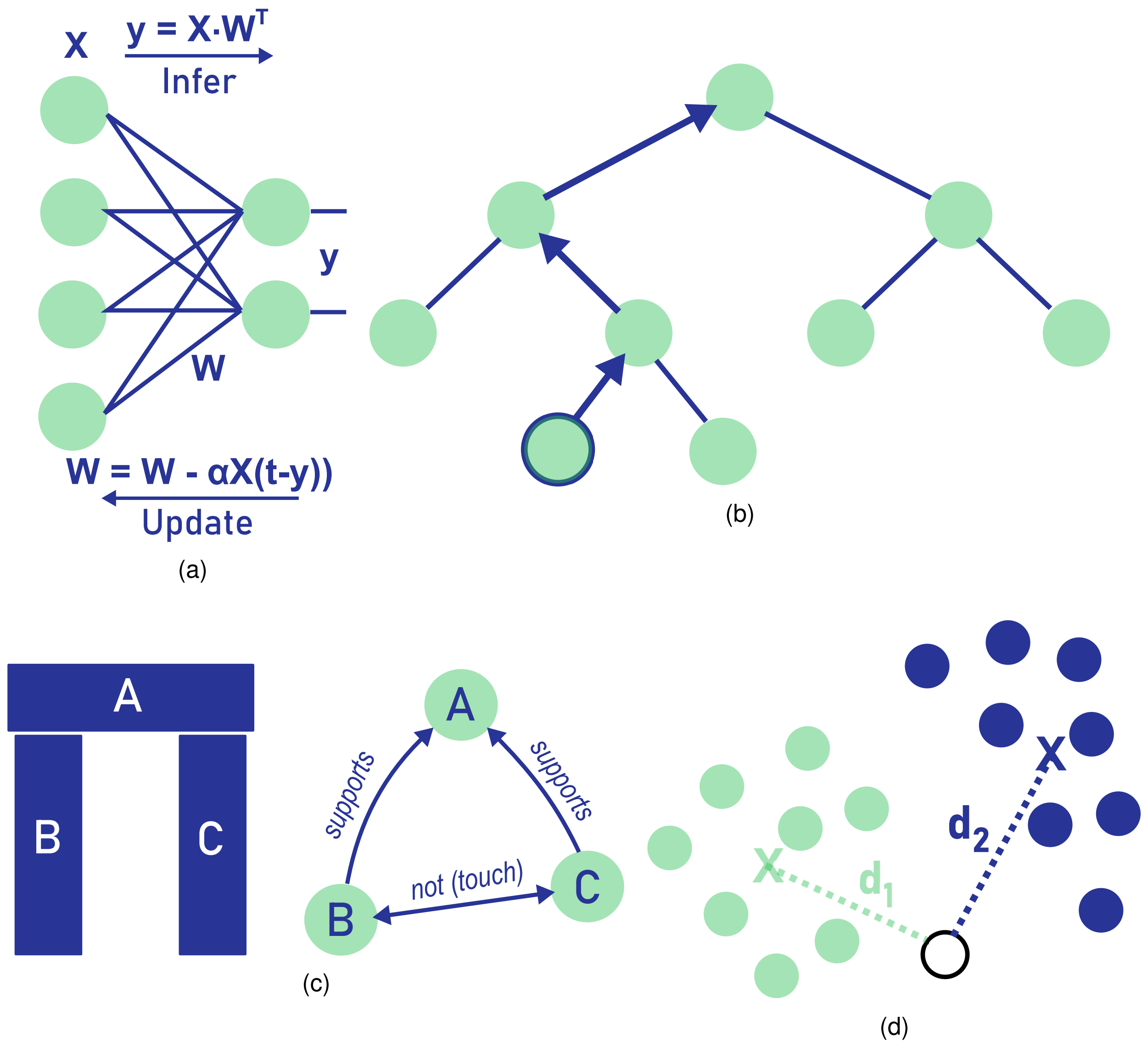


Figure 1.3: **Examples of computer models applied in artificial intelligence.** (a) The perceptron model composed of an input layer of neurons, representing the input data points, and an output layer of neurons each of which corresponds to a prediction, normally a label, about the input. These outputs, y, are inferred using the dot product between the data X and the weight matrix W and then the weight vector can be updated, or trained, through the multiplication of the difference between a teacher signal t and the actual outputs y with the input data X and a learning rate α. (b) A search tree model whereby each green circle represents a node, a specific state in a game of chess for example, and each blue branch that extends downwards a possible decision taken from that node. Blue upward pointing arrows provide a notion of backing-up the search tree, such that information about future states can inform what decision is taken in the current state - drawn as the green node at the top of the tree. (c An example of arch learning, using the task of learning the properties of an arch - drawn as the blue rectangular blocks. Arch learning allows the construction of a semantic network with relations (blue directed arrows) between nodes (green circles) that characterise the object to be recognised. (d) An example of a nearest neighbours unsupervised learning model whereby the distance between the centroid of a data cluster (green and blue crosses) and a new data point (white circle) can be used to determine to which cluster it belongs.

1.1.3 1980's and 90s

The 1980's began with a proposal for what would become a hugely important neural network architecture, named the 'neocognitron' by Japanese computer scientist Kunihiko Fukushima [50]. The modelling approach known at the time of writing as convolutional neural networks would descend from this idea. In his paper, Fukushima draws heavily upon research into the architectural mechanisms uncovered in the visual cortex of the cat several decades earlier [31]. This 'bio-inspired' neural network architecture employed a combination of fixed and modifiable synapses which, contrary to later approaches with convolutional neural networks, would be trained using an unsupervised machine learning algorithm. Later in 1986, the automatic differentiation technique originally proposed in 1970 [37], was famously applied to train a multi-layer perceptron model - solving the credit assignment problem between network layers that had previously limited perceptron models [51]. This technique came to be known as backpropagation. Three years later, the 'universal approximation theorem' was presented by George Cybenko stating that a finite hidden layer of fully-connected neurons of such a multi-layer perceptron model was capable of approximating any continuous function [52]. The combination of backpropagation with universal approximators gave rise to tremendous expectation about the potential of such a modelling approach, although it would take further decades of innovation, by in large at the hardware level, before that potential would finally be realised. Backpropagation was extended to the setting of recurrent neural networks simultaneously by several scientists towards the end of the 1980s using a technique called backpropagation through time [53], whereby recurrent connections between neurons were defined, not with cyclic loops that were forbidden by backpropgation, but by 'unrolling' the network into several timesteps. In 1997 an important recurrent neural network architecture was proposed called the long short term memory network [54]. Somewhat like the neocognitron, long short term memory networks, incorporated specific connectivity patterns which permitted, for example, architectural mechanisms enabling functionality such as 'forgetting' which proved useful in the tasks related to time-series data.

In the 1990's Christopher Watkins would take forward the ideas of Donald Michie, thankfully this time with a digital computer, and Arthur Samuel and propose the reinforcement learning approach known as Q-learning [55]. Instead of a set of drawers containing beads, Q-learning uses a state-action 'Q-table'. Each cell of a Q-table contains a Q-value corresponding to the expected, temporally discounted, future reward based on taking optimal actions from that state. The Q-learning algorithm explores these states through interaction with an environment and uses the Bellman equation from dynamic programming [56] to populate, or learn, the state-value pairs in each cell. Around the same time, Richard Sutton would formalise the work of Arthur Samuel into what is now called temporal difference learning [57]. Temporal difference learning algorithms are characterised by a temporally decaying parameter called an eligibility trace that tracks the time since, for example, a particular state has been visited. This approach was applied in the 1994 by engineers working at IBM whereby a temporal-difference model learned, from self-play, to play the game of backgammon, eventually obtaining a 'master level' of performance [58]. IBM would also produce DeepBlue two years later that, despite being unsuccessful the first time around, would go on to beat the reigning world chess champion in 1997. DeepBlue employed a search-tree algorithm, largely similar to the minimax algorithm used by Shannon four decades before, called alpha-beta [59]. The success therein was not a substantial modelling innovation but rather the development of a massively parallelised computer architecture that allowed faster and deeper search to be performed. With DeepBlue, IBM would go on to set two trends in artificial intelligence in the subsequent decades - that of applying 'brute force' computing to AI and also the use of public competitions as a marketing strategy of large corporations.

Similarly to how the neocognitron took architectural inspiration from biology, an American scientist called John Hopfield would, in 1982, take inspiration from the dynamical properties of biological neural networks and propose a new idea for how memories could be stored and accessed in a recurrently connected population of neurons [60]. The neurons of early 'Hopfield networks' were McCulloch-Pitts neurons which output either +1, if it's weighted input exceeded a threshold, or -1 otherwise. Under the condition of symmetrical connectivity, it was proven that, upon presentation of a static input pattern, the state of the network would tend towards a stable attractor of +1 and -1 activations - serving as an activation 'barcode' that represented a memory. New attractors could be formed in Hopfield networks through the Hebbian modification of synaptic parameters based on the co-activation of the pre- and post-synaptic neurons pairs. Two years later Hopfield networks would be implemented using neurons implementing a sigmoid activation function [61] and later with dynamical, Hodgkin-Huxley, neuron models [62] culminating in a research field dedicated to investigating 'attractor networks'. In the 1990's American physicist Carver Mead would draw analogues between the modulation of conductivity in neuronal ion channels in biological neuronal membranes and transistors [63] giving rise to a new field called 'neuromorphic' computing. Neuromorphic computing would concern itself with the bottom-up emulation of biological neural networks and biological Hebbian learning mechanisms, attractor networks for example, using sub-threshold analogue transistor circuit models of neurons and synapses.

These two decades also saw the proposal and demonstration of many other important supervised and unsupervised learning models and algorithms such as the self-organising map proposed in 1990 by Finnish scientist Teuvo Kohonen [64]. Additionally, in 1995, Vladimir Vapnik published a paper that combined a linear classification approach he had been developing since the 1960's called support vector machines [65], with a set of mathematical tricks called kernel methods that were also originally proposed in the 1960's [66]. The incorporation of the kernel trick into the support vector machine permitted an implicit projection of data, which was perhaps not linearly separable in its original space, into new space where it becomes separable using linear support vector machine classifier [67]. Another important idea, called boosting, was developed during the 1990's [68]. Boosting recognised the predictive power of a large ensemble of individually weak predictive models which, while alone were perhaps only slightly better than random guessing, together allowed particular subsets of models to specialise in the detection of specific features of a complex dataset. Boosting was combined with decision trees by Tim Kam Ho in 1995 to realise the important random decision forest machine learning algorithm [69].

1.1.4 21st Century

In the current epoch it is perhaps not yet clear which new ideas will stick or not. At the time of writing, it seems like a period with little in the way of fundamentally new or radical ideas in how to build models of artificial intelligence and instead one where existing ideas have been supercharged by ever increasing computational resources. The key piece of computing hardware would become the graphics processing units, the first of which was launched by NVIDIA, rather appropriately, at the turn of the century in 1999 [70]. Graphics processing units would ultimately pave the way for ‘deep’ multi-layer perceptron models to be trained using the backpropagation algorithm to achieve state of the art performance across a wide range of application domains. The approach was re-branded as ‘deep learning’ in order to distance it from previous failures regarding the optimisation of multi-layer perceptron models using backpropagation [71]. If artificial intelligence were a garden, deep learning is a massive Sycamore tree that is blocking out the light for a lot of the smaller, more fragile, plants. The first great success of deep learning was the application of a deep convolutional network model, named AlexNet, in the annual ImageNet image classification competition held in 2012 where it achieved a resounding victory [72]. AlexNet, a somewhat monstrous evolution of the 1980’s neocognitron counting $60,000,000$ free synaptic parameters, proposed a regularisation technique called dropout that prevented the model, which with the large number of parameters had so many degrees of freedom, from over-fitting. Dropout, another concept loosely inspired from biology, simply requires that, upon successive forward propagations, synaptic parameter values be temporarily set to be zero with some probability such that they do not contribute to the models prediction.

The next year, a British artificial intelligence company called DeepMind would demonstrate a deep reinforcement learning approach, called Deep-Q learning, that learned to play Atari games from experience [73] where a deep convolutional neural network model was employed as an approximation of the Q-table proposed in the 1990’s [55]. Again it was a bio-inspired idea, based on theories of the mammalian hippocampus, called experience replay was the would prove to be the trick that would enable the Deep-Q model to obtain such an ‘expert-level’ of performance. Echoing the marketing strategy of IBM’s DeepBlue in the 1990’s, DeepMind famously defeated the European champion of ‘Go’ with their Alpha-Go deep reinforcement learning model in 2016 [74]. Similarly to DeepBlue, Alpha-Go was based on a search-tree algorithm called Monte Carlo tree search [75] combined with of value and policy encoding deep learning models.

Deep learning models have also had a profound impact in natural language processing. Initially, this was thanks to deep recurrent neural network models, the long short term memory network for example. However, in 2014 ‘attention’ models were proposed which would come to dominate natural language processing. Attention provided an architectural mechanism allowing models to focus on particular parts of a data sequence - whether it being an a certain area of an image or particular word in a phrase - similar in concept to the mammalian fovea. Initially deep attention models were incorporated into an architecture composed of recurrent neural networks [76]. In 2017 however, engineers working at Google proposed an attention model called Transformer. Transformer improved on the previous state of the art using an attention architecture that did not feature any recurrence [77] - leading many in the field to claim that recurrent neural networks were now redundant in such applications.

Another important deep learning architecture was proposed in 2014 called the generative adversarial neural network [78]. These models are actually a composite of two competing deep neural networks. The first neural network, the generator, is fed with noise it then uses to produce a ‘fake’ output data point - a fake image for example. The second network, the discriminator, is tasked with distinguishing this fake from a real data point. The two models are optimised simultaneously using backpropagation, whereby the objective of the discriminator is to minimise the error in the detection of the real data points while the generator is trained to maximise the error of the discriminator. Interestingly, the generator uses a de-convolutional neural network architecture to generate the fake images [79] - effectively a convolutional network flipped back-to-front.

Despite the truly seismic success of deep learning, serious concerns about model size, therein the number of synaptic parameters in the model, as well as the computational resources required to optimise all of those parameters during training throw its practicality into question [3, 4]. At the time of writing the largest models make use of a few hundred billion synaptic parameters [80]. This is particularly relevant in the context of memory and energy constrained 'edge' computing applications and has led to research into deep learning architectures and algorithms that allow model size to be significantly reduced. One such model, proposed in 2016, is SqueezeNet [81]. SqueezeNet offers a 50x reduction in model size with respect to the sixty million parameter AlexNet while being able to obtain an equivalent performance. The principal innovation of SqueezeNet was the proposal of the 'fire' module. Each fire module in the architecture is composed of a layer of point-wise convolutional filters followed by an expansions layer containing more point-wise filters but also larger convolutional kernels.

Another direction of research, resonating more with the ideas of John Hopfield and attractor networks, called reservoir computing emerged at the beginning of the 21st century. Reservoir computing is based on the idea that static, randomly recurrently connected neurons could project an input time-series into a new, possibly higher dimensional, space where temporal information pertaining to different classes of data points could be integrated into linearly separable neuron activation patterns. The two main approaches in reservoir computing are echo state networks [82], which employ neurons with continuous activation functions, and liquid-state machines [83], that use spiking neuron models. The two approaches can therefore be broadly dichotomised into favouring either the McCulloch-Pitts or Hodgkin-Huxley paradigms. Both approaches have been applied to problems in temporal pattern recognition, although their achievements have been largely overshadowed by the achievements of deep learning models applied to the same tasks.

Despite being unable to match the performance of deep learning models based on graphics processing units, 'non-von Neumann' reservoir computing models implemented on an emerging class of computer called a 'neuromorphic processor', offers the prospect of a more energy efficient solution. Rooted in the ideas outlined by Carver Mead in the 1980's [63], the first system bearing the hallmarks of a neuromorphic processor was demonstrated in 2006 and consisted of an array of programmable analogue silicon spiking neuron circuits which could be inter-connected through further circuits models of the synaptic connections between neurons [84]. A more developed system, supported by an additional digital 'event routing' protocol was proposed in 2014 [85] and would be followed by a similar chip in 2015 which incorporated additional circuit implementations of bio-inspired Hebbian learning algorithms [86]. Fully digital neuromorphic processors were then released released by IBM in 2015 [87] and then Intel in 2018 [88] confirming the interest of large corporations in the development of neuromorphic, event-based, hardware. Such systems are referred to as non-von Neumann since they offer a means of distributed memory and parallelised computing, similar to the computation of the animal nervous systems on which the chips are based, in contrast to the centralised and sequential nature of the computing machines developed by the likes of Alan Turing and John von Neumann. However, it is important to note that both Turing and particularly von Neumann [14] also argued in favour of such brain-inspired computing solutions in the 1940's and 50's.

1.2 Hardware in artificial intelligence

Artificially intelligent computer models are intrinsically tied to computing hardware. It is no coincidence that the fathers of computer science, like Alan Turing, Donald Michie and Claude Shannon, were also the pioneers that developed early concepts in AI that still persist to this day. At the time of writing, the predominant computing architecture is the von Neumann stored-program computer. Von Neumann machines are coarsely defined as having separate memory and processing centres. Information encoded as strings of binary bits, describing either variables or arithmetic and logic instructions, are stored as digital words in the memory centre. The processing unit fetches these variables and instructions from defined addresses in memory, performs an operation on the variables as defined by the corresponding instruction and then writes the result back to another pre-defined location in the memory. Relative to the modern computers available at the time of writing, early systems were almost unimaginably constrained in terms memory and speed. Around the time that Arthur Samuel wrote his draughts playing program [40] for example, the state of the art computer was the IBM 1401. It disposed a mere 4kb of 8-bit memory and was clocked at a frequency of 87kHz. This permitted the equivalent of approximately one thousand operations per second [89]. You can imagine how aware of the underlying hardware the early AI practitioners had to be. As time progressed however, von Neumann computers would become faster and leverage more memory as well as benefit from architectural innovations such as parallelisation, cache memory [90] and multi-threading [91], such that gradually less and less thought was given to the underlying computation. However, for memory intensive techniques in AI, such as deep learning, computing hardware has once again become a limiting factor. This is not due to memory capacity, which continues to increase year on year, nor is it due to a limit in clock speed. The problem arises from the time and energy incurred in the transportation information between memory and processing centres. In reference to this, the term 'von Neumann bottleneck' was coined in 1978 [92]. This has driven the pursuit of alternative, non-von Neumann, hardware paradigms wherein the objective has been to reduce as much as possible the physical distance separating memory and processing - ideally to zero.

1.2.1 LISP machines

In the first decades of AI research and application, the go-to language for the implementation of tree, logic and knowledge based models was LISP [93]; deriving its name from LISt Processor in reference to the data structure and syntactic style central to by the language - the linked list. As a result of its popularity, a special purpose computer called a LISP machine was developed in the 1970's [94] whereby dedicated circuits allowed the acceleration of various LIPS specific operations and memory structures. These machines offered a speed-up of several factors compared to equivalent programs implemented on general purpose machines. However, as the use of the LISP language, and the approach to AI which it facilitated, declined during the 1980's and 90's, many LISP machine manufacturers would ultimately go out of business and general purpose von Neumann computers would go on to serve as the predominant substrate for AI.

1.2.2 Field programmable gate arrays

The earliest example of a non-von Neumann computing hardware is the field programmable gate array (FPGA), proposed in 1985 by the American company Xilinx [95]. FPGAs are effectively a two-dimensional array of configurable logic blocks (Fig 1.4a). Each block can be configured independently to implement basic logic operations, typically using a look-up table. The connections between neighbouring blocks can also be configured offering the possibility of realising large reconfigurable digital circuits. FPGAs distribute blocks of memory over the periodic 2D array which are responsible for storing the information which determines the logic and inter-connectivity for a subset of blocks. Naturally a body of work has focused on the implementation neural network architectures and deep learning algorithms, such as backpropagation [51], on FPGAs [96]. In particular the parallelisation of the dot(inner)-product operation, which underpins the forward and backward propagations required in deep learning, has been investigated. However, when benchmarked against application-specific integrated circuits (ASICs) implementing neural networks [97], FPGAs are generally acknowledged to incur penalties an order of magnitude greater in silicon area, power consumption and latency [98].

1.2.3 Graphics and tensor processing units

Graphics and, more recently proposed tensor, processing units (GPU/TPUs) represent an evolution of central processing units (CPUs) through the incorporation of a parallelised architecture and multi-threading that allows the acceleration of mathematical operations, namely operations on matrices (Fig. 1.4b). Originally proposed, as suggested by the name, to accelerate the processing of images and video by NVIDIA in 1999 [70], the GPU has also been leveraged to accelerate AI models based on the manipulation of large matrices - principally deep learning [71]. Like CPUs, current GPUs [99], are configured to execute stored programs through the use of an interface called Compute Unified Device Architecture (CUDA) which provides access to the arithmetic units of the GPU from a high-level programming language and, although memory and processing and more distributed with respect to a CPU, they are still considered as von Neumann computers. Recently Google released the TPU that, unlike the GPU which is intended to be applied to a broad class of applications, is targeted solely on accelerating inference and learning algorithms related to deep learning [100]. While GPUs and TPUs achieve extremely high throughput and accelerate significantly deep learning models, they require a large amount of energy - a typical GPU at the time of writing will consume on the order of hundreds of Watts of power [101]. Their application is therefore largely limited to cloud computing and data centre settings. This however restricts the class of tasks to which such GPU-based AI models can address - in particular applications at the edge where energy and memory resources are constrained and perhaps intermittent.

1.2.4 Neuromorphic processors

The energy requirements of GPUs and TPUs, when applied to artificial intelligence, are often contrasted with the estimated energy requirements of animal nervous which are capable of performing the same task (while of course performing countless other parallel tasks) but requiring many orders of magnitude less energy. This has driven interest in the field of neuromorphic computing [63], which aims to develop computing systems that imitate the bottom-up computational primitives of animal nervous systems in configurable, often self-configuring, silicon chips. It is believed that such a bottom-up approach, emulating the massive parallelism and distributed computation of the biological system, will lead to the realisation of a more energy efficient hardware that could support AI models which can be applied in the energy constrained setting of the edge. Numerous analogue circuits modelling the dynamical properties of biological neurons [102, 103] and synapses [104] have been proposed, leading to the first re-configurable arrays of such models in 2006 [84]. Later systems would be composed of multiple 'neuromorphic cores' [86, 85] that make use of content addressable memories [105] and a routing protocol called address event representation (AER) [106] to send information between neurons on the same core and between cores (Fig. 1.4c). AER assigns an address to each neuron and, since neuromorphic processors implement spiking neuron models like that proposed by Hodgkin and Huxley [28], transmit events from a source address neuron to, potentially, multiple target address neurons asynchronously.

Neuromorphic processors have also been realised using digital, instead of analogue, circuit models of neurons and synapses [87, 88]. The digital approach is appealing since, as technology nodes scale down, digital circuits become more energy efficient whereas analogue circuits face increasing design challenges and are not able to extract an equivalent benefit [107]. Furthermore, due to the increased mismatch between transistor dimensions and properties in more advanced nodes, large behavioural differences can exist between analogue circuit models that would ideally behave identically. To mitigate this, the dimensions of the transistors composing analogue circuit models are generally increased well above the minimum feature size of the node [108] - negating somewhat the benefit of using the more advanced technology nodes.

Other than providing a silicon substrate for defining dynamical neural network topologies, neuromorphic processors also support biologically-plausible learning and adaptation algorithms, such as Hebbian spike-timing-dependent-plasticity [109] for long-term memory formation as well as circuits implementing short-term plasticity [110] and also spike frequency adaptation [111]. At the time of writing, the field of neuromorphic computing, in particular the development of neuromorphic processors, is still emerging and currently faces a broad spectrum of challenges ranging from conceptual ones, regarding how to transfer results from computational neuroscience into neuromorphic computing models, to technical ones such as addressing bandwidth limitations and reducing levels of static power consumption due the AER based communication protocol [112]. The field would do well however, to appreciate the long timescales observed throughout the history of AI between the emergence of new ideas and their successful implementation.

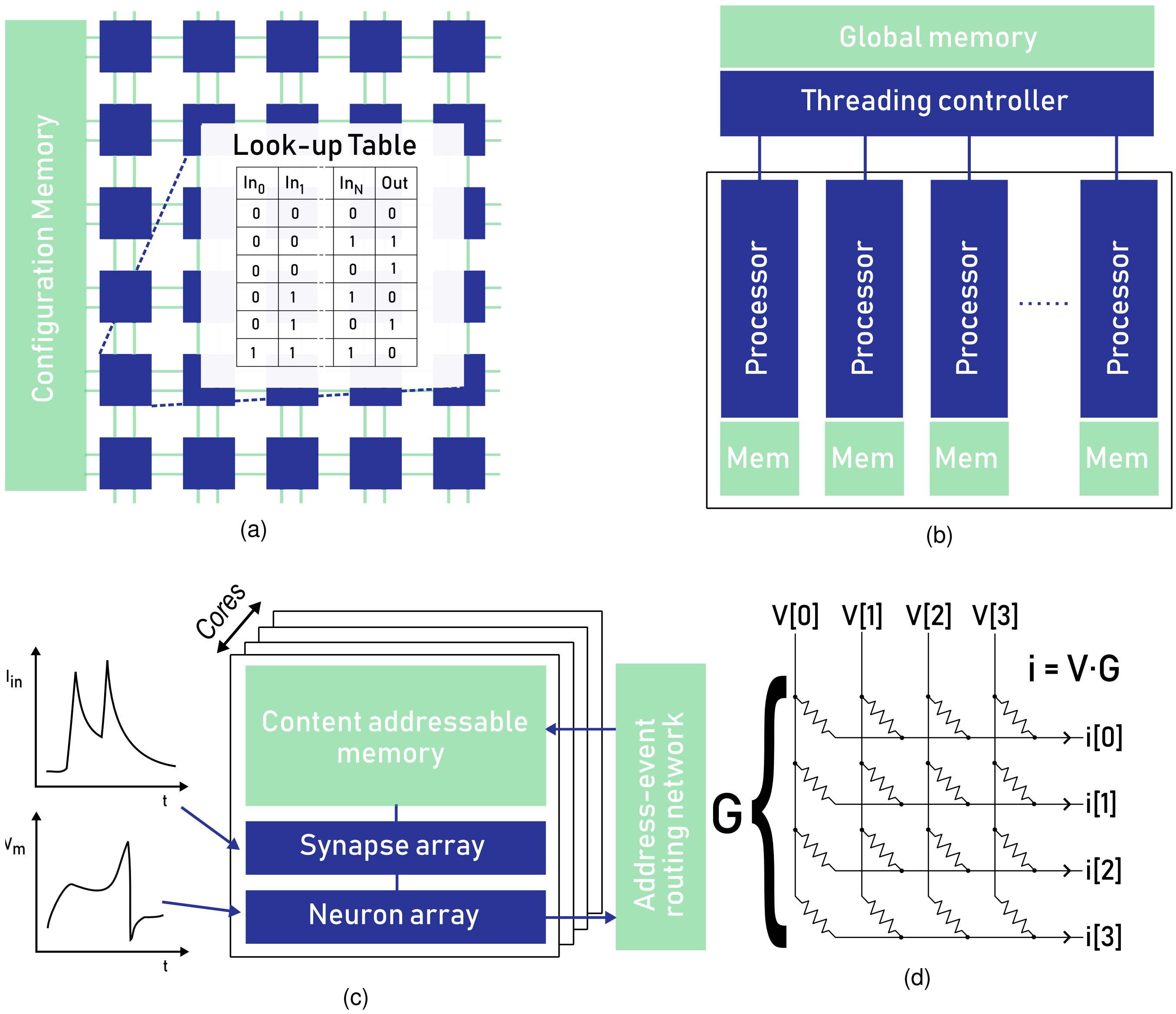


Figure 1.4: **Computing hardware supporting AI computer models.** (a) A field programmable gate array (FPGA) is composed of an array of configurable logic blocks (blue squares) which receive and send digital signals with their neighbouring blocks. The logic of each block is generally implemented using a look-up table. The logic of the look-up table and block inter-connectivity is determined using a memory that is generally shared by a subset of of blocks (b) Graphics processing units are generally composed of multiple processors, each with their own local memory, that execute in parallel on multiple cores. Their execution is determined by a global threading controller and a global memory. (c) Neuromorphic processors are composed of multiple neuron cores, each containing an array of neurons and synapses that, with either analogue or digital circuits, model their dynamical properties. A content addressable memory is used to interconnect neurons on the same core as well as neurons on other cores which communicate through a global address-event representation (AER) interface. (d) A dot(inner)-product operation can be implemented by applying features (V[0], V[1], V[2] and V[3]) of a voltage vector **V** of the columns of a cross-point array of resistive (conductive) elements **G**. The currents (i[0], i[1], i[2], i[3]) that flow out of each row correspond to the vector **i**.

1.2.5 Memory technologies

Computing hardware is dominated by memory. This is as true for the distinct memory centres of von Neumann chips as it is for the distributed memory blocks in FPGAs and neuromorphic processors. Models of artificial intelligence also hinge on memory which provides structure and brings them to life - whether it being the giga-bytes of memory required to define the synaptic weights of AlexNet [72] or the transient memories that are accessed through attractors in Hopfield networks [60, 61]. Memory is a concept that unifies concepts of learning, behaviour and planning in psychology but at the same time is something entirely tangible from a technological perspective. The right kind of memory technology could be the spark that illuminates new opportunities and methods for constructing the next generation of non-von Neumann machines that can support models of artificial intelligence more efficiently than those that exist today.

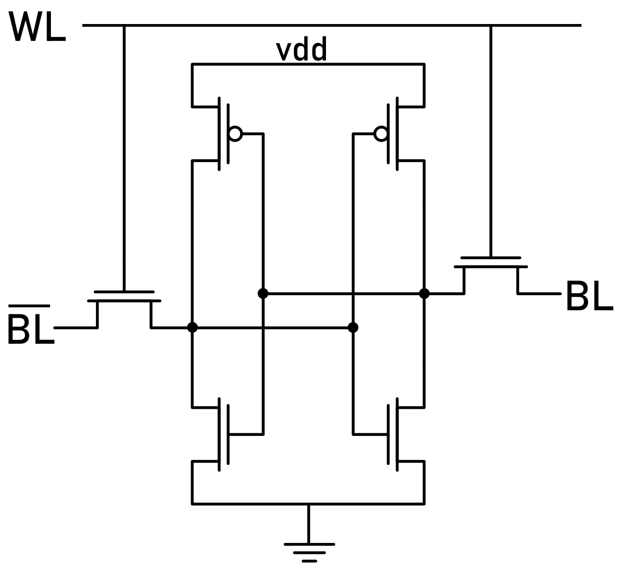
Like AI, memory technologies also have an interesting history, whereby a multitude of physical principles have been harnessed over time as they have become available or were sufficiently well understood. From the time of the Jacquard loom [10], up until approximately the mid-twentieth century, the best way to store information was using a punch card - literally a thick sheet of paper with holes punched into it. The punch card was eventually replaced by binary patterns written onto, and read off of, ferromagnetic strips on the outer surface of a rotating drum in 1940's and 50's [113]. Early computers would go on to use magnetic tapes, a sensible descendent of the rotating drum, as well as delay-line memories in which signals, often a series of rectangular ultrasonic waveforms, would propagate indefinitely in a delay loop medium, liquid mercury for example, and be restored once per loop using a pulse shaping circuit [114]. A computer could then read out serial sections of this circling waveform and compute with it. In the 1960's, both static random access memory (SRAM) [115] (Fig.1.5a) and dynamic random access memory (DRAM) (Fig. 1.5b) were developed that were respectively based on a transistor circuit with restorative feedback and a transistor gated capacitor which was periodically charge-refreshed. These memory cells have since become favourable as they can be realised using the same technological processes that are also used to make arithmetic logic units and today remain the dominant solutions for on-chip working memory'. They are referred to as volatile memory technologies since the electric charge, stored by the respective cells that represents the bit they encode, leaks away in the absence of a power supply. The major downsides of such volatility are the static power consumption, required for information to be retained, and the fact that each time a system is power cycled information has to be reloaded into working memory from a storage class non-volatile memory. In order to store information in such a non-volatile fashion, magnetic hard-disk drives are largely used in modern computers. In 1968, an alternative means of realising a non-volatile memory, the first since the punch card that did not rely on magnetic properties, called the floating-gate transistor [116] was proposed. The floating gate transistor works on the principle of injecting charge into a 'floating gate' by means of electron tunnelling, or hot electron injection, that takes place under application of a large voltage over the device. The charge injected into the floating gate persists in the absence of a power supply and, when a read voltage is applied to the standard gate of the transistor, the current that flows in the transistor channel is determined by the quantity of this stored charge - therefore denoting the value of the stored bit. The floating gate transistor would later be used by engineers working at Toshiba in Japan to realise NOR and NAND flash non-volatile memory cells in 1984 and 1987 which respectively

use series or parallel combinations of these transistors [117, 118].
Since the turn of the 21st century, resistive memory (RRAM) devices, which use the resistance of a two-terminal nano-device as a means of non-volatile storage, have been proposed based on a variety of physical mechanisms (Fig 1.5c). Such mechanisms include the presence of conductive oxygen vacancies (OxRAM) [119] or a conductive bridge of metal-ions (CBRAM) [120] in an insulating oxide, the extent of crystallisation of a glass alloy in phase change memories (PCM) [121] and the difference in magnetic polarisation between two ferromagnetic material layers separated by a thin insulating tunnelling layer [122] (MRAM). Ferroelectric materials have also been investigated in a similar device topology to the floating gate transistor whereby a ferroelectric material is used as a transistor gate-dielectric (FeFET) and, through the non-volatile modification of its polarisation, the read current of the transistor also changes to reflect the value of the stored bit [123]. RRAM are claimed to be a 'missing' fourth circuit element predicted in 1976 called the memristor: a device whose resistance varies as a function of the history of charge that has flowed through it [124, 125]. This is hotly disputed however [126], and in this thesis the term resistive memory, or RRAM, will be used exclusively. In any case, the resistive state (or conductive state) of RRAM can be programmed by applying voltage and current waveforms over the two device terminals to store binary bits [127] or multiple bits per cell [128] in a non-volatile fashion. However, the resistive states assumed after programming operations are subject to a high degree of randomness whereby, upon identical programming operations, devices assume, often drastically different, resistance states between successive programming iterations and between devices - respectively termed 'cycle-to-cycle' and 'device-to-device' variability [129]. Such variability imposes severe limits on the number of bits that can be reliably stored in a cell with respect to other memory device approaches.

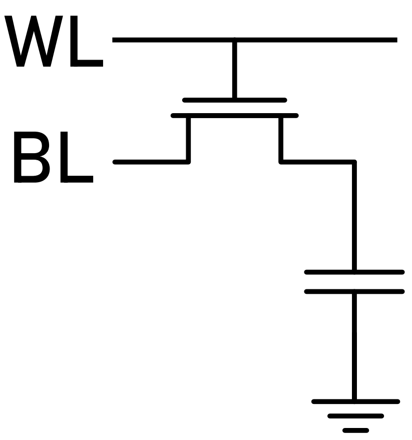
One of the principal interests in resistive memory devices is to leverage their non-volatile conductance states as analogue weight elements in non-von Neumann hardware implementations of neural network models. In particular, much like GPUs, as a means of parallelising the dot(inner)-product operations that are required in the forward propagation of an input data point to an output inference and also in the back-propagation operations required during model training. Specifically, a dot-product operation is evaluated between a matrix of conductances, organised in a cross-point array, and a voltage vector, corresponding to features of an input data point, that is applied across the array columns. The currents flowing out the array rows correspond to each element of the vector resulting from the dot-product between the voltage encoded data and the conductance based model (Fig. 1.4d). The key advantage of such an approach is that the computation occurs inside of the memory, exploiting nothing more than the physical laws of Ohm and Kirchoff - potentially circumventing a von Neumann bottleneck in the computing system [130]. To accommodate this analogue-domain parallelisation of the dot-product operation however, serious practical issues remain regarding the energy and area required due to the repeated digital-to-analog and analog-to-digital conversion [131, 132] and 'pitch matching' the low-area memory cross-points with large area peripheral circuits [133, 134]. Further problems exist in the high programming voltages, high programming currents and high read currents required [135] that not only entail large quantities of energy but pose challenges in the integration with advanced technology nodes that use much smaller nominal supply voltages. The first such neural network inference chip, whereby a neural network model was trained 'ex-situ', or 'off-line', on a von Neumann computer and then transferred onto the hardware which would perform only inference with the model, was proposed in 1991 and used the drain-source conductance of transistors, whose gates were biased by charged capacitors, as the analogue memory element [136]. This was shortly followed by further inference chips employing the conductance states of FeFETs [137] and floating gate transistors [138] as the non-volatile synaptic weight elements. The first inference chips based on arrays of re-configurable two terminal RRAM devices were then proposed in the next decade [139]. Despite proposals to realise neural networks without overlapping memory cross-points [140], the crossbar structure has come to dominate proposals for such dot-product (sometimes called multiply and accumulate) 'engines' and their design and fabrication remains, at the time of writing, an active research topic. In order to solve leakage current, or 'sneak path', issues in RRAM arrays, multiple device selection solutions have been proposed whereby diodes [141], transistors [142] or ovionic threshold switching thin-films [143] act to select specific devices in a memory array for reading and programming. The configurability of resistive memories, under the application of relatively simple voltage pulses that can be generated locally has also prompted research into the implementation of machine learning algorithms 'in-situ' - also referred to as 'on-chip' or 'on-line' - for feed-forward [144, 145, 146], long-short-term-memory [147, 148] and spiking neural networks [149, 150, 151, 152]. When contrasted with ex-situ model training, in-situ training allows for models to be updated locally and, as such, the approach is therefore a promising avenue for bringing adaptability and learning to edge systems. Additionally, in-situ trained models have also been observed to accommodate for local hardware and device specific defects and variability [146].

Despite open questions regarding how to best harness its properties, RRAM is certainly a candidate for the 'right' kind of memory that could well lead to a new generation of low-energy non-von Neumann computers. Although their application has thus far been largely confined to either ex-situ [153] or in-situ [145, 146, 147, 148, 152] implementations of backpropagation for training deep learning models, other examples of non-von Neumann computing based on RRAM properties also exist. This includes their use in realising logic circuits [154] as well as in true random number generation [155]. Additionally, intrinsic resistive memory randomness has been leveraged as a means of solving optimisation problems in RRAM-based Hopfield networks [156]; in a similar fashion to how the spin-spin coupling of superconducting quantum bits has been used to implement quantum annealing algorithms in conceptually similar Ising machines [157]. Furthermore, while strictly not acting as a memory, the random switching behaviour of MRAM when driven by a constant DC bias current has been applied to perform Bayesian inference; using a tree of C-Muller elements [158] to evaluate the joint probability of input random variables [159], each represented by an MRAM device, or by interconnecting the devices to realise a Bayesian network [160].

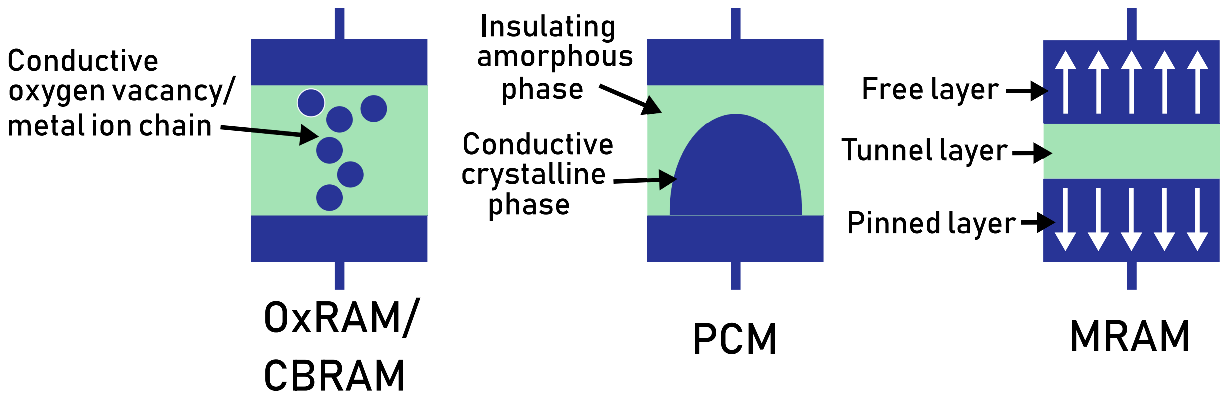
Future memory concepts such as magnetic racetracks [161] which store bits as a sequences of magnetic 'domain walls' in nanowires, molecular memory cells that use switchable properties of organic molecular structures to store information [162], quantum-dot based flash [163] as well as quantum mechanical memory [164] promise further properties and opportunities that can also be one day harnessed in computer models of artificial intelligence.



(a)



(b)


Conductive oxygen vacancy/ metal ion chain
Insulating amorphous phase
Conductive crystalline phase
Free layer
Tunnel layer
Pinned layer
OxRAM/ CBRAM
PCM
MRAM

(c)

Figure 1.5: **Common memory bit cells** (a) A six transistor static random access memory (SRAM) cell whereby the cell is selected for programming or reading using the wordline (WL) and the binary bit that is present within the cell can be readout based on the current flowing out of the bitlines (BL). (b) A dynamic RAM (DRAM) cell whereby the wordline (WL) is used to read the bit, stored as a voltage on the capacitor, of the cell. DRAM requires a refresh controller in order to keep the capacitors charged or discharged appropriately. (c) Three different types of resistive memory (RRAM) cell are shown - (left) oxide/conductive-bridge RAM (Ox/CBRAM), (centre) phase change memory and (right) magnetic RAM. Arrows on each two-terminal device indicate the conductance mechanism which is used to store the bits within the resistive/conductive states of the cell. White arrows in the MRAM cell indicate the spin polarisation of the magnetic layer.

1.3 Challenges in applying artificial intelligence

There are obvious conceptual challenges in artificial intelligence, namely the question of what intelligence even is and how to go about building models of it on computers are still very much open questions. The best black-box and tree-based approaches developed at the time of writing, although arguably not viable models for real intelligence, are certainly capable of reproducing specific intelligent and useful behaviours in isolation. As such it is now possible to apply AI models, neural networks for example, to solve real-world problems. However, considerable challenges still need to be confronted such that existing approaches can be applied in a practical and responsible fashion.

1.3.1 Training energy consumption

Artificial intelligence, deep learning above all, has benefited enormously from the aggressive transistor scaling that has increased the speed of computers and the availability of on-chip memory. However, to obtain state of the art performance in a specific task the energy and time required to train a model is often truly staggering [3, 4]. The AlphaGo model from DeepMind, for example, was trained over the course of a month using on the order of thousands of CPUs and hundreds of GPUs [74]. Knowing that a single GPU has a power consumption on the order of hundreds of Watts [101], a back of the napkin calculation suggests the total energy demanded by the GPUs alone is on the order of millions of kilowatt hours. To put this number in context, an average French household consumes on the order of a thousand kilowatt hours of energy in an entire year [165]. Is it reasonable that the same amount of energy to sustain one thousand households in a year was used to train a deep learning model to play Go in a month? At the time of writing, the AlphaGo model was produced four years ago. Since then model sizes have increased exponentially in order to achieve incremental performance gains [5]. To train the Transformer model proposed by Google in 2017 [77], greenhouse gas emissions equivalent to six hundred plane tickets between New York and San Francisco or the total fabrication and running costs of five cars during their lifetimes was estimated to have been required [3]. The worlds largest model at the time of writing uses almost two hundred billion parameters [80]. It has been estimated that the electricity usage alone to train it cost five million US dollars. Limited to a single GPU, this training process would take 355 years to conclude [166]. Considering the rapid growth in the application of these extremely energy intensive AI models, which is projected to be sustained well into the future [1], serious questions have to be posed about the sustainability of allowing the field to continue in this fashion and, ultimately, end up as a major contributor to climate change.

1.3.2 Bringing learning to the edge

As a result of these energetic requirements, the majority of applications for AI hinge on centralised cloud computing infrastructures [167]. This means that models are trained, stored and often queried by edge computing hardware in the cloud - where massive amounts of energy, memory and computing resources are readily available. To reduce the latency and the volume of data transferred between the edge and the cloud, models can also be trained on the cloud and then transferred to dedicated edge hardware that acts to perform local inference with that trained model [168, 169, 170, 171]. However, no commercially available hardware exists that can meet energy usage and memory size constraints that would enable intelligent systems to be trained and adapt themselves locally at the edge [169, 2]. While for many application domains this cloud-centric approach to AI is perfectly adequate, it does mean that a number of potentially revolutionary use cases are being overlooked and not subject to investigation. For example, implantable medical systems such as cardioverter defibrillators [172], which are required to intervene with an electric shock upon detection of dangerous heart arrhythmia, have existed for some decades. Such systems might benefit tremendously from the local and patient specific application of AI that could allow their operation to be continuously adapted and optimised. Once implanted however, these medical systems may have intermittent, or no access at all, to an external power supply. Other than the tremendous heat that would be generated, a deep learning model based on a GPU within the implantable system would evidently quickly exhaust any local battery. Further than restricting the areas in which AI can be applied, the cloud-centric approach also influences the direction of research into new modelling and algorithmic ideas. In recent years , research interest has been overwhelmingly focused on the brute-force mathematical optimisation, framed as supervised learning problems, of extremely large models using pre-labelled 'big data' datasets on cloud platforms. Such a learning approach is not suitable in the edge setting. At the edge, by contrast, data is input to the model in a continuous and unlabelled stream in real-time from the environment and, due to memory constraints, likely will not be stored. Rather, ideas focusing on how to understand, learn from and respond to patterns in such a label-less deluge in real-time, on how to deal with previously unseen types of data and compensate for environmental changes and drift where the edge system exists would be preferred. At the time of writing however, since they are not required in the cloud-centric context, such modelling and algorithmic approaches either do not exist or are not being actively pursued at large by the community.

In spite of this, the market for edge learning is expected to emerge and grow rapidly over the coming decade [2]. To enable this, the solution can be viewed from two angles: either edge hardware must be made more efficient or models of AI must be improved such they demand fewer resources. Aspects of both are undoubtedly true. To bring locally adaptive intelligence and learning to the edge, potentially opening up new application domains that are not being considered today, simultaneous and cooperative innovation in hardware and modelling are required.

1.3.3 Ethical application of AI

Unlike the goal-tree based approaches applied in the 1960's [41] which, after performing an action, could offer a reason as to why, the outputs of black-box models like neural networks often suffer from a lack of, or zero, interpretability [6]. While harmless in applications such as product recommendation or in customer service chatterboxes, serious ethical questions arise for safety-critical applications. For example, autonomous driving and medical diagnosis - two of the most potentially transformative emerging markets for AI. For example, if a deep learning model takes an erroneous action that results in a fatal traffic accident, how would a justice process proceed without being able to interpret the actions of the model that lead to it? Or in the medical setting, if a model arrives at a certain diagnosis in a complex problem studied in a patient, should a doctor continue with, and take responsibility for, that course of treatment if they themselves are not able to arrive at the same conclusion and not able to interpret why the AI model itself has arrived to such a conclusion?

Following a similar set of arguments as the case for interpretability, various approaches to AI are not able to hold-up their hands and say 'I don't know' or 'I am uncertain about this output'. Examples of this exist whereby an image of the hand-written digit '0' is presented to a model which has been trained to recognise cats and dogs leading to a prediction that the input data point was a cat. Reciprocally, for models trained to recognise hand-written digits, given an input of a cat, they might tell you a '7' has been presented to it. This problem arises by viewing, in the case of neural networks, output layers of neurons as a discrete probability distribution through, most often, the application of a 'softmax' function. The output softmax ensures all of the output responses, encoding the probability of an input data point belonging to a pre-determined set of classes, sum to one [72]. Therefore, when presented with classes of data on which a model has not been trained, or has observed very few examples of, it is obliged to make a prediction such that the total probability across the output sums up to one in any case without communicating any level of uncertainty about this predictions. For AI models to communicate uncertainty, they have to incorporate uncertainty into their constituent parameters. This uncertainty will then in turn propagate through a model into its output predictions [173, 174].

A class of models which describe and communicate uncertainty in this fashion are 'Bayesian models' - named in homage to the Reverend Thomas Bayes. Bayesian models, instead of using deterministic values to describe something like a synaptic weight in a neural network, make use instead of probability distributions (i.e., random variables) [7]. In this fashion, uncertainty in the estimation of each parameter can be incorporated into each of the probability distributions which describe them [175, 173, 174] and in fact becomes and integral part of their training process. For example a synapse with the value of 2.5 in a deterministic neural network might be described by a normal distribution in a Bayesian neural network where, if it was quite sure about this value, the normal distribution would be narrow (have low standard deviation) around a median of 2.5. If it were less sure about this value, the normal distribution might be very wide and spread its total probability density over a larger range of values. This parameter uncertainty, as aforementioned, then propagates through Bayesian models and manifests in their output prediction uncertainty. As a result, a model trained to recognise cats and dogs, when presented with the digit '0', might tell you, via its output prediction distribution, 'that looks more like a cat than a dog although, in all honesty, I have never seen anything like that before and I am very uncertain'. For some applications, a cloud-based model tasked with selecting car insurance policies for example, the consequence of taking wrong actions with a low frequency is largely unimportant. If the model achieves a prediction accuracy of 99.99$\%$, the company will certainly lose some money in a small number of large money payouts, but the money likely saved through incorporation of AI into their process will overall most likely make it worthwhile. Uncertainty is extremely important however, just like interpretability, in safety-critical applications where a single wrong decision could cost a human life. Consider an AI model tasked with controlling the actions taken by a cardioverter-defibrillator: an implantable medical device which applies electrical shocks to the hearts of patients in the event of detecting dangerous heart arrhythmias in order to restore a normal heart rhythm [176, 177]. While the detection of a true positive could save a patients life, the erroneous application of a shock to a properly functioning heart could prove fatal [178]. A model which takes such heavy decisions, where there exists an imbalance between inaction and incorrect action, should not be obliged to act when it is unsure. While a deterministic model does not know when it is unsure, it is possible to look at the output prediction distributions of a Bayesian model to understand, with what level of certainty, an action should be taken or not.

1.4 Scope of this thesis

The objective of the work presented in this thesis is to explore the inter-disciplinary boundary between the fields of artificial intelligence, emerging technologies and biological nervous systems. Specifically it will consider how resistive memory technologies can be used to support ultra-low energy approaches to AI based on neural network, or connectionist, models that can be applied in the resource constrained setting of the edge. The key will therefore be the exploitation of the non-volatility and in-memory dot-product capabilities of resistive memory, while finding methods to work with, or leverage, the other non-ideal characteristics of resisitve memory. For example, the large levels of conductance variability that currently hold back its practical application.

In order to achieve this, Bayesian machine learning approaches, compatible with the fundamental device properties of RRAM, are developed and applied experimentally to the training of RRAM-based models. Additionally, a computing fabric based on distributed resistive memory devices, capable of supporting re-configurable neural network models and inspired by the organisational principles observed animal nervous systems, is proposed. Circuits, implementing the building blocks of this computing hardware are designed, fabricated in a hybrid 130nm CMOS-RRAM process and then experimentally verified. Two neural network models inspired by two different insect sensory systems, the cricket cercal system and the *Drosophila* elementary motion detection system, are also developed. These cricket and *Drosophila* models respectively show that such bio-inspired architectures permit the memory requirements of neural network models to be reduced by orders of magnitude with respect to conventional architectures, and that bio-inspired dynamical neuron and synapse models are important for addressing tasks with temporal dependencies.

The thesis is organised into three chapters, addressing each of these three aspects independently. While they are written to stand on their own, each chapter compliments aspects of the others and together they are intended to paint a picture of a Bayesian neuromorphic computing approach based on resistive memory. The first chapter presents the two bio-inspired models and their advantages, the second describes RRAM-compatible Bayesian machine learning algorithms that can be applied to train resistive memory based neural networks and the third chapter presents a circuit and system level description for a re-configurable RRAM-based computing fabric on which such models can be implemented. The thesis concludes with a future vision of a full Bayesian neuromorphic computing system. It is discussed how properties of the bio-inspired neural network model architectures from the first chapter might be important for addressing potential limitations foreseen in the presented machine learning and hardware approaches.

Ultimately, the work provides a blueprint for a new approach to artificial intelligence by combining ideas at the interface (Fig.1.6) of machine learning, technology and biology, each of which plays an important, although normally independent, role in the field. The resulting computing system, by drastically reducing the energy required to implement machine learning, offers a means of bringing learning to extremely resource constrained environments at the edge. Due to certain properties of the bio-inspired and Bayesian AI models developed, namely their abilities to be interpreted and express uncertainty, their value in addressing questions regarding the ethical application AI are also discussed. As a result, it opens up the door on a new set of extremely low-energy safety-critical application domains, which cannot be addressed with existing approaches – implantable intelligent and adaptive medical systems for example.

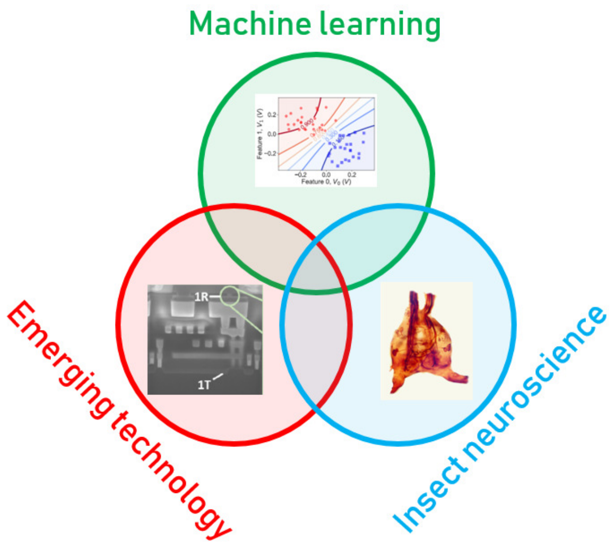


Figure 1.6: A Venn diagram showing the three predominant research fields, and the emphasis on studying their intersections, covered in this thesis.

Chapter 2

Bio-inspired neural network architectures

2.1 Chapter introduction

Intelligence in animal nervous systems, contrary to prevailing notions in AI, is more than just learning. A great number of animal behaviours such as the detection and appetitive response to blood in sharks [179], escape responses in mice, flies and crickets [180, 181, 182, 183, 184] as well as complex courtship and burrowing behaviours observed in a wide variety of species [185, 186] are built-in innately through neural network architecture and neural dynamics from birth. The genome of an animal, carved out and refined during many hundreds of millions of years of evolution, contains the full instruction set which details as to how all of the neural architectures that compose a nervous system are built and interconnected. The mammalian genome however, composed of some billions of nucleotides, stores the equivalent of a mere gigabyte of information [187] - and that is to build the entire animal, not just its brain. A single gigabyte pales in comparison to the memory requirements of even modest neural network models currently being applied in AI. This alludes to the existence of a smaller set of fundamental, task specific, neural architectures and computational principles which are recycled across multiple neural systems. In fact, even across wildly different species separated by half a billion years of evolution, common architectures and computing principles are observed [188]. This chapter will seek to develop bio-inspired model architectures based on neuro-physiological and neuro-anatomical research currently being conducted into two insect sensory systems - the cricket cercal system and the *Drosophila* elementary motion detection system. Overall, the purpose is to understand as to how connectionist AI approaches might unshackle themselves from ‘pure learning’ [189], and what the advantages may be in doing so, by taking inspiration from the innate mechanisms uncovered by evolution and stored in the animals constrained genome.

2.2 Model of the cricket cercal system escape response

2.2.1 Introduction

Neurons are the fundamental computational units of connectionist approaches to artificial intelligence [27, 28]. They can be understood computationally as defining a linear hyper-plane in an input feature space through application of an activation function to a weighted sum of these features. While alone these neurons can solve only simple ‘linearly separable’ tasks; they can, when networked together in a neural network architecture, solve more complex non-linear ones [190]. The predominant approach in neural networks in recent years has been that of deep learning [71], which can attribute its success to the combination of two important ideas. The first is the universal approximation theorem [52] which states that a fully-connected feed-forward hidden layer of neurons is capable of approximating any continuous function - given an appropriate combination of synaptic parameters. The second is the use of batch backpropagation stochastic optimisation [37, 51] as a means of determining these parameter values that link together successive layers of neurons. However, deep learning models with hidden layers are characterised by non-convex loss surfaces. As such, backpropagation will converge to a locally-optimal configuration of parameters - not necessarily the global optimum - as a function of their random initialisation. In order to improve the quality of these local minima the solution has been to include many, hence deep, wide neural network layers [191], although this ultimately leads to a unwieldy and memory inefficient model which is difficult, arguably impossible, to interpret [6].

An important approach within deep learning, first proposed in the 1980’s as the ‘neocognitron’ [50], has developed into the field of convolutional neural networks (CNNs) which achieve state of the art performance in applications related to image processing. This is in spite of the fact CNNs are composed of a subset of the parameters of a fully-connected feed-forward network of an equivalent depth [72]. Crucially, CNNs, originally inspired by research into the cat visual cortex [31], have demonstrated the potential of applying backpropagation to task-appropriate neural network architectures inspired by biology - as an alternative to the use of universal approximators. Other deep learning architectures, namely attention models [76] and generative adversarial networks [78], have lead to respective leaps in machine translation and novel data-point generation that further reinforce the importance of architecture in neural networks.

In the field of neuromorphic computing, where a more ‘bottom-up’ approach is taken to artificial intelligence, research into biological nervous systems has, as in the case of CNNs, also provided a source of architectural inspiration. This has led to models which, for example, incorporate dynamical and topological motifs inspired by the *Drosophila* visual, the honey-bee olfactory, cricket auditory and the cockroach motor systems into motion detection [192], contrast enhancement [193], temporal pattern detection [194] and locomotion [195] models respectively. However, such approaches have been somewhat limited by lack of an effective means of defining model parameters, whereby manual parameter tuning or correlation-based Hebbian learning rules [196] are typically employed.

Here, we propose a computational model for the air current evoked jumping escape response studied in the cricket cercal system. Instead of manual tuning or Hebbian learning, we employ the backpropagation algorithm, more commonly used in the deep learning setting, as a means of model parameterisation. In contrast with deep learning, however, backpropagation is not used to find an arbitrary local loss minimum based on a random parameter initialisation, but instead as a means of steering the parameters towards optimal values consistent with the logical structure of the bio-inspired architecture that relates neurons to one another.

We find that, when applied to the detection of a simulated attacking predator, the optimised cercal system model is able to obtain the same performance as benchmark universal approximators, although requiring between one and two orders of magnitude fewer parameters. Additionally, we also find that, because the model contains neurons and synapses with well defined functional roles, it is possible to interpret how the optimised model functions and makes decisions. Ultimately, the results provide a strong basis for the incorporation of biologically inspired architectures into memory efficient and interpretable neural network models.

2.2.2 Model definition

The cricket cercal system

The cricket cercal system has been under investigation for several decades, leading to an understanding of many of its neural and biomechanical components (reviewed extensively in [197, 183]). The nervous system of the cricket is characterised by a chain of ganglia along a nerve cord which runs from the head of the animal down to the rear of the body pictured in Fig. 2.3b [198, 199]. Signals from various sensory and motor systems ascend up or descend down the nerve cord between the ganglia where information is processed to mediate the animal's behaviour. The interneurons making up the cercal sensory system are contained in the Terminal Abdominal Ganglion (TAG) at the very posterior end of the nerve cord (Figs. 2.1b,2.1c,2.1d). This cercal sensory system mediates the detection and analysis of air currents which activate sensory receptors on the crickets two rear cercal appendages. These appendages, or cerci, are long antenna-like structures, each of which is covered by up to a thousand filiform hairs as pictured in the electron microscopy image of Fig. 2.1a [200, 201, 202]. Below each hair there is a sensory neuron which, under mechanical displacement of the hair, will fire action potentials that propagate along an axon into the TAG (see Fig 2.1c). The ensemble of all of the sensory afferents on the two cerci project to specific locations within the TAG and form a sensory feature map [203, 204, 205, 206]. Different spatial regions of this map represent information about air-current direction and velocity transduced around the animals cercal appendages.

Within the TAG, there are approximately twenty large spiking interneurons, in addition to some thousands of smaller local-interneurons, (Fig. 2.1d) which receive direct excitatory synaptic input from the filiform sensory afferents [207]. These are 'projecting' interneurons: they send axons up the nerve cord to higher ganglia (Fig. 2.3b). There are also an unknown number of small local interneurons within the TAG which do not send axons up the nerve cord [208, 209]. Many of these are non-spiking 'graded release' neurons, that interconnect between themselves and the large spiking interneurons. Each of the large spiking interneurons has a unique anatomical projection of its dendritic trees within the sensory afferent map [210, 204]. As a result, each of these interneurons has a unique specific responsiveness to the different properties of the external air current stimuli. The ensemble activity pattern of these interneurons encodes these stimulus parameters, and sends a compressed representation of this stimulus up the nerve cord towards higher ganglia, which in turn use this information to initiate the appropriate behaviours in other areas of the nervous system. For this study, we restrict our terminal abdominal ganglion model to neurons and stimuli that concern the escape response behaviour.

We have developed a reduced model of the cercal system. It consists of twenty-four neurons: sixteen neurons that represent an integrative input layer, seven neurons in a hidden layer which represents the projecting interneurons within the TAG, and a single neuron in the 'output layer', supposed to exist in a higher motor ganglion in the animal, whose response determines whether a jumping escape response should be initiated or not. The characteristics of the neurons in these three layers, and the rationale

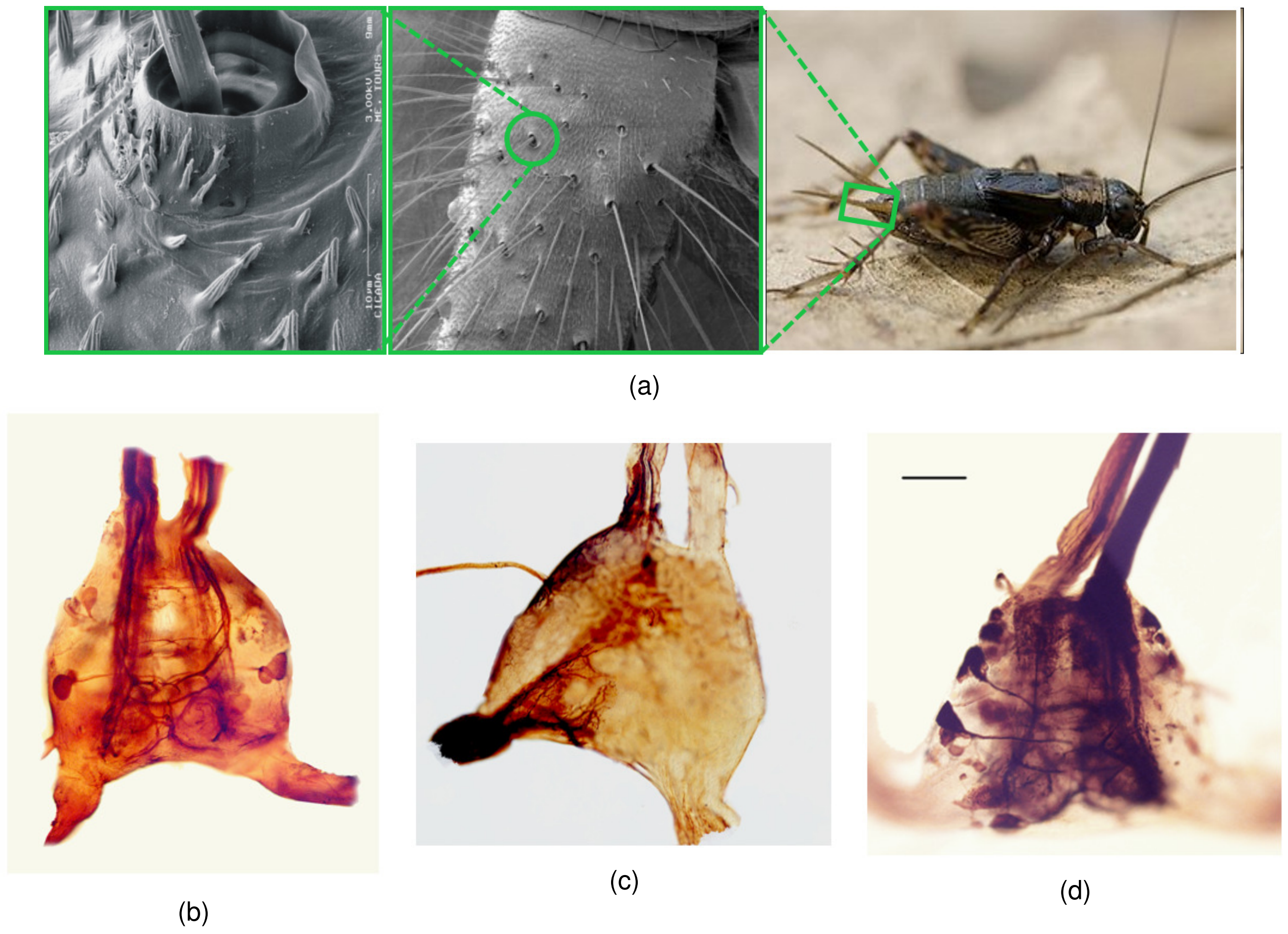


Figure 2.1: **Optical and scanning electron microscopy images of the cricket and its terminal abdominal ganglion.** (a) Three panel image showing different extents of zoom of the cricket cercal hairs. (right) An image of a wood cricket. A green rectangle highlights the base of its right cercal appendage. (centre) a scanning electron microsopy image of the base of a crickets cerci. A green circle highlights one of the sockets in which cercal hairs pivot within. (left) A scanning electron microscopy image of a cercal hair embedded within a socket. (b) An optical microscopy image of an isolated terminal abdominal ganglion. The two extensions at the base of the image correspond to the neuropil where the sensory neurons on the two cerci project their sensory afferent. The two vertical extensions at the top of the image are the two axon bundles composing the nerve chord that ascends up to higher ganglion. (c) An image of the TAG where a red cobalt chloride dye, appearing almost black in high concentrations, has been used to stain a subset of the afferents projecting from the left cerci into the ganglion. (d) An image of the TAG where cobalt dye, appearing purple in the image, has been used to stain the cell bodies, dendrites and the axonal projections up the nerve chord, of a subset of giant interneurons. Neuron cell bodies can be see clearly on the left-most border of the TAG which send projections, both dendritic and axonic, into the middle of the ganglion.

for their inclusion and inter-connectivty with other neurons, is described in the following sections and diagrammed in Figs. 2.2. Before that however, it is important to introduce the statistical simulation method used to generate the spiking activity of the crickets ensemble of cercal hairs that act as the input for our model.

Statistical Model of Filiform Hairs

In the cricket cercal system, neural processing begins at the cercal filiform hairs, which act as extremely sensitive mechanical filters [211, 201, 212, 213]. There are hairs of many different lengths on each cercus, and the length of each hair determines the range of frequencies, or speeds, of air currents to which it is most responsive. Longer hairs are more responsive to slower air currents and short ones to faster air currents. Each filiform hair has a spiking neuron at its base which is mechanically activated when the hair is displaced. These sensory neurons have been observed to fire even under extremely low background air current intensity, at rates of up to hundreds of spikes per second [212]. The base of each hair is constrained in a complex mechanical socket such that it can pivot freely along only one specific axis. Characterisation of socket orientation distributions on the cerci have revealed four distinct populations - hairs that are sensitive to air currents directions coming from 45°, 135°, 225° or 315° relative to the head of the animal, which by convention points to 0°. Further studies on the distribution of hair lengths revealed a bi-modal distribution of hairs into two broad but overlapping length categories [200]. In functional terms, this general distinction can be used to categorise the two groups of hairs as responding more readily to either slow air currents (long hairs) or fast air currents (short hairs). Given that there are two cerci, each containing populations of long and short hairs that are sensitive to four directions of air current, there are sixteen sub-populations of cercal filiform hairs which respond optimally to specific, restricted combinations of air current direction and speed. A further study noted another response characteristic of a hair as a function of its length. For an accelerating burst of fast air, shorter filiform hairs respond with a reduced latency with respect to longer ones [214]. Whether an anatomical adaptation or a happy coincidence this latency provides a useful signature in the recognition of an attacking predator which would generate such a high speed burst of air.

Based on these findings we developed a statistical model for the generation of spikes for sixteen sub-populations of hairs: one for each air current angle (45°, 135°, 225° or 315°) for each of two hair lengths (long or short) for each of the two cerci (left or right). Each sub-population is composed of N hairs.

This model was used to generate individual simulation runs, each one second in duration, of the ensemble spiking activity of all 16 $\times$ N sensory neurons. Each simulation is characterised by five parameters. Two of these parameters are (1) the *background speed* and (2) the predominant *background direction* of the ambient air currents. These background air currents correspond to the main source of 'noise' that would be expected in a realistic situation that could confound the animal's ability to detect the presence, distance and direction of a predator. Essentially the background air current imposes a variable level of activity in the receptor cells that, in turn, would lead to a background activity in the interneurons of the TAG. The other three simulation parameters correspond to the simulated predator attack. They specify (3) the *attack angle*, (4) the *attack speed* and (5) the amplitude of the air current that is assumed proportional to the *attacker size*. For each individual one second simulation, values for these five variables are random-uniformly sampled between a lower and upper bound. An attack stimulus is presented at either 350ms or 700ms during this one second. The statistical model generates sixteen sets of N sensory neuron spike-times: one per sub-population.

The specific protocol for defining the neuron firing timestamps in an attack sequence was is as follows. For each receptor in a long hair sub-population, a number of spike-times are sampled from a uniform random variable between zero and one second (each sample therefore corresponding to a firing time) equal to the *background speed*. For each receptor in a short hair sub-population, samples were drawn at half of this rate, reflecting their lower baseline firing levels [212]. The receptors in the sub-populations whose hairs were oriented towards the *background direction* drew an extra 25% of samples while those contrary to *background direction* drew 25% less. At either 350ms or 700ms in the simulation, a simulated attack was initiated from behind the animal. The attack is represented by long hair sensory receptors sampling a firing timestamp from a normal random variable centred on the attack time, and the short hair sensory receptors sampling from a normal random variable shifted forwards from the attack time by $-10ms$ - reproducing the response latency effect reported in [214]. The standard deviation of the normal random variable which samples the neuron firing timestamps is equal to the inverse of the *attack speed*. The probability, p, of each sensory receptor sampling a firing timestamp from the normal random variable, therein the probability of the hair responding during an attack, depends on the *attack angle* and the *attacker size*. Hair sub-populations with a preferred direction oriented towards the *attack angle* and the sub-populations on the cerci that is on the same side of the animal as the attack originate from respond with a higher probability. The probability of a single hair sampling a spike-time during an attack is defined as;

$$p = p_\theta \times p_{side} \times s, \tag{2.1}$$

$$p_\theta = (1 - \beta_1) \times cos|\theta_p - \theta| + \beta_1, \tag{2.2}$$

$$p_{side} = ((1 - \beta_2) \times exp(\frac{-|\pi - \theta|}{\alpha}) + \beta_2, \tag{2.3}$$

where θ and θ_p are the *attack angle* and the preferred orientation in radians of the hair respectively. The *attack angle* is bounded to be between 120° and 240°. The *attacker size*, s, is a number between 0.75 and 1, $\beta_{1/2}$ are the minimum response probabilities for p_θ and p_{side}, and α defines how the response intensity decays on the contra-lateral cerci to the attack as a function of the *attack angle*, θ. See Appendix 6.7 for examples of the raster plots generated by this model for different sets of parameters.

Input layer: Feature map neurons

In the cricket TAG, this information on air-current direction and speed, represented by the ensemble firing pattern of the sensory neurons, is preserved through sub-population specific projections of sensory neuron spikes to specific locations of a sensory feature map. In each location of this feature map, the thousands of sensory neuron spikes that propagate from the cerci merge into continuous analogue signals which denote the intensity of a particular environmental feature [203, 215, 204, 205, 206]. TAG interneurons are then observed to 'tap' into various properties of this feature map through specific dendritic arborisations in particular spatial locations of the feature map.

We propose to model this feature map using the input layer of sixteen neurons depicted in Fig.2.2a. Each of these sixteen feature map neurons integrates the spiking input from one of the sixteen sub-populations of hairs on the left and right cerci. Each of our input layer neurons implements a leakyintegration model, in other words a parallel resistor-capacitor circuit fed by a pulsing current source. They can be described using the differential equation;

$$\frac{dV_c}{dt} = \frac{i_{in}R - V_c}{RC}, \tag{2.4}$$

whereby a current pulse i_{in}, is injected onto a parallel combination of a resistor R and capacitor C, resulting in a capacitor voltage V_c which decays in time to zero in the absence of an input current. This model allows the neuron to integrate temporal information within a time window defined by the product of R and C. For an analogue circuit implementation of such a model, see Appendix 6.24a. Based on observations of interneurons in the TAG, which appear to receive excitation above and below a set-point due respective increases and decreases from ambient background air-current levels, we apply a hyperbolic tangent function to the difference between the instantaneous capacitor voltage, V_c, and its average over D previous instances [212]. In other words, under stimulation by average instantaneous background air currents the output of the feature map neurons should be around zero;

$$V_{out} = tanh(V_c - \sum_{x=0}^{D} V_x). \tag{2.5}$$

This results in an activation V_{out} for each of the neurons in the input layer that provides the input for the neurons in the next hidden, layer. Examples of the activations as a function of time for each of these sixteen feature map neurons are plotted in Appendix 6.7.

Hidden layer: Cercal interneurons

The hidden layer is composed of a seven neuron circuit, based on our interpretation of numerous results that have been published in past neurophysiological studies. This group of seven interneurons includes four neurons ‘tuned’ preferentially to four specific air current directions, two interneurons tuned preferentially to different air current speeds, although relatively insensitive to direction of those air currents, and a final interneuron whose activity indicates the overall ‘global’ background air current intensity in all directions. These neurons represent these specific quantities through logical connections to specific subsets neurons in the input layer and, through additional lateral interactions between interneurons, shape their responses in the optimal manner. This network architecture is depicted in the three panels of Fig. 2.2. Each of the sixteen feature map neurons in the input layer are identified using a grid which categorises them by angle, speed and side in Fig. 2.2. The rationale for the connections between the neuron elements are as follows.

- **Directional interneurons**: Past studies have documented several projecting interneurons that are responsive to low velocity air currents from a very restricted range of directions, and suppressed by air currents from the opposite directions; i.e., they are directionally selective [210, 215]. These interneurons have been observed to have dendritic arbors in locations of the TAG feature map which receive sensory excitation from air currents from the direction the interneurons is responsive to [203, 204]. In our model these neurons are labelled as *d45*, *d135*, *d225* and *d315*. Each of these interneurons receive input from the corresponding slow feature map neurons from both the left and right cerci as in Fig. 2.2a.

- **Speed interneurons**: Other interneurons have been observed to be responsive to a much broader range of directions and instead appear predominantly sensitive to different air current speeds from the rear of the animal [215, 208]. In Fig. 2.2a, these are labelled as *slow* and *fast* that respectively receive input from the feature map neurons encoding air currents coming from behind due to slow and fast stimuli (originating from long and short receptor sub-populations on the cerci respectively).
- **Global regulation interneuron**: The seventh interneuron in the model is defined as the 'global regulation' cell, labelled as *glob* for short in Fig. 2.2c. Extracellular recordings of the ascending spiking activity from the TAG have shown, somewhat counter-intuitively, that the giant interneurons are more active in a relatively calm laboratory setting than in the field, where background air current intensity is considerably greater [180]. That has lead to the proposal of a global regulation, or a background subtraction mechanism, at play in the cercal system. Furthermore, interneurons have been reported in the cricket TAG which are responsive to low speed air currents coming from all directions [209]. Inspired by these two findings we propose to include the interneuron *glob* in the cercal escape system model as depicted in Fig. 2.2c which receives input from the slow feature map neurons in all directions on the left and right cerci and then sends regulatory synapses to the other six interneurons in the hidden layer.

As well as the specific connection pattern from the feature map neurons in the input layer to the interneurons in the hidden layer, we also incorporate functional lateral connections between these interneurons. Lateral inhibitory connections have been proposed in the cercal system [210, 215] based on observations of direction encoding neurons being inhibited under air current stimuli coming from the opposite direction to which they are sensitive. Detailed neuro-physiological data about the neural basis for these interconnections is however very sparse, although several non-spiking local interneurons have been studied [208, 209] which could well implement these connections. In our model, we do not explicitly include any extra local interneurons to mediate these lateral interactions. Rather, we define direct all-to-all synaptic connections between each of the four directionally selective neurons and between the two speed encoding neurons in addition to uni-directional synaptic connections from the speed encoding neurons to the four directionally selective ones. These connections are depicted in Fig. 2.2b. Instead of defining these as inhibitory connections we allow their sign and magnitude to be determined through the synaptic parameter optimisation process that is described in the following section. Each of these seven interneurons, like those in the input layer, implement hyperbolic tangent activation functions on a weighted summation of their inputs and a bias term. The bias terms of each of the neurons determines the activation of the neuron in the absence of external input, a parallel to the resting membrane potential in the biological cells.

Output layer: Jump neuron

In the animal, it is supposed that the resulting cercal representation ascends to a higher motor ganglion, one of those pictured in Fig. 2.3b, which, given a certain activation pattern, triggers a central pattern

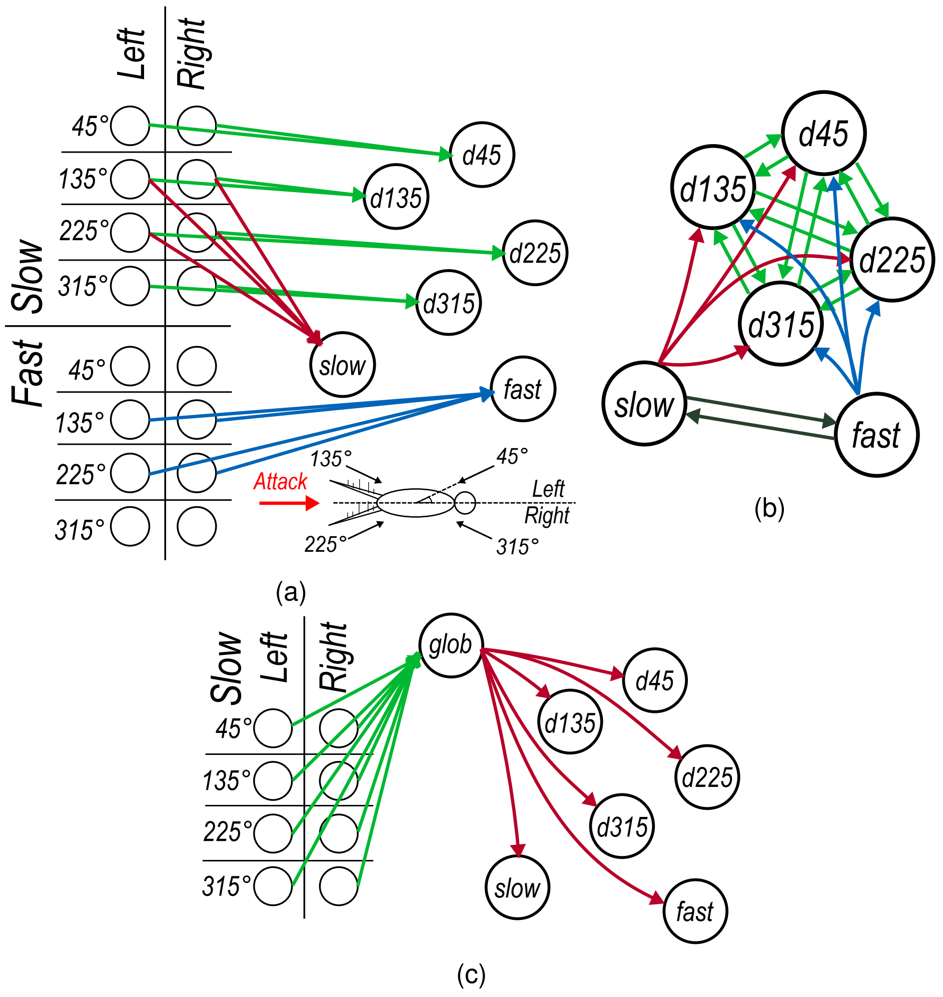
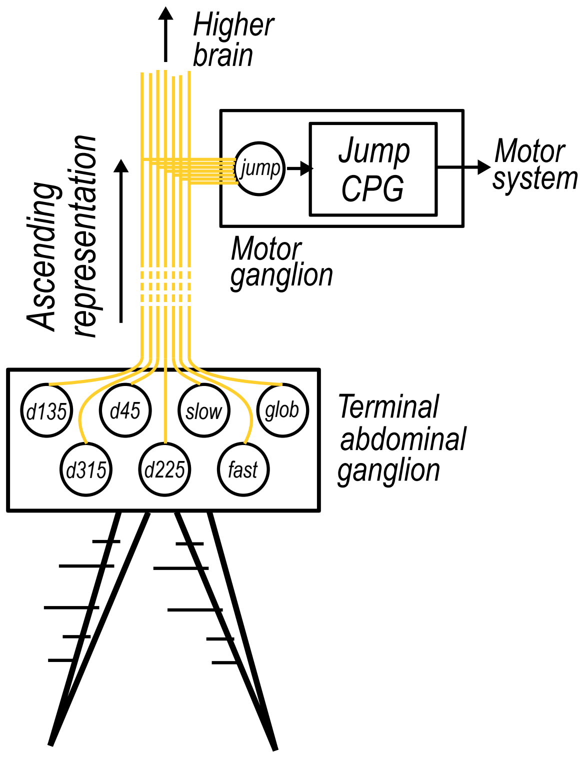


Figure 2.2: **Architecture of the cercal system escape response model.** Colours of synaptic connections do not correspond to synaptic inhibition or excitation but instead are used to help distinguish between sets of synapses. (a) The feed-forward connections between each of the feature maps neurons and the six interneurons each of which encodes a particular property of the input. (b) The lateral connections between interneurons, including the (green) mutual connectivity between the directionally selective neurons, (red and blue) the lateral connections from the air speed sensitive neurons onto the directionally selective interneurons and (black) the mutual lateral connections between the air speed sensitive neurons. (c) The global regulation network which receives (green) synapses from all of the low frequency air current feature map neurons and sends regulatory synaptic connections to each of the six interneurons.

generator to coordinate an escape motor response. Therefore we propose that each of these seven interneurons in the hidden layer of the model synapse onto a final *jump* neuron as depicited in Fig. 2.3a.

In our model, the jump neuron implements a sigmoid activation function on the weighted sum of the interneuron activations, and a bias term. In this fashion, the output can be viewed as the probability that the original ensemble spiking activity pattern of the sensory neurons on the two cerci represents the signature of an attacking predator.

In order to determine the synaptic parameters that connect the feature map neurons to the interneurons, the interneurons to the jump neuron as well as the lateral connections between the interneurons, we apply the backpropagation algorithm [51]. To obtain a dataset that can be used to train and then verify



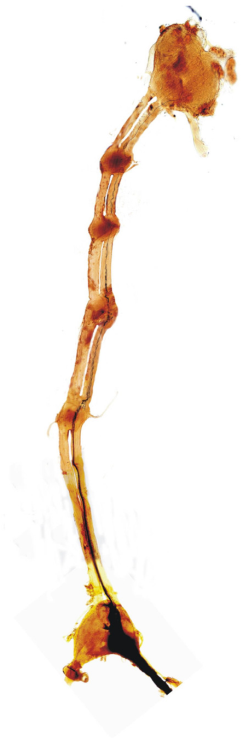


Figure 2.3: (a) The activations of the seven interneurons included in the cercal escape system model ascending up the nerve cord to a higher ganglia. Such a higher ganglion is proposed to contain a neuron or group of neurons, here labelled jump, that looks for certain patterns of activation in the seven interneurons to activate a jump central pattern generator, labelled as jump CPG, to initiate an escape motor routine. (b) An optical microscopy image of the ganglion chain of the wood cricket, stained using a cobalt chloride dye. The terminal abdominal ganglion sits at the base of this chain which ascends up to the central brain contained within the head of the animal. Intermediate ganglion, which for example control one of the insects motor systems, appear as more densely stained red dots along the chain.

the trained model, the feature map neuron activations, V_{out}, at 350ms and 700ms of 1000 randomly parameterised simulation episodes are used. The two resulting V_{out} vectors, of the sixteen activations per simulation, are each then labelled as 1 (attack) or 0 (ambient) as well as with the predominant *background direction* during that episode. This yields labelled dataset of 2000 points of sixteen features each. A random subset of half or these points is used to train the model and the remaining half is reserved to test the model.

A multi-objective cost function was defined as the sum of the binary cross-entropy loss of the jump neuron, with respect to the attack or ambient label, and the categorical cross-entropy of the four directional sensitive interneurons, with respect to the one-hot encoded *background direction* label. Intuitively this optimises the model to simultaneously detect an attack signature and determine the prevailing direction of the background air currents. An additional weight decay term [216] was summed with these two losses in order to penalise the optimisation converging to a solution with large parameters values and, possibly, over-fitting to the test data set.

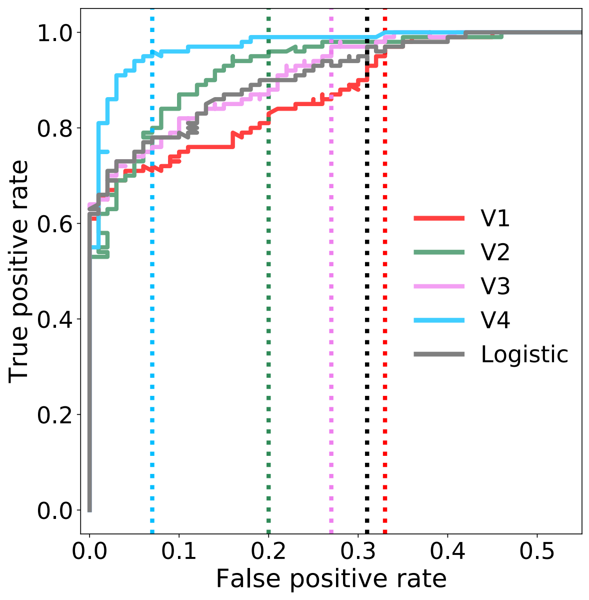
In order to implement the lateral connections between the interneurons without introducing cyclic loops into the model, these synaptic connections were ‘unrolled’ by one step. This is required since the gradients that are backpropagated would otherwise circle indefinitely. This was achieved by applying the activation function on the weighted sum of the inputs from the feature map neurons and the *glob* neuron in a first step and then applying a second activation to the sum of this first activation, the weighted sum of the lateral connections and the bias term.

The derivative of the mean of this loss over a mini-batch of eight training data points was used to update model parameters inline with the root mean square propagation stochastic optimisation algorithm [217] over 50 training epochs. The PyTorch automatic differentiation python framework was used to build and train the proposed TAG model [218].

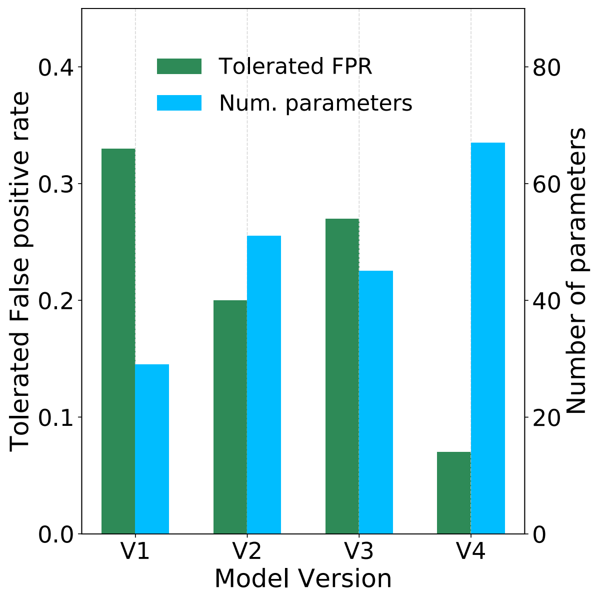
2.2.3 Model evaluation

In order to evaluate the proposed terminal abdominal ganglion (TAG) cercal escape system model we use a receiver operating characteristic (ROC) curve. The ROC curve plots the true (TPR) and false positive rate (FPR) as the probability threshold, that rectifies the activation of the sigmoidal *jump* neuron into a jump (1) or don’t jump (0) output, is increased from zero to one in regular steps. To reflect the imbalance between the consequences of not detecting true positive (succumbing to a predator) and responding to false positives (wasting a relatively unimportant amount of energy [219]), in the context of the insect, we define the metric ‘tolerated’ FPR which is the false positive rate that is be tolerated in order to achieve a minimum true positive rate. We fix the minimum TPR to 0.95 in this evaluation.

Fig. 2.4a plots the ROC curves resulting from an ‘ablation’ study of the proposed TAG model whereby combinations of individual components are considered separately. Specifically, the curves correspond to V1 - only the interneurons, V2 - the interneurons plus their lateral connections, V3 - the interneurons plus the global regulation network and V4 - the interneurons plus the lateral connections as well as the global regulation network. For means of comparison, the ROC curve of a logistic regression model, effectively a single neuron taking input from all sixteen feature map neurons, is plotted in black. In the case of model V1, it is observed that, despite increasing the number of neurons from one (i.e. the logistic regression model) to seven, there is a decrease in the tolerated FPR. This indicates that the extraction and combination of specific features alone does lead to an effective model. In model V2, the addition of lateral connections between the interneurons is seen to greatly reduce the tolerated FPR and noticeably increase the area bounded under the ROC curve. This demonstrates the importance of allowing neurons, each encoding different properties of the total sensory landscape, to communicate and compete, via lateral synapses - increasing the contrast between their responses. In model V3, the lateral connections are removed and the global regulation network is added. While the tolerated FPR deteriorates with respect to model V2, there is still a slight improvement with respect to the ROC curve of the logistic model and greatly improved with respect to the neurons acting alone. This suggests a certain importance in regulating the activations of the interneurons as a function of the background air current intensity which, intuitively, should make it easier for the *jump* neuron to detect the activation pattern of an attack in the presence of fluctuating background conditions. Finally, in model V4, the lateral connections and global regulation network are included together. The resulting ROC curve, plotted in blue, bounds noticeably more area than any of the other models and achieves tolerated FPR of 0.07 - well below that of the other models. This demonstrates that, while the lateral and global connections independently provided modest gains with respect to the logistic model, it is their interplay and the simultaneous application of contrast enhancement and background subtraction mechanisms that give rise to the most suitable model. While lateral connections are a commonly observed feature in animal nervous systems, this result provides a foundation for the proposal in [180] that a global regulatory mechanism was present in the cricket cercal system. The tolerated FPR of each model version, as well as the number of synaptic parameters required per model, are summarised in the bar-chart of Fig. 2.4b. Here it is noteworthy that, for a relatively modest two-fold increase in the number of parameters, the tolerated FPR reduces by more than three times. For a study into the robustness of the performance of the TAG model under random permutations to its optimised parameter vales, see Appendix 6.10.



(a)



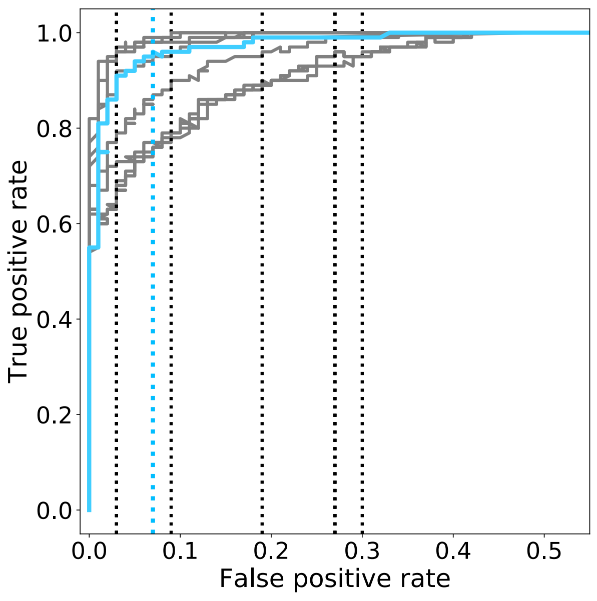
(b)

Figure 2.4: **Performance of the four ablated versions of the TAG model**. (a) Four receiver operating characteristic (ROC) curves are plotted showing how the true and false positive rates vary as the probability threshold on the sigmoidal output *jump* neuron is increased. Vertical dashed lines show the tolerated false positive rate (FPR) for a minimum true positive rate of 0.95. (b) Pairs of bar plots for each ablated version of the TAG model show the tolerated FPR (green bars, left y-axis) and the number of parameters required by the model version (blue bars, right y-axis).

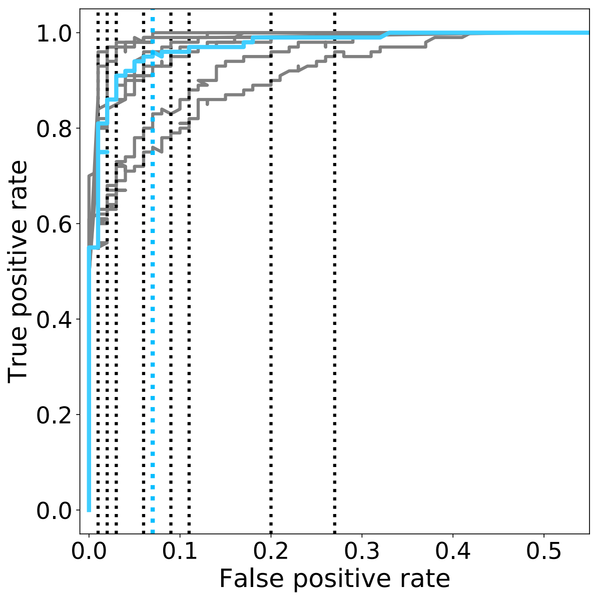
2.2.4 Universal approximator benchmarking

In order to benchmark the test accuracy and memory requirements of the optimised TAG model, single and three fully-connected hidden-layer neural network models are used - characteristic of a generic deep learning approach based on the universal approximation theorem [51, 71, 52]. These were also implemented using the PyTorch framework. They were trained using backpropagation with the adaptive moment estimation optimisation algorithm [220]. The models were trained over one thousand epochs using the same mini-batches as the TAG model. On average this was observed to be the number of epochs required for the parameters to converge to a minimum loss. It is also informative to note that this represents over an order of magnitude more training than that required by the TAG model, which required only fifty epochs.

In Figs. 2.5a and 2.5b we plot the ROC curves obtained by the optimised deep learning models, with a single hidden layer and three hidden layers respectively, when trained and tested using the same training data split as the TAG models. Numerous ROC curves are plotted for deep learning models with an exponentially increasing number of neurons per layer. As the number of neurons is increased the resulting ROC curves are pushed towards the top left hand-side of the plots, bounding more area and reducing the tolerated FPR. The ROC curve of the TAG model V4 is re-plotted in blue for comparison. The bar-plots in Figs. 2.6a and 2.6b show the resulting tolerated FPR and the number of parameters required by each of these single hidden layer and three hidden layer models respectively. In each bar plot the tolerated FPR and the number of parameters for the TAG model V4 are plotted as horizontal green and blue dashed lines for comparison. What is striking is that, for the deep learning models to obtain en equivalent performance to the TAG model, they are seen to require between one and two orders of magnitude more synaptic parameters. Specifically, to match the TAG model performance, the single hidden layer neural network is seen to require between 128 and 256 neurons and, for the three hidden layer model, between 16 and 32 neurons per layer.



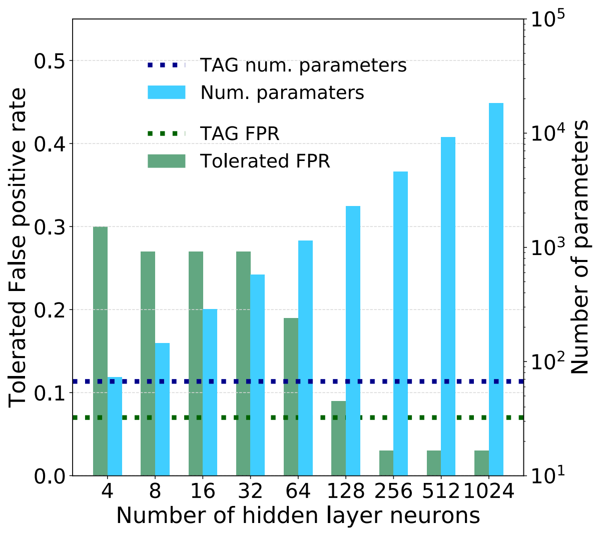
(a)



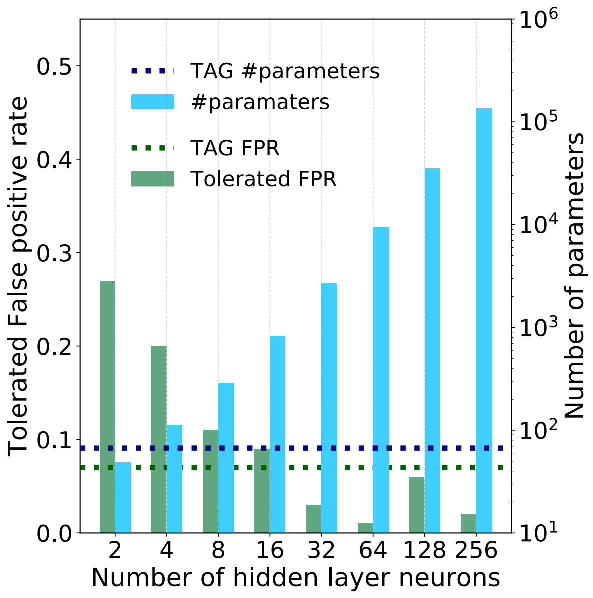
(b)

Figure 2.5: (a) ROC curves over a range of layer widths for a single hidden-layer model. The ROC curve plotted in blue corresponds to that of the TAG model V4. (b) ROC curves over a range of layer widths for a three hidden-layer model. The ROC curve plotted in blue corresponds to that of the TAG model V4.

However, as the number of hidden layer neurons are further increased the tolerated FPR of these extremely large models drops below that of the TAG model - in some cases obtaining a tolerated FPR of 0.03. In order to understand whether this large investment in memory is worthwhile, we use the Akaike information criterion (AIC) [221]. The AIC provides a number proportional to the sum of the model size and the negative log-likelihood of the model, evaluated on the test data set, and offers a means of comparing



(a)



(b)

Figure 2.6: (a) Pairs of bar plots over a range of hidden layer sizes for a single hidden layer universal approximator. The bars show the tolerated FPR (green bars, left y-axis) and the number of parameters required by the model version (blue bars, right y-axis). Dashed horizontal lines show (green) the TAG model tolerated FPR and (blue) the number of parameters required in the TAG model. The minimum TPR is 0.95. (b) Pairs of bar plots over a range of hidden layer sizes for a three hidden layer universal approximator. The bars show the tolerated FPR (green bars, left y-axis) and the number of parameters required by the model version (blue bars, right y-axis). Dashed horizontal lines show (green) the TAG model tolerated FPR and (blue) the number of parameters required in the TAG model. The minimum TPR is 0.95.

the efficiency of the solutions. It should be noted that a lower AIC indicates a more efficient model. The AIC of the single and three hidden layer deep learning models are plotted as a function of the hidden layer size in Fig 2.7. Additional horizontal dashed lines show the AIC score of the logistic regression (grey) and the TAG model (blue). It can be clearly seen that the TAG model is comfortably the most efficient solution to the problem and that, for the extremely large deep learning models which obtain the lowest tolerated FPRs, the AIC score explodes owing to the complexity of the model. Furthermore, it is indicative to note that, even for small hidden layer sizes, it is difficult to justify the choice of a deep learning model over the use of a simpler logistic regression model.

In order to complete this comparison, we extend it to the case of pruned multi-layer perceptrons - whereby memory requirements can be reduced by deleting synaptic weights with optimised values close to zero [222]. Specifically we consider a pruned version of the MLPs in Figs.2.6a and 2.6b and an additional version trained where an $L1$-norm regularisation term is summed with the loss function in order to enforce sparsity in the resulting model [223]. The weight distributions of these MLP models are plotted in Supplementary Fig. 7 where it can be seen that the models trained using the L_1-norm loss function have weights grouped very tightly around zero - appearing almost like a delta function. The tolerated FRP as a function of the percentage of pruned weights in the single and three hidden layer MLPs containing 256 and 32 neurons in each of their hidden layers respectively is plotted in Figs. 2.8a and 2.8b. These dimensions

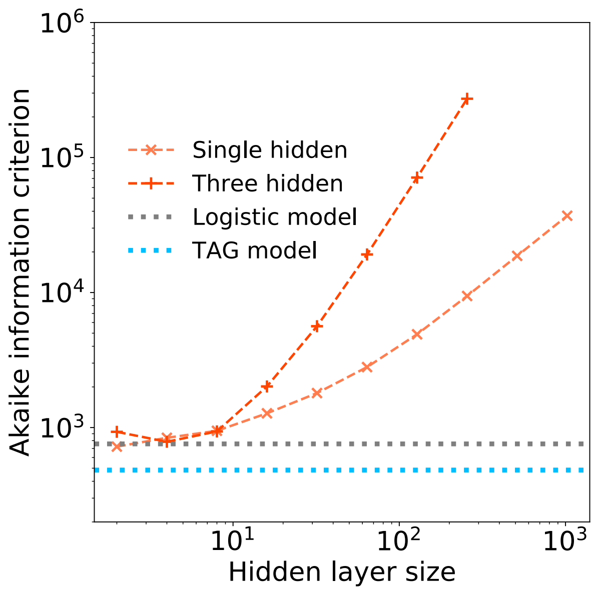
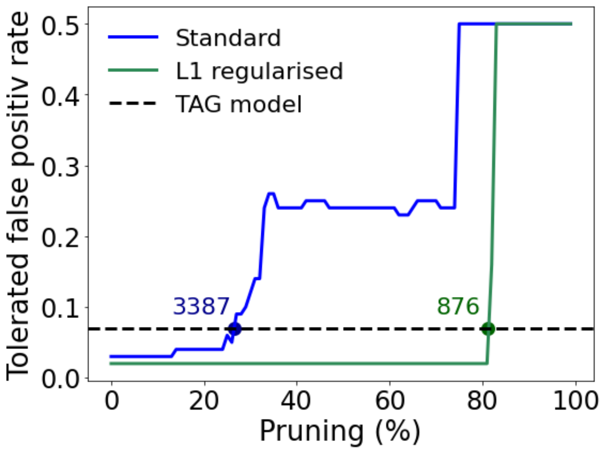


Figure 2.7: The Akaike information criterion (AIC) is plotted in log-log scale over range of hidden layer width for (orange) a single hidden layer and (peach) a three hidden layer deep learning models. Horizontal dashed lines show the AICs of a (grey) logistic regression model and (blue) the TAG model.

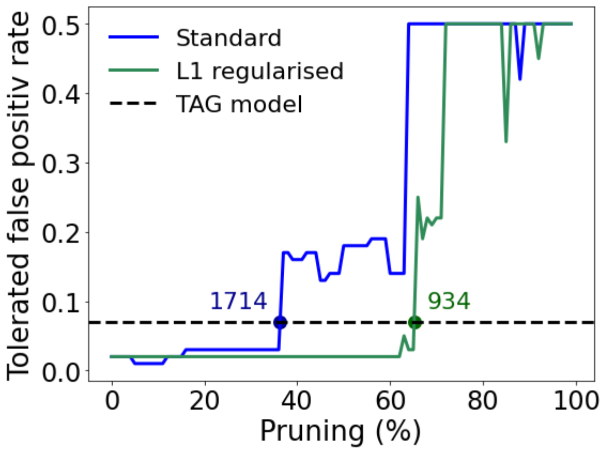
were selected in each case since this was the number of neurons per hidden layer, per MLP, that allowed a lower tolerated false positive rate than the TAG model. In each case, as the extent of pruning increases, the tolerated FPR, after a varying period of robustness, also increases - the point at which the tolerated FPR exceeds that of the TAG model is indicated using a labelled dot in Figs. 2.8a and 2.8b. While the L_1-norm regularised models permit a considerable reduction in the number of weights with respect to the standard MLP before a degradation in the tolerated FPR is observed, the number of parameters required in order to maintain a performance equivalent to that of the TAG model architecture was still over an order of magnitude greater than this bio-inspired architecture.

2.2.5 Model interpretation

In addition to their low memory efficiency, universal approximation based deep learning approaches suffer from another major drawback in their lack of interpretability [6]. Namely, it is difficult to ascertain as to why certain combinations of input features lead to certain output predictions as a function of the cascading layers of synaptic weights that have resulting from the application of backpropagation. This poses ethical and practical problems for many applications of artificial intelligence such as that addressed here which, certainly from the perspective of the cricket, can be considered as safety-critical - the model output entails potentially dangerous consequences (i.e., not jumping in the presence of an attacking predator). In contrast to the wide fully-connected layers of neurons in universal approximators, the proposed TAG model contains eight neurons and 63 synapses - each defining a set of logical relationships between the neurons which each have well defined functional roles. As such, based on its optimised synaptic weights, we make first-order structural and behavioural interpretations of the TAG model.



(a)



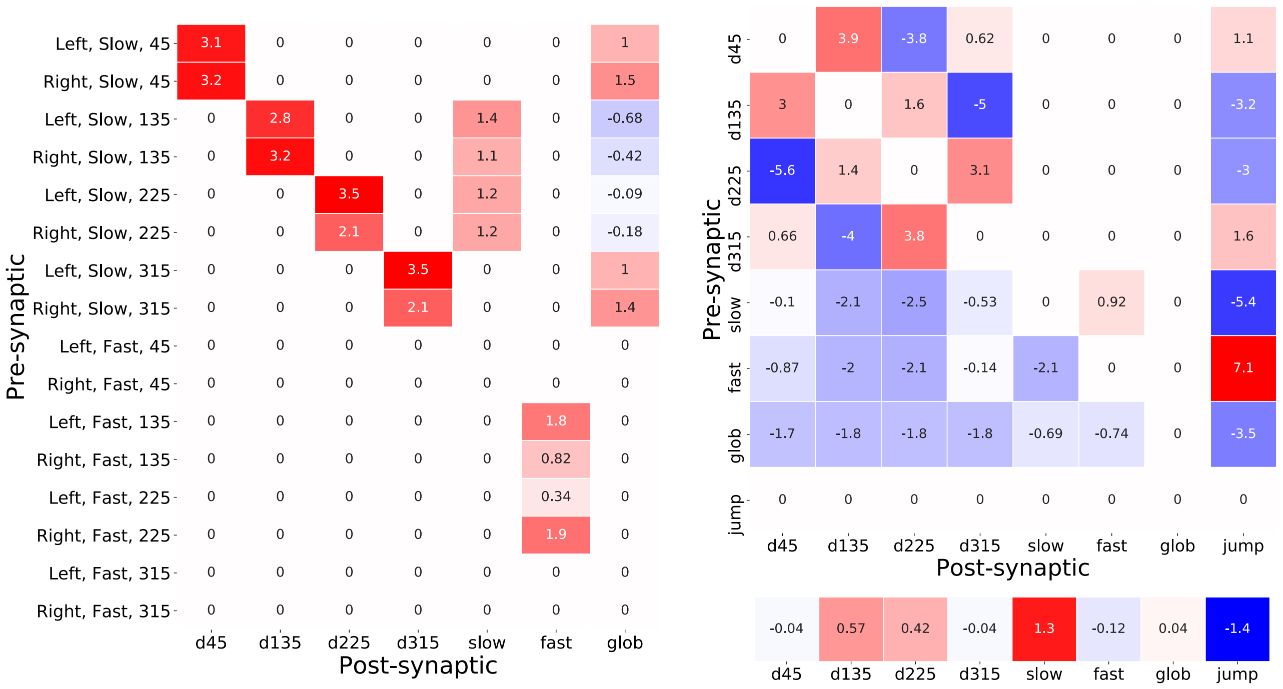
(b)

Figure 2.8: **Relationship between the tolerated false positive rate and the percentage of the weights closest to zero pruned for the standard (blue) and L_1-norm regularised (green) models.** The horizontal dashed line shows the tolerated false positive rate achieved by the TAG model. The number associated to each dot, drawn when the relationship crosses the horizontal dashed line, denote how many parameters the model required to match the tolerated FRP for the TAG model. (a) For the case of a single hidden layer MLP with 256 neurons in the hidden layer. (b) For the case of a three hidden layer MLP with 32 neurons per hidden layer.

Structural interpretation

A table of the optimised synaptic weights inter-connecting the input layer neurons with the interneurons is shown in Fig. 2.9a. Each cell in the table contains the synaptic weight value resulting from the backpropagation based training. The majority of these feed-forward connections are excitatory such that, without consideration of the lateral connections, the interneuron activations would be proportional to the activations of the input layer neurons that connect to them. Therefore, the activation of each of the directional neurons for example, encodes the instantaneous intensity of slow air currents coming from each of the four input air current directions. It is interesting to note the imbalance in the excitation received from the left and right cerci in the cases of *d135*, *d225* and *fast* whereby sensory activity coming from 135° is stronger from the left cerci and activity from 225° is stronger from the right. This in fact appears to model the dependency on attack direction described in Equations 2.2 and 2.3 whereby attacks coming from the left-hand side excite the sensory neurons oriented at from 135° more; and those from the right-hand side excite the 225° sensory neurons more.

Elevated co-activations of *d135*, *d225* and *fast* will therefore correlate with an attack event - in particular the activation of *fast*. Another feature that stands out is the weak negative feed-forward connections from the input neurons, corresponding to air-flow from attack angles, that seems to help decouple the high air-currents resulting from an attack and the activation of *glob* which is intended encode the overall background air-current intensity. Intriguingly, the inhibition coming from the 135° input neurons on the left and right cerci is, in this case, elevated with respect to that coming from 225°. This angular asymmetry is also observed in the weak excitatory inputs from 135° and 225° inputs from the right and left cerci respectively that are pre-synaptic to the *fast* neuron. It is unclear what is the purpose of this subtle adaptation.



(a)

Pre-synaptic \ Post-synaptic	d45	d135	d225	d315	slow	fast	glob
Left, Slow, 45	3.1	0	0	0	0	0	1
Right, Slow, 45	3.2	0	0	0	0	0	1.5
Left, Slow, 135	0	2.8	0	0	1.4	0	-0.68
Right, Slow, 135	0	3.2	0	0	1.1	0	-0.42
Left, Slow, 225	0	0	3.5	0	1.2	0	-0.09
Right, Slow, 225	0	0	2.1	0	1.2	0	-0.18
Left, Slow, 315	0	0	0	3.5	0	0	1
Right, Slow, 315	0	0	0	2.1	0	0	1.4
Left, Fast, 45	0	0	0	0	0	0	0
Right, Fast, 45	0	0	0	0	0	0	0
Left, Fast, 135	0	0	0	0	0	1.8	0
Right, Fast, 135	0	0	0	0	0	0.82	0
Left, Fast, 225	0	0	0	0	0	0.34	0
Right, Fast, 225	0	0	0	0	0	1.9	0
Left, Fast, 315	0	0	0	0	0	0	0
Right, Fast, 315	0	0	0	0	0	0	0

(b)

Pre-synaptic \ Post-synaptic	d45	d135	d225	d315	slow	fast	glob	jump
d45	0	3.9	-3.8	0.62	0	0	0	1.1
d135	3	0	1.6	-5	0	0	0	-3.2
d225	-5.6	1.4	0	3.1	0	0	0	-3
d315	0.66	-4	3.8	0	0	0	0	1.6
slow	-0.1	-2.1	-2.5	-0.53	0	0.92	0	-5.4
fast	-0.87	-2	-2.1	-0.14	-2.1	0	0	7.1
glob	-1.7	-1.8	-1.8	-1.8	-0.69	-0.74	0	-3.5
jump	0	0	0	0	0	0	0	0

d45	d135	d225	d315	slow	fast	glob	jump
-0.04	0.57	0.42	-0.04	1.3	-0.12	0.04	-1.4

Figure 2.9: **Tables of synaptic weight parameters between pre- and post-synaptic pairs of neurons**. The number in each cell shows the value of the weight and the colour of the cell denotes the sign and magnitude of the weight. (a) The table of synaptic weights between each of the sixteen feature map neurons and the seven interneurons. (b) The table of synaptic weights used for each pre- and post-synaptic interneuronal connection. The values of the interneuron biases, corresponding to their resting activations, are shown below the table.

A second pair of tables in Fig. 2.9b shows the sign and weight of the optimised synaptic weights inter-connecting the hidden layer neurons as well as their optimised biases. An immediately eye-catching feature is the symmetry observed in the connectivity pattern between the direction encoding interneurons. The interneurons are all seen to strongly inhibit their opposing counterpart - for example the complimentary negative synaptic connections between *d45* and *d225*. Excitingly this is consistent with predictions that have been made regarding the biological system whereby the reduction in directional interneuron activity, when presented with an air-current stimulus contrary to its preferred direction, has been attributed to the existence of laterally inhibiting interneurons between them [215]. Furthermore, these hidden layer interneurons send excitatory lateral connections to their ipsa-lateral and contra-lateral direction encoding neurons - notably the excitation towards the contra-lateral neuron is consistently greater. Based on the presence of the predicted lateral inhibitory interactions, this result hints at the possible existence of additional lateral excitatory connections in the biological system.

The post-synaptic connection weights of *glob* onto the six interneurons seen in Fig. 2.2a suggest a very interesting functional role of this neuron. Since these weights are negative and *glob* implements a hyperbolic tangent activation function, therein activating negatively for low *background speed* and positively for high *background speed*, *glob* acts as an excitor given lower background air current levels but, intriguingly, as an inhibitor when the background levels are higher. This latter effect results from the positive product of the negative neuronal activation and the negative synaptic weight value. This is once again an exciting result given the prediction on the presence of a global regulatory or background subtraction mechanism in the biological system [180].

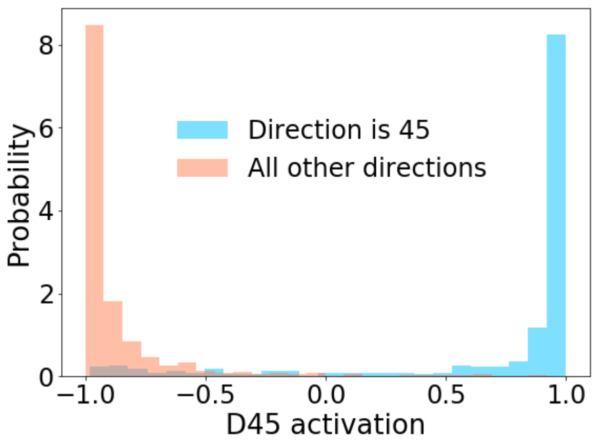
Finally, it is informative to read off the synaptic weights connecting the hidden layer interneurons to the output *jump* neuron as a logical 'bar-code' - offering a means of understanding what combinations of hidden layer activations should result in elevated activations of the jump neuron. The *jump* neuron is inhibited due to pre-synaptic connections from *d135*, *d225* and *glob*. This is likely an adaptation that allows the model to reduce the false positive rate under high background air current levels or high prevailing air-currents originating from the same directions as an attack. Another noteworthy feature of this synaptic bar-code is the opposing strong inhibition and strong excitation from the pre-synaptic *slow* and *fast* interneurons. Once again, since the neurons implement hyperbolic tangent functions, the respective negative and positive co-activation of *slow* and *fast* will result in a joint strong positive excitation of *jump*. This suggests that a key distinguishing feature of an attack is the divergence between the activations of *fast* and *slow*. The optimised TAG model further enhances this divergence through the lateral connections between*slow* and *fast*. In the hidden layer, *slow* synapses positively onto *fast*, and *fast* negatively onto *slow* - the more positive the activation of *fast* therefore, the more negative the resulting activation of *slow* and vice versa.

Functional interpretation

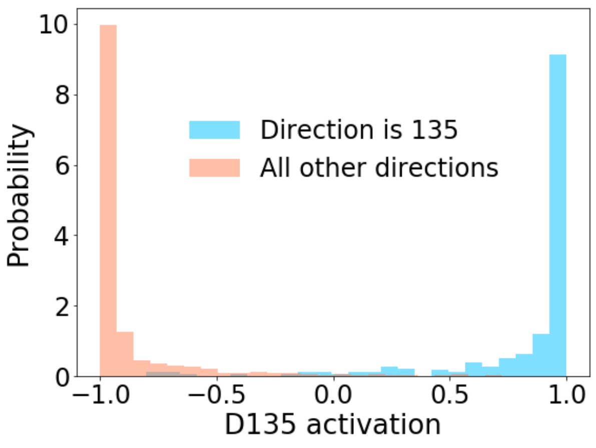
Although it arises as a direct consequence of the structural properties of the model, another interpretation of the model can be made from a functional perspective - therein how the neurons activate when presented with input data-points. In order to build up this picture, the probability distributions of interneuron activations over the 1000 test data-points are plotted and discussed.

The activation distributions of the directionally selective interneurons are shown in Fig. 2.10. Resulting from the excitatory input from the relevant feature map neurons, and the response shaping due to the lateral connections, each of these distribution shows a clear bi-modal response. Specifically, for a *background direction* in the preferred direction of the interneuron, the activation is strongly positive, and, for a *background direction* coming from any other angle, the response is strongly negative. Functionally these interneurons are seen to implement a 'one-hot' encoding of the *background direction*, reminiscent of winner-take-all computational mechanisms that are equally achieved through competitive lateral interactions between neurons [224, 225].

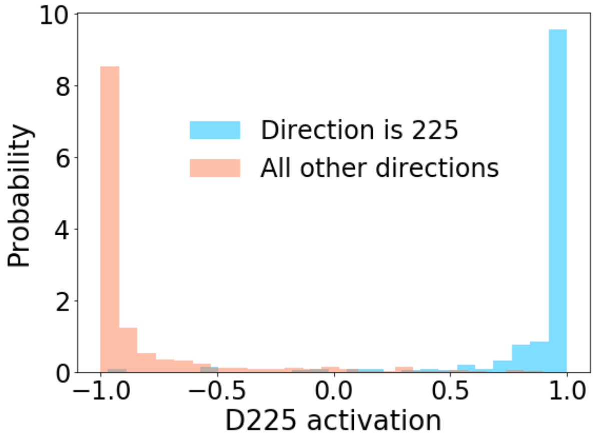
The fascinating computation carried out by the *slow* and *fast* interneuron pair is depicted in Fig. 2.11. Specifically the interneurons are drawn in their 'unrolled' format and the intermediate and final activation distributions of the *slow* and *fast* neurons are plotted as insets. The intermediate activation distribution of *slow* spans predominantly positive values, reflecting its strong excitatory input from the feature map neurons in the input layer and the comparatively weaker input from the *glob* neuron. In this intermediate stage therefore, its activation is proportional to the slow air current intensity coming from behind, given a low background intensity. Since *glob* is inhibited by air current stimuli coming from behind the animal (Fig. 2.2a), *slow* is seen to, on average, activate most due to input data points that are due to an attack - although the distribution of responses due to ambient conditions has a considerable overlap. The



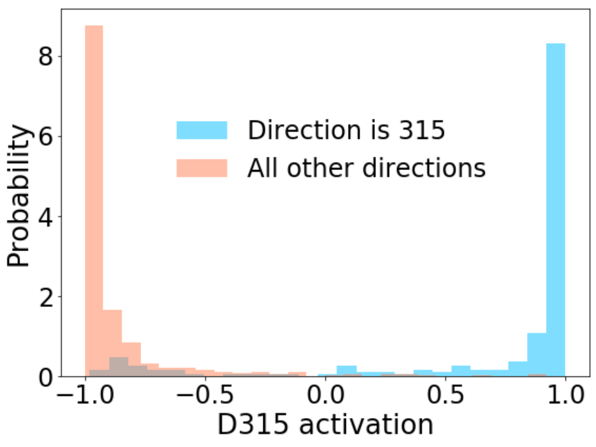
(a)



(b)



(c)



(d)

Figure 2.10: **The activation distributions over the 1000 test-points for each of the directional encoding interneurons**. The blue distributions show the hyperbolic tangent activations when the prevailing air-current direction was equivalent to the preferred direction of the sensory neurons that feed into the feature map neurons that each neuron receives excitation from. The red distributions correspond to when the prevailing direction of the air-currents was not in the neurons preferred direction. (a) Directional encoding neuron for air-currents from 45°. (b) Directional encoding neuron for air-currents from 135°. (c) Directional encoding neuron for air-currents from 225°. (d) Directional encoding neuron for air-currents from 315°.

intermediate attack and ambient activation distributions for the *fast* interneuron also exhibit a significant overlap. However, due to the particular sensitivity of this neuron to fast air currents coming from 135° and 225° from respective left and right cerci (Fig. 2.9a), one of the hallmarks of an attack modelled by equations 2.2 and 2.3, *fast* responds more positively to attack stimuli and more negatively to ambient stimuli on average. Through repulsive lateral interaction of these intermediate activations, the final activation distributions of *slow* and *fast* due to an attack are seen to respond more negatively and more positively respectively - reflecting the divergence between their post-synaptic connections to *jump* seen in Fig. 2.2a.

Finally, to complete the functional picture, the activation distributions of *glob* under high and low background air current intensities and of *jump* when the model is presented with *attack* and *ambient* data

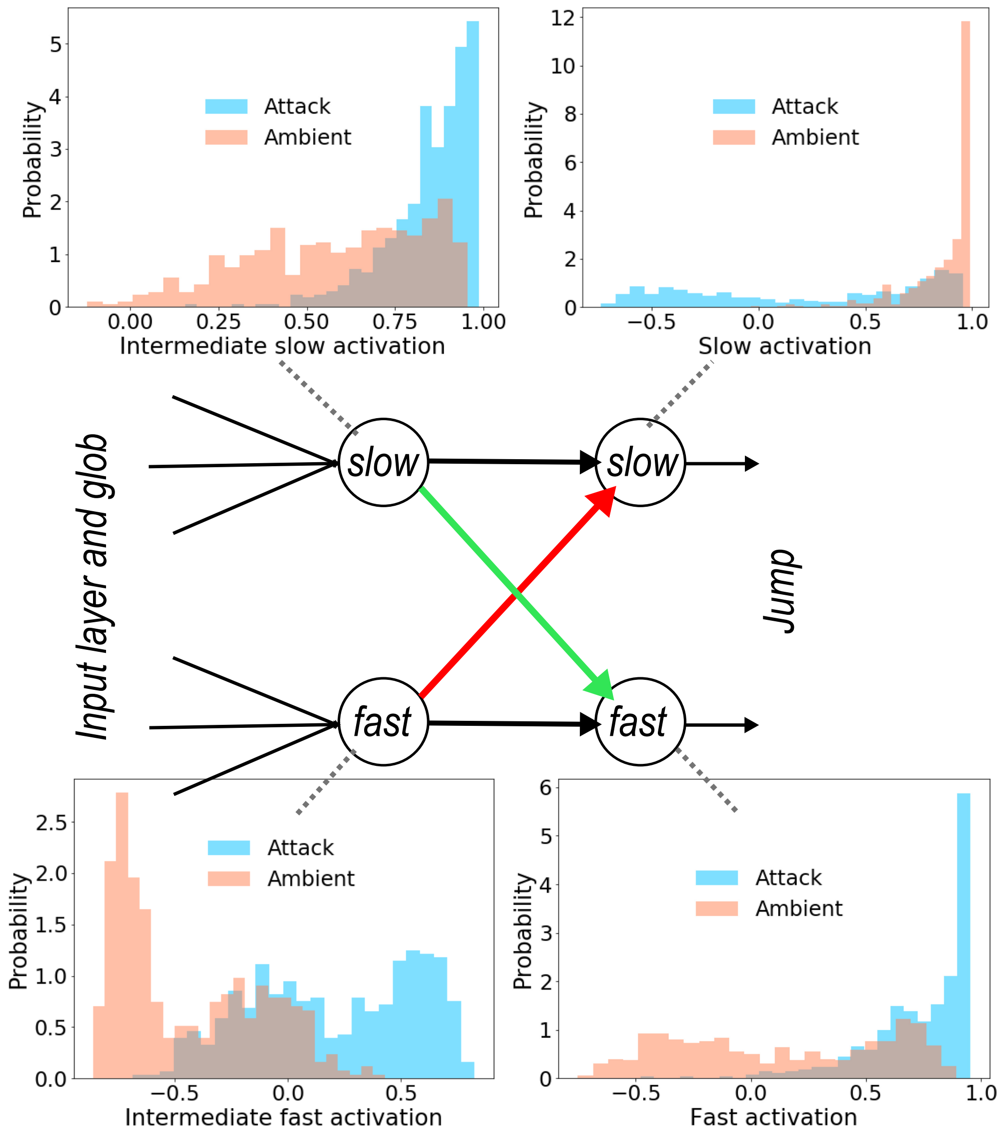
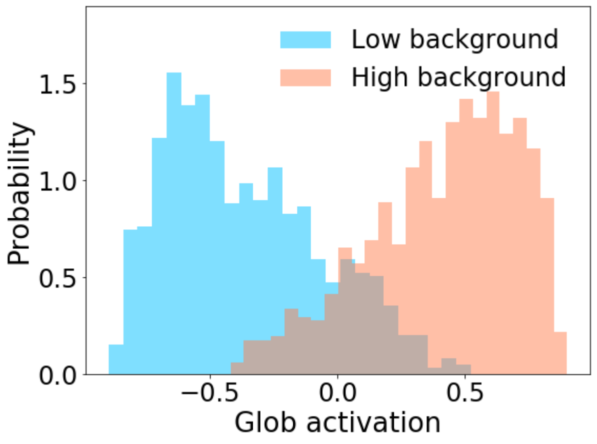
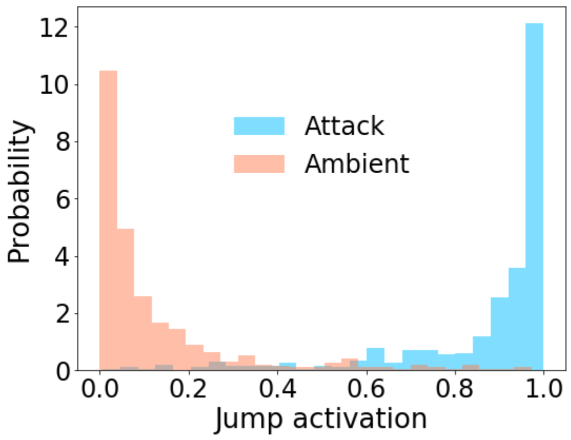


Figure 2.11: **The activation distributions over the 1000 test-points for each of the air current speed encoding interneurons in their intermediate and final, unrolled, states** The blue distributions show the hyperbolic tangent activations when the data-point pertains to an attacking predator and the red distribution to ambient conditions. (top left) The intermediate hyperbolic tangent activation of the *slow* neuron due to the feed-forward excitation from the feature map neurons. (top right) The final, unrolled hyperbolic tangent activation of the *slow* interneurons after the lateral inhibition from the intermediate state of the *fast* neuron. (bottom left) The intermediate hyperbolic tangent activation of the *fast* neuron due to the feed-forward excitation from the feature map neurons. (bottom right) The final, unrolled hyperbolic tangent activation of the *fast* interneurons after the lateral excitation from the intermediate state of the *slow* neuron.

points is shown in Fig. 2.12. Consistent with its proposed dual role as an inhibitor and as an excitor the *glob* activation responds with broad, largely negative or positive, activation distributions for low and high background air current intensities respectively. As reflected by the large area under the ROC curve of the optimised TAG model in Fig. 2.4a, the eventual attack and ambient *jump* distributions over all of the test data-points for the sigmoidal *jump* neuron are well separated. The response is predominantly a binary one, with the output layer activating very close to zero or one in most cases. Adaptation of the probability threshold, which is effectively a vertical line drawn through the distribution of Fig. 2.12b, allows the tolerated false positive rate to be tuned.



(a)



(b)

Figure 2.12: (a) The hyperbolic tangent activation of the *glob* neuron due to the feed-forward connections from the feature map neurons. The blue distribution shows the hyperbolic tangent activations for the test-points where the background air-current intensity was low and the red distribution when the background air-current intensity was high. (b) The sigmoidal activation distributions of the *jump* neuron for the (blue) attack test-points and (red) ambient test-points. A decision on whether to initiate a jump escape routine can be made by comparing the activation with a probability threshold along the x-axis.

2.3 Model of the *Drosophila* elementary motion detection system

2.3.1 Introduction

In recent years detailed partial connectomes of insect neural networks have been produced. An example is the elementary motion detection (EMD) network of the *Drosophila* visual system [226, 227, 228]. The insect has two large eyes composed of repeating hexagonal columns called ommatidia (Fig. 2.13). Each ommatidia has an identical structure which processes visual information from a small region of the full visual field. It is composed of four distinct layers of neuropil - the retina, lamina, medulla and the lobula as shown in Fig. 2.14(a). The neural pathway for detecting elementary motion within this structure is depicted schematically in Fig. 2.14(b). It begins at the retina where cells transduce light into electrical signals. These then synapse onto L1 and L2 cells in the lamina. At this stage the information is rectified into ON and OFF pathways - ON pathways carry information on luminescence increments and the OFF pathways luminescence decrements. The L1 and L2 cells in the lamina synapse predominantly onto the Mi1, Tm3 and Tm2, Tm1 cells in the medulla. These cells are implicated in implementing temporal delays between adjacent activity in spatial regions in the visual field [229]. The pairs of cells in the medulla synapse onto T4 and T5 cells in the lobula which are then excited if a spatiotemporal correlation exists between it's pre-synaptic cells indicating motion. Subsets of T4 and T5 cells are sensitive to motion in one of the four cardinal directions up, down, left and right and terminate independently in one of four distinct layers in the lobula plate (LP). Groups of lobula plate tangential cells (LPTC) are excited by activity in one layer of the LP and fire to indicate motion in a specific direction. Another group of lobula plate/lobula columnar type two (LPLC2) cells are excited only by diverging motion in each layer of the LP such that they indicate a looming stimulus on a collision course [182] - like the '*jump* neuron in the TAG model of section 2.2 this neuron triggers an escape response in the animal. Furthermore, it has also been found that the neuromodulator octopamine tunes dynamics of the EMDs in the *Drosophila* visual system as a function of whether the insect is resting or flying [230, 231, 232]. This allows the insect to adapt its sensitivity to different velocities of stimulus as well as reduce power consumption whilst in a resting state. In Fig. 2.14(c) the response of an LTPC cell is reported for Drosophila stimulated with a moving grating when it is in resting and flying states. The area under the curve for the insect in its resting state is greatly reduced relative to that of its flying state which is thought to be an evolutionary adaptation to optimise its consumption of energy. The Hassenstein-Reichardt EMD (HR-EMD) is a popular model which reproduces experimental observations of the *Drosophila* EMDs [233]. Photo-excitation at adjacent regions in the visual field propagates signals through crossing low pass filters, performing a delay function in the temporal domain, before being recombined at a multiplication unit to detect spatiotemporal correlations. The output of the unit can be either positive or negative indicating motion in one of two directions as in Fig. 2.14(d). A number of previous works have implemented the HR-EMD in analogue very large scale integrated (aVLSI) systems for point [234], 1D [235], 2D [236] and rotational motion [237]. Another perspective on hardware based motion detection are token and feature based EMDs [238, 239, 240, 241]. Here we propose an alternative approach for motion detection based on dynamic exponential synapse and leaky-integrate and fire neuron models - that are relatively simple to implement in a computer model as well as with analogue electronic circuits [103, 242]. Furthermore, we use a model of the dynamic vision sensor (DVS) to provide spiking, or 'event-based', input for the resulting spiking neural network model.

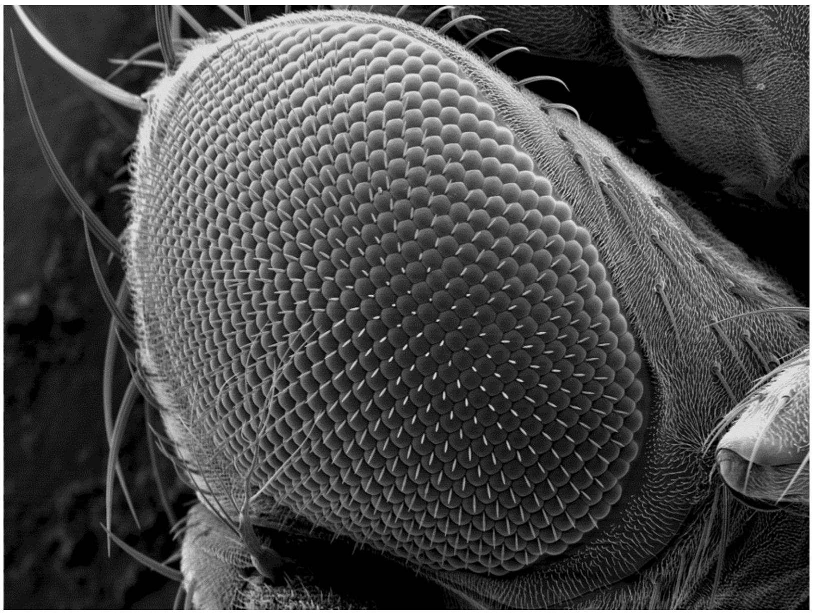
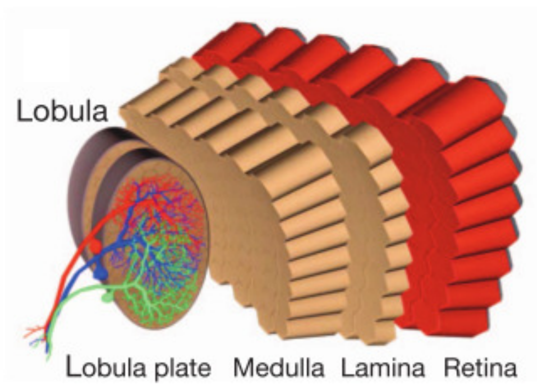
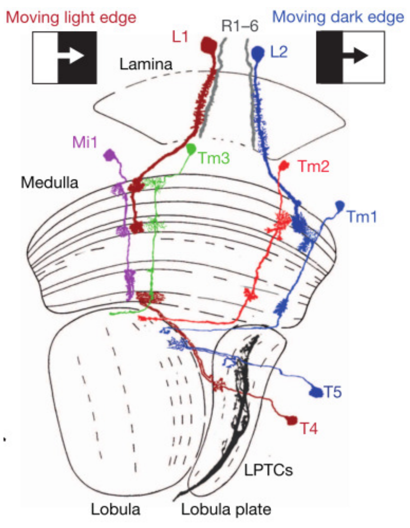


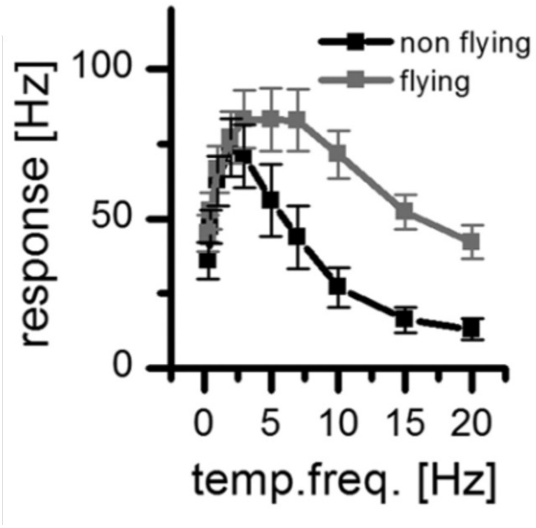
Figure 2.13: A scanning electron microscopy image of an the eye of *Drosophila. Each 'dot-like' ommitada on the surface extends down through the retina, lamina and medulla of the insect*



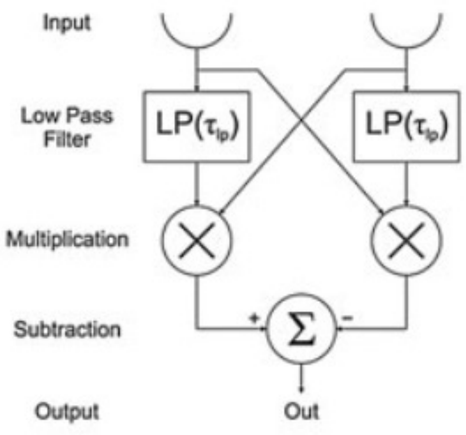
(a)



(b)



(c)



(d)

Figure 2.14: The architectural layout of the Drosophila visual system and its detection of motion. (a) The four layers of neuropil in the Drosophila visual system [227]. (b) The specific cells identified in the pathway from the retina to the lobula plate involved in elementary motion detection [229]. (c) Temporal frequency sensitivity tuning curve of the mean response of LPTC neuron in Drosophila in resting and flying states as modulated by octopamine. [230]. (d) A schematic of the HR-EMD model often implemented in hardware motion detectors [237].

2.3.2 Model definition

The final model is realised by a two-dimensional tiled matrix of 'coincidence detection modules' whose outputs are networked in a specific fashion to a small number of readout neurons which provide information on global motion in the visual field. The visual field corresponds to the ON and OFF events produced by a simulation of a grating moving either UP, DOWN, LEFT or RIGHT in front of a DVS camera [243, 244] at a range of different speeds. The objective of the model is to report the global direction of motion in the presence of local 'salt and pepper' noise.

Coincidence detection module

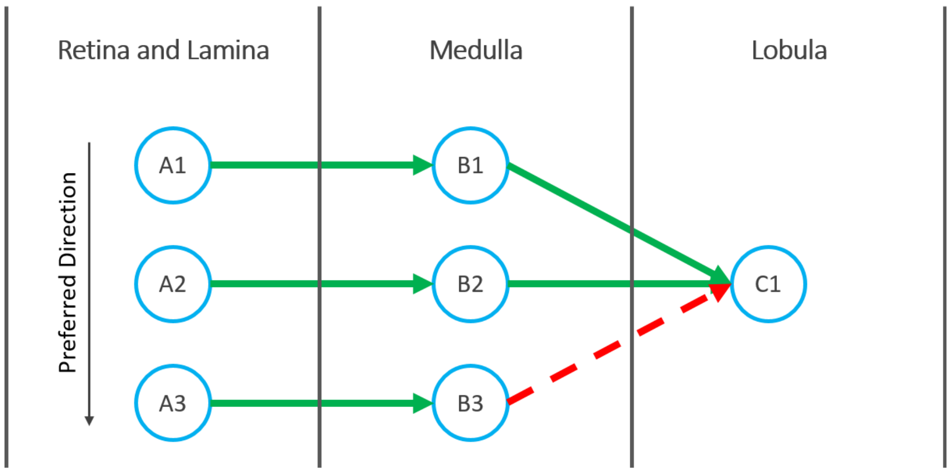
To detect motion in one direction along a single dimension a temporal delay can be implemented between two spatially adjacent inputs followed by a downstream mechanism for detecting spatiotemporal correlations. In *Drosophila* this delay is thought to be implemented between pairs of neurons in the medulla [229] and the correlation performed by the postsynaptic neurons in the lobula. Such a delay can be implemented by injecting the current of an exponential synapse into a leaky-integrate and fire neuron model;

$$\frac{di_{syn}}{dt} = \frac{-i_{syn}}{\tau_{syn}}, \tag{2.6}$$

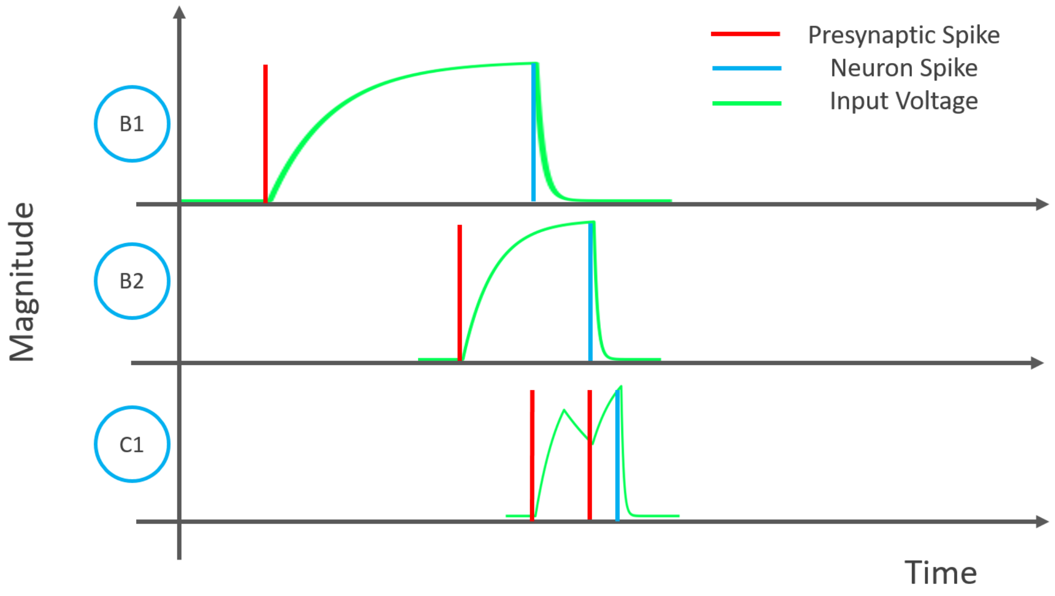
$$\frac{dV_{neu}}{dt} = \frac{i_{in}W - V_{neu}}{\tau_{neu}}, \tag{2.7}$$

where i_{syn} and V_{neu} are the synaptic current and neuron input voltage and τ_{syn} and τ_{neu} control the decay in time of these two variables respectively [245]. The input current of each neuron, i_{in}, is defined as the instantaneous sum of i_{syn} from all pre-synaptic neurons. The value i_{syn} of each synapse can be masked by $+1$ or -1 to define it as either excitatory or inhibitory. By adjusting the variables τ_{syn} and τ_{neu}, the delay properties can be modified. The variable W is the synaptic weight that determines how much V_{neu} is affected for a non-zero i_{syn}. Although in an electronic circuit implementation of this model tau_{syn} is often a function of W, the parameters are treated independently in this computational model for the means of simplicity. If the value of V_{neu} exceeds a threshold value V_{th}, the neuron will 'fire'. At this point, each of the synapses to connecting the firing neuron to a post-synaptic neuron will have their value of i_{syn} stepped by i_{spk}. A correlation operation can be performed by parameterising a neuron to fire only when two pre-synaptic spikes arrive within a short, pre-determined, time window. Chaining delay and correlation operations in sequence can be achieved using the coincidence detection module shown in Fig. 2.15(a). Neurons in the input layer fire to denote spatial activity, an event produced by the DVS camera for example, and synapse with excitatory connections onto a second delay layer. In this layer the delays implemented by the B1 and B3 neurons are larger than the central B2 neuron. B1 and B2 synapse with excitatory connections onto the correlator neuron C1 in the output layer. Neuron B3 synapses with an inhibitory connection that acts to suppress C1 from firing. As depicted in Fig. 2.15(b), if the input layer is excited in the sequence A1, A2 then A3, therein along its 'preferred direction', the firing times of B1 and B2 should be 'pushed' together in time, and the input from B3 will arrive at C1 last. If the two excitatory spikes from B1 and B2 to arrive at C1 within a sufficiently short time window, it will fire before the arrival of the inhibition from B3. To suppress C1 from firing due to motion contrary to the preferred direction, referred to as the 'null' direction, of the module (i.e. an excitation in the sequence A3, A2 then A1) the firing times of B3 and B2 will be pushed together in time and that of B1 will be delayed. Due to the inhibition resulting from B3, C1 will not be sufficiently excited to fire. In a third case where all input elements are excited simultaneously, due to sensory noise for example, B1, B2 and B3 will fire at approximately the same time and negate each others contribution. The parameters of the model can be set such that C1 does not receive sufficient excitation in this scenario to fire.

To allow for the detection of motion in two dimensions, four identical coincidence detection modules, sharing a common central short delay node, can be arranged orthogonally - such that each module is able to detect local motion in only one of the four cardinal directions of motion (i.e., UP, DOWN, LEFT or RIGHT). A ‘flattened’ network diagram of this 2D motion detector is depicted in Fig. 2.16.



(a)



(b)

Figure 2.15: The one dimensional delay and correlate spiking neural network and its functional basis. (a) A 1D delay and correlate SNN. Open blue circles represent LIF CMOS neurons, green arrows excitatory synapses and red dashed arrows inhibitory synapses. Vertical lines separate the three layers of the network into the abstracted functions as performed by the different layers of neuropil in the elementary motion detection system. (b) A time domain plot for the three neurons involved in detection of motion - B1 and B2 being excited in sequence and their resulting signals meeting at C1. Red spikes correspond to the activity of a presynaptic neuron, blue spikes to the activity of the neuron associated with the plot and the green trace is the neuron input voltage resulting from the incoming pre-synaptic spikes.

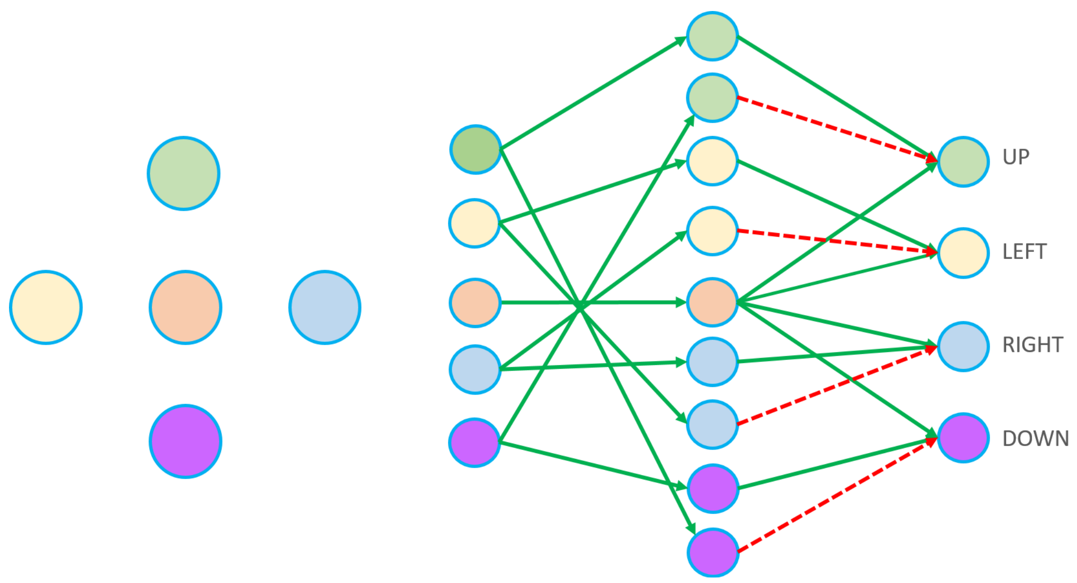


Figure 2.16: The two dimensional equivalent of the delay and correlate network. Blue filled circles represent LIF CMOS neurons where the fill colour indicates the pathway involved in detecting the directions UP (green), DOWN (purple), LEFT (yellow), and RIGHT (blue) while the central shared low time constant pathway is coloured red. Green arrows represent excitatory synapses and red dashed arrows inhibitory ones. The spatial organisation (left) of the inputs to a 2D delay and correlate SNN (right) is shown.

Readout Network

Inspired by the connectivity pattern observed in *Drosophila* between the elementary motion detection circuits in each of the animals ommatidia and the global feature encoding neurons in the lobula plate, we propose to connect the local 2D motion detection networks presented in this model to the five readout neurons depicted in Fig. 2.17. To readout the direction of motion, four neurons, modelling the LPTCs in the lobula plate, each sensitive to one of either UP, DOWN, LEFT and RIGHT motion are excited by the corresponding directional output of each of the 2D motion detector networks across the visual field.

2.3.3 Model evaluation

A DVS of resolution 10×10 is stimulated using a grating (a series of bars, 1 pixel in width, at 2 pixel intervals) moving UP, DOWN, LEFT or RIGHT in time provides OFF pathway spikes for the inputs of a matrix of the 2D motion detection networks (Fig. 2.16). Noise is simulated through setting each DVS pixel to spike in the absence of an OFF pathway event with a probability of 0.025 per timestep. The frequency of the grating is defined as pixels crossed by the grating per second of simulation time. Twenty 2D motion detection networks are used to span the visual field - each of which contains thirteen neurons and twenty one synapses. In total the topology requires 256 neurons and 580 synapses for the 10×10 visual field. The simulation advances in discrete time steps at which the synaptic currents and neuron input and output voltages are updated with respect to their values at the previous timestep in accordance with equations 2.6 and 2.7. The desired outcome is for the readout layer neuron corresponding to the direction of motion of the grating to fire at an elevated rate relative to the others upward motion to fire at an elevated rate relative to the others. The F_1 score [246] in the detection of upward motion is used to quantify the performance of

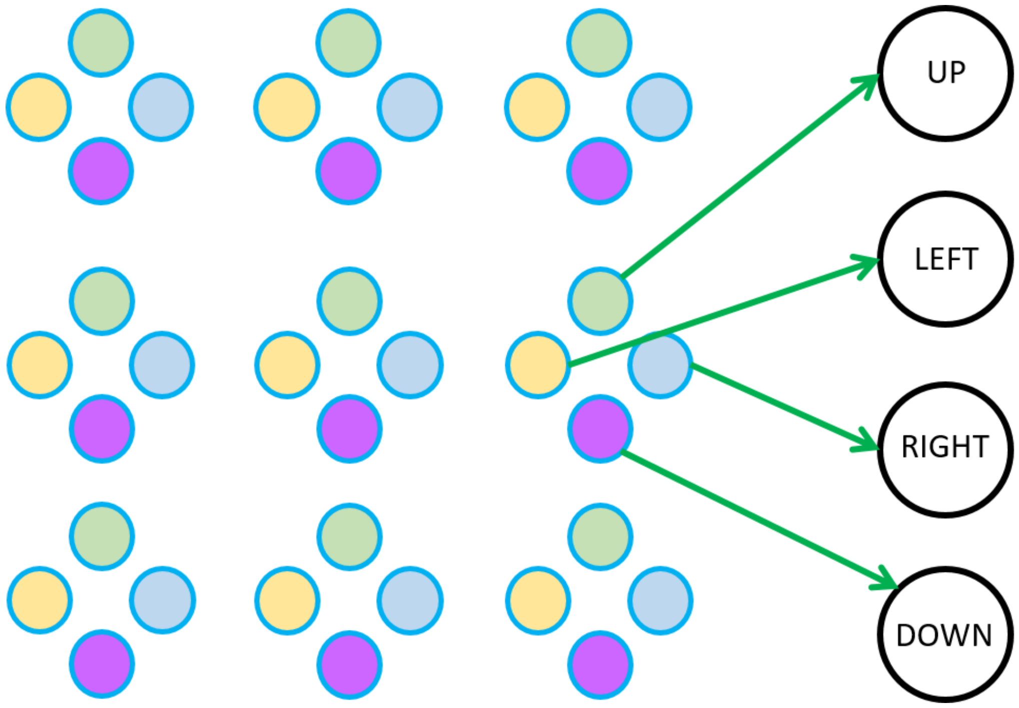


Figure 2.17: The connection pattern required between the 2D elementary motion detection SNNs that span the visual field in an array, and the four readout neurons. The coloured blue circles correspond to the output layer neurons from Fig. 2.16 with their spatial organisation indicating which direction they detect. Only one set of connections from one 2D motion detector to the four readout neurons has been shown for simplicity.

the model;

$$F_1 = \frac{2UP}{2UP + (DOWN + LEFT + RIGHT)}, \tag{2.8}$$

where UP, DOWN, LEFT and RIGHT are the number of times each of the direction encoding readout neurons fired during the simulation. F_1 score ranges from a minimum value of zero to a maximum value of one. The grating is swept over the input at a range of frequencies and the F_1 score is calculated per frequency. We refer to the plot of F_1 score with grating frequency as the sensitivity tuning curve (STC). As plotted in Fig. 2.14(c), the LPTCs of *Drosophila* have been measured to be responsive over a broad range of grating frequencies.

In order to determine the parameters of the model, a genetic algorithm is applied [247, 248]. Specifically the genetic algorithm determines the parameters of one coincidence detection module that are common to each orthogonal rotation in the 2D motion detection network and to each of twenty modules that span the visual scene and also determines the parameters of the readout neurons - each of which also share the same values. The parameters are the synaptic weights of all synapses W the synaptic and neural time constants τ_{syn} and τ_{neu} as well as the threshold voltages of all neurons v_{th}. The model parameter space therefore has twenty-four dimensions.

Sixty networks are created per generation. First generation parameters are assigned by sampling from a uniform distribution between a lower and upper bound per parameter. A sensitivity tuning curve (STC) is produced for each of the sixty networks. The ten with the largest area under the STC curve (AUC), therefore the ten networks with the best F_1 score over the grating frequency sweep, in addition to two randomly selected ones are recombined in pairs to produce the next generation of networks. Parameters are subject to 'soft' and 'hard' mutations with a certain probability. Hard mutations occur with a probability of 0.05 whereby the parameter is randomly assigned a value from a uniform distribution between an upper and lower bound. A soft mutation occurs with a probability of 0.5 whereby the parameter is reassigned a value from a normal distribution around the value inherited from the parent with a standard deviation of 1%. The average F_1 score of the ten best networks as a function of the generation is shown in Fig. 2.18 where it increases rapidly during the first twenty generations before levelling off an improving in a more incremental fashion.

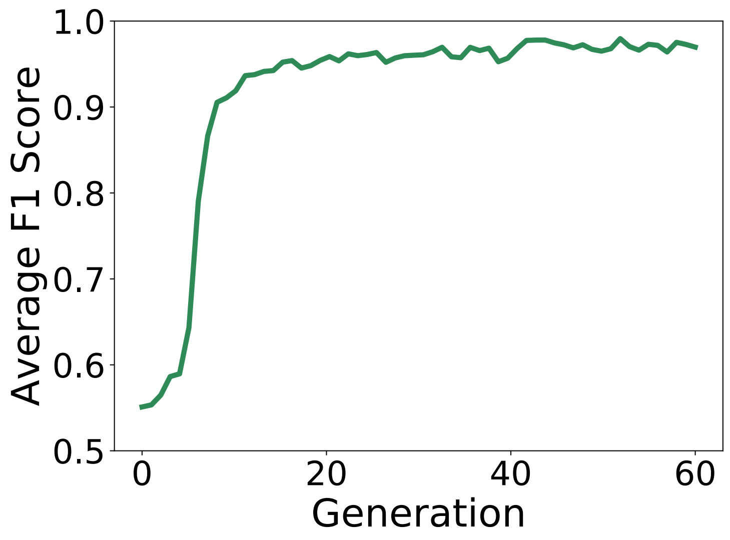


Figure 2.18: The average F_1 score as a function of the generation during the genetic optimisation of the model parameters.

The sensitivity tuning curves resulting from the best performing networks after termination of the genetic optimisation after sixty generations are plotted in Fig. 2.19. In total two network configurations were determined inspired by the adaptation observed in *Drosophila* whereby the neurotransmitter octopamine adapts the speed sensitivity of its elementary motion detectors between flying and resting states. Therein, one configuration is optimised to detect lower frequency gratings and the other to detect higher frequency gratings - they are referred to as the Slow and Fast network configurations respectively. The Slow configuration was optimised first and then, using the resulting optimal values for V_{th}, the Fast configuration was determined in a second step. The Slow and Fast configurations are plotted with respective red dashed and green traces in Fig. 2.19(a). The Slow state accurately detects the direction of motion within a range of grating frequencies of 0.7-3Hz while the Fast state does so within a range between 2-20Hz. Further, to strike a greater parallel to experimental measurement of this behaviour, reported in Fig. 2.14(c), firing-rate tuning curves are plotted in Fig. 2.19(b). This plots the firing rate of the UP neuron, when stimulated with an upward moving grating, over the same range of grating frequencies for both the Slow and Fast network configurations. Since the area under the firing-rate tuning curves of Fig. 2.19(b) is greater over the range of grating frequencies when the network is in the Fast state than the Slow state the energy consumed by firing activity in the readout network in the Slow configuration is reduced by 42% relative to the Fast one. The result of Fig. 2.19 shows that, like *Drosophila*, the range of velocities of stimulus to which the system is most sensitive to can be adapted by switching synaptic weights and time constants between two sets of values. Additionally, when the network is switched to the Slow state the power consumption of the system is reduced since it responds less to non-relevant higher speed stimuli.

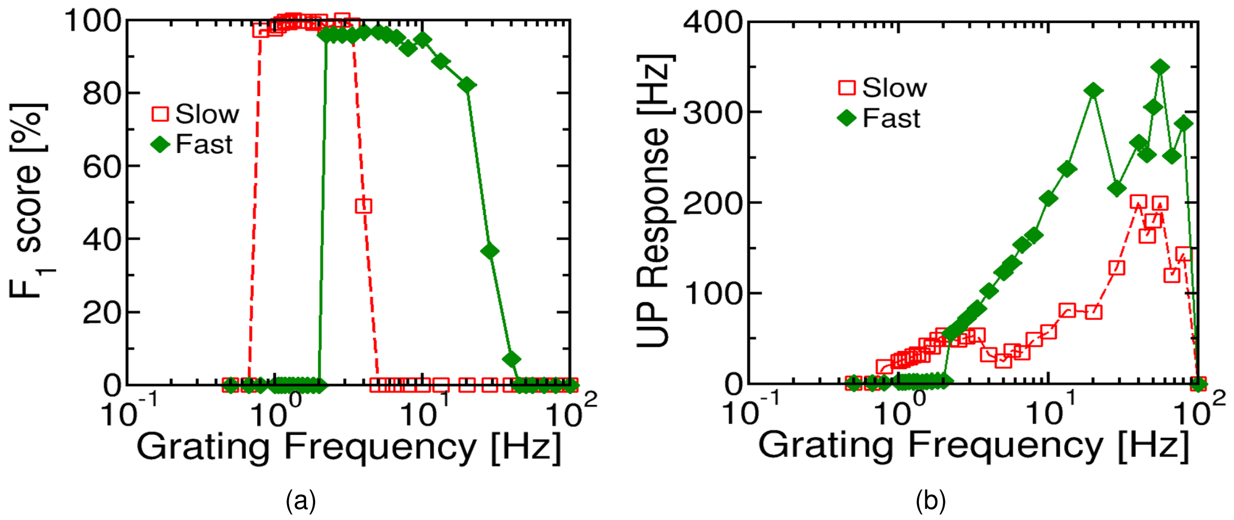


Figure 2.19: Results of the spiking neural network simulation demonstrating the performance and power consumption of the two network configurations over a range of grating frequencies. (a) The sensitivity tuning curve corresponding to the performance of the topology in detecting the correct direction of motion over a range of grating frequencies. The red points correspond to the Slow network configuration and the green points to the Fast network configuration using the parameters from the genetic optimisations. (b) The firing-rate tuning curve corresponding to the response frequency of the neuron defined as the number of times the UP neuron fires per second over the same range of grating frequencies. It can be seen that, resulting from the reduced response in the Slow configuration, the power consumption is reduced in this state relative to that of the Fast configuration inline with the reduced area under the tuning curve.

2.4 Chapter discussion

Although the two presented models were inspired by different perception systems found in two different insect species, it is fascinating to note the parallels between them. For instance, just as the filiform hairs on the cricket cerci sort local air-flow into one of four angles, the elementary motion detection circuits in the the lamina and medulla of *Drosophila* sort local motion into one of four cardinal directions [249]. Also, just as specific regions of the TAG integrate spikes from the sensory neurons of the cerci into distinct regions corresponding to direction, the array of elementary motion detection circuits integrate their global activity in specific regions of the *Drosophila* lobular plate - forming a directional and spatial feature map of its visual scene. Furthermore, in the same fashion as the seven interneurons in our TAG model form a compressed representation of the thousands of spikes triggered on the two cerci, *Drosophila* boils down the high dimensional space of its original optical input into a representation of 27 interneurons. As in our model each of these interneurons encodes certain properties of the visual scene [250]. One of the most interesting parallels with the optimised TAG model, although the escape response was not incorporated into the elementary motion detection model, is the observation that the *Drosophilas* giant fibre neuron, that descends towards the animals motor ganglion to initiate a jumping escape response, sums excitatory and delayed inhibitory input from two of these interneurons - one encoding stimulus speed and the other stimulus size [184]. This resonates with the excitation and delayed (inline with [214]) inhibition from the *fast* and *slow* neurons in the optimised TAG model. This alludes to a common set of design principles in, at least, insect sensory systems consistent with the observation that, since the animal genome is relatively constrained in the information it can store [187], animals should be applied to recycle the sets of neural architectures and computational mechanisms across different systems. In the these two systems, information is sorted into fundamental attributes as early-on as possible: the cricket sensory neurons reporting air current in a specific direction for example, and thereafter using a global set of such attributes to build a sensory feature map. By tapping into, and combining in a non-linear fashion, various activity in various spatial locations over this feature map, the ensemble of information is compressed aggressively into a representation, arguably an abstraction, comprising a much smaller set of neurons - seven in our model and twenty-seven in the *Drosophila* visual system. The output of these neurons is then sent only to where it needs to be , in order to drive appropriate behaviours - escape responses for example.

The objective of this chapter was to develop two neural network model architectures inspired by biology. While the model architectures were inspired from research into the biological systems, several trade-offs were made from the standpoint of practicality. For example, in the case of the cercal system model, the sensory feature map that is realised in the animal through a complex network of sensory afferents, was simplified into sixteen leaky-integrating hyperbolic tangent neurons. Also, instead of modelling the seven interneurons in the model as spiking neurons, which they are in the biological system, hyperbolic tangent functions were used instead. Despite deviating from the exact biology, these practical modelling choices were still able, however, to capture the fundamental underlying computing principles. This highlights the difference between bio-mimicry and bio-inspiration, at least in the context of neural networks, whereby the former aims to reproduce biology exactly and the latter to extract and apply its key principles - although not necessarily in the same fashion. Surprisingly, despite this deviation from the exact biology, two predictions from electrophysiological studies into the cricket terminal abdominal ganglion were validated in our model to be computationally important in the efficient detection of simulated attacking predators. Specifically the presence of lateral inhibitory connections between the directionally selective interneurons predicted in [215] and the global regulation, or background subtraction, mechanism proposed by [180]. These results raise the interesting question as to what else such bio-inspired neural network models might tell us about the mechanisms at play in the cercal system. For example, could the lateral excitatory connections between directionally sensitive interneurons or the divergent connections between air current speed encoding neurons also exist in the biological system?

The two sections of this chapter respectively showed the potential of incorporating bio-inspired architectures, for the purposes of memory efficiency and interpretability, and the importance of dynamical neuron and synapse models in representing time. Therefore an interesting future direction of work is in understanding how to best marry these two facets together - for example, how to properly leverage dynamical neuron and synapse models in the TAG model architecture. Work in this direction has already explored how backpropagation can be applied to training feed-forward spiking neural networks [251, 252] using gradient approximations to the non-differential spiking neuron activation as well as topologies based on dynamical sigmoidal neurons [152].

What is clear is that, contrary to the direction of deep learning, intelligence in animal nervous systems is more than learning. Rather innate architectures, that have been discovered over the course of evolution, provide built-in solutions for many tasks. Not only looming as discussed here, but also more complex behaviours concerning courtship and burrowing too [185, 186]. With the advent of recent methods that permit increasingly detailed study of animal nervous systems, ranging from electron microscopy based connectome reconstruction [253] to opto-genetic-based electrophysiology [254], subsequent years promise that such architectures can be more easily discovered and then transferred into neural network models like those presented here. An important question to address with future work is how to scale the approach to larger models whereby many such bio-inspired modules can be inter-linked in a larger network - resemble something more like an ‘artificial nervous system’ as opposed to merely an artificial neural network.

Chapter 3

Bayesian machine learning with resistive memory

3.1 Chapter introduction

In the previous chapter it was discussed and demonstrated how principles from animal nervous system could be used to inspire neural network model architectures. The two models presented were implemented in software on a von Neumann computer and, as such, their parameters could be determined in an arbitrary fashion - respectively, backpropagation and a genetic algorithm were used. However, this thesis is concerned with exploiting the non-volatility of resistive memory in order to support non-von Neumann RRAM-based neural network approaches. In using resistive memory, the determination of an optimal set of parameters is non-trivial owing to their random properties and the resultant lack of precision with which parameters can be determined. Therein, this chapter deals with the open question of what approach to machine learning can be compatible with these non-ideal device properties that currently impede the practical application of RRAM-based machine learning models. Unlike the genetic and gradient-based approaches used the previous chapter, this chapter turns instead to the Bayesian machine learning framework where inherent device randomness can be actively harnessed rather than acting as the principal roadblock.

3.2 In-situ resistive memory based Markov Chain Monte Carlo

3.2.1 Introduction

A tantalising prospect for the future of computing is the realisation of standalone systems capable of acting, adapting and learning from new experience locally at the edge [169] - independent of the cloud - while simultaneously observing severe constraints on energy consumption, data availability and memory size. While there are no commercial systems currently available that can meet these requirements, edge learning is an application domain that is projected to emerge and grow over the coming decade [2]. Edge learning faces particular challenges given the noisy and limited raw sensory information encountered in the complex environments where such systems could be deployed - an implanted medical system required to locally update its operation based on the evolving state of a patient for example. The models and algorithms within the domain of machine learning offer the enabling tools for such systems. However, until recently, little attention has been given to the hardware that underpins their inherent computation. Machine learning models are trained using general purpose hardware which inherits from the von Neumann organisation [255]. This entails spatially separate processing and memory and does not owe itself to energy-efficient learning. For example, state of the art performance in machine learning is currently being obtained with neural network models which feature a very high number of parameters [256] that are determined using the backpropagation algorithm [51] and a massive volume of data. The energy required to train such models can be staggering due to the transfer of vast quantities of information between memory and processing centres on the hardware [3, 257]. Although cloud computing platforms offer a centralised solution for such energy and data intensive training, these demands are not consistent with the requirements of edge learning [169]. For edge applications therefore, it is required to abandon the von Neumann paradigm in favour of another, where memory and processing can co-exist at the same location.

Resistive random access memory (RRAM) technologies, often referred to as memristors [258, 259], hold fantastic promise for realising such in-memory computing systems, owing to their extremely efficient implementation of the dot-product (or multiply-and-accumulate) operation that pervades machine learning - relying simply on Kirchoffs current law [260, 261, 262, 263, 264, 265, 266] (Fig. 3.2). These technologies come in many flavours [121, 122, 120, 119], and intense effort is currently directed towards their use as synaptic elements in hardware neural networks for edge computing systems [260, 261, 262, 263, 264, 265]. Currently, approaches for training such systems in-situ revolve around in-memory implementations of the back-propagation algorithm [261, 267, 263, 264, 265]. However, implementing back-propagation in such hardware remains a formidable challenge due to multi-fold non-ideal device properties: non-linear conductance modulation [268], lack of stable multi-level conductance states [269, 270], as well as device variability [271, 272]. Considering these real device properties, the performance of systems can be lower than that obtained on conventional computing systems [273, 274, 275]. Several non-ideality mitigation techniques [261, 267, 276, 264, 265, 277] enhance system accuracy but ultimately curtail the efficiency of the in-memory computing approach. On the contrary, approaches based on neuroscience-inspired learning algorithms, such as spike-timing dependent plasticity feature, resilience and, in fact, sometimes benefit from device non-idealities [278, 279, 280]. However, these brain-inspired models cannot yet match state of the art machine learning models when applied to practical tasks. Further research has taken a bolder stance on the issue of resistive memory non-idealities and instead propose that they should be actively embraced. For example, the cycle-to-cycle conductance state variability, which has been leveraged as a source of entropy in random number generation [281, 282], has also been exploited in stochastic artificial intelligence algorithms such as Bayesian reasoning [283, 284], population coding neural networks [285] and in-memory optimisation [286, 287]. Unfortunately however, these approaches sacrifice the key property of conductance non-volatility - the basis of resistive memory's potential for efficient in-memory computing.

In this work, we present an alternative approach which simultaneously exploits conductance variability and conductance non-volatility without requiring mitigation of other device non-idealities. From the perspective of cycle-to-cycle conductance variability, we propose that resistive memories can be viewed as physical random variables which can be exploited to implement in-memory Markov Chain Monte Carlo (MCMC) sampling algorithms [288]. We show how a resistive memory based Metropolis-Hastings MCMC sampling approach can be used to train, in-situ, a Bayesian machine learning model realised in an array of resistive memories. Crucially, the devices that perform the critical sampling operations are also those which store the parameters of the Bayesian model in their non-volatile conductance states. This eliminates the need to transport information between processing and memory and instead relies on the physical responses of nanoscale devices under application of voltage pulses inside of the memory circuit itself.

In order to demonstrate the practicality of RRAM-based MCMC sampling, we have implemented an experimental system consisting of a computer-in-the-loop with a fabricated array of $16,384$ resistive memory devices in a one-transistor-one-resistor (1T1R) configuration that are organised by the computer to realise a Bayesian machine learning model (Fig. 3.1). The computer is responsible for performing calculations based on the conductances read from the devices in the array and, as a function of the result, configures the programming voltage waveforms that then are applied over the devices - thereby implementing the learning algorithm in-situ. For information on the experimental setup, the device arrays and the device programming conditions used, see Appendix 6.3.

In a first experimental realisation, we train the system to solve an illustrative classification task, in a second we apply it to the detection of malignant breast tissue samples and, in a third, we address the detection of arrhythmic heartbeats. Finally, we extend the approach using a behavioural simulation calibrated on an array level variability characterisation, to the Cartpole reinforcement learning task. In each of these tasks, we benchmark the resulting performance against software-based neural network models and find that the resulting Bayesian models are not only able to outperform these software benchmarks but also that RRAM-based MCMC is able to train the full model with some orders of magnitude fewer device programming operations relative to existing resistive memory based backpropagation approaches. Finally, through the design and simulation of a fully-integrated implementation of our proposed approach, we compare the energy required to train the models presented in here relative to conventional CMOS solutions and, once again, observe a reduction comprising several orders of magnitude.

3.2.2 In-memory implementation

Arrays of resistive memory devices are capable of implementing extremely efficient in-memory machine learning models [260, 261, 262, 263, 264, 265]. Considering an RRAM-based logistic regression classifier, one of the most canonical models in machine learning, the circuit of M parallel devices shown in Fig. 3.2 defines a hyper-plane that can separate two classes of data. Each parameter of the logistic regression model is defined by the conductance of one of the M devices. The response of this conductance-based model, **g**, can be inferred by presenting a voltage vector **V**, encoding a data point, to the top terminals of the devices and sensing the current that flows out of the common, bottom node. This current is equivalent to the dot-product between the two vectors $\mathbf{V} \cdot \mathbf{g}$.

The dominant approach in RRAM-based machine learning for training the conductance parameters of a model is gradient-based optimisation whereby an error, or cost function, is iteratively differentiated with respect to the current parameters of the model. The resulting derivative provides precise conduc-

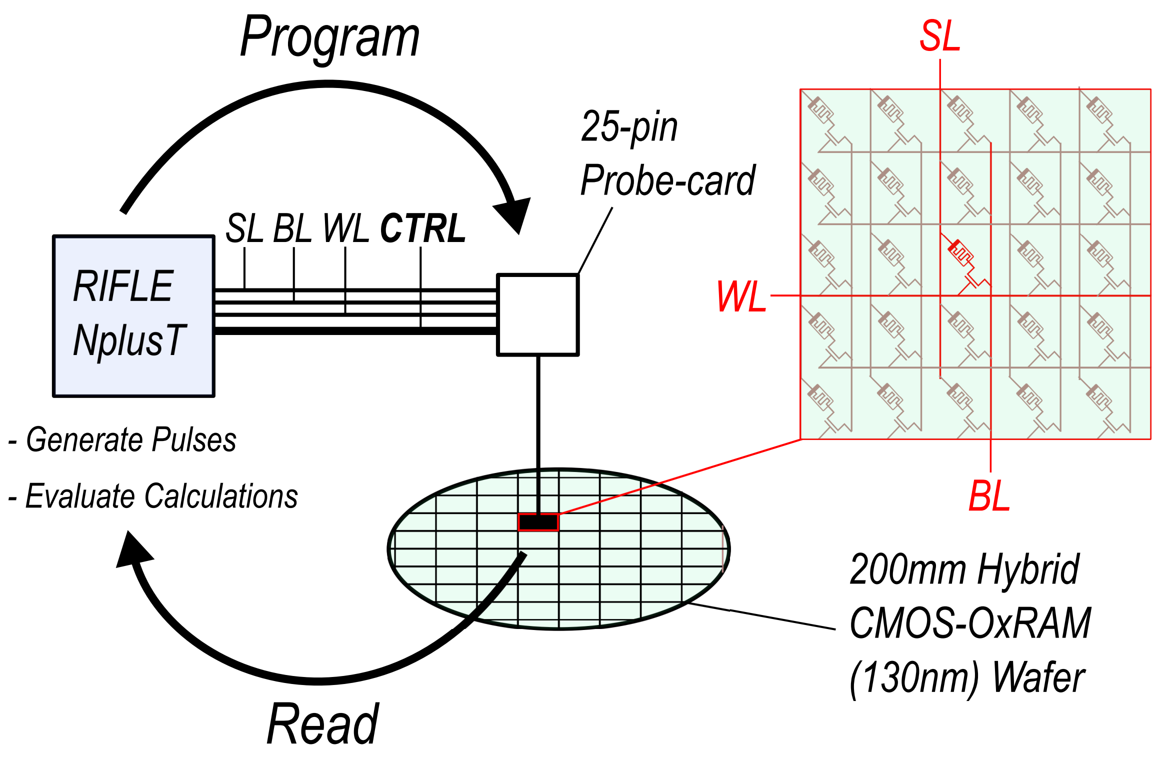


Figure 3.1: **Diagram of the computer-in-the-loop with the resistive memory array.** The computer-in-the-loop experiment using the RIFLE NplusT engineering test system, which incorporates a digital sequencer, 100 MHz arbitrary waveform generators, 70Msample/sec cell current measurement capability, as well as a C++ programmable computer. The computer can configure pulses applied by the arbitrary waveform generator to source (SL), bit (BL), and word lines (WL) signals. The CTRL bus controls periphery circuitry integrated within the hybrid CMOS-OxRAM wafer and determines to which device in the array the SL, BL and WL signals will be applied to (here the red coloured 1T1R structure). All these signals are interfaced to our 200 mm (8-inch) wafer through a 25-pin probe-card, which contacts to 25 metal pads integrated on top of the back-end-of-line of the wafer. Using this setup, the computer is therefore able to program the conductance states of devices integrated in the array, read the resulting states, and then, based on these results, re-program the devices in the array in a continuous program-read loop. In this fashion, we can implement RRAM-based algorithms, whereby programming operations can be determined based on the feedback of the result of the previous programming steps. In the case of the RRAM-based MCMC sampling algorithm implemented here, after reading programmed device conductance values, the computer is then responsible for calculating the dot-product between the data points and the conductance model, evaluating the likelihood of the conductance model and finally performing the acceptance ratio calculation. The ultimate aim of such a system is to experimentally verify RRAM-based algorithms, before a full system can be designed and integrated on a standalone chip.

tance updates which, after being applied over a sufficient number of iterations, guide the model down the slope of an error gradient such that it settles into a minimum (Fig. 3.3a). This results in a locally-optimal model that can then be applied to tasks through inference. However, performing this type of training in-memory is extremely challenging as resistive memory technologies feature highly non-linear and variable conductance updates that do not naturally support the high precision updates required by this class of algorithm [268, 269, 270, 271, 272]. Furthermore, in such a deterministic modelling approach (Fig. 3.2), each parameter is described by a single value, and it is not possible to account for parameter uncertainty.

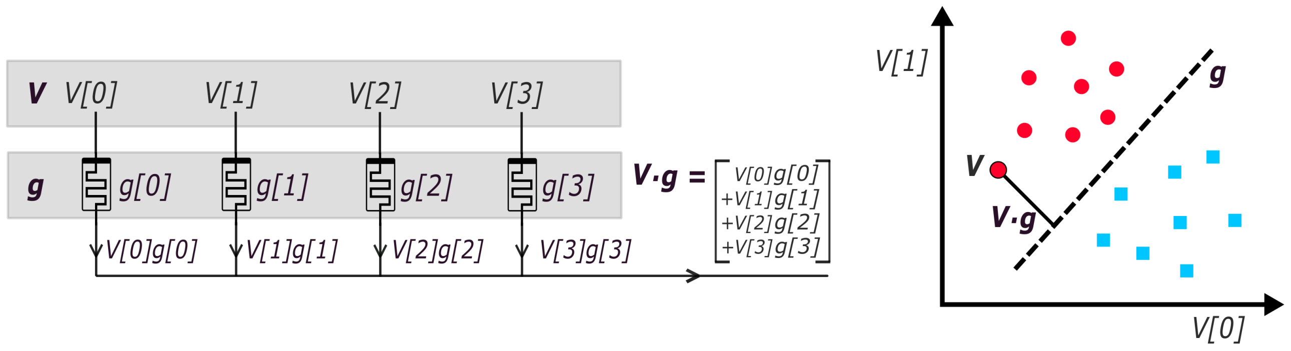
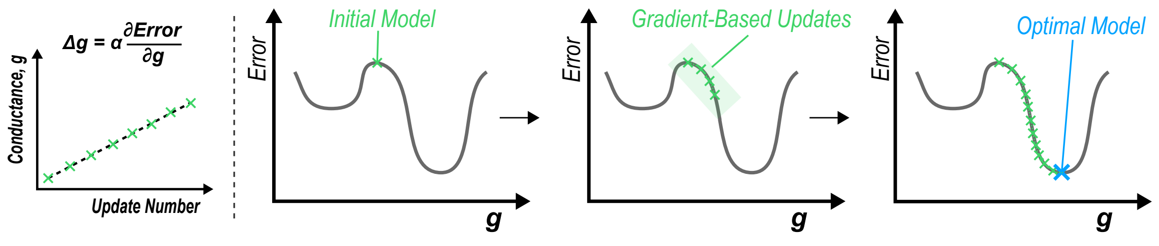


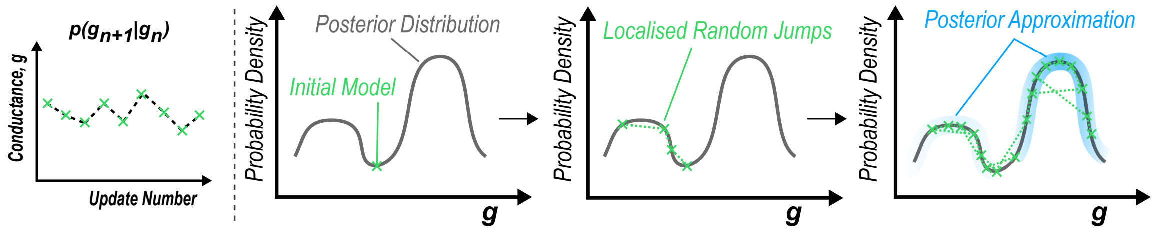
Figure 3.2: (left) A conductance model **g**, composed of four resistive memory elements, defines a linear boundary which (right) separates two classes of data (red circles from blue squares). Through application of a voltage vector **V** to the top electrode of the parallel resistive memories, the summed current flowing out of the common node at the the bottom electrode is equivalent to the dot-product $\mathbf{V} \cdot \mathbf{g}$, which can then be used to determine to what class the data point **V** belongs.

Capturing the uncertainty in parameters is important when faced with the limited, possibly incomplete, data and noisy sensory readings encountered at the edge [289, 175]. This uncertainty in parameter estimation also allows for a description of uncertainty in the model's predictions, which would be desired in a safety-critical edge application - like an implanted medical system. To account for uncertainty, it is preferable to construct a Bayesian model. In this case, parameters are represented, not by single values, but by probability distributions. The distribution of all of the parameters of a Bayesian model, given observed data, is called the posterior distribution, or simply the posterior. As an analogue to deriving an optimal deterministic model through gradient-based updates, the objective in Bayesian machine learning is to learn an approximation of the posterior distribution. When the posterior has been approximated it can be applied to tasks through model inference. To approximate the posterior, sampling algorithms, most commonly Markov Chain Monte Carlo (MCMC) sampling [288], are often employed. Instead of descending an error gradient, MCMC sampling algorithms make localised random jumps on the posterior distribution and continuously store models that lie on it (Fig. 3.3b). The algorithm jumps from a current location in the model space **g** to a proposed location $\mathbf{g}_\text{p}$, according to a proposal distribution $p(\mathbf{g}_\text{p}|\mathbf{g})$ which is usually a normal random variable. By comparing the proposed and current models, a decision is made on whether to accept or reject the proposed model. If accepted, the next random jump is made from this newly accepted model. MCMC sampling algorithms are configured to accept more samples from regions of high probability density on the posterior, that would correspond to a lower error in the gradient-based case of Fig. 3.3a. After a sufficiently large number of such iterations, the accepted samples can be used together as an approximation of the posterior distribution.

In this thesis we realise that, in stark contrast to the case of gradient-based learning algorithms, the properties of resistive memories are incredibly well suited to the requirements of sampling algorithms. This is because the cycle-to-cycle conductance variability, inherent to device programming, is not a nuisance to be mitigated, but instead, a computational resource that can be leveraged by viewing resistive memory devices as physical random variables. More specifically, we exploit the random variable available in a hafnium dioxide-based filamentary random access memory [119] (OxRAM), co-integrated at array level



(a)

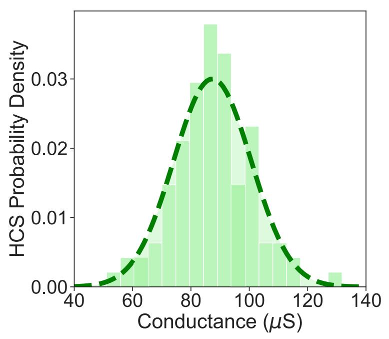


(b)

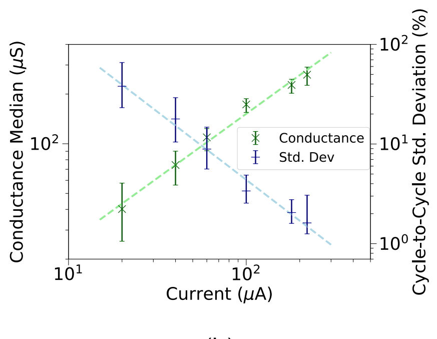
Figure 3.3: **Strategies for training RRAM-based models.** (a) (left) Gradient-based learning algorithms iteratively compute the derivative of an error metric with respect to a conductance model **g**, multiplied by a learning rate α, to determine updates to be applied to the g parameters. The ideal RRAM device should be capable of high precision and linear conductance updates. (right) The three panels show the gradient-descent algorithm for an increasing number of model updates (green crosses). From an initial model the algorithm performs gradient-based updates until it converges to a local minimum in error. (b) (left) Sampling algorithms use a proposal distribution ($p(g_{n+1}|g_n)$) to propose random updates to model conductance parameters which are then either accepted or rejected. The ideal RRAM device for sampling algorithms should offer random conductance updates deriving from a known probability distribution. (right) The three panels illustrate how a sampling algorithm performs local random jumps on the posterior distribution for an increasing number of sampling operations. From an initial model, a proposal distribution generates a series of localised random jumps (dashed green lines) which are then either accepted (green cross) or rejected. The algorithm tends to accept models of a higher probability density on the posterior distribution. After a sufficient number of iterations the accepted models can be used together as an approximation of the posterior distribution (blue haze).

into a 130 nm complementary metal oxide semiconductor (CMOS) process [290]. Each memory point in the array is connected in series to an n-type transistor. The conductivity of an OxRAM device can be modified by the application of voltage waveforms which, through reduction-oxidation reactions at an interfacial oxygen reservoir between the oxide and top electrode, create or rupture a conductive oxygen-vacancy filament between the electrodes. The device can be SET into a high conductive state (HCS), by applying a positive voltage to the top terminal of the device, while grounding the bottom, and thereafter RESET into a low conductive state (LCS), by applying a positive voltage to the bottom electrode while grounding the top.

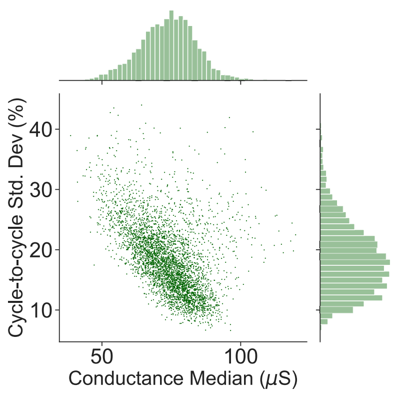
Each time the device is SET, a unique HCS conductance is achieved - resulting from the random re-distribution of oxygen-vacancies within the oxide [271, 272] on consecutive programming cycles. If the HCS conductance is measured over successive cycles, a normally distributed cycle-to-cycle conductance probability density emerges (Fig. 3.4a). The SET operation is therefore analogous to drawing a random sample from a normal distribution. In addition, the median conductance of this probability distribution can be controlled by limiting the SET programming current (I_{SET}) via the gate-source voltage of the series transistor. The relationship between the conductance median and SET programming current follows a power law [291], and the standard deviation of the distribution also depends on the SET programming current (Fig. 3.4b). Therefore, manifested in the physical response of these nanoscale devices, we find the essential computational ingredient required to implement in-memory MCMC sampling algorithms - a physical normal random variable which can be harnessed to propose new models based on the current one. In addition to this cycle-to-cycle variability, device-to-device variability is also present in the technology. The distribution of the normal physical random variable median and standard deviation of each cell in a memory array, cycled under identical conditions, are plotted in Fig.3.4c in order to demonstrate the device-to-device variability that exists in this technology.



(a)



(b)



(c)

Figure 3.4: **Electrical characterisation of OxRAM cycle-to-cycle and device-to-device variability.** (a) Probability density of the HCS cycle-to-cycle variability for a single OxRAM device, measured over 100 RESET/SET cycles (see Appendix 6.3), and fitted with a normal distribution (dashed line). (b) Cycle-to-cycle conductance median and standard deviation for a population of $4,096$ devices, for a range of SET programming current (see Appendix 6.3). Both relationships are fit with a power law. The device-to-device variability in the quantities over the 4096 devices, extending to the 95th and 5th percentiles (two standard deviations), is shown with error bars at each point. (c) Joint distribution of the conductance median and standard deviation of each device in the population. $4,096$ devices have been RESET/SET cycled 100 times under the same programming conditions, and the resulting median conductance and standard deviation of each device has been plotted as a single green point - illustrating the device-to-device variability within a population. Two histograms on opposing axes show the probability densities for the conductance median and standard deviation independently.

We propose that the $N \times M$ resistive memory array depicted in Fig. 3.5a can be trained through MCMC sampling and then store, in the distribution of its non-volatile conductance states, the resulting posterior distribution of a Bayesian model. A single deterministic model, $\mathbf{g}_\text{n}$, is stored in each of the rows where its parameters are encoded by the conductance difference between positive $\mathbf{g+}_\text{n}$ and negative $\mathbf{g-}_\text{n}$ sets of devices - allowing for each parameter to be either positive or negative (Fig. 3.5b).

The principle of our in-situ learning approach is to generate at each row a proposed model, based on the model in the previous row, inline with the Metropolis-Hastings MCMC sampling algorithm [288]. Each parameter of the proposed model can be generated naturally, using the OxRAM physical random variable.

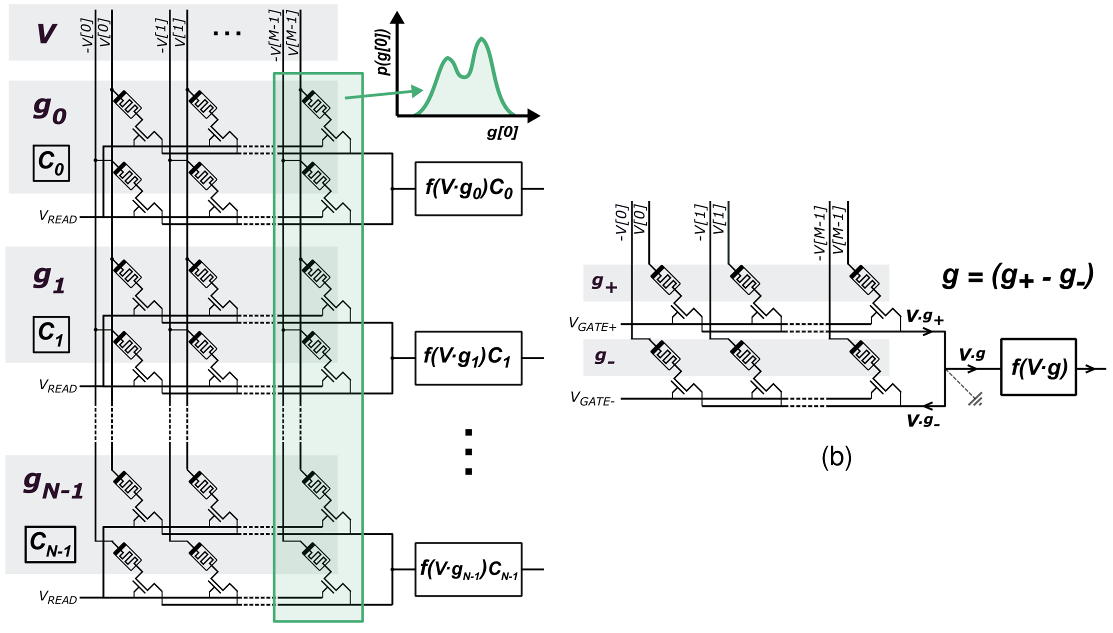


Figure 3.5: (a) Memory array architecture where RRAM conductances store the posterior approximation and calculation used to perform inference with a learned posterior. Each of the N rows store a single conductance model $\mathbf{g}_\mathrm{n}$ and features a digital counter element, C_n. (b) A single array row is the differential conductance between conductance vectors $\mathbf{g}_+$ and $\mathbf{g}_-$. A positive voltage vector, **V**, is applied over the top electrodes of $\mathbf{g}_+$ and a negative opposite voltage vector $-\mathbf{V}$ is applied over the top electrodes of $\mathbf{g}_-$. If the common, bottom node is pinned at a virtual ground, the current flowing into the function block is equal to the dot-product $\mathbf{V} \cdot \mathbf{g}$.

This is achieved through performing a SET operation on each device in the row with a programming current that samples a new conductance value from a normal distribution, provided by its cycle-to-cycle variability, centred on that of the corresponding device in the previous row - as depicted in Fig. 3.6.

By computing a quantity called the acceptance ratio, a decision is made on whether this proposed model should be accepted or rejected. If rejected, the row is reprogrammed under the same conditions - thereby generating a new proposed model. Additionally, the value of a digital counter, C_n, which is associated with the previous row is incremented by one. By tracking the number of rejections in this manner, the contribution of the model in each row to the overall probability density of the posterior approximation (Fig. 3.3b) is accounted for. If the proposed model is instead accepted, the process is repeated at the next row, and so on until the algorithm arrives at the final row of the array. At this point the distribution of programmed differential conductances in each of the array columns corresponds to the learned distributions of each of the Bayesian model parameters - the posterior distribution. After training, the learned posterior distribution in the array can be applied to a task through inference.

Specifically, an MCMC sampling algorithm called Metropolis-Hastings [288] was implemented using the computer in the loop with the 16k device array pictured in Fig 3.7. The devices in the array, which physically exist as a 128×128 array of 1T1R structures, are re-mapped into a virtual address space which realises the structure presented in Fig. 3.5a.

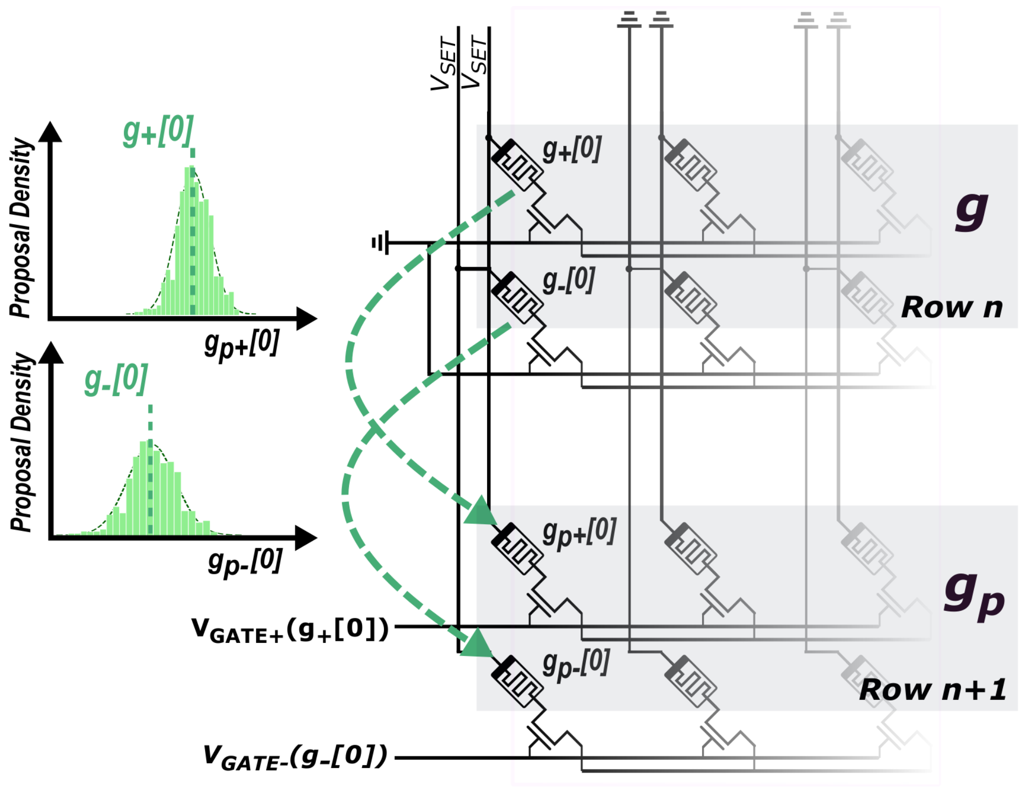


Figure 3.6: Model proposal step between two array rows. Using the known relationship between SET programming current and the read conductances in row n, devices (pointed to by the arrows) $g_{+}[0]$ and $g_{\text{-}}[0]$ in the $n{+}1$th array row are SET. Their SET programming currents are proportional to the conductances of the corresponding devices in the nth row. The new conductance values of $g_{+}[0]$ and $g_{\text{-}}[0]$ in the $n+1$th row are thereby sampled from normal random variables with medians equal to the conductances of the devices in the nth array row (the green probability distributions, left). The SET programming currents are fixed by applying the appropriate voltages $V_{\text{GATE+}}$ and $V_{\text{GATE-}}$ to the transistor gates of row $n+1$.

This is achieved by allocating pairs of sequential banks of M devices (for an M-parameter model) corresponding to the $\mathbf{g}_{+}$ and $\mathbf{g}_{\text{-}}$ conductance vectors to each of the N rows in Fig. 3.5a. A dot-product is performed by reading the conductances of the devices composing $\mathbf{g}_{+}$ and $\mathbf{g}_{\text{-}}$ and subtracting them in the computer-in-the-loop to arrive at **g** and then performing the dot-product between **V** and **g**. Note that in a future version of the system, by integrating appropriate circuits within the device array the dot-product can be performed by simply applying the data points as read voltages and reading the output current consistent with the row design of Fig. 3.5b.

Resistive memory based MCMC sampling begins by performing a RESET operation on each device in the array, rendering all devices in the LCS. Using the variable n to point to the row containing the current model, the algorithm begins with $n = 0$. The devices in row n are SET. Because we do not have strong a priori belief on the initial model parameters, each device is SET using the lowest available V_{WL} (in this case $1.4V$) which corresponds to the lowest SET programming current (40μA). As a result the standard deviation of the initial samples will be high (resulting from the relationship measured in Fig. 3.4b). The devices in the following row, $n+1$, are then programmed with SET programming currents proportional to the conductances read from the corresponding devices in row n. This results in a proposed model,

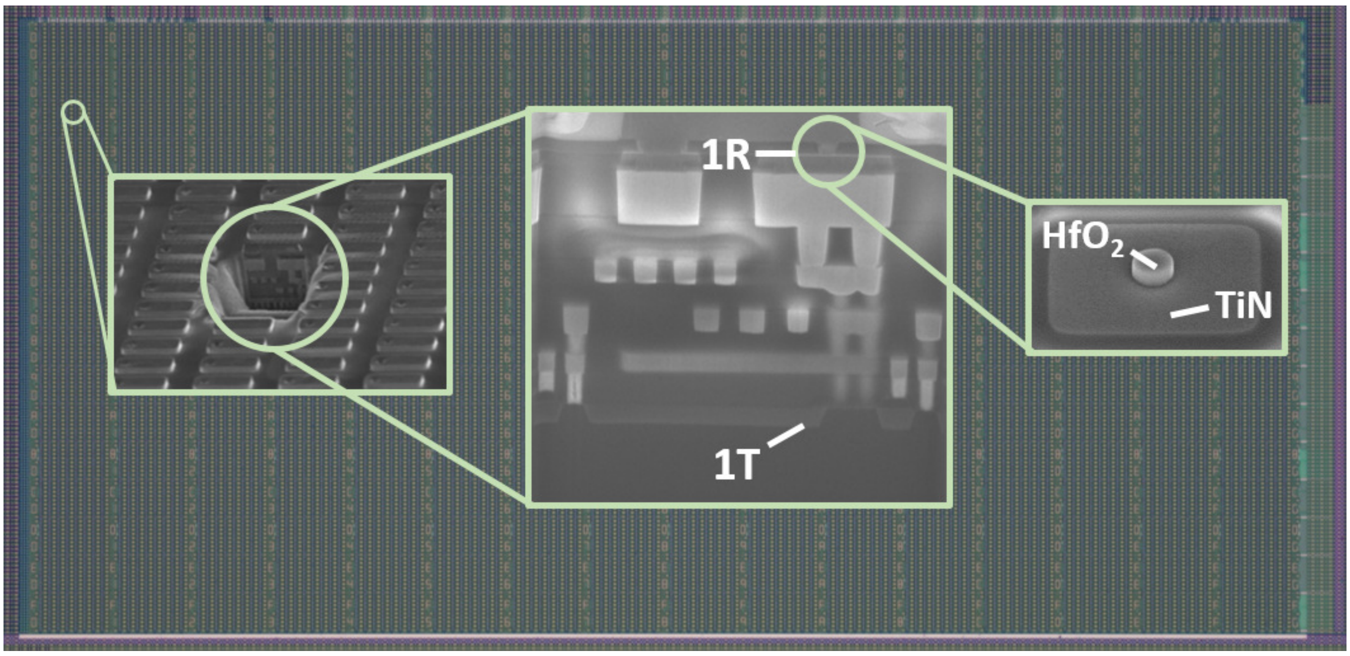


Figure 3.7: OxRAM array used in the experiments. An optical microscopy image of the fabricated array is shown in the background. Scanning electron microscopy images are superimposed on top. (left) A focused ion beam a etch reveals the cross-section of a 1T1R structure (centre). In the front-end-of-line a transistor (1T) acts as a selector for the OxRAM device (1R) integrated in the back-end-of-line. (right) Imaged before deposition of the top electrode titanium layer, a 10 nm thick, 300 nm wide mesa of HfO_2 rests upon a TiN bottom electrode.

$\mathbf{g}_\text{p}$ being generated in row $n+1$ inline with the proposal distribution offered by the cycle-to-cycle HCS conductance variability (Fig. 3.4a):

$$p\left(\mathbf{g}_\text{p}|\mathbf{g}\right) = \mathcal{N}\left(\mathbf{I}_\text{SET}(\mathbf{g}), \sigma(\mathbf{g})\right). \tag{3.1}$$

In doing this, a new conductance value is sampled for each device in row $n+1$ from a normal random variable with a median value corresponding to the same device in row n, offset by device-to-device variability (Fig. 3.4c) that is introduced when moving between successive rows.

The SET programming current in the proposal step is determined by the value of V_WL used to program each device in row $n+1$. To achieve this a look-up table is determined during an initial sweep whereby the entire 16k device array is RESET/SET cycled once per V_WL value that will be used in the experiment. For each of these values of V_WL, the median conductance read across the 16k device array is calculated and inserted in the corresponding entry in the table. Therefore, when programming a device in the row containing the proposed model is required to read the conductance of the corresponding device in row n, and use the value of V_WL with the closest corresponding conductance, as the V_WL used in the SET operation. In the experiments, the look-up table extended from 1.4V to 1.8V in discrete 20mV steps, corresponding to SET programming currents in the range of 40μA-100μA and permitted median conductances in the range of 40μS- 80μS to be used. Programming the devices in row $n+1$ implements the model proposal step depicted in Fig. 3.6.

In Metropolis-Hastings MCMC sampling, after proposing a new model, it is required to make a decision on whether to accept and record the proposed model $\mathbf{g}_{\mathrm{p}}$ or reject and record, once again, the current model $\mathbf{g}$. This decision is made based on the calculation of a quantity named the acceptance ratio a. Because the proposal density is normally distributed (Equation 3.1), and therefore symmetrical [292], it can be written as:

$$a = \frac{p(\mathbf{g_p})}{p(\mathbf{g})} \frac{p(\mathbf{t}|\mathbf{g_p}, \mathbf{V})}{p(\mathbf{t}|\mathbf{g}, \mathbf{V})}. \tag{3.2}$$

This acceptance ratio is a number proportional to the product of the likelihood of a proposed model ($p(\mathbf{t}|\mathbf{g_p}, \mathbf{V})$) and a prior on the proposed model ($p(\mathbf{g_p})$), divided by the product of the likelihood and prior of the current model. Intuitively the acceptance ratio indicates how much more likely is the current model, given a set of observations, than the previous one and how much more does it correspond to prior beliefs on the parameters than the previous one. Given a dataset of D data points where A data points belong to the class the model is required to recognise ($t = 1$) and B other data points that the model should not recognise ($t = 0$) the Bernoulli likelihood of a model is given by:

$$p\left(\mathbf{t}|\mathbf{g}, \mathbf{V}\right) = \prod_{a=0}^{A} f\left(\mathbf{V}_{\mathrm{a,t=1}} \cdot \mathbf{g}\right) \times \prod_{b=0}^{B} \left(1 - f\left(\mathbf{V}_{\mathrm{b,t=0}} \cdot \mathbf{g}\right)\right). \tag{3.3}$$

The function $f(\mathbf{V} \cdot \mathbf{g})$ depends on the specific formulation of the model. The prior of a model is given by:

$$p\left(\mathbf{g}\right) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{\left(\mathbf{g} - \mu\right)^2}{2\sigma^2}\right), \tag{3.4}$$

where the constant σ, corresponds to the prior belief that the posterior distribution is a multi-dimensional normal distribution with a standard deviation of σ in each dimension. In all examples in this thesis, the value of μ was set to zero. These quantities are calculated on the computer-in-the-loop in logarithmic scale, constituting a summation over D points. Because Markov Chain Monte Carlo sampling methods work with small datasets, where overfitting can be avoided through the incorporation of uncertainty into parameter estimations [175], it would be possible to sub-sample the original D data points if the dataset was unreasonably large. In order to decide if the proposed model should be accepted or rejected, a is compared to a uniform random number between 0 and 1, u, generated on the computer using the C++ standard random math package. If a is less than u then $\mathbf{g_p}$ is rejected. This is achieved by programming the devices in row $n+1$ back into the LCS and incrementing the counter at row n, C_{n}, by one. The counter is a variable in the C++ on the computer although, as is the case for all functionality of the computer-in-the-loop, could be integrated as a digital circuit on a future implementation of the system. The devices in row $n+1$ are then SET once more under the same programming conditions - generating a new proposed model $\mathbf{g_p}$ at row $n+1$. This process repeats until a is found to be greater than u. When this is the case, $\mathbf{g_p}$ is accepted whereby the counter at row $n+1$ is incremented by one and the model at row $n+1$ becomes the new current model ($n = n+1$). This new current model $\mathbf{g}$ is then used to propose a model, $\mathbf{g_p}$, at the next row in the array. The model stored at row $n-1$ is then left preserved in the non-volatile conductance states of the resistive memory devices, weighted by the counter value $C_{\mathrm{n-1}}$ which denotes the probability density of that accepted model. As this process repeats, and progresses down the rows of the memory array, the algorithm randomly walks around the posterior distribution leaving information on its probability density imprinted into the non-volatile conductance states of the OxRAM devices and the row counter values. Upon arriving at the final row of the array, $n = N - 1$, the training process terminates resulting in a physical array of resistive memory devices which contains an approximation of the posterior distribution that can then be used in inference.

Performing inference consists of computing the dot-product between a new, previously unseen data point ($\mathbf{V_{new}}$), and the model recorded in each array row. The response of each row is then multiplied by the value in each row counter. The summed response of each row in the array is then divided by the sum of all row counter values. This is also referred to as taking the expectation, and results in a scalar value which has been inferred from the N weighted samples from the posterior distribution approximation:

$$P\left(T_{\text{new}} = 1|\mathbf{V}, \mathbf{t}\right) = \frac{1}{Tot} \sum_{n=\beta}^{N-1} C_{\text{n}} f\left(\mathbf{V}_{\text{new}} \cdot \mathbf{g}_{\text{n}}\right). \tag{3.5}$$

The summation considers only rows of an index greater than β which determines the number of rows discarded to account for the burn-in period. The variable Tot is the sum of all row counter values recorded after the burn-in period.

3.2.3 Application to supervised learning

In this section it will demonstrated experimentally how this approach can be used to address three supervised learning tasks.

Illustrative 2D separation

First, as an illustrative example, we train a Bayesian logistic regression model, realised in a 2048×2 array, to separate two classes of artificially generated data - the red circles from blue squares in Fig. 3.8b To configure the memory array as a Bayesian logistic regression the logistic function is applied on the current flowing out of the row:

$$f(\mathbf{V} \cdot \ \mathbf{g}) = \frac{1}{1 + e^{-\mathrm{S}(\mathbf{V} \cdot \mathbf{g})}}, \tag{3.6}$$

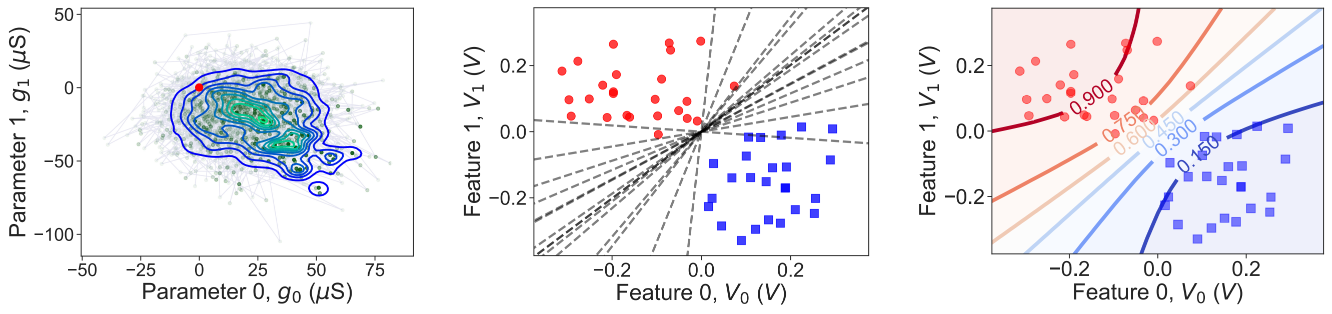
where S is a scaling parameter. This logistic function limits the response of the row dot-product into a probability between 0 and 1. When configured as such, the Bernoulli likelihood of a conductance model is calculated as:

$$p\left(\mathbf{t}|\mathbf{g}, \mathbf{V}\right) = \prod_{a=0}^{A} \left(\frac{1}{1 + e^{-\mathrm{S}(\mathbf{V}_{\mathbf{a,t=1}} \cdot \mathbf{g})}}\right) \times \prod_{b=0}^{B} \left(1 - \frac{1}{1 + e^{-\mathrm{S}(\mathbf{V}_{\mathbf{b,t=0}} \cdot \mathbf{g})}}\right). \tag{3.7}$$

After training, inference can performed on a new data-point $\mathbf{V}_{\text{new}}$ whereby it is assigned a probability of belonging to the class $t = 1$ by computing:

$$P(T_{\text{new}} = 1|\mathbf{V}, \mathbf{t}) = \frac{1}{Tot} \sum_{n=\beta}^{N-1} \frac{C_{\text{n}}}{1 + e^{-\mathbf{S}(\mathbf{V}_{\mathbf{new}} \cdot \mathbf{g}_{\mathbf{n}})}}. \tag{3.8}$$

After the algorithm terminates, the non-volatile conductance states of the devices in the array give rise to the multi-modal posterior approximation plotted in Fig. 3.8a. Two distinct peaks emerge, denoting regions of high probability density where many of the accepted models are tightly packed. A randomly selected subset of these accepted models are plotted as hyper-planes in the space of the data in Fig. 3.8b, each defining a unique linear boundary separating the two clouds of data points belonging to each class. Through combination of all the accepted models, therein using the posterior approximation which now exists in the array, a probabilistic boundary between the two classes emerges in Fig. 3.8c. Any previously unseen data point can hereafter be assigned a probability of belonging to the class of red circles as a function of where it falls on this probability contour.



(a) (b) (c)

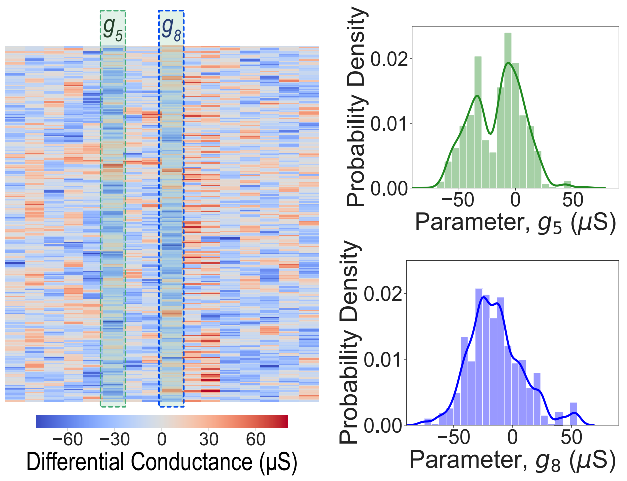
Figure 3.8: **Experimental results on the illustrative 2-D dataset.** (a) Posterior distribution stored within the memory array after the training experiment. The two conductance parameters of each accepted model are plotted as points in the conductance plane (or model space). The initial model stored in the zeroth row is shown by a red dot. The models accepted into the subsequent array rows are plotted as green points with an opacity proportional to the associated row counter value. The transparent lines between green points show the jumps on the posterior made between successive array rows. The resulting posterior distribution is superimposed in a contour plot whereby blue and green contours denote low and high probability density respectively. (b) The two classes of data (red circles and blue squares) and a subset of fifteen models stored in randomly selected rows of the memory array. (c) The probabilistic boundary that is described by the posterior distribution stored within the resistive memory array. Each of the contour lines is annotated with a probability that corresponds to the probability that any point lying on it belongs to the class of the red data. The bounded regions between contours are coloured from red to blue whereby red denotes high confidence that a point within that shaded region belongs to the red class, and blue a low confidence.

Malignant tissue recognition

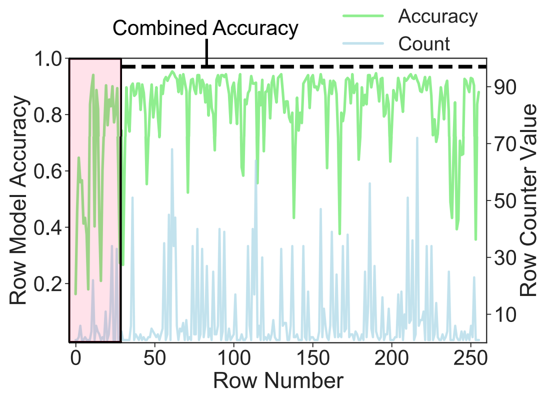
For the second supervised learning task, the Wisconsin breast cancer dataset was used which consists of 569 data points with class labels malignant ($t = 1$) or benign ($t = 0$) [293]. The dataset was shuffled into a training split of 369 points and test split of 200 points that were used in each of the 100 train/test iterations. The Chi2 feature selection algorithm [294] was used to select sixteen features and these features were then scaled around zero such that the model does not require an additional bias parameter. If this step were not performed an extra column could be added to the array which would then learn the distribution of the bias parameter. During training, the algorithm was configured to recognise data points corresponding to malignant tissue samples ($t = 1$). At inference time the 200 previously unseen data-points from the test split were assigned a probability of being malignant using Equation 3.8. Output probabilities greater than or equal to 0.5 corresponded to a prediction of the sample being malignant ($t = 1$) and probabilities less than 0.5 corresponded to a prediction of the sample being benign ($t = 0$). The reported test accuracy corresponds to the fraction of the 200 test data points which were correctly classified.

A Bayesian logistic model based on a 256×16 array was applied to this task. After training has been completed, the differential conductance parameters programmed into the resistive memory array are plotted in a heatmap in Fig. 3.9a. Probability distributions of two parameters from the resulting posterior are shown alongside. In order to visualise the learning process, the classification accuracy of the accepted conductance model in each array row, in addition to its corresponding counter value, are plotted in green and blue traces respectively in Fig. 3.9b. From an initial conductance model, which achieves a poor classification accuracy, the algorithm quickly converges onto the posterior after approximately 32 rows. After this 'burn-in' period, the algorithm tends to accept models into array rows which have a higher probability density, corresponding to models with higher classification accuracy. Strikingly, the accuracy does not saturate but rather increases sharply during burn-in and then proceeds to oscillate between high and medium accuracy conductance models. This is an important property of MCMC sampling algorithms whereby sub-optimal models are also accepted, although less frequently and with a smaller row counter value, ensuring that the true form of probability density of the posterior is uncovered. After the algorithm terminates, the accuracy achieved on the testing set was 97%, indicated by the horizontal dashed black line in Fig. 3.9b. Notably, the combined accuracy of all of the models in the posterior approximation is greater than the accuracy of any of the single accepted deterministic models alone.

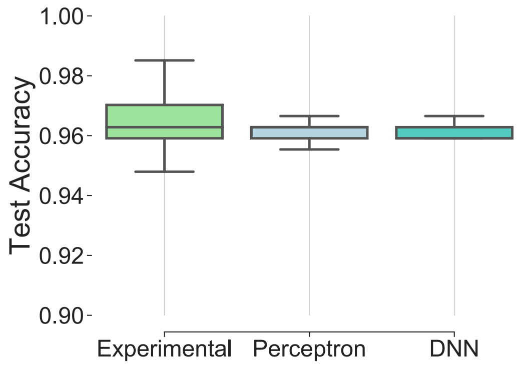
In order to gather statistics on variability between training iterations, the training process was repeated over 100 further experiments and the resulting accuracy distribution is reported in Fig. 3.9c - achieving a median accuracy of 96.3% and, on some iterations, classifying over 98% of test points correctly. It was determined that this median accuracy could be sustained for array sizes down to 96×16 (Fig. 3.9d). In order to benchmark this result, a software-based perceptron as well as a single hidden-layer neural network, using a number of synapses equal to the number of RRAM devices used in the experimental demonstration trained using backpropagation are used. The resulting accuracy distribution of the perceptron and neural network software benchmarks over the 100 training iterations are largely similar - obtaining a median accuracy of 95.8% as plotted in Fig. 3.9c.



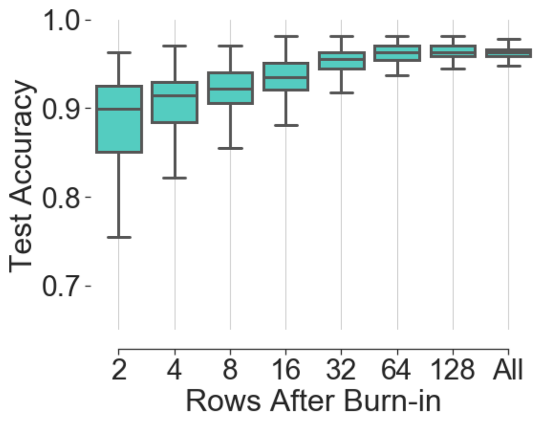
(a)



(b)



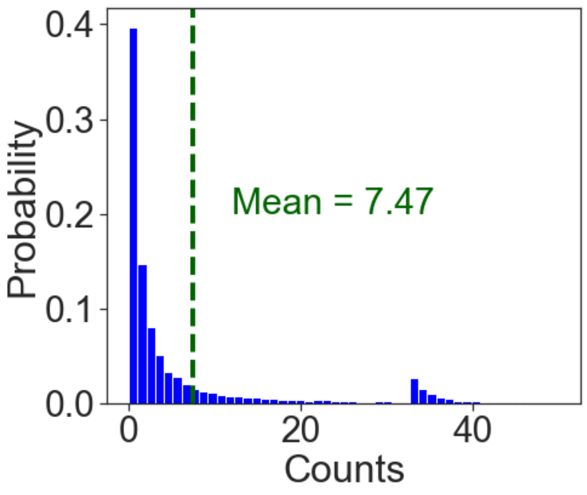
(c)



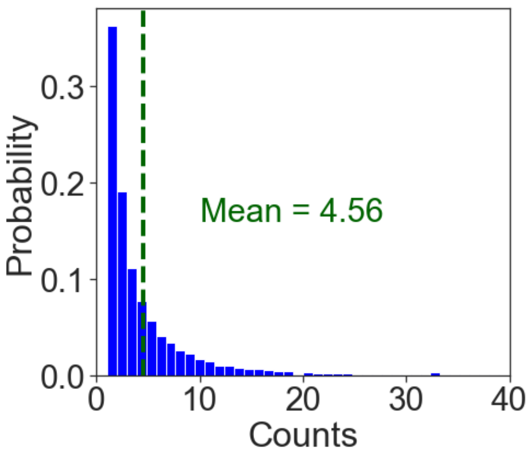
(d)

Figure 3.9: **Experimental results on the supervised classification of breast tissue samples.** (a) (left) Heatmap of the differential conductance pairs in the 256×16 array after a training experiment. Cells within the heatmap are coloured from blue to red indicating the sign and magnitude of each conductance parameter. The row counter weighted distributions within columns five and eight, corresponding to two learned model parameters, are plotted in respective green and blue histograms and fitted with kernel density estimations. (b) Accuracy, evaluated on the dataset, of the conductance model (green) and the row counter value (blue) for each of the 256 rows. The first 32 rows, contained within a red rectangle, have been accepted during the burn-in period. Discarding these rows, the combined accuracy of the remaining rows on the test dataset is 97%, indicated by the horizontal dashed line. (c) Boxplots showing the test accuracy distributions over 100 separate train/test iterations for the malignant tissue recognition task using the same train/test split for (left, light green) the experimental setup, and (center and right) software-based neural network (labelled as DNN) benchmark models. The coloured boxes span the upper and lower quartiles of accuracy while the upper and lower whiskers extend to the maximum and minimum accuracies obtained over the 100 iterations. The median accuracy is indicated with a solid horizontal line. (d) The effect of the array size on the test accuracy. The test accuracy distribution shown as a boxplot for an increasing number of rows after the fixed burn-in of 32.

We next compare the total number of programming operations required to train a full model in our experiment relative to the estimated number required to train RRAM-based implementations of the benchmarks inline with two state of the art RRAM-based backpropagation approaches [261, 266] - labelled as ‘Ambrogio’ and ‘Yao’ in Fig.3.11a. This is a useful metric in understanding the programming efficiency of the approach. Consistent with the observation that, in our experiments, each device was required on average to be RESET/SET cycled only 7.5 times (Fig. 3.10a); we found that RRAM-based MCMC required between two and four orders of magnitude fewer total programming operations than the neural network benchmarks and between one and two orders of magnitude fewer than the perceptron benchmarks (Fig. 3.11a). The substantially smaller number of programming operations highlights the particular harmony between device property and algorithmic requirement achieved through the combination of RRAM and MCMC sampling when contrasted with RRAM-based learning algorithms based on gradients.



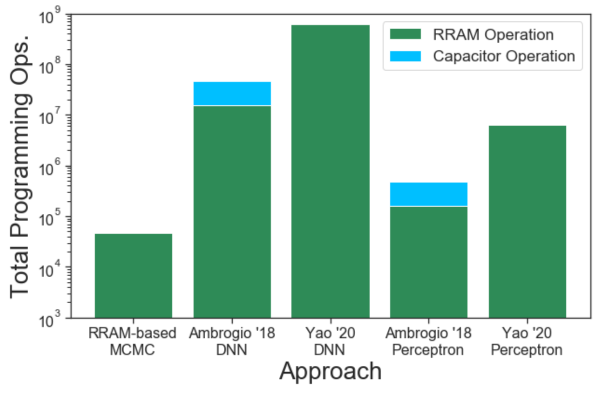
(a)



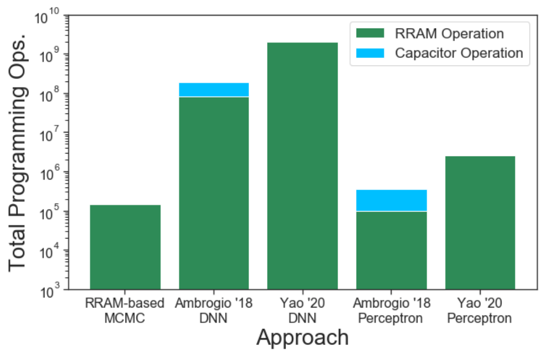
(b)

Figure 3.10: **Probability distribution of row counter values for the supervised learning tasks.** Probability histogram of the values held in the row counters, labelled as counts on the x-axis, over the 100 training iterations for the (a) malignant tissue recognition task and (b) the arrhythmic heartbeat detection task. The mean row counter value over each set of 100 training iterations is shown with a vertical green dashed line – 7.47 and 4.56 for (a) and (b) respectively. Because the value in each row counter denotes the number of times the devices in the subsequent row were RESET/SET cycled, this mean value corresponds to the number of times each device in the array is RESET/SET cycled.

Alongside conductance variability, another major drawback of resistive memory technologies is the rapid degradation of their conductive states when subjected to repeated programming operations - referred to as endurance cycling. In order to asses the impact of endurance cycling on RRAM-based MCMC sampling, we RESET/SET cycled an RRAM array $10,000$, $100,000$ and $1,000,000$ times and, after each decade of cycling, apply RRAM-based MCMC sampling to the same task. After only $100,000$ RESET/SET cycles the array is already no longer suitable for use in a standard memory application due to the overlap of the LCS and HCS cumulative probability distributions plotted in Fig. 3.12a. This situation is further exacerbated by the emergence of permanent write failures over the next two decades of cycling [295] (green curves in Fig. 3.12b). In spite of this however, the accuracy of the models trained after each decade, plotted as dark blue points in Fig. 3.12b, remain comfortably within the bounds observed during the 100 training iterations using the fresh memory array (Fig. 3.9c). A further light blue point is plotted



(a)



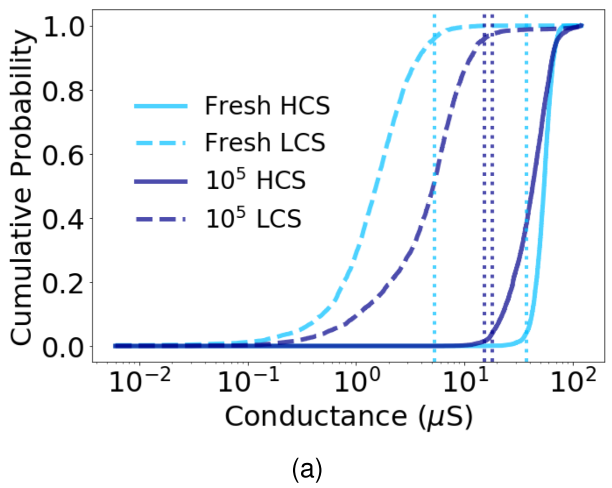
(b)

Figure 3.11: **Bar plots summarising the total number of programming operations required to train a full RRAM-based model for the two supervised learning tasks.** Each bar shows the total number of programming operations observed in the (a) malignant tissue recognition task and (b) arrhythmic heartbeat detection task. The proposed RRAM-based MCMC approach (first column) is compared with state of the art RRAM-based backpropagation approaches [261, 266] applied to both the hidden-layer neural network and perceptron benchmark models. Green (bottom) bars show the total number of operations applied to the resistive memory devices and blue bars, in the case of Ambrogio '18 which uses the change in the voltage on a capacitor for the majority of parameter updates [261], show the number programming operations applied to the capacitors. Each bar is based on the application of the approaches in [261] and [266] to train RRAM-based implementations of the benchmark perceptron and neural network models used here.

whereby the error probabilities measured on the same technology for $10,000,000$ cycles are artificially imposed during the experiment by masking read conductances with values indicative of the two error types with a given probability. Here, even when faced with 4% of devices stuck in the HCS and 1.5% of devices stuck in LCS, the accuracy of the resulting model did not degenerate below that of the fresh array.

Heart arrhythmia detection

Finally, to address a task more representative of learning at the edge, we apply RRAM-based MCMC to train a multi-layer Bayesian neural network experimentally to detect heart arrhythmias from electrocardiogram recordings [296] (see Appendix 6.6). Bayesian neural networks are the probabilistic counterparts of deterministic neural networks whereby probability distributions are used to represent the synaptic connections between neurons, allowing the model to incorporate uncertainty into these synaptic parameters and therefore be robust to over-fitting [175]. Each neuron in the network, like the in the previous two demonstrations, implements the logistic function and does not use a bias parameter. Like in the previous cases the output neuron of the network is interpreted as a probability and the Bernoulli likelihood is used to train the model in the same fashion. As depicted in Fig. 3.13a, ten features obtained from the fast Fourier transform of each heartbeat (see Appendix 6.6) are used to train the Bayesian neural network. Two examples of the synaptic distributions learned during the experiment are also plotted. The trained model was then tasked with the detection of arrhythmic beats in a new, previously unseen, subject. The accuracy distribution over 100 training iterations is plotted in Fig. 3.13b, whereby the model obtains a median



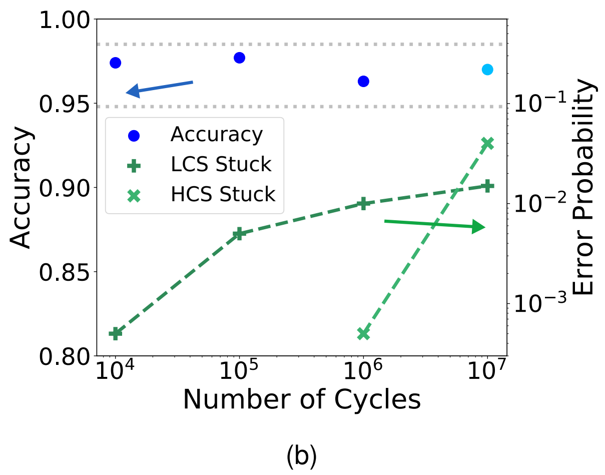
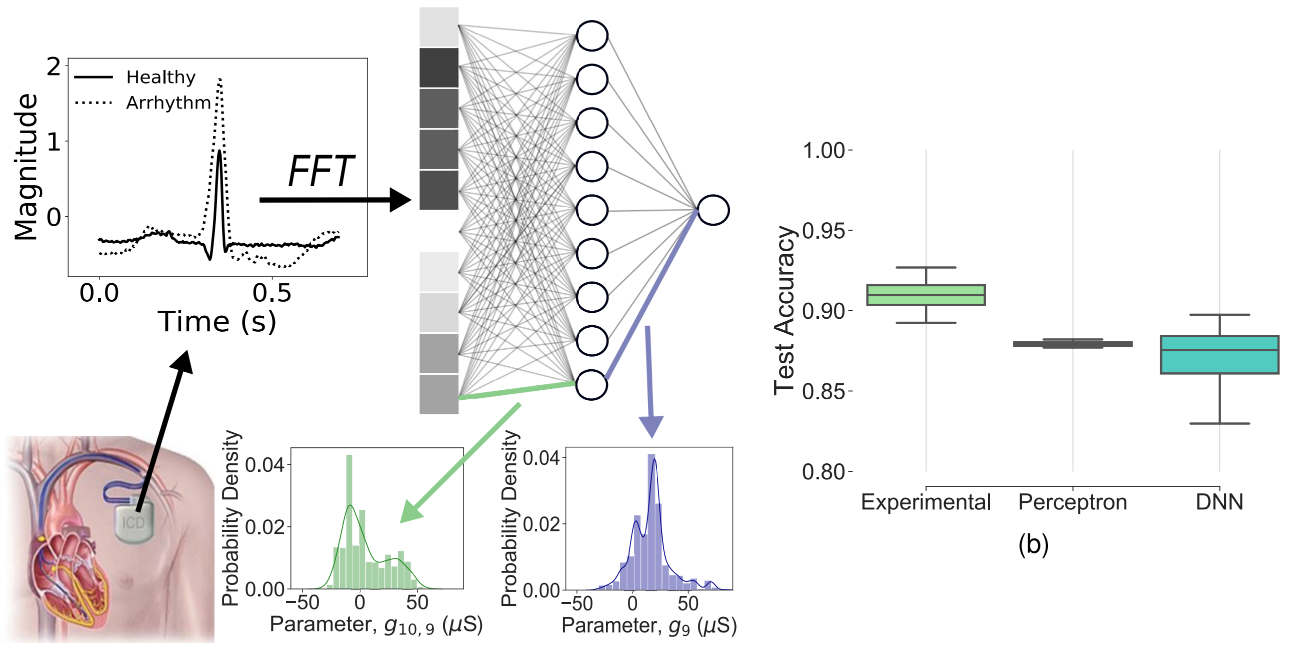


Figure 3.12: (a) The experimental accuracy of the model trained on an array after an increasing number of decades of endurance cycles. The permanent write error probabilities of stuck in LCS and stuck in HCS devices are also plotted on a secondary axis. Arrows have been annotated to indicate which curves correspond to which axes and grey horizontal dashed lines denote the lower and upper accuracies obtained over 100 training iterations using a fresh array. (b) The experimental accuracy of the model trained on an array after an increasing number of decades of endurance cycles. The permanent write error probabilities of stuck in LCS and stuck in HCS devices are also plotted on a secondary axis. Arrows have been annotated to indicate which curves correspond to which axes and grey horizontal dashed lines denote the lower and upper accuracies obtained over 100 training iterations using a fresh array.

test accuracy of 91% and, on some iterations, recognises 93% of beats correctly. This result was benchmarked against software-based perceptron and neural network models which, applied to the same task, obtained median accuracies of approximately 87.5%. The comparison between the total number of programming operations required in our experiment and with the RRAM-based backpropagation approaches was repeated and, once again, it was observed that RRAM-based MCMC sampling required fewer total operations than with the RRAM-based backpropagation approaches (Fig.3.11b) - reflecting the observed average of only 4.5 RESET/SET cycles required per device during the experiment (Fig.3.10b). Considering that the RRAM-based Bayesian neural network outperformed the deterministic software benchmarks (Fig. 3.13b), this task highlights that, beyond being RRAM-compatible, Bayesian machine learning offers an entirely different modelling method that appears well suited to the characteristics of edge learning.



(a) (b)

Figure 3.13: **Experimental results on the supervised detection of arrhythmic heartbeats.** (a) Depiction of the arrhythmic heartbeat experiment. Electrocardiograms (top left) recorded from the heart, potentially using an implanted cardioverter defibrillator [172] (bottom left), are used to extract ten features through a fast Fourier transform (labelled as FFT) which are then used as the input for a multi-layer Bayesian neural network (top right). The Bayesian neural network is trained to detect arrhythmic beats. Two of the resulting synaptic weight distributions after training are also plotted (bottom right). (b) Boxplots showing the test accuracy distributions over 100 separate train/test iterations of the arrhythmic heartbeat detection task using the same train/test split for (left, light green) the experimental setup, and (center and right) software-based neural network (labelled as DNN) benchmark models. The coloured boxes span the upper and lower quartiles of accuracy while the upper and lower whiskers extend to the maximum and minimum accuracies obtained over the 100 iterations. The median accuracy is indicated with a solid horizontal line.

Calibrated on an array level variability characterisation, a behavioural simulator was developed that was determined to correspond well with the experimental results (see Appendix 6.4). It was used to investigate the impact of cycle-to-cycle and device-to-device variability on RRAM-based MCMC. The algorithm was seen to be robust to considerable deviations from the measured values of variability although, for unnaturally small cycle-to-cycle variability (Fig. 3.14a) or unnaturally large values of both cycle-to-cycle and device-to-device variability (Fig. 3.14b), the test accuracy and the total required number of RESET/SET cycles respectively were adversely affected.

3.2.4 Application to reinforcement learning

We now demonstrate that the approach can be extended to the reinforcement learning setting using the behavioural simulator. In contrast to supervised learning, reinforcement learning does not require a labelled dataset - a potentially appealing prospect for a practical edge learning system. Instead, a model

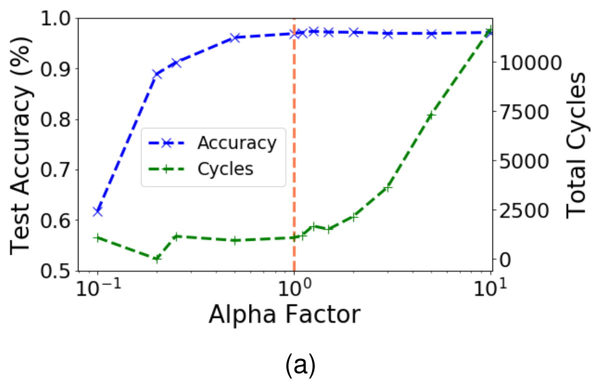
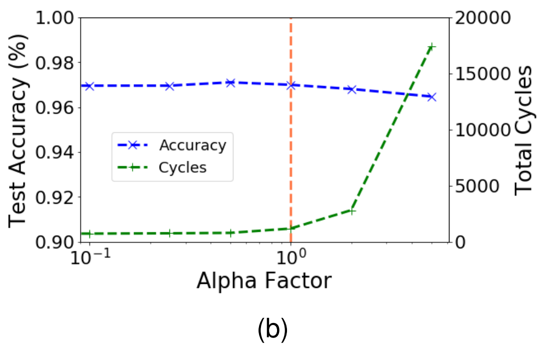


Figure 3.14: **Simulation of the impact of the cycle-to-cycle and device-to-device variability on the test accuracy, for the supervised malignant tissue recognition task.** (a)/(b) The cycle-to-cycle / device-to-device variability was multiplied artificially by a constant alpha factor in 10 iterations of simulation per alpha factor. The mean test accuracy of over the resulting ten iterations was plotted, as well as the total number of RESET/SET cycles.



is tasked with determining the actions of an agent in a physical or simulated environment in real-time [297] based on interaction with that environment. The agent observes, at each timestep, input information pertaining to the current state of the environment (**V**) and, as a function of the actions taken by the agent (**a**), a scalar reward (r) is received. The objective in reinforcement learning is to realise a model (often referred to as a policy) that allows an agent to take actions in an environment which maximise its expected reward. Here, we apply RRAM-based MCMC as a policy-search algorithm [298] and learn a posterior distribution in terms of reward.

Specifically, we address the Cartpole control task [299] that is becoming an important validation task for RRAM-based reinforcement learning approaches [263, 300]: an agent learns how to control a pole balanced on top of a cart by accelerating to the left or right as a function of four observed environmental variables describing the velocity and position of the cart and pole (Fig. 3.15a). To achieve this, we employ a perceptron model, effectively realised using two 512×4 memory arrays. The output neuron with the largest response, as a function of the input features, dictates the action taken by the agent on each timestep.

The python library gym was used to simulate the Cartpole environment. The Cartpole environment provides four features to the behavioural simulator at each timestep of the simulation. The behavioural simulator then specifies the actions to be taken by the agent in the environment at the next simulation timestep. Under application of a new observation from the environment **V** the response

$$f(\mathbf{V} \cdot \mathbf{g}) = S\,\mathbf{V} \cdot \mathbf{g}, \tag{3.9}$$

was calculated, where S is a scalar constant. Rows of equivalent index in both arrays share a common row counter and are programmed and evaluated at the same time. The devices within the zeroth row of both arrays are initially SET by sampling from a normal random variable with the lowest available conductance median (in this simulation the conductance range extended from 50μS-200μS). The initial model is evaluated during a training episode where the cumulative reward received during the episode is recorded. For each timestep that the agent does not allow the pole to rotate 15 degrees outwith of perpendicular, or does not move outwith the bounds of the environment (both resulting in early termination of the episode), the agent receives a +1 reward. If the agent maintains the pole balanced for 500 timesteps the episode terminates resulting in an episode in which the agent has received the maximum possible score of 500. Respective models are then generated at the first row of each array by sampling new models from normal random variables with medians equal to the conductances of the corresponding devices in the zeroth row, offset by device-to-device variability. The two proposed models in the pair of rows determine the actions of the agent during the following training episode whereby the agent accelerates to either the left or the right at each timestep of the episode as a function of which array exhibited the greater response (Equation 3.9). As a function of the cumulative reward achieved during this training episode a decision is made on whether to accept or reject the proposed model using the acceptance ratio:

$$a = \frac{p(\mathbf{g_p})}{p(\mathbf{g})} \frac{p(r|\mathbf{g_p}, \mathbf{V}, \mathbf{a})}{p(r|\mathbf{g}, \mathbf{V}, \mathbf{a})} \kappa^{-1}. \tag{3.10}$$

Instead of using the ratio of likelihoods of the proposed and current models as in the supervised case, the ratio of episodic rewards for a model **g**, episodic observations **V** and episodic actions **a** are used: this is simply the scalar reward value obtained after an episode acting according to a proposed model $\mathbf{g}_\text{p}$ divided by the reward received when the current model **g** was accepted. This demonstrates a particular strength of applying MCMC sampling in a reinforcement learning setting, relative to supervised learning, since the calculation of the acceptance ratio can be achieved in a single calculation instead of an operation involving a sum over all data points in the log-domain. The prior calculation is the same as in Equation 3.4. The acceptance ratio is then multiplied by a constant κ^{-1} which acts a hyper-parameter that determines the extent of exploration from higher to lower reward regions on the posterior. If the acceptance ratio is less than a uniform random number generated between 0 and 1 then the proposed model at the first row is rejected and the row counter, C_0, is incremented by one. A new model is then proposed at the first row. If this new model achieves a cumulative reward during the next training episode, $(p(r|\mathbf{g_p}, \mathbf{V}, \mathbf{a}))$, such that the acceptance ratio is now greater than a uniform random number, the two models within the first rows of both arrays are accepted and then become the current models. The reward which was obtained when these current models were accepted is recorded and thereafter used as $p(r|\mathbf{g}, \mathbf{V}, \mathbf{a})$ in the calculation the acceptance ratio. After training has been completed upon the algorithm reaching the final array row, actions are determined during 100 testing episodes by calculating:

$$P(a_\text{new} = 1|\mathbf{V}, \mathbf{r}) = \frac{1}{Tot} \sum_{n=\beta}^{N-1} C_\text{n}(\mathbf{V_{new}} \cdot \mathbf{g_n}), \tag{3.11}$$

for each array at each simulation timestep. The summation considers only rows greater than index β, which determines the number of rows discarded to account for the burn-in period. The variable Tot is the sum of all row counter values recorded after the burn-in period. The array with the largest response at each timestep determines the action taken during inference in a winner-take-all fashion. It should be noted that although we have used the notation $p(r|\mathbf{g}, \mathbf{V}, \mathbf{a})$, for means of consistency with the rest of this section, this quantity is not a probability but in fact a reward.

The training process is one again visualised by plotting the reward received by the agent when each

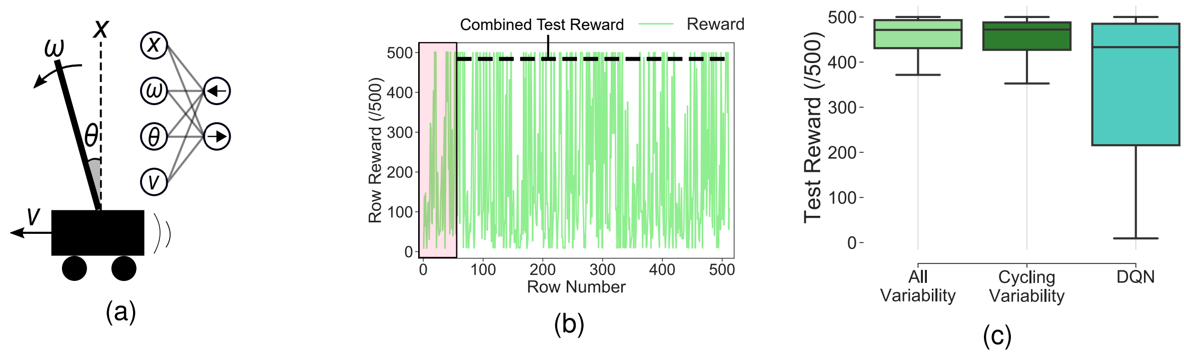


Figure 3.15: **Behavioural simulation results on the Cartpole reinforcement learning task.** (a) Diagram of the Cartpole task: learning how to accelerate left or right in order to maintain a pole balanced on top of a cart within 15 degrees of normal (the vertical dashed line). The environment is described by four features; X - x-position of the cart, θ - angle of the pole to vertical, ω - angular velocity at the tip of the pole and ν - velocity of the cart. These four features serve as input to the perceptron model (upper right) where the two output neurons determine whether the agent accelerates to the left or right. (b) Reward obtained during the training episode when each of the conductance models was accepted into each row of the 512 memory array rows (the maximum reward is 500). A burn-in period of 64 rows is denoted with the red rectangle. The mean reward obtained over 100 test episodes using the combination of the models accepted after the burn-in, equal to 484 out of a maximum score of 500, is denoted with a horizontal dashed black line. (c) Boxplots showing the distribution of the mean test reward obtained during 100 testing episodes achieved in the Cartpole task over 100 separate train/test iterations. (left, light green) Behavioural simulation considering device-to-device and cycle-to-cycle variability, (centre, dark green) behavioural simulation considering only cycle-to-cycle variability and (right, blue) a DQN benchmark model. The coloured boxes span the upper and lower quartiles of mean reward, the upper and lower whiskers extend to the maximum and minimum mean rewards obtained, and the median mean reward is indicated with a solid horizontal line.

row was accepted in Fig. 3.15b. The reward is seen to oscillate during training as the algorithm explores the posterior distribution. After training, the agent then uses the learned posterior approximation to select actions over 100 testing episodes - achieving a mean test reward of 484 out of 500. To determine the variability between training iterations, this procedure was repeated 100 times and the distribution of mean test reward is plotted in Fig. 3.15c. Over the 100 training iterations the median mean test reward obtained was 475. In order to benchmark this result, a Deep-Q Network [301] (DQN) reinforcement learning model was applied to the same task. The DQN employed one hidden layer of neurons with a total number of synapses equal to the number of differential conductance pairs used in the two memory arrays. The distribution of mean test reward obtained by the DQN is plotted in Fig. 3.15c where it obtained a median mean test reward of 420 - less than that of the RRAM-based perceptron - while also exhibiting greater variability between training iterations.

In order to assess the impact of device-to-device variability (Fig. 3.4c) on this task, the simulation was repeated without its consideration. The resulting test reward distribution is plotted in Fig. 3.15c where it is seen to be largely equivalent to that case where device-to-device variability was considered. This result is consistent with studied impact of device-to-device variability in the supervised learning task, challenging the longstanding conception that device-to-device variability is a disadvantage in RRAM-based machine learning that requires mitigation. However, this should not necessarily come as a surprise: unlike in gradient-based optimisation, where device-to-device variations impede the proper descent down an error gradient, MCMC sampling is rooted in randomness.

3.2.5 System level energy estimation

To understand the overall energy cost of a system based on RRAM-based MCMC sampling, we evaluated the energy consumption of all the elements of a full system using commercial (Cadence) integrated circuit design tools. This discussion was made in the case of the reinforcement learning task which features a straightforward computation of the acceptance ratio and therefore less designer dependent choices than in the supervised case. Reinforcement learning shows particular promise for eventual edge applications since the system does not need to store a labelled training dataset. As detailed, the reinforcement learning process alternates between three types of phase: evaluating a model (Step 1), accepting or rejecting the model (Step 2), and programming the same row again if the proposal is rejected, or the next row of the array if accepted (Step 3). We evaluated the energy consumption of the operations necessary in these three phases. Our evaluation was made in the 130 nm CMOS process of the test chip featured throughout the chapter, as well as in a more modern technology to evaluate how our approach scales using the validated physical design kit of a commercial 28 nm technology that targets low-power applications. In the 28 nm evaluation, RRAM properties are supposed identical to those in the 130 nm measurements: due to the filamentary nature of RRAM switching, the RRAM electrical properties of the HCS are relatively independent of the RRAM critical dimension [302]. We used the following methodology. The energy consumption of the digital circuits were obtained by writing a synthesisable register-transfer level (RTL) description of the circuit in SystemVerilog, generating Value Change Dumps (VCD) files representative of the Cartpole task using the Cadence ncsim simulator, and synthesising the systems to standard logic gates using the Cadence RTL Compiler. Energy estimates are provided by Cadence RTL Compiler using the appropriate VCD files. For analogue circuits, the circuits were designed using Cadence Virtuoso, and their power consumption estimated using the Cadence Spectre simulator. We realised specific designs for all parts of the system, except for Analog-to-Digital conversion and SRAM access, where energy numbers from a commercial vendor and from the literature [303] were used, respectively.

The three different steps required by the system are as follows;

- **1. Evaluation of a model**: In our approach, to evaluate a model, a cartpole episode is run. The system receives input from each of the four sensors (Fig. 3.15a), which are presented as analogue voltages to the columns of the two memory arrays of the system using low precision 5-bit digital-to-analogue converters. Then the decision to move the cartpole left or right is obtained by comparing the V·g values of both arrays. To calculate these two quantities and compare them, we evaluated the circuit of Fig. 3.16a, using conservative assumptions. The length of the read operation was chosen based on a worst case (smallest difference between the V·g values of both memory arrays), extracted from the Cartpole results. The simulations were performed in the 130 nm process, and we assumed that the energy consumption would not scale in 28 nm. Then, the system receives a binary reward from the environment, which is counted by our system. The energy consumption of this counter and of the state machine of the process were evaluated using a dedicated SystemVerilog description, in 130 nm and in 28 nm.

- **2. Acceptance/rejection of a model**: In our proposed design, the process of accepting and rejecting the model is performed using digital electronics, using 9-bit fixed point (integer) computation. A full SystemVerilog description was designed for accurate evaluation. The different aspects of the computation are summarised in the simplified data path presented in Fig. 3.16b. For increasing energy efficiency, the prior was not computed according to equation 3.4, but using an approximate value $|g|$, which was verified to provide equivalent accuracy by simulation of the cartpole task. This prior value can be obtained using a read operation of the memory array, and an 8-bit analogue-to-digital conversion (ADC). We did not design an ADC, but used typical energy values from the literature on 8-bit low power ADCs [303]. We performed all computation in the log domain, so that equation 3.10 is transformed into integer sums and subtractions, which consume very low power. Depending on the value of the resulting log-acceptance ratio, three situations are possible. If the value is lower than a rejection threshold, the model is definitively rejected. If the value is higher than an acceptance threshold, the model is definitively accepted. If the value is between these two values, the actual value of the acceptance ratio (coded in fixed point) needs to be obtained from its log value. For this purpose, we store in a static memory (SRAM) the values of acceptance ratio corresponding to their log value, in the intermediate range. Energy consumption for the SRAM access was not simulated, but obtained using data from a commercial solution. Then, to decide if the model is accepted, an 8-bit pseudo-random number, generated using a linear-feedback shift register (LFSR), is compared to the acceptance ratio. The choice to use fixed point representation in logarithmic scale to represent probabilities, the reliance on a LFSR to generate random numbers, and the use of the SRAM to get acceptance ratio make this step very low power, although it performs relatively sophisticated operations.

- **3. Programming of the RRAM cells of the next row**: For programming the next row, each device needs to be RESET and then SET with compliance current proportional to the conductance of the corresponding device of the previous row. To perform this, we evaluated the circuit of Fig. 3.16c, which can perform this operation precisely: it mirrors a multiplied version of the read current of an RRAM device while imposing a virtual ground. The operational amplifier is based on a standard two-stage operational transconductance amplifier design. The current mirror has asymmetrical transistors, so that the read current is multiplied into the appropriate compliance current. This takes advantage of the fact that the relationship between conductance median and SET programming current can be well approximated with a linear relationship in conductance range used in our experimental system. In Fig. 3.16c, the read current from a device in the left memory array is used for programming a device in the right memory array. This is in contrast with the schematics in Fig. 3.5a where this operation is done within the same array. Because these operations are performed in the analogue domain, and in parallel, it would be required to read and write on the same column of the array presented in the main text – something that is of course impossible. In our design therefore the rows of the two memory arrays alternate between the two arrays – in other words the odd rows of the "go left" actions and the even rows of the "go right" action exist on the same array and vice-versa. This choice has no impact on functionality, simplifies the RRAM write circuitry and, notably, allows

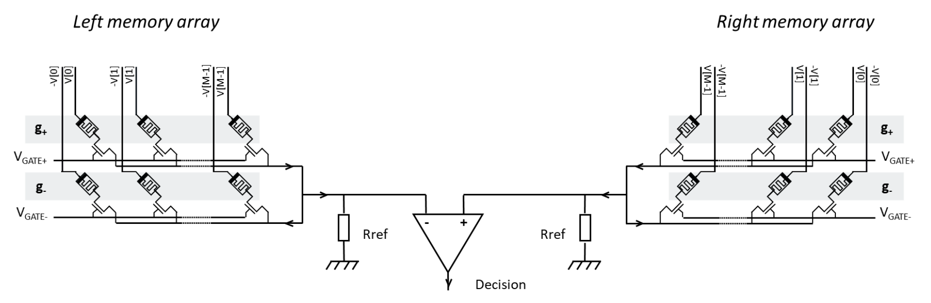
	Step 1 (Model evaluation)	Step 2 (Model acceptance/rejection)	Step 3 (RRAM programming)	Total
Number of repetitions	$500 \times 10 \times 512$	10×512	10×512	
Total energy (130nm)	5.8μJ	120nJ	1.1μJ	**6.9μJ**
Total energy (28nm)	2.5μJ	34nJ	1.1μJ	**3.6μJ**

Table 3.1: Table showing the number of operations required in each step of the MCMC sampling process for the Cartpole reinforcement learning task in addition to the energy required in each of these steps for 130nm and 28nm technology nodes. A final column shows the total energy required to train the full model with each technology.

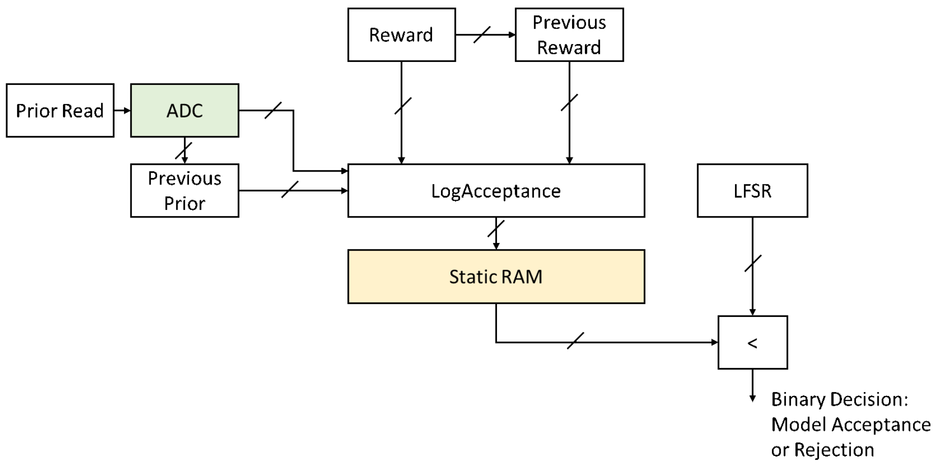
for analogue-domain parallelisation.

The results plotted in Figs. 3.17a and 3.17b show the programming of RRAM devices (step 3) consumes considerably more energy than steps 1 and 2. However, these steps are repeated a highly non-uniform number of times, as presented in Table 3.1. This Table also shows that when training the full model, step 1 consumes the most energy, for both the 130 nm and the 28 nm technology. This suggests that effort should be spent on optimising step 1, which may be achieved by using comparator designs optimised for low power consumption. However, the relative importance of the different steps will depend tremendously on the details of a task: e.g., if the cartpole experiments are shorter than 92 time steps (215 in the 28 nm technology), step 3 becomes the dominant source of energy consumption – at which point it becomes most important to reduce as much as possible the number of device programming operations. This resonates well with the results seen in Figs. 3.11a and 3.11b where RRAM-based MCMC sampling was observed to orders of magnitude fewer total programming operations than RRAM-based backpropagation approaches. The total energy consumption summing all three steps is 6.9 μJ in 130 nm, and 3.6 μJ in 28 nm. These values are highly attractive.

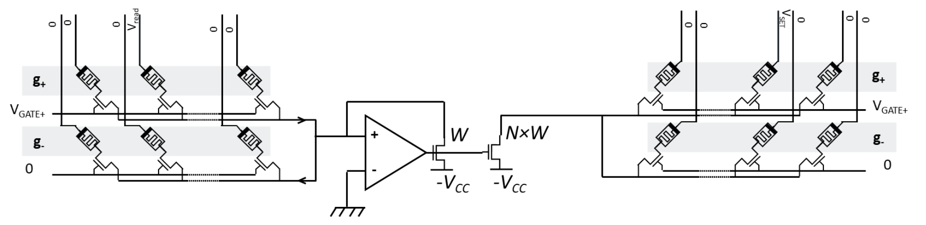
To perform an order of magnitude comparison with a conventional CMOS implementation, we implemented the same algorithm (using the same assumption for the prior) in low-level C code, and simulated it on an Intel Xeon Gold 5220 processor (using a single core), based on a 14 nm CMOS technology that notably is more aggressively scaled than the nodes considered in our design. This implementation consumed around 600 mJ. A frequently used metric for evaluating artificial intelligence dedicated hardware is the number of (giga)teraoperations per second and per Watt ((G)TOPS/W). In our case, considering that the total energy consumption is dominated by step 1, and counting only model evaluation/inference, we arrive at estimates of 3.5 TOPS/W in 130 nm and 8.2 TOPS/W in 28 nm when our design is used to support the perceptron model used in the Cartpole task which features four inputs. This TOPS/W metric would increase proportionally for a model with more inputs (i.e. a memory array with more columns), since the read operation energy consumption is largely independent from the model size. This metric, which is very



(a)



(b)



(c)

Figure 3.16: (a) Circuit to perform the comparison between the linear activation of two rows from the left and right arrays. The binary output of the comparator determines the action taken at each timestep. (b) Simplified flow chart of the digital circuits used to implement RRAM-based MCMC. (c) Circuit showing the simultaneous read of a device the current rows and programming of the corresponding device in the proposed row.

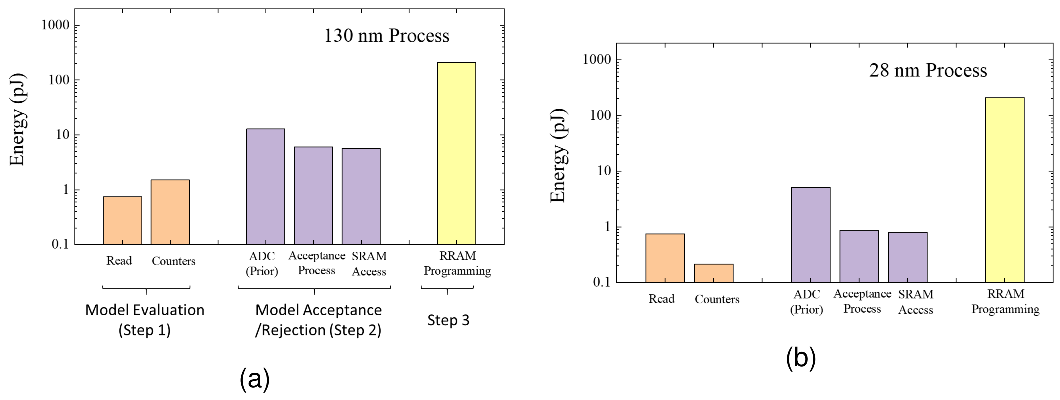


Figure 3.17: (a)/(b) Bar plot showing the energy required per operation for each of the aspects of the designed RRAM-based MCMC system for 130nm/28nm technology nodes.

adapted for deep neural network accelerators, ignores the supplementary operations, such as Gaussian random number generation, that are required in MCMC learning algorithms. Therefore the most useful metric of efficiency, that allows it to be compared with approaches based on other fundamental operations, appears to be the total learning energy for a task.

We now discuss, in a qualitative manner, how these results would project to the supervised learning tasks. In these tasks, the acceptance process requires the value of V·g, instead of comparison between two memory arrays. This means that step 1 will require an ADC conversion. This makes this step more energy hungry, and likely to be more dominant than in the reinforcement learning case. Based on this assumption, in the case of the Wisconsin malignant tissue recognition task, we find that the training energy cost is 3.4 μJ in the 130 nm technology, and 1.3 μJ in 28 nm. In the case of the arrhythmic heartbeat detection task, we find a training energy cost of 12 μJ in the 130 nm technology, and 4.7 μJ in 28 nm. These numbers are once again highly attractive. Finally, to perform another order of magnitude comparison we estimate the energy required to train the supervised learning benchmark neural networks using the GOPS/W metrics reported by state of the art GPUs [304]. It should be noted however that the GOPS/W reported for these systems are normally valid for only very large neural networks (considerably larger than those considered in our benchmarking) trained under optimal conditions. We used the Profiler module available in TensorFlow, where the benchmark models were trained, to determine the number of operations required in each backpropagation update. Multiplying this number by the GOPS/W metric of the most performant GPUs lead to a required energy of approximately 6mJ – although in reality this number would be expected to be greater. Once again, however, this estimate is considerably larger than the energy estimated to experimentally train each of the full models through RRAM-based MCMC. The considerable differences between traditional CMOS implementations and our approach highlights the benefits of a dedicated in-memory implementation of machine learning algorithms – particularly if their fundamental operations can be realised naturally through inherent nano-device physics.

3.3 Ex-situ transfer of a Bayesian neural network

3.3.1 Section introduction

In recent years, neural network models [256] have demonstrated human-level competency in multiple tasks, such as pattern recognition [305], game playing [301], and strategy development [306]. This progress has led to the promise that a new generation of intelligent computing systems could be applied to such high complexity tasks at the edge [169]. However, the current generation of edge computing hardware cannot support the energetic demands nor the data volume required to train and adapt such neural network models locally at the edge [3, 257]. One solution is ex-situ training: a software model is trained on a cloud computing platform and then subsequently transferred onto a hardware system that acts only to perform inference [170, 171]. The engine-room of such inference hardware is the dot-product (multiply-and-accumulate) operation that is ubiquitous in machine learning. Non-von Neumann dot-product implementations based on non-volatile resistive memory [121, 122, 120, 119] (RRAM) technologies, otherwise known as memristors, are a particularly promising path towards reducing the energy required during inference.

As was the case in the in-situ learning setting, intrinsic cycle-to-cycle and device-to-device conductance variability once again constitute a considerable challenge. This random variability constrains the number of separable multi-level conductance states that can be achieved, preventing the high-precision transfer of the model parameters in a single programming step. To mitigate against this variability, iterative closed-loop programming schemes (also referred to as program-verify schemes) are often employed whereby devices are repeatedly programmed until their conductance falls within a discretised window of tolerated error. However, such approaches entail costly circuit overheads as well as energy and time during model transfer [307, 308, 264]. Other approaches propose to employ multiple 1T1R structures in parallel at each array cross-point [309, 310, 277] which, although allow for a higher precision in the transferred weight, entail additional costs in area and transfer energy. Other approaches propose to sidestep intrinsic randomness altogether through the quantisation of conductance-levels into one of either a low-conductance or a high-conductance state in binarised neural networks [311]. However, such models require a significantly higher number of neuron elements and model parameters to approach the performance of conventional neural networks [312]. It should be noted that, unlike a separate body of work that leverages the random switching properties of magnetic RRAM [158, 160] or stochastic electronic circuits [313, 314] to perform Bayesian inference, this chapter is concerned with performing inference of ex-situ trained Bayesian neural networks transferred onto an RRAM-based hardware.

The reality is that the programming of resistive memory is an inherently random process and, the devices therefore, are not well suited to being treated as deterministic quantities. Fortunately however, as has been seen in the previous chapter, this randomness follows stereotyped probability distributions and allows resistive memory devices to be instead yielded as physical random variables [280, 315]. This opens up an alternative direction of exploration whereby RRAM conductance states can be manipulated within the frameworks of probabilistic programming and Bayesian modelling [289]. In this section, we propose and demonstrate experimentally an approach for the ex-situ training and subsequent transfer of a Bayesian neural network on to a resistive memory based inference hardware. Because Bayesian neural networks, like resistive memory conductance states, describe model parameters as probability distributions, and not single high-precision values, it suggests a more natural pairing of algorithm and technology. In this setting, the objective is no longer to precisely transfer a single parameter from a software model to the corresponding device in a resistive memory array but to transfer a probability distribution from the software model into a distribution of device conductance states.

In this work, we first propose to use an expectation-maximisation algorithm to decompose a probability distribution into a small number of random variable components; each corresponding to the cycle-to-cycle conductance distribution of RRAM programmed under a given programming current. We then demonstrate experimentally, with an array consisting of $16,384$ fabricated hafnium dioxide 1T1R structures, that these random variable components can be used to transfer the original probability distribution into the conductance states of a column of RRAM devices. We show that this approach can be leveraged to achieve the transfer of a full Bayesian neural network in a single programming step. We also describe an RRAM-based hardware capable of storing, and performing inference with, such a Bayesian neural network. Finally, a Bayesian neural network model is trained ex-situ and transferred using the proposed technique to the experimental 16k device array. The transferred model is used for inference in a simple illustrative classification task where the decision boundaries that were learned ex-situ are seen to be well preserved in the model transferred into the inference hardware.

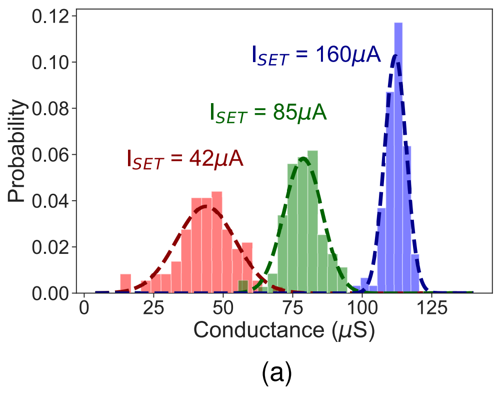
3.3.2 Expectation-maximisation based parameter decomposition

As was leveraged in the previous chapter, RRAM devices are normal random variables in their high conductance state. As a further example of this three of the random variables available from a single device under three programming conditions are plotted in Fig. 3.18a. What is evident from this plot is that, as the median of the random variable increases, its standard deviation decreases - the two quantities are intrinsically tied. This relationship between the cycle-to-cycle conductance standard deviation and median, over the full population of $16,384$ devices in the 1T1R array, is plotted in Fig. 3.18b where it is seen to be well approximated by a linear function.

Bayesian neural networks are variants of conventional neural networks whereby parameters are not single values, but probability distributions [175]. The distribution of each parameter encapsulates the uncertainty in its estimation which allows for a model to avoid over-fitting given, for example, a small training dataset or noisy sensory observations [289] - this is in contrast to deterministic neural networks which often suffer from severe over-fitting under similar conditions.

Therefore, the challenge is to transfer these software-based probability distributions into a plurality of device conductance states on an RRAM-based inference hardware. The fundamental insight here is that, because RRAM conductance states are also probability distributions, they owe themselves more naturally to the transfer of ex-situ trained Bayesian neural network models than deterministic ones.

We propose that the probability distribution of each Bayesian neural network parameter can be approximated by a linear combination of weighted normal random variable components - determined using a Gaussian mixture modelling approach [316]. In a Gaussian mixture model, each of the K Gaussian distributions, also referred to as normal random variable components, are characterised by a median, a standard deviation and a weighting factor. These three parameters per component are updated itera-



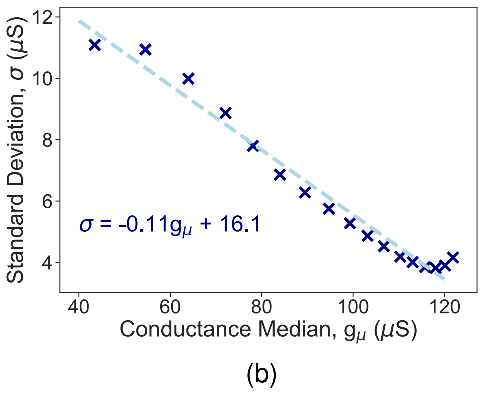


Figure 3.18: (a) Three cycle-to-cycle conductance variability probability distributions for a single device programmed 100 times under three different SET programming currents. The raw data is plotted as three histograms which have each been fitted with a normal distribution (dashed line). (b) The median standard deviation of the cycle-to-cycle and device-to-device conductance variability distribution of all $16,384$ devices is plotted as a function of the median of the medians of all device conductance variability distributions for a sweep of the SET programming current. The relationship can be approximated with a linear function (dashed line)

.

tively using an algorithm called expectation-maximisation until a mixture of components is found that best ‘explains’ the target parameter distribution [316]. As seen in the previous section, OxRAM devices are normal physical random variables in the high conductance state [315] (Fig. 3.18a) and can therefore act as the random variable components in such a probabilistic mixture.

However, although the median of each RRAM random variable component can be freely determined by the SET programming current (Fig. 3.18a), the standard deviation is intrinsically tied to this value (Fig. 3.18b). This requires that, instead of treating the standard deviation of each component as free parameters during the expectation-maximisation algorithm, its value must be assigned based on the a known relationship with the median (here the equation in Fig. 3.18b). This may require, for example, additional circuitry on a practical chip to perform an initial calibration step owing to the die-to-die variability that exists across chips on a wafer as well as between wafers [317].

We apply this technique to decompose the single target probability distribution plotted in green in Fig. 3.19 into K physical random variable components. These components, determined through expectation-maximisation, can then be used to program experimentally a column of N RRAM memory cells such that the distribution of conductance states in the column approximates that of the target distribution. This result is achieved by programming subsets of devices in the column with a SET programming current such that their conductance states are sampled from the Gaussian corresponding to each component. The number of devices programmed per component is equal to the nearest integer value resulting from the multiplication of the total number of available devices by its weighting factor. For this target distribution it was found that five normal components ($K = 5$) were required to well approximate the target distribution. This result was obtained by performing the expectation-maximisation algorithm over a sweep of K and observing at which value of K the resulting log-likelihood of the mixture saturated - as plotted in Fig. 3.20a. The five resulting RRAM random variable components are superimposed over the original target distribution in Fig. 3.19. These five components are then used experimentally to program a column of 1024 1T1R RRAM devices as described - the number of devices programmed per component is specified in the caption of Fig. 3.19. The resulting probability distribution, plotted as a histogram in Fig. 3.19, is seen to well approximate that of the original target distribution. This also suggests that the linear approximation of the relationship between conductance median and standard deviation, which has a non-negligible error at the conductance extremities, is not detrimental.

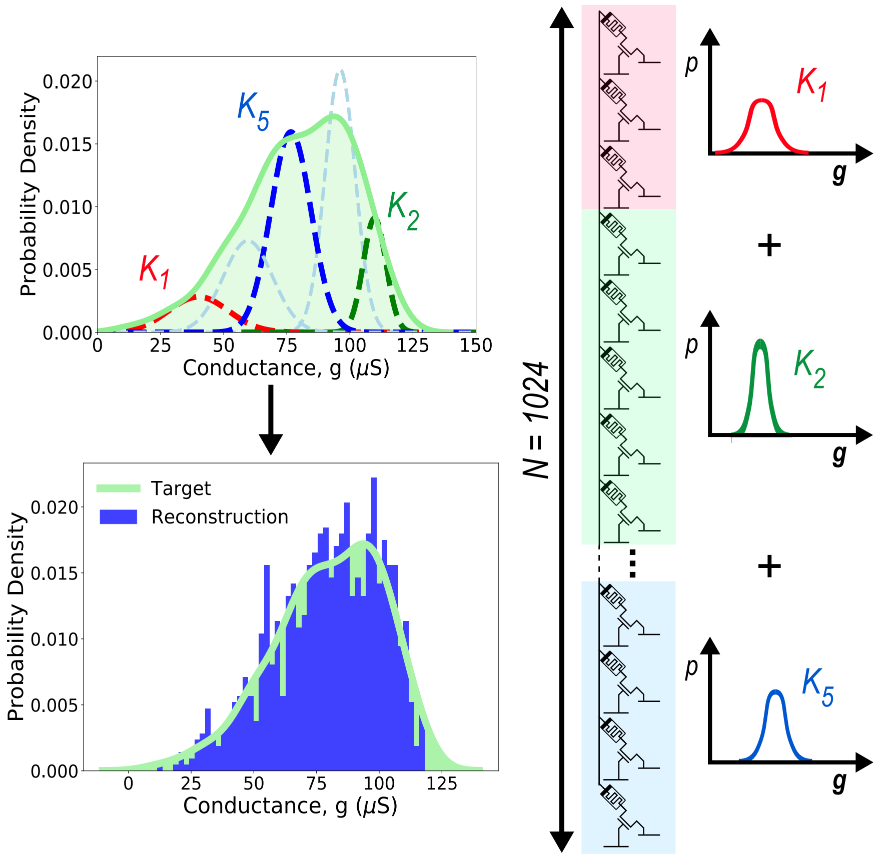


Figure 3.19: (upper left) A single target distribution (green) is plotted alongside five superimposed normal distributions (dashed lines), corresponding to the five RRAM random variable components determined through the adapted expectation-maximisation algorithm. The red, blue and green components use 85, 333 and 107 devices respectively and the leftmost and rightmost unlabelled components use 177 and 322 devices. (right) Diagram of a column of N resistive memory devices connected in a one-transistor-one-resistor configuration. Three groups of devices down the column are highlighted and correspond to the weighted number of devices that will be used to generate samples from the corresponding red, green and blue normal components in the upper left figure. (lower left) Experimentally transferred probability density histogram of device conductances (blue) to a column of $1,024$ devices, superimposed over the target distribution (green), obtained using the medians and weighting factors of the normal components shown in the upper left figure.

In order to quantify the closeness of the approximation transferred to the hardware we evaluate the Kullback-Leibler (KL) divergence from the transferred to the target distributions over a range of column sizes. The resulting mean KL divergence, over ten experimental transfers, is plotted in Fig. 3.20b for an increasing number of RRAM cells per column. The KL divergence reduces rapidly as the number of devices in the RRAM column is increased, consistent with the law of large numbers [318].

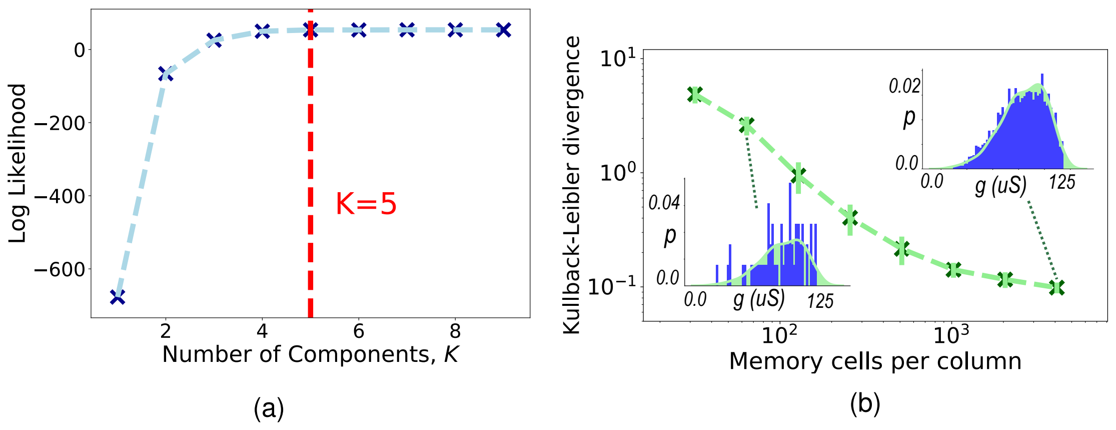


Figure 3.20: (a) Maximum value of log-likelihood obtained for an increasing number of components, K. For this target distribution, it was determined that five components were required to approximate the target distribution (dashed red line). (b) Kullback-Leibler divergence from the target to the transferred distributions calculated for an increasing number of RRAM memory cells per column. For each number of memory cells, the distribution was transferred ten times. The resulting variability in the KL divergence is shown using vertical green bars at each point indicating one standard deviation. An example of the transferred distributions for 32 and $4,096$ devices are plotted as an inset.

3.3.3 Ex-situ training of an RRAM-based Bayesian neural network

Before applying the presented technique to the transfer of a full Bayesian neural network model, we first describe how to perform the ex-situ training of an RRAM-based Bayesian neural network and how the parameters from this software model can be represented using an array of resistive memory devices [315]. In the Bayesian framework, training is typically performed with Markov Chain Monte Carlo (MCMC) sampling or by variational inference algorithms [319, 320, 321]. We employ the No-U-Turn sampler (NUTS) MCMC algorithm [322]. In contrast to gradient-based approaches, which result a deterministic locally-optimal model, NUTS MCMC results in a collection of sampled models, each with their own parameters (synaptic weights and biases). The distribution of each learned parameter, in other words the distribution of parameters over all of the sampled models, can then be transferred to a distribution of device conductances in a column of RRAM cells. In contrast to deterministic models, which generally require a single device per parameter, the use of a distribution comprising multiple devices per parameter allows uncertainty to be incorporated into its estimation. The ability to represent uncertainty in this manner is advantageous for Bayesian models, permitting them to account for factors such as sensory noise, small training dataset size as well as representing uncertainty in their output predictions. [289, 175].

One neuron in an RRAM-based Bayesian neural network can be realised as depicted in Fig. 3.21b. The neuron receives input synapses from M (here three) neurons in the previous network layer (Fig. 3.21a) - each connecting to one of its three columns. The distribution of the three input synaptic parameters are each stored in a column of size N and therefore necessitates an $N \times M$ array of 1T1R structures per neuron.

By applying a voltage vector **V** across these M columns, corresponding to the activations of the neurons in the previous network layer or the input data for neurons in the first layer, a current equal to $\mathbf{V} \cdot \mathbf{g}_\mathrm{n}$ flows out of each array row and into a neuron circuit. Since the conductance values of $\mathbf{g}_\mathrm{n}$ are on the order of microsiemens, the neuron circuit must first multiply this current value by a scaling factor S and then apply an activation function $h()$ to the scaled quantity, resulting in a neuron output voltage $z_n = h(S(\mathbf{V} \cdot \mathbf{g}_\mathrm{n}))$. This voltage can then in turn be applied to a column of each of the neuron arrays in the next layer. The distribution of these N neuron activation voltages, **z**, constitutes the output distribution of the neuron. We use the hyperbolic tangent activation function for all neurons besides those in the output layer where the softmax function is used. Use of the softmax function at the output allows the likelihood of the model to be evaluated as a categorical random variable during training such that it can be applied to multi-class datasets.

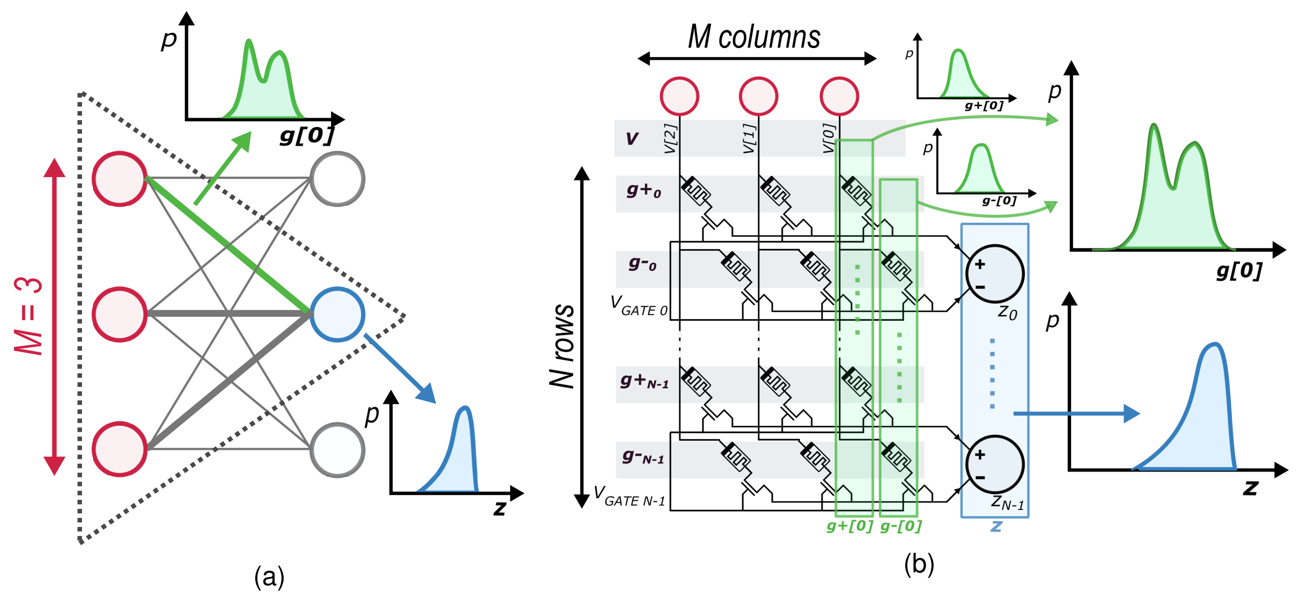


Figure 3.21: (a) A feed-forward neural network of three neurons ($M = 3$) in the first layer and three neurons in the second layer. For the case of a Bayesian neural network the synapses (for example that highlighted in green) and neurons (for example that highlighted in blue) are described by probability distributions. The RRAM-based realisation of the Bayesian neuron and synapses enclosed in the dashed grey triangle is shown in part (b). (b) A proposed structure for realising a Bayesian neuron (and the synapses fanning into it) based on an $N \times M$ array of resistive memory. The distribution of conductance states of the devices in a column corresponds to the distribution of a synaptic parameter (for example that highlighted in green). Each row of the array uses devices that code for positive (g_+) and negative (g_-) values that enables each parameter to be positive or negative. The inputs to the columns are the output voltages generated by the M neurons in the previous network layer. As a result of these input voltages, two currents will flow out of each row and into a neuron circuit which subtracts them and then evaluates an activation function. This activation produces an output voltage as a function of this current that can then be applied to the column of neuron arrays in a subsequent layer. The distribution of the N output voltages (blue probability distribution) is the output distribution of the neuron.

In practice, as each parameter distribution can assume positive and negative values, each model parameter should be described by the difference between positive and negative distributions: $p(g) = p(g_+) - p(g_-)$ (Fig. 3.21b). Therefore, during MCMC sampling, the parameters which are sampled are $p(g_+)$ and $p(g_-)$ and not $p(g)$ directly. In addition, each neuron of the Bayesian neural network requires a bias distribution $p(g_b)$, that can be realised with an extra column of devices, identical to the others, to which a constant voltage V_b is applied.

One further technological constraint must be taken into account. Each RRAM device has a limited conductance range; in the technology applied here, extending approximately from 20μS to 120μS (Fig. 3.18b). As a result, the sampled distributions of each parameter, $p(g_+)$ and $p(g_-)$, of the Bayesian neural network must be bounded within these limits during the ex-situ training. Fortunately, in the Bayesian framework, such a bounding can be achieved naturally by placing an appropriate prior distribution over each parameter. To account for this we therefore place a normal prior over each parameter, with a median of 80μS, and a standard deviation of 20μS, such that the sampled distributions exist within the limited conductance range.

3.3.4 Transfer to resistive memory based inference hardware

We now combine the ideas of the two previous sections and present an approach to achieve the transfer of an ex-situ trained Bayesian neural network onto the RRAM-based hardware depicted in Fig. 3.22.

The No-U-Turn Sampler (NUTS) Markov Chain Monte Carlo (MCMC) algorithm is used to explore the posterior distribution of the Bayesian model by generating proposed models through random permutations to the current model parameters – i.e. exploration through construction of a Markov Chain [322]. Models are accepted or rejected based on a ratio of their likelihoods and priors, and all of the accepted models are stored in memory. After NUTS MCMC has collected a pre-defined number of samples, *S*, the algorithm terminates. All of these stored samples constitute an approximation of the posterior distribution of the model. Considering a Bayesian neural network model, with *Q* parameters (synapses and biases) and where NUTS MCMC has been used to record *S* samples, this posterior distribution is described in an $S \times Q$ matrix (upper left in Fig.3.22) where each matrix cell is, typically, a floating-point precision number. Each of the *S* rows of the matrix is an accepted model and each of the *Q* columns of the matrix store all of values of a single parameter over all of these models. Each of the Q columns therefore corresponds to the distribution of each synaptic parameter in a Bayesian neural network. The objective is to transfer the posterior approximation in the $S \times Q$ matrix into multiple $N \times M_i$ resistive memory arrays. This requires three matrix pre-processing steps before the eventual single step transfer (see Fig. 3.22 for a depiction of the steps).

- **Quantise** – The expectation-maximisation is applied independently to each of the *Q* columns of the $S \times Q$ samples matrix. This allows *K* component medians to be extracted per synaptic distribution – where the number of *K* can be different for each parameter. For each matrix column the values in the matrix cells are then quantised by setting them to a new value equal to the nearest component median value.

- **Sub-sample** – This quantised $S \times Q$ samples matrix will likely have many more rows than the number of rows available on the RRAM-based inference hardware (i.e. $S \gg N$). This is because, to allow NUTS MCMC to adequately explore the posterior distribution of the model, it is useful to collect a large number of samples. Therefore, to reduce it to the same dimensions as the RRAM-based hardware, the matrix must be sub-sampled to an $N \times Q$ matrix – ideally by uniform-random removal of rows.

- **Slice** - Because the hardware is not composed of one large $N \times Q$ array of RRAM, but lots of smaller $N \times M_i$ arrays (one array per neuron) the $N \times Q$ matrix must be sliced, column-wise. This groups together all of the pre-synaptic distributions for each neuron into a single array. In this fashion, each cell of each processed $N \times M_i$ software matrix has a one-to-one correspondence with a device on an RRAM array.

After these processing steps, it is now possible to make a single step transfer by programming each physical device on the hardware with a SET programming current which samples a conductance value from a Gaussian distribution with a median equal to the value in the corresponding software matrix cell. It is important to preserve the order of parameters by transferring in a row-wise fashion: since the sampled parameters obtained through NUTS MCMC will often co-vary with one another (i.e. certain parameters will be correlated per sample). For instance, if we were to mix-up parameters between two of the recorded models from the posterior approximation, the resulting model would not necessarily lie on the posterior distribution of the Bayesian model. Note that the weighting factor resulting from the expectation-maximisation algorithm is not used to determine the number of devices programmed in line with each component as in Fig.3.19, but is instead incorporated implicitly by the samples that have been accepted by NUTS MCMC.

After the model has been transferred, the hardware can then perform inference on previously unseen data points whose features are presented as voltages to the columns of the neuron arrays in the first hidden-layer. These voltages drive the forward propagation of neuron voltage activation distributions through the subsequent network layers, resulting finally in an activation distribution per output neuron. These output distributions can then be used to make a prediction regarding as to what output neuron class the input data point belongs. Additionally, the standard deviation of each prediction distribution can be calculated and used to quantify the uncertainty in the prediction of each output neuron.

Since the pre-synaptic distributions of each of the neurons in a Bayesian neural network are stored in multiple rows (i.e. multiple samples), but the physical connections between each neuron consist only of single metal wires, inference with a Bayesian neural network must be performed one row (sample) at a time. At the output layer then, a separate memory structure is required to temporarily store each of the output neuron activations that result from each of the transferred samples - an SRAM memory for instance. In this fashion, after all of the rows have been read in an inference, the prediction distribution of each output neuron is readily available. To achieve this we propose, in Fig. 3.22, that each neuron array contain only a single neuron circuit that is multiplexed between each of the N rows sequentially. By applying voltage pulses to the gates of the devices in only one row, while grounding the others, this multiplexing can be achieved cheaply and without a dedicated multiplexing circuit. Additionally, the use of a shared neuron circuit also reduces the required circuit overhead to implement the activation function or to perform any required analogue-to-digital conversions by a factor of N. For example, by applying the red pulses in Fig. 3.22 to row $N = 0$ of all of the neuron arrays simultaneously at time t_0, each neuron in the output layer will produce a voltage activation, z_0. In other words, these output activations result from

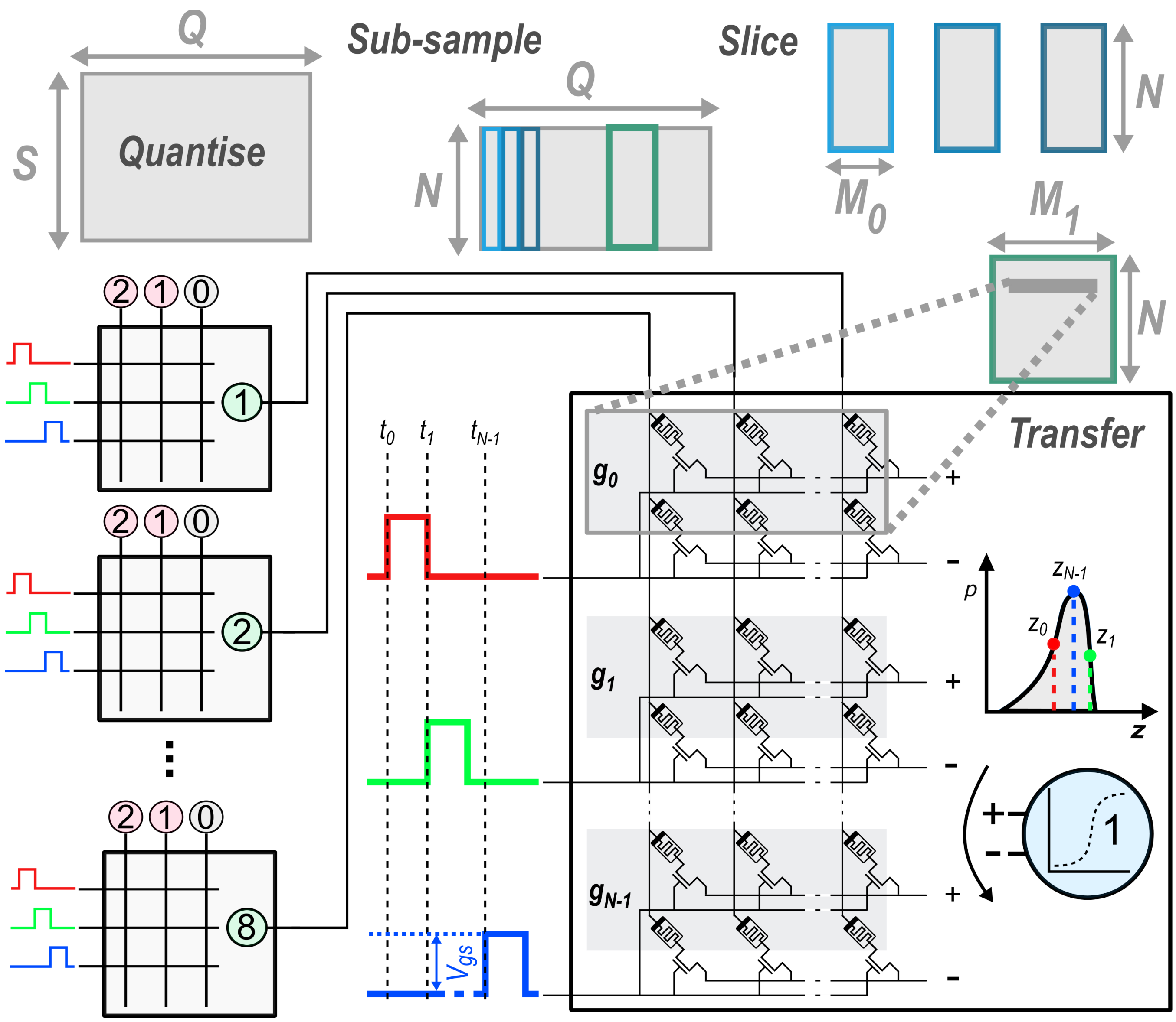


Figure 3.22: Proposed hardware realisation of feed-forward layers in the RRAM-based Bayesian neural network shown in part Fig 3.23. The output of three hidden-layer neuron arrays, corresponding to neurons one, two and eight in part Fig. 3.23, are connected to the inputs of three columns of RRAM of another neuron array, neuron one in the output layer in Fig. 3.23. As a function of the input data feature voltages (from the red coloured neurons in the first layer), the hidden layer neurons will produce activation voltages that are in turn applied over the columns of the output layer neurons causing the output layer neurons to produce activation voltages. This forward propagation of voltage continues for an arbitrary number of network layers until reaching the output layer. By sequentially applying gate voltage pulses to each row of all the arrays - the red pulses at t_0, the green pulses at t_1 and finally the blue pulses at t_{N-1} - output neuron one will sequentially produce voltage activations z_0, z_1 and z_{N-1}. The distribution of all activations, $p(z)$, gives rise to an output distribution of neuron one.

the forward propagation of the input through the devices in row $N = 0$ of each array only. Thereafter, applying the green pulses at t_1 the outputs z_1 will result from the forward propagation through the devices at rows $N = 1$ and so on. By proceeding in this fashion up until row z_{N-1} an output distribution $p(z)$ will be available for each output neuron at t_{N-1}. The mean value and standard deviation of this distribution can be used to respectively make a prediction and quantify prediction uncertainty.

To demonstrate this technique, we perform the ex-situ training of the Bayesian neural network depicted in Fig. 3.23. We apply it to an illustrative example in the generative moons classification task [323]: each of the two output neurons of the network must learn a non-linear decision boundary that separates its respective class of noisy data points from the other. To evaluate the transfer of the Bayesian neural network model, we perform a hybrid hardware/software experiment. After termination of NUTS MCMC, the normal random variable components required for all of the model parameters are identified using expectation-maximisation. Then, $1,024$ devices in the experimental RRAM array are programmed, using the corresponding SET programming currents based on the median value of each of these random variable components. The resulting conductance values are then used to build up a computer model of the the proposed hardware depicted in Fig. 3.22 in order to perform an inference. Each RRAM cell of this computer model is randomly assigned one of the $1,024$ transferred conductances which resulted from the SET programming current that would have been used to program the equivalent device on the physical array. Examples of the resulting distributions transferred to the synaptic parameter highlighted in green in Fig. 3.23 are plotted for $1,024$, 128 and 16 rows. On average, based on the measured SET programming currents, the programming energy required to perform the transfer of the full Bayesian neural network model to the array was 1.37μJ, 172nJ and 21.5nJ for the models based on $1,024$, 128 and 16 rows respectively.

Upon performing inference with the hybrid hardware/software model, the decision boundaries for each of the two output neurons for the model transferred to the $1,024$, 128 and 16 row arrays, shown in Fig. 3.23, arise. The output neurons appear, in all situations, capable of discerning the underlying structural separation between the two types of data point that was learned in the software model. The probability contours of the two output neurons are largely similar for the case of $1,024$ and 128 rows, while those for 16 rows appear more erratic. Despite this appearance however, the boundaries drawn at the interface of the two moons with N=16 rows still capture the fundamental curvature of their division. Based on the read currents of the programmed devices, the energy required to read all of the device conductances during inference was 110nJ, 13.7nJ and 1.72pJ for the models transferred to the $1,024$, 128 and 16 row arrays respectively. However, that is important to note that, based on the design of the in-situ RRAM-based MCMC system, the energy required by read circuitry, analogue-to-digital and digital-to-analogue conversions as well as circuits for implementing the neuron activation functions have not been considered and would lead to a considerable increase in these values depending on design choices.

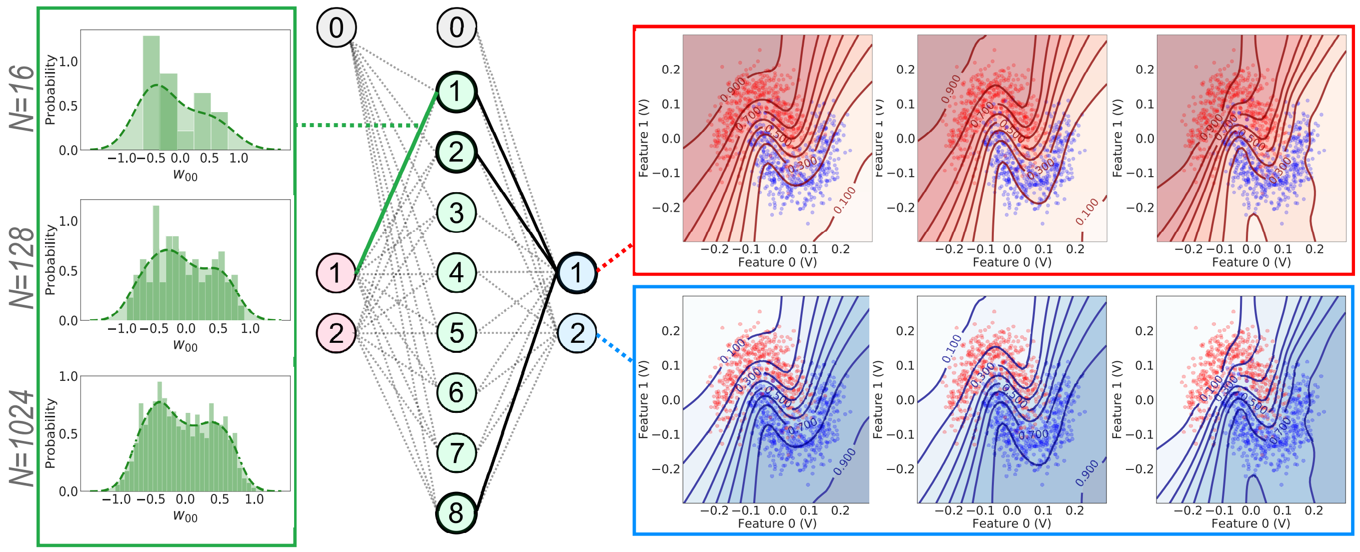


Figure 3.23: (centre) A single hidden layer feed-forward Bayesian neural network. Circles and lines in bold correspond to the neuron arrays and connections shown in part Fig. 3.22. (left) The probability density histograms and kernel density estimates for a synaptic parameter (green) using 16, 128 and $1,024$ memory cells per column. (right) The predictive probability contours of neuron one (recognising points from the red moon) and neuron two (blue moon) for 16, 128 and $1,024$ memory cells per column. Each of the red and blue moons data points are described by two feature voltages that are applied as inputs to the columns of the green neuron arrays.

The prediction uncertainty of each of these transferred Bayesian models is plotted in Fig. 3.24. This uncertainty, captured in the distribution of each synaptic parameter, naturally propagates through a Bayesian neural network to the output layer where, as might be expected, it is seen to be greatest at the interface between the red and blue points. While the prediction uncertainty contours are largely similar for $N = 1024$ and $N = 128$, they are once again degraded for $N = 16$. In safety-critical edge inference applications the ability of a Bayesian neural network to quantity uncertainty, with respect to deterministic models, is potentially invaluable and, perhaps, indispensable from an ethical perspective [324]. For example, in a medical system, such as the implantable cardioverter-defibrillator [172] like that considered in the in-situ setting, these prediction uncertainties can be leveraged to avoid the erroneous application of an electric shock to the heart which can, in some instances, prove fatal [178]. If the system were presented with a data-point close to a noisy decision boundary (as in Fig. 3.24) or with a data-point from a location in the feature space that the model had not observed during training, perhaps due to a damaged or drifting sensor, the prediction uncertainty of the model will be large. By placing a threshold on a tolerated level of prediction uncertainty, above which the system should not take action, the model is able to express that it dos not known with certainty what action should be taken and erroneous, potentially dangerous, interventions can be avoided.

3.3.5 Allowing a model to say ‘I don’t know’

We now present a final example that demonstrates how ex-situ Bayesian model transfer can be performed in the closed-loop (program and verify) fashion [264, 307]. The ex-situ trained model is applied

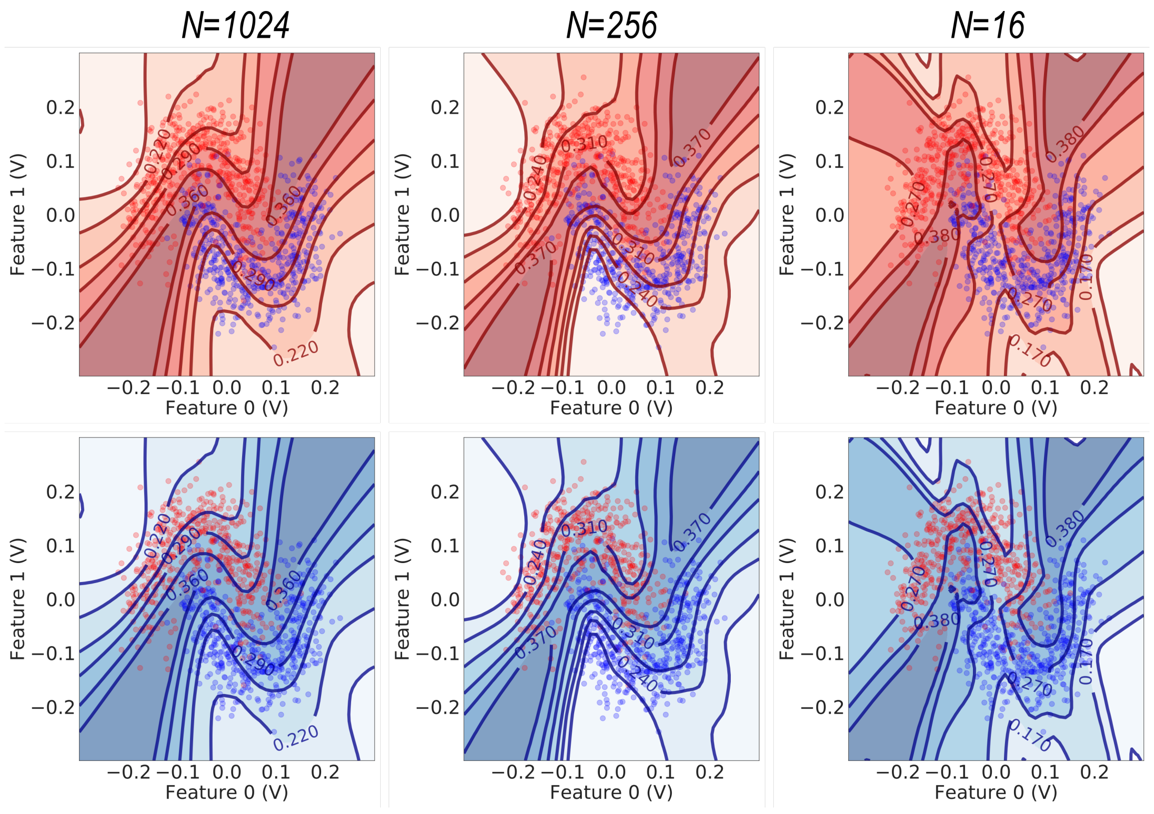
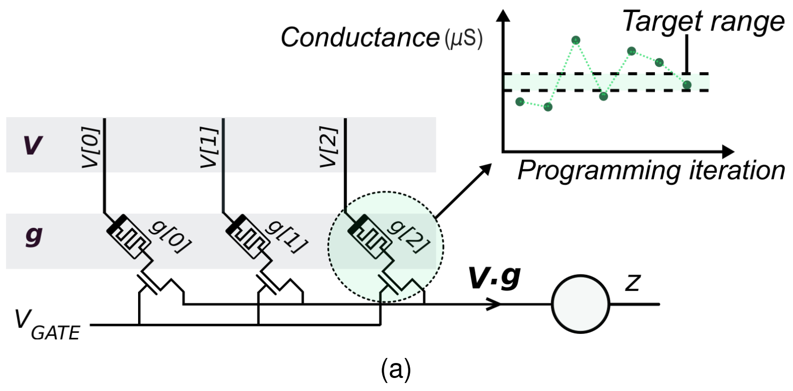


Figure 3.24: The uncertainty contours for the models transferred to the 1024, 128 and 16 row RRAM-based Bayesian neural network hardware. This uncertainty is determined by calculating the standard deviation of each output neuron's prediction distribution. The uncertainty in all cases is seen to be greatest at the noisy interface between the moons.

to the heart arrhythmia datatset [296] which, crucially, also allows for a practical demonstration of how Bayesian modelling uncertainty can be applied in a safety-critical application. Specifically, a technique allowing for a model to say 'I don't know', such that it doesn't not carry out a potentially harmful action, is presented. The model is trained to classifying four different types of potentially heart arrhythmia as well as healthy heartbeats. Considering a futuristic use case whereby an implantable cardioverter defibrillator is equipped with an ex-situ trained RRAM-based Bayesian neural network model, the model would be responsible for application of an electric shock to the heart upon detection of certain types of arrhythmia. As described however, the application of a shock to a normally functioning heart can prove fatal [178]. The model therefore must apply a shock only when sufficiently certain in order to avoid performing potentially dangerous actions, while also taking actions frequently enough to intervene in the event of sustained arrhythmic beats.

A Bayesian neural network model, with two hidden layers of eight neurons each, is trained ex-situ by NUTS MCMC [322] using sixty four input features resulting from the fast Fourier transform of each heartbeat (see Appendix 6.6). The training set is composed of two hundred examples of each of the five classes of beat. Once trained it is then transferred in a closed-loop fashion to the hybrid hardware-software experimental 16k device array. Here, instead of exploiting the devices as random variables as presented in the rest of this section, each device is iteratively programmed until its conductance falls within a pre-defined conductance error range - as depicted in Fig.3.25a. Therefore the high precision weights of the model in the cloud are discretised into a much smaller number of levels. In Fig. 3.25b it is shown how each resistive memory in the 16k device array can be programmed into eight separable conductance levels while being in the HCS, in addition to one further level corresponding to the LCS, using a basic RESET/SET closed-loop programming technique. In this scheme; a specific SET programming current is defined for each of the eight HCS levels and a device is iteratively programmed until its conductance falls within a pre-determined error margin (Fig. 3.25a).

Using the software model of the inference architecture presented in Fig.3.22, each of the simulated devices is assigned a conductance value from the desired level in the device data presented in Fig. 3.25b. Considering that subtraction of two devices encodes the positive or negative weight of each parameter, this allows for seventeen distinct conductance combinations. The combination of conductances levels per pair of devices is chosen such that, on average, the resulting differential conductance should be as close as possible to the original high-precision value of the software model.



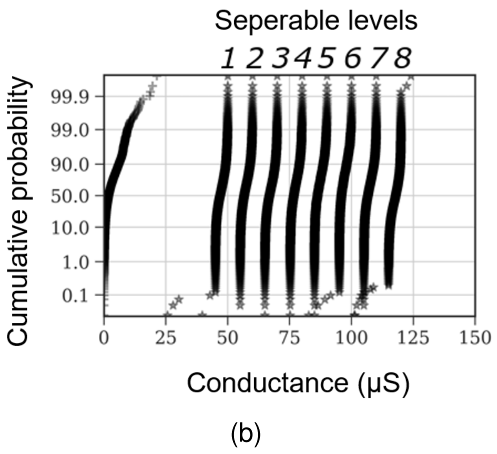


Figure 3.25: (a) Scheme indicating how a row of three resistive memory devices realise three synaptic connections between input activations **V** and a post-synaptic neuron. By applying these activations over the columns of the row of resistive memory the current that flows out of the rows will be equal to the dot product **V** $\cdot$ *g*, where **g** is the conductance vector of the row. In the inset of device an arrow indicates the closed-loop programming routine that is used in order to program the device conductance state, $g[2]$, whereby the memory is iteratively programmed until its conductance falls within a pre-determined range of error. (b) The multilevel conductance distributions obtained with the 16k device array using a RESET/SET closed-loop programming technique. Each cumulative distribution is composed of a series of points which are ordered, per separable level, from lowest to highest conductance.

Using the transferred model, the uncertainty of all of predictions is calculated over the training set. The uncertainty of each prediction is plotted in Fig. 3.26a whereby correct predictions are shown in green whilst incorrect ones are shown in red. The lowest uncertainty amongst the misclassified points per class, or the highest uncertainty of all points if there were no errors as was the for class one in Fig. 3.26a, was then multiplied by a discounting factor lower than one (here equal to 0.75) to give, for each class, an uncertainty threshold. Predictions made on the test set, with an uncertainty greater than this uncertainty threshold, would then result in no action being taken by the model in this application setting. In applying these uncertainty thresholds, the RRAM-based Bayesian neural network was still sufficiently certain to take actions (applying a shock or doing nothing) over 71% of the test data. Remarkably, even at this relatively high rate of action, the model makes no errors. In other words, in this realistic safety-critical setting, the model took zero potentially dangerous action by recognising when it was uncertain and therefore was able to say ‘I don’t know’.

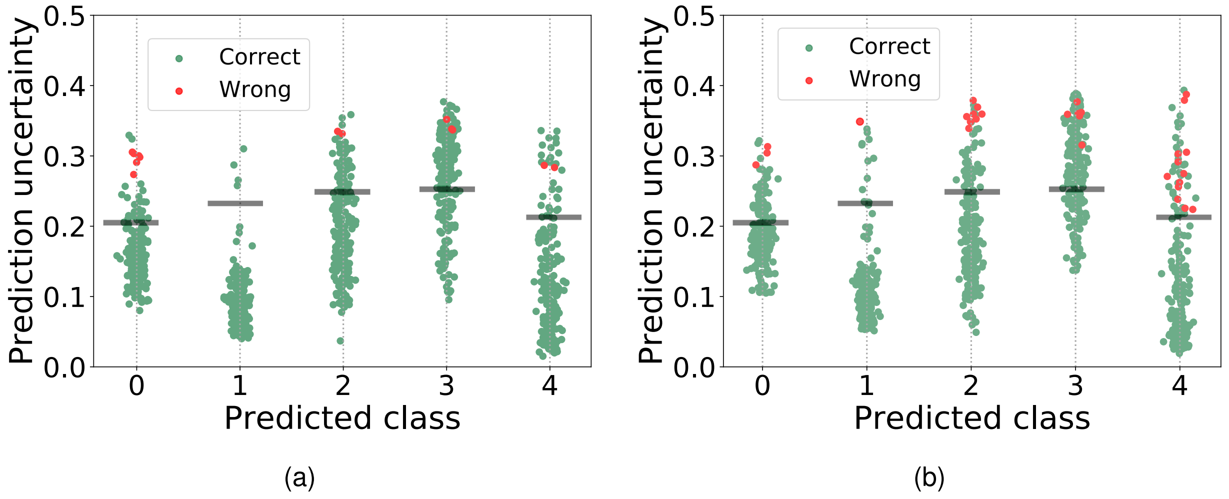


Figure 3.26: (a) For each predicted class of heartbeat of the model over the training split of the heart arrhythmia dataset of the dataset, the prediction uncertainty (the standard deviation of the prediction distribution) is plotted. Correct predictions are plotted as green points, and wrong ones as red points. Horizontal black lines show the uncertainty thresholds, calculated using the training set, that can be used as a means of determining whether or not the model takes an action as a function of the prediction uncertainty over the test set. These thresholds correspond to the lowest uncertainty of an incorrect prediction in the training set multiplied by a discounting factor less than one. (b) For each predicted class of heartbeat of the model over the test split of the heart arrhythmia dataset of the dataset, the prediction uncertainty (the standard deviation of the prediction distribution) is plotted. The uncertainty thresholds calculated on the test set, the same ones drawn in part (a), are plotted for each class. Without the use of these uncertainty thresholds, what would have corresponded to correct and incorrect predictions are plotted respectively as green and red points. Each uncertainty threshold rests below all of the red points of each predicted class meaning that, by leveraging model uncertainty, no predictions were made on the test set.

3.4 Chapter discussion

In this chapter it has been demonstrated that, contrary to the case of gradient-based algorithms and deterministic modelling techniques, Bayesian modelling and resistive memory devices result in a harmonious pairing of algorithm and hardware.

In the first section the potential of simultaneously harnessing resistive memories as physical random variables and as high efficiency dot-product engines in the realisation of in-situ Markov Chain Monte Carlo sampling algorithms was demonstrated. This in-situ learning method, which actively exploits cycle-to-cycle conductance variability, was shown to also be resilient to device-to-device variability and extensive device aging - orders of magnitude beyond the point where devices could be used as traditional memories. The appeal of RRAM-based MCMC was seen in two key sets of results. First, the dramatic reduction in the total number of programming operations required to train a full model, with respect to RRAM-based backpropagation approaches, underlined the remarkable compatibility of resistive memory and Markov Chain Monte Carlo sampling. Additionally, the orders of magnitude reduction in the energy of a full RRAM-based MCMC system relative to conventional CMOS solutions, compounded the potential of the approach based on the application of brief voltage pulses, in-memory, to nanoscale devices. In fact, the energy totals which were on the order of micro Joules, correspond to the energy estimated to be consumed by a biological neuron spiking tens of times or less [325]. This is particularly eye-catching given the current intense focus within several research communities of developing spike-based brain-inspired learning algorithms in order to reduce the energy requirements of machine learning systems. Secondly, when faced with data representative of the edge, namely the noisy and multi-modal electrocardiograms in the arrhythmia detection task, the robustness of Bayesian machine learning to over-fitting by incorporating uncertainty into learned parameters [175, 289], was seen to be crucial in outperforming the software-based deterministic benchmark neural networks trained through backpropagation. This second result puts in evidence that, beyond simply being compatible with the fundamental properties of resistive memory devices, RRAM-based Bayesian models exhibit a particular suitability for the edge setting.

The Metropolis-Hastings MCMC sampling algorithm presented in the first section is known to lose efficiency when the dimensionality of the model it is applied to train becomes very high. While this could pose a potential issue going forward as we look to scale this approach to higher complexity tasks, a potential solution was stumbled upon in section 2.2. One of the key results from the neural network model of the cricket cercal system was that such bio-inspired neural network models could allow for a reduction in the number of model parameters by some orders of magnitude. The exploration and discovery of bio-inspired architectures, to which RRAM-based MCMC can be applied to whilst remaining at its most effective, may therefore be key to scaling the approach to higher complexity tasks. Furthermore, Hamiltonian Monte Carlo algorithms [326, 327], such as the No-U-Turn sampler used in section 3.3, are able better address higher dimensional problems and therefore such MCMC sampling algorithms can be incorporated into evolution of the presented Metropolis-Hastings RRAM-based MCMC system presented here. To apply MCMC sampling to larger datasets than those considered in this section, where the Metropolis-Hastings algorithms can also be limited, mini-batch MCMC algorithms should also be investigated [328, 329] that allow for samples to be accepted based on subsets of the data.

Ultimately section 3.2 demonstrated that, by embracing what have been previously considered as non-ideal device properties, resistive memory based Markov Chain Monte Carlo sampling hardware can sit at the core of a new generation of intelligent edge learning systems that promise to open a door to new, extremely resource constrained, applications. For example, intelligent and adaptive implantable medical devices which is a currently revolutionary application domain within machine learning that is currently of out reach through existing commercial approaches [2].

The second section of the chapter approaches the learning problem from the ex-situ perspective. Because Bayesian models require many samples per parameter, existing ex-situ transfer approaches could entail a prohibitive energy cost and latency to transfer a full Bayesian neural network. Therefore the expectation-maximisation based transfer approach was introduced which permitted an ex-situ trained Bayesian model to be transferred in a single step - i.e, each device was required to be programmed only once. The importance of describing uncertainty in the parameters of the Bayesian neural network was also discussed in terms of how the resulting output prediction uncertainty could be leveraged in a safety-critical application. We studied the use case of the application of an electric shock upon the detection of dangerous heart arrhythmias. It was demonstrated that by putting a threshold on the output uncertainty (here called uncertainty thresholding), it was possible to make a Bayesian neural network model say ‘I don’t know’ and take zero potentially dangerous actions. In a variety of emerging, or yet to be discovered, edge computing applications, where systems will interact with humans and whose actions may compromise their safety, it is difficult to imagine using a modelling approach that does not quantify prediction uncertainty. Alongside Bayesian models, other techniques based on deterministic neural networks such as Monte Carlo dropout [330] and deep ensembles [331] have also been recently found to offer good uncertainty estimates. It will therefore be important to compare the uncertainties available through these approaches with those naturally incorporated into Bayesian models.

Going forward from this initial proposal and experimental demonstration, future work should aim to understand how the expectation-maximisation transfer technique can scale to larger network models and to higher-complexity datasets as well as looking for adaptations to the algorithm that can more naturally incorporate covariance in the posterior - ideally obviating the ‘row-wise’ transfer protocol used here. In doing so it will be also be useful to explore the application of training algorithms like variational inference [319, 320, 321] which, through use of ‘mean-field’ variational approximations, can be used to eliminate parameter covariance altogether and allows each synapse to be treated independently. Alternatively, it has also been found that as Bayesian neural networks become deeper, parameter covariance largely disappears altogether [332]. A solution therefore, could simply be to perform ex-situ training on models with a greater number of layers.

Finally, it will also be instructive to perform a quantitative comparison between ex-situ trained RRAM-based Bayesian and deterministic neural network models to understand the advantages and trade-offs between the two approaches in terms of inference accuracy, the energy and latency incurred in model transfer and inference as well as memory requirements.

Chapter 4

A non-von Neumann neuromorphic computing fabric

4.1 Chapter introduction

The first two chapters have addressed the potential of bio-inspired neural network architectures and a method for training such models, in the ex-situ or in-situ setting, if its parameters were implemented using resistive memory devices. This final chapter completes the picture by proposing how resistive memory based bio-inspired models can be realised in a full non-von Neumann computing system. The two sections of this chapter show respectively how RRAM can be incorporated into neuromorphic circuit models and into a neuromorphic architecture that can interconnects these models in a configurable architecture. Crucially, in both cases, the non-volatility of RRAM is used a means of distributing memory continuously through the computing fabric and, as observed in biological nervous systems, co-localising the configurable memory elements with the computational units in a fashion that makes them almost indistinguishable.

4.2 Hybrid neuromorphic circuits

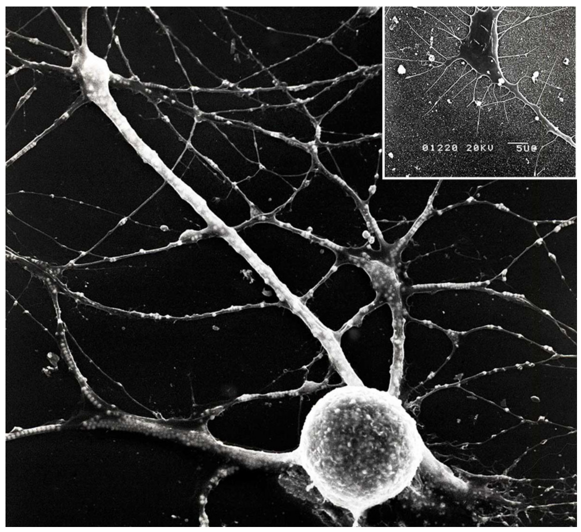
Proteins exist within the membranes of biological neurons (such as that pictured in Fig. 4.1(a)) which transport ionic charge into and out of the cell [333]. Under excitatory stimulation positive charge accumulates inside whilst leaking out at a constant rate. In the case of spiking neurons, if the rate of charge accumulation sufficiently exceeds the leak, a chain reaction triggers the generation of an action potential (spike) and a wave of ionic charge propagates down the neuron's axon. As the axon branches, this spike proceeds and divides along parallel bifurcating channels until it reaches synapses composed of pre-synaptic processes and post-synaptic sites belonging to the afferent neuron. At these synapses, electrical excitation elicits release of neurotransmitter that diffuses across a nanoscale cleft from the pre- to the post-synaptic site and subsequently modulates ion uptake at the destination cell. Properties of neurons (like those depicted in Fig.4.1(b)), such as the time constant of the input charge integration, the synaptic weight that governs how much charge is injected upon the arrival of neurotransmitter and the length of the refractory period during which the neuron cannot spike are determined by the conductance of these membrane proteins. Neurons continuously adapt these conductances inline with different neural plasticity mechanisms for the purposes of maintaining homeostasis [334, 335] and to form memories and learn [336, 337].

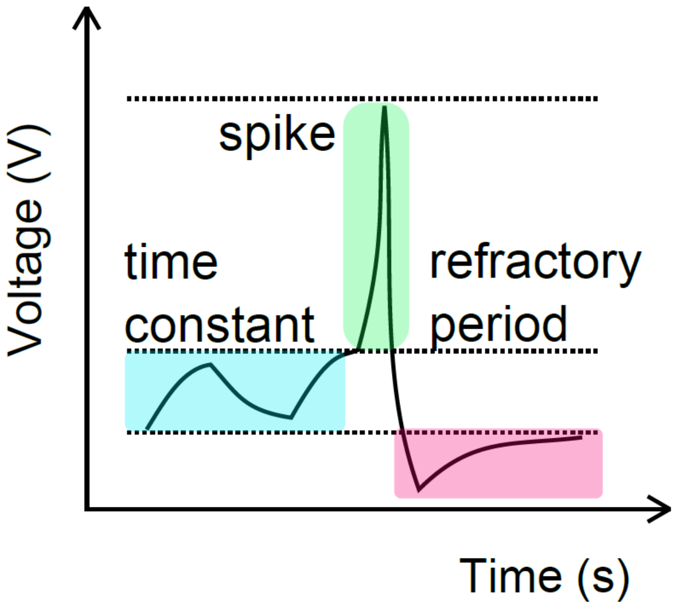
Neuromorphic synapse and neuron models, which aim to mimic the dynamic behaviours of the biological cells using electronic circuit components [63, 102], use the drain-source conductance of a transistor under a ‘current-mode’ gate bias, mimicking neuronal ion channels, as a means of implementing these conductance parameters [84, 86, 85, 103, 338]. Essentially, these biased transistors act as the conductances (resistances) used in Hodgkin-Huxley style neuron models [28], which regulate at what rate charge flows onto and off of capacitors.

In the class of non-von Neumann computer chips referred to as neuromorphic processors, these biases are programmed and generated in a centralised unit on the chip [85, 86, 339, 340]. These bias generators generally store parameter values using a volatile SRAM memory. Such an approach, however, has many drawbacks. To limit the size, and therefore the static power consumption of the volatile memory of the current source generator, as well as the metallic area required to physically deliver biases to individual neurons; groups of thousands of neurons and synapses in neuromorphic processors are generally obliged to share the same set of parameters. In addition, this volatile memory is typically programmed once, via an external computer, either when the neuromorphic processor is power cycled or re-purposed for a new application. However, for networks more closely emulating the computational principles of biological systems where neuron and synapse parameters are updated continuously, such as the *Drosophila* elementary motion detection model presented in section 2.3, the von Neumann bottleneck that persists between processing and memory in neuromorphic processors becomes a limiting factor. At this point, the separation of memory and the computational elements becomes critically expensive in the implementation of parameter adaptation algorithms due to the large volume of data which is required to be ferried between the neurons and this central memory centre. It should be noted that in some neuromorphic processors that support ‘on-line’ plasticity this is not necessarily the case for the parameters that encode the synaptic weight. In these cases, whereby synaptic weights are modified inline with spike-timing dependent rules, the synaptic weight parameters are stored and updated locally at each synaptic array using an additional volatile memory. Solutions exploiting certain properties of resistive memory devices, such as the volatile resistance state of metal-insulator-transition Mott devices [341] and the gradual crystallisation of phase change memories [342] also been demonstrated. These solutions use the physics of the device to store the information locally, without a static power draw, and therefore allow parameter memory to be distributed throughout a computing substrate instead of centralised in a bias generator. However, the properties of these RRAM-based neurons based on single devices are fixed and cannot be configured.

4.2.1 CMOS-RRAM analogue circuit models

In this section, hybrid CMOS-RRAM electronic circuits models which model biological neurons and synapses like those used in section 2.3 are proposed, designed and fabricated. These analogue circuits incorporate resistive memory elements in order to store parameters that define the behaviour of these models in a non-volatile fashion whilst also playing an role in the physical operation of the circuit. Crucially, this



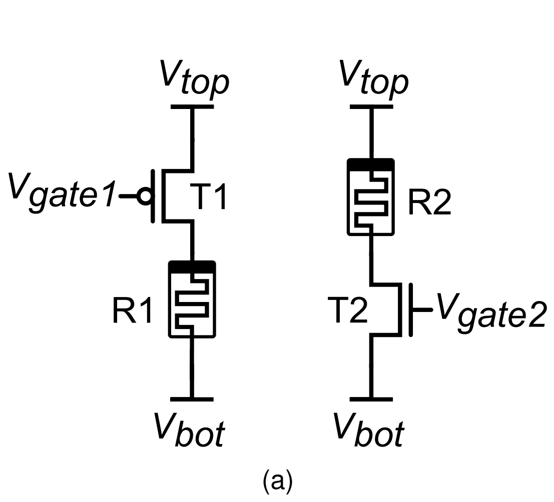


(b)

Figure 4.1: (a) SEM image of a neuron. The spherical structure is the soma where ions are integrated. Further linear structures are the dendrites and axon [343]. (b) The neuron time constant (blue) is defined by the rate of change of input voltage upon a stimulus. The neuron spikes (green) above a threshold followed by a refractory period (red) during which input signals cannot be integrated.

allows configurable memory to be distributed in space throughout a neuromorphic computing system in a truly non-von Neumann fashion - corresponding more closely with the underlying organisational principles observed in animal nervous systems.

The basis of these hybrid neuromorphic circuits is the 1T1R structure [344, 290] depicted in Fig. 4.2a. A resistive memory (R1 or R2) is connected in series with either a PMOS or NMOS selector transistor (T1 or T2). The transistor has two roles; (1) to determine the share of total programming voltage V_{top}-V_{bot} (V_{prog}) that is seen over the resistive memory and (2) to limit the current flowing through the device during a programming operation. Both objectives are achieved by modulating V_{gate} when a non-zero V_{prog} exists. As seen in the previous chapter there are two standard RRAM programming operations called SET and RESET which render the device in a low (LRS) or high (HRS) resistive state. In the case of oxide-based RRAM (OxRAM) the resistance of a thin layer (tens of nanometres) of a transition metal oxide (TMO) material is sandwiched between two metal electrodes and can have its resistance modified through application of electrical pulses which redistributes the conductive oxygen vacancies in the device oxide (Fig. 4.2b).



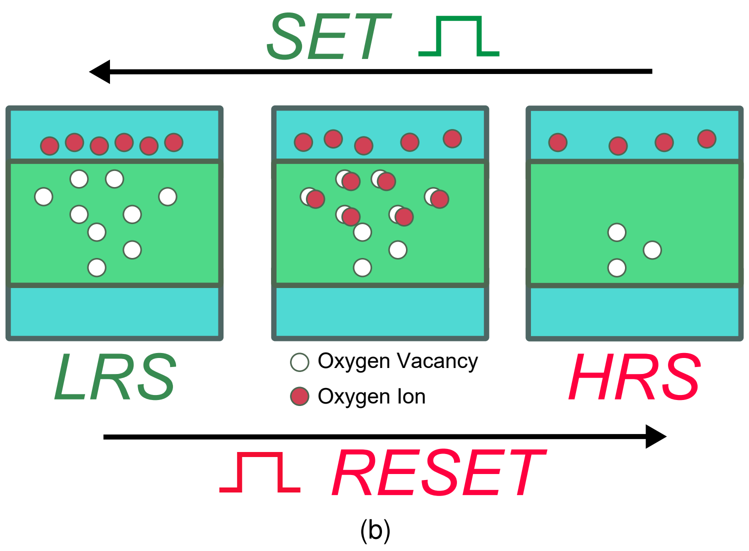
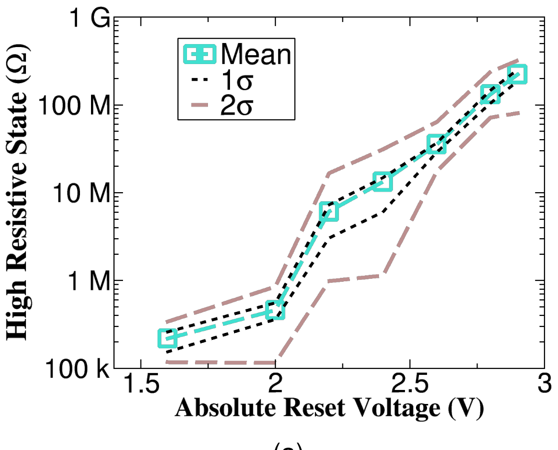


Figure 4.2: (a)The one-transistor-one-resistor (1T1R) structure whereby a resistive memory device is connected in series with either a (left) PMOS or (right) an NMOS. (b) The mechanism of resistance modulation in an oxide based resistive memory. Oxygen vacancies, which form a conductive filament between electrodes, are removed and instantiated due to reduction-oxidation reactions with oxygen ions through the application of RESET and SET pulses. Oxygen ions exists in an 'oxygen reservoir' between the top electrode of the device and the oxide layer.

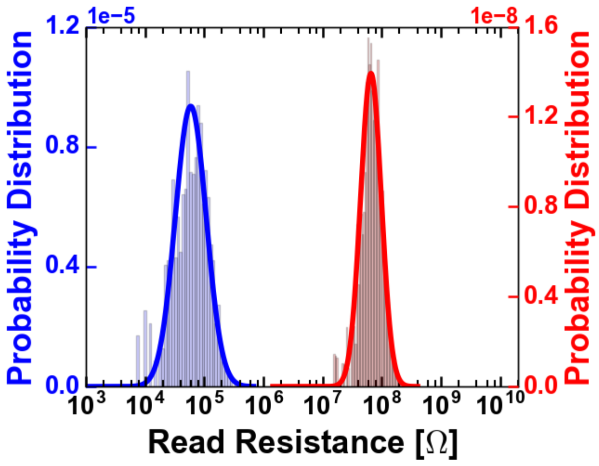
As previously noted, ion channels within neuronal membranes regulate the flow of ionic current into and out of the cell's somatic body which acts as a capacitor. Essentially they represent transient or fixed resistances which regulate flow of charge between an extra-cellular battery and this capacitor and serve as a fundamental building block of animal nervous systems. In the same fashion we propose to use the programmed resistance state of resistive memory devices as a direct means of realising the resistance (or conductance) of biological ion channels in neuromorphic circuit models.

In chapter 3, the devices were used in their low resistance state (LRS) (high conductive state). The resistance extremes of devices in the LRS span from 5kΩ to 50kΩ. However, in order to achieve neural and synaptic dynamics which evolve over 'biological timescales', therein the rate at which relevant environmental variables themselves change, much larger values of resistance than this are required. For example, given a capacitor of 1pF and a desired resistor-capacitor (RC) time constant, τ_{RC}, of some hundreds of micro seconds, a resistance on the order of hundreds of MΩ would be required. Therefore it is required to operate the resistive memory in its high resistive state (HRS). In the case of OxRAM, the HRS resistance is not determined by the properties of a conductive oxygen vacancy filament but, on the contrary, the absence of such a filament. Specifically the read resistance across the two terminals is dominated by the distance of the tunnelling gap between the bottom electrode and the remnants of the disrupted conductive filament [272]. Since the tunnelling current decays exponentially as a function of the tunnelling gap size, and the magnitude of the voltage applied in a RESET operation is proposed to be linearly proportional to the size of the gap, the relationship between the HRS resistance and the applied RESET voltage follows the exponential relationship as plotted in Fig. 4.3a. Two examples of the cycle-to-cycle probability distribution of the HRS are shown in Fig. 4.3b, whereby a single device has been cycled

1000 times under two different RESET voltages. Similarly to how the HCS variability (LRS) was seen to follow a normal distribution, the HRS cycle-to-cycle variability is well modelled using a log-normal distribution. OxRAM devices are therefore log-normal random variables in the HRS, where the mean value of the distribution can be determined by varying the RESET voltage.



(a)

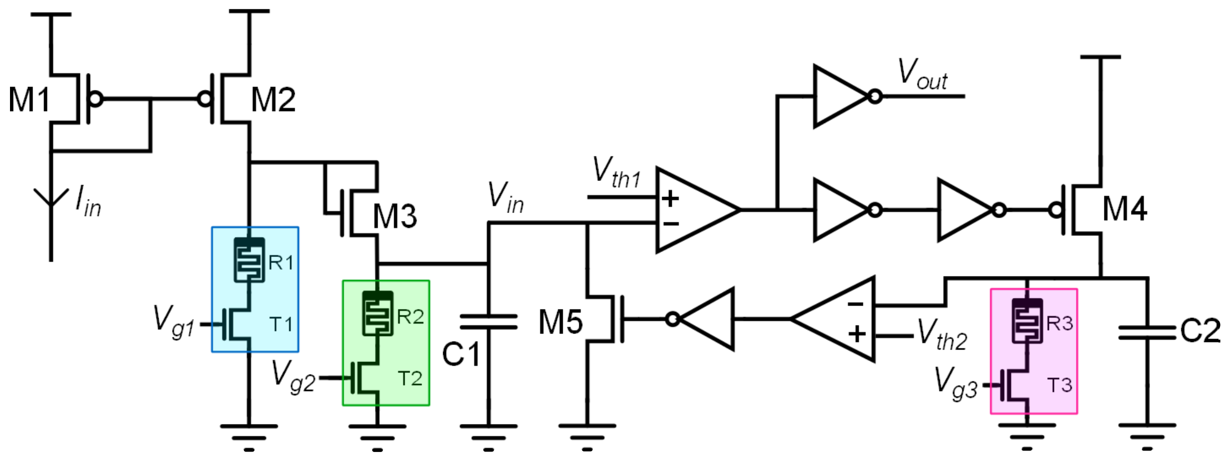


(b)

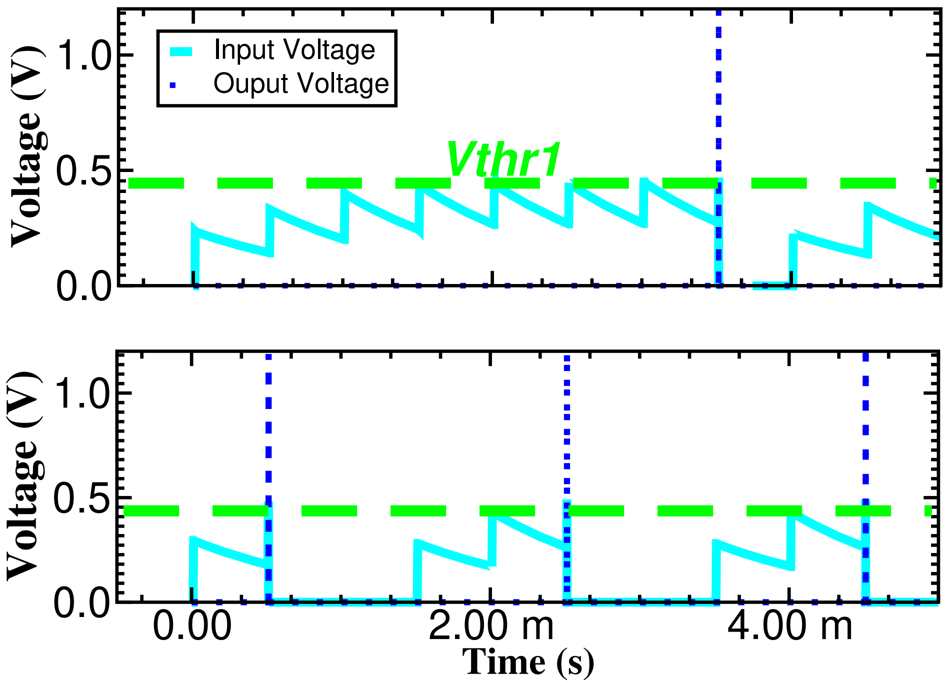
Figure 4.3: **The high resistive state of oxide-based resistive memory devices** (a) The relationship between the high resistive state resistance and the absolute RESET voltage applied over the device terminals for a single device. The mean (blue) as well as one (black) and two (brown) standard deviations are shown that correspond to the cycle-to-cycle variability distribution of the device. (b) Two cycle-to-cycle variability distributions, comprising 1000 SET/RESET cycles, are shown for a single device due to two different absolute RESET voltages - blue: V_reset=1.5V, V_reset=4V and red: V_reset=2V, V_reset=4V. The selector transistors had a gate length of 6.7μm. Each histogram of read resistances over these cycles has been fit with a log-normal probability distribution function which here appears as a normal in log-resistance.

Hybrid neuron circuit

The most straightforward, yet still computationally useful, neuron models are the leaky-integrate and fire (LIF) models [345]. They capture the essence of a neuron's ability to integrate charge on its somatic membrane upon synaptic excitation while simultaneously leaking away this charge in time. Further, upon reaching a threshold of accumulated charge the neuron fires and emits an output pulse which can be propagated to the synaptic inputs of other LIF model neurons. The hybrid differential pair integrator [338] neuron circuit model in Fig. 4.4a captures these behavioural features in an hybrid CMOS-RRAM circuit. Importantly, the device incorporates three 1T1R structures that influence its dynamical properties. Upon the injection of input current, charge is integrated onto capacitor C1. The amount of integrated charge depends on the ratio of the resistance values 1T1R$_2$ (green) to 1T1R$_1$ (blue) and therefore allows for gain tuning. The charge which is integrated onto C1 leaks to ground at a rate defined by the resistance of 1T1R$_2$ (green). If the rate of integration sufficiently exceeds the rate of the leak then a threshold voltage is reached (V_{th1}) (here defined using an OPAMP comparator) and an output inverter sets V_{out} to a logic high. During this firing event, capacitor C2 is charged via the current source M4. As soon as the capacitor exceeds V_{th2} transistor M5 opens and shunts V_{in} to ground - bringing to an end the pulse. Transistor M5 remains shunted to ground for the period the voltage on capacitor C2 remains in excess of V_{th2}, defined by the rate the charge leaks to ground through $1T1R_3$. In summary, the $1T1R_1$ and $1T1R_2$ affect the neuron input time constant and input gain while $1T1R_3$ defines the neuronal refractory period. Two example waveforms obtained with different resistance configurations, obtained through SPICE simulation, of V_{in} and V_{out} under a periodic input current pulse train (1μs pulse-width of 100nA every 250μs) are shown in Fig. 4.4b. For the layout of a hybrid CMOS-RRAM neuron circuit see Appendix 6.15a.



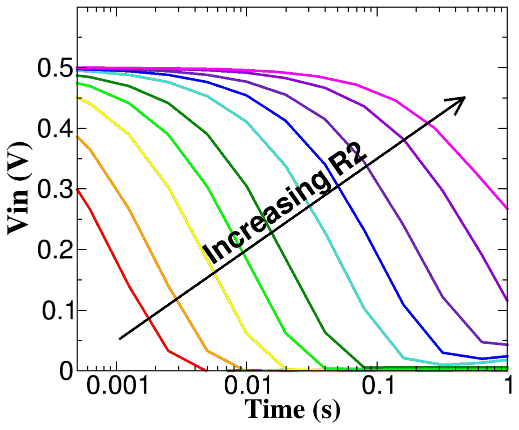
(a)



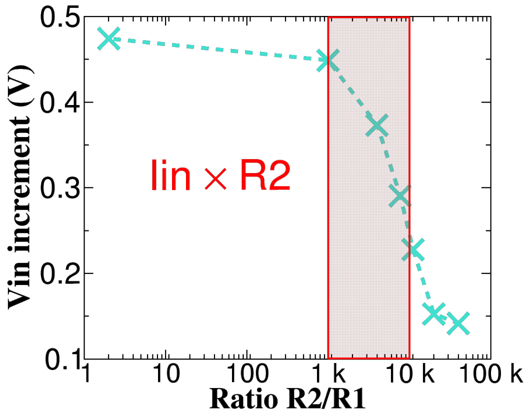
(b)

Figure 4.4: **The hybrid differential pair integrator neuron circuit and its behaviour.** (a) Hybrid differential pair integrator neuron circuit where 1T1R structures are used to set input gain, time constant and the refractory period. All NMOS transistors have a width/length of 650nm/250nm while the PMOS transistors are 1.2μm/250nm. Capacitors C1 and C2 are both 1pF. (b) The circuit is stimulated with a train of square current pulses (1μs pulse-width of 100nA every 250μs) where the input voltage, output voltage and neuron firing threshold are shown. The supply voltage is 1.2V consistent with the voltage rating for the 130nm CMOS technology used to design the circuits. Two resistance configurations are presented which are (top) R1=1GΩ,R2=1GΩ,R3=1GΩ and (bottom) R1=40MΩ,R2=1GΩ,R3=40MΩ.

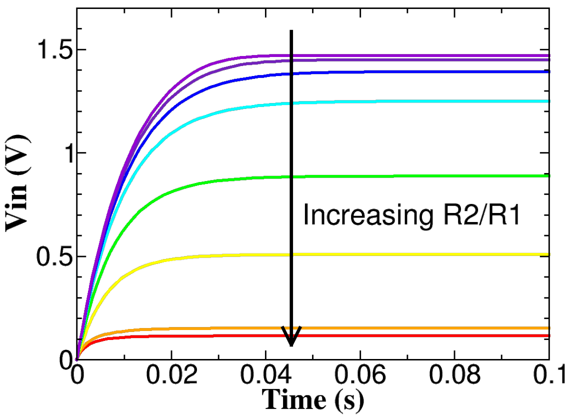
Further SPICE simulations explore the relationship between the RRAM resistances and the neurons behavioural properties. In Fig. 4.5(a), an increasing value of *R2* is seen to increase the RC time constant governing how quickly an initial capacitor voltage of 0.5V decays of 0V. In addition, increasing the ratio between *R2* and *R1* is seen to reduce the voltage increment resulting from a single input pulse starting from an initial *C1* voltage of 0V as well as limiting the maximum steady state value of V_{in} when charged with a constant DC current source in Fig. 4.5(b) and (c) respectively. This ratio should be on the order of the product of *R2* and I_{in} for the gain to be significantly affected (Fig. 4.5(b)). Finally, the effect of *R3* in tandem with the capacitance of *C2* on the refractory period is plotted in Fig. 4.5(d). It is seen that, for a fixed capacitance, the refractory period increases following a power-law with the resistance value.



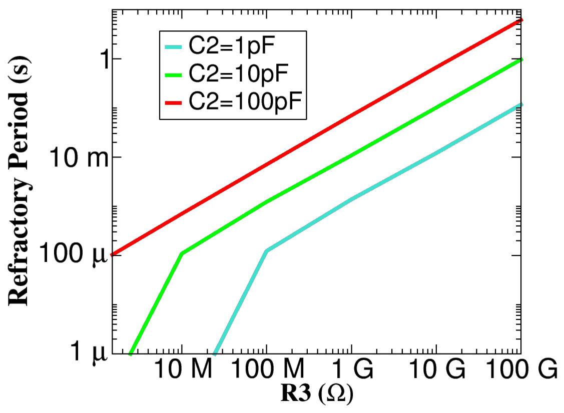
(a)



(b)



(c)

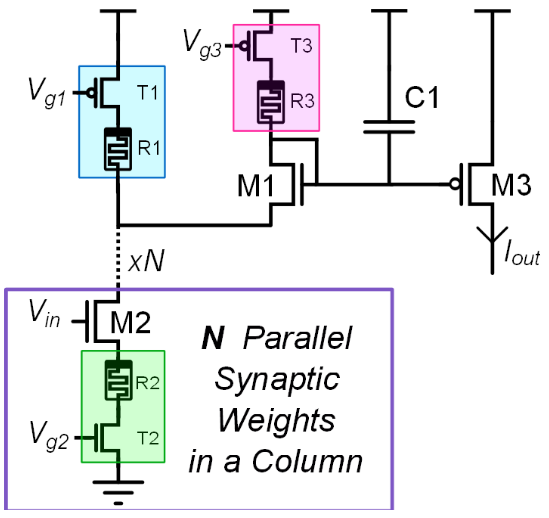


(d)

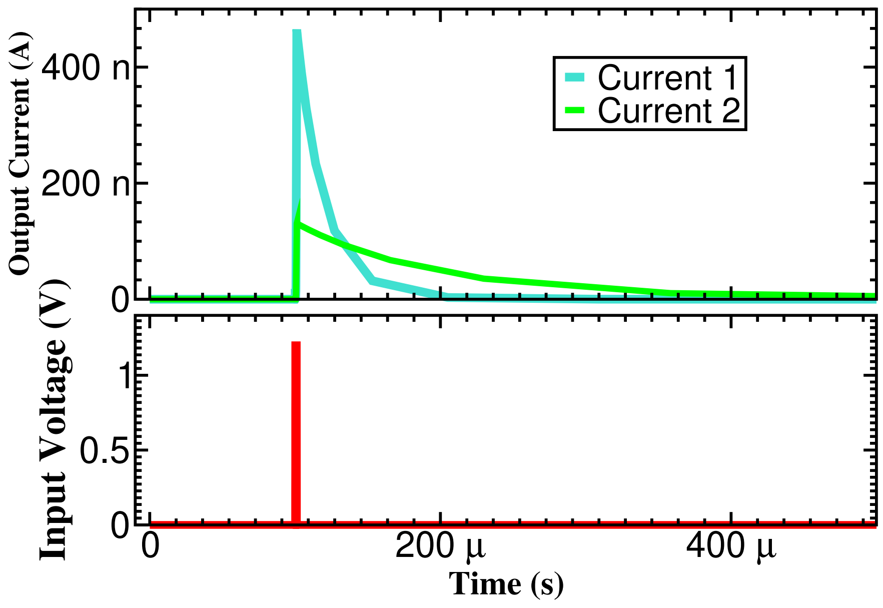
Figure 4.5: (a) *Vin* plotted in log time for *R2* swept from 10kΩ to 10GΩ for a fixed *R1*. As *R2* increases so does the input time constant and the rate of change of voltage decay following an input spike decreases. (b) For a spike at*Iin* and *Vin* at 0V the instantaneous increment of *Vin* is plotted with the ratio of *R2* and *R1* (fixed *R2*). Only when the ratio is on the order of magnitude as *R2* multiplied by *Iin* (red box) is the gain affected. (c) For a DC *Iin*, *Vin* is plotted in time for a range of ratios of *R2* over *R1* from 2000 (red) to 0.1 (violet). The steady state value indicates the maximum *Vin* obtainable for a given ratio. As the ratio reduces the maximum obtainable *Vin* increases (for a given *Iin*). (d) The effect of *R3* and *C2* on the refractory period (during which the neuron cannot integrate incoming signals). The refractory period increases with both quantities.

Hybrid synapse circuit

While the input currents injected into the neuron circuit of Fig. 4.4a were simple rectangular pulses, the synaptic currents injected into biological neurons exhibit temporal properties, reflecting the rate of release and uptake of neurotransmitter in the synaptic cleft. Such temporal behaviours are believed to be important for neural computation [346]. Circuit models exist for mimicking biological synaptic dynamics for use in neuromorphic processors [103]. The simplest model is that of the exponential synapse whereby, during an input voltage pulse (modelling a pre-synaptic action potential), the output current is stepped up and then decays exponentially in time [104]. This is the behaviour of the hybrid differential pair synapse circuit in Fig. 4.6a. Upon an input voltage pulse, V_{in}, a current proportional to the value of $1T1R_2$ (green) flows from C1 to ground for as long the voltage drop over the diode-connected transistor M1 is turned on. As this current flows, during an active high V_{in} pulse, the voltage at C1 reduces and turns on transistor M3 allowing an output current to flow (which can be injected into a neuron circuit model). This voltage over C1 continues to reduce for as long as the voltage difference between C1 and the potential divider node formed between $1T1R_1$ and $1T1R_2$ is sufficient for the diode M1 to remain turned on. Therefore, $1T1R_1$ imposes a limit on the magnitude of the output current that flows in M3 as a function of the capacitor voltage at node *C1*. After the input pulse comes to an end, current stops flowing from *C1* to ground. Instead the capacitor charges up again linearly via a leakage current from $1T1R_3$ (red). Due to the sub-threshold relationship between gate voltage and drain-source current, this results in an exponential reduction in the output current. A SPICE simulation shown in Fig. 4.6b gives two examples of output current waveforms after the arrival of an input voltage pulse for two configurations of the three 1T1R structures which augment the hybrid circuit. It should be noted that, although in Fig. 4.6a one synapse circuit contains one capacitor, inside neuromorphic processors [86, 339], multiple synapse circuits share (along a row or column of a synaptic array) a single capacitor and superimpose their currents onto it. This helps to reconcile the small footprint of a synapse circuit with the large footprint of a 1pF capacitor without compromising on large (biological) time constants. For the layout of a hybrid CMOS-RRAM synapse circuit see Appendix 6.14a.



(a)



(b)

Figure 4.6: **The hybrid differential pair integrator synapse circuit and its behaviour** (a) Hybrid differential pair integrator synapse circuit where 1T1R structures are used to set a weight per synapse and current gain and the time constant of the exponential current decay per column in a synaptic array. All NMOS transistors have a width/length of 650nm/250nm while the PMOS transistors are 1.2μm/250nm. Capacitor C1 is 1pF. (b) During an input voltage pulse (red pulse of 1.2V with a 1μs pulse-width) the output current is incremented. After the pulse the current exponentially decays to zero. The current waveform for two different configurations are shown - Current 1 (R1=10MΩ,R2=500kΩ,R3=1GΩ) and Current 2 (R1=10MΩ,R2=375kΩ,R3=500MΩ). The supply voltage is 1.2V consistent with the voltage rating for the 130nm CMOS technology used in simulation.

4.2.2 An OxRAM based intrinsic plasticity algorithm

One of the main motivations for the incorporation of resistive memory devices into neuromorphic circuit models is the opportunity of non-volatile memory elements to reduce, or eliminate altogether, static power consumption - since energy is not required in order to maintain information once programmed. Another is that, contrary to the case of neuromorphic processors based on a centralised volatile memory, it can permit the realisation of massively parallelised local adaptive plasticity mechanisms. One such local mechanism, desirable in the context of neuromorphic spiking neural network models, is neuronal intrinsic plasticity (IP) [334]. Intrinsic plasticity is a homeostatic mechanism which, along with synaptic scaling [335], has proven important in order to maintain healthy neural dynamics in recurrently connected networks [347]. The basis of IP is to modify the properties that govern the excitability of a neuron such that it is simultaneously able to minimise the energy it consumes due firing while maximising the amount of information its spike train can encode. For example, if the firing rate of a neuron is bounded between zero and a maximum value determined by its refractory period, then, to transmit the maximum volume of information, its firing rate probability density function (PDF) should be a uniform distribution such that information is transmitted equally over the available bandwidth. However, if the neuron should simultaneously minimise its power consumption while firing around a target rate, its firing rate PDF should assume an exponential distribution [348]. This exponential PDF is observed in neural recordings across multiple organisms such as that from the macaque inferior temporal cortex plotted in Fig. 4.7. In this experiment, the animal was habituated to a set of natural scenes, to which the neurons in its visual cortex had adapted to and thereby assumed an exponential firing rate PDF. The animal was then presented with unfamiliar noisy, unnatural, images - resulting in the non-optimal firing rate PDF plotted using a red dashed line in Fig. 4.7.

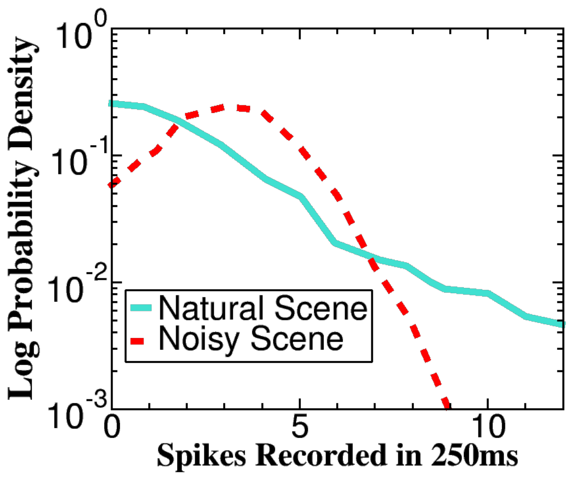


Figure 4.7: Two probability density functions (PDFs) of the firing rate activity recorded from the temporal inferior cortex of a macaque monkey. The PDF recorded from neurons when the animal was presented with natural scenes to which it has been habituated is plotted in blue while the response of the neurons to an unnatural and noisy scene is plotted in red (dashed). Adapted from [348].

To implement a naive 'technologically-plausible' intrinsic plasticity algorithm based on OxRAM device properties, we propose that the log-normal random variable, available in the HRS of OxRAM, can be combined with another random property of OxRAM to implement an in-memory Markov Chain [349] that searches for a stable configuration of the neuron. This additional stochastic property we call the SET Bernoulli random variable.

The SET Bernoulli random variable

Traditionally a SET programming pulse is applied which ensures deterministically that a functioning device transitions from the HRS to the LRS. However, for the case of sub-threshold SET pulses (here below V_{SET}=1.4V) the HfO_2 based RRAM considered in this thesis exhibits a non-deterministic switching mechanism whereby the probability of a device being SET has a dependence on the SET voltage applied over the device [350]. In order to characterise this probability-voltage relationship, devices in a 4k device array (see Appendix 6.3) were subject to a sweep of sub-threshold SET pulses. Devices were re-initialised to an initial HRS state between V_{SET} steps. A resistance threshold of 20kΩ defines a SET device from one which remains in the HRS. The fraction of SET devices after the sub-threshold SET pulse had been applied defines the SET probability per voltage across the array. The cumulative distributions (CDFs) of the 4096 devices in the array for a sweep of V_{SET} are plotted in Fig. 4.8. As V_{SET} increases, devices are more likely to transition from the HRS distribution (right) to the LRS distribution (left). Furthermore, it's interesting to note that even for deep sub-threshold pulses the resulting LRS resistance values fall under the 20kΩ threshold and into the LRS distribution in spite of a small (relative to standard SET conditions) applied programming voltage.

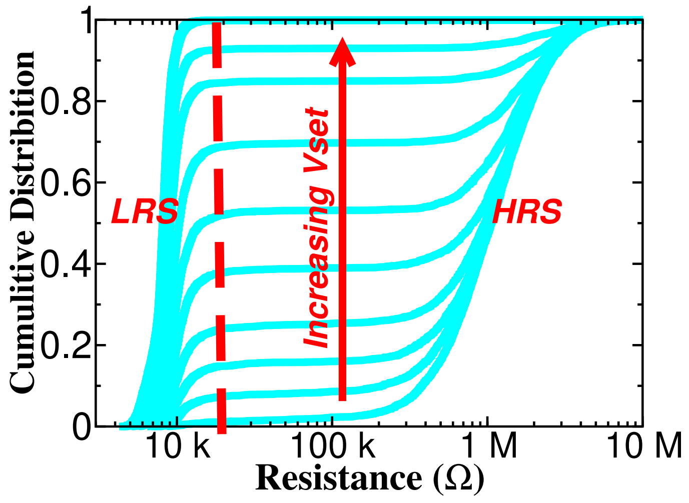
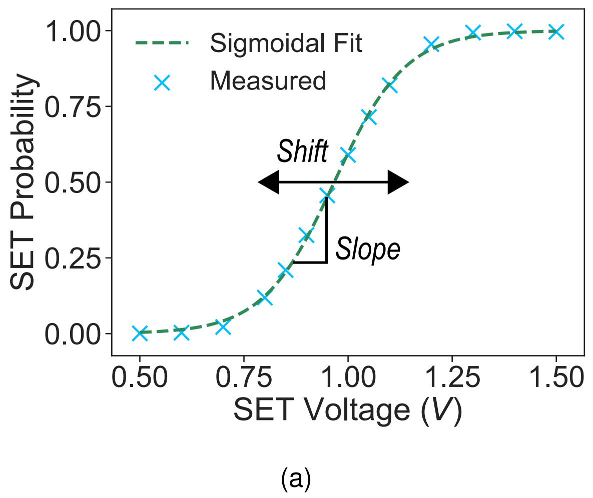
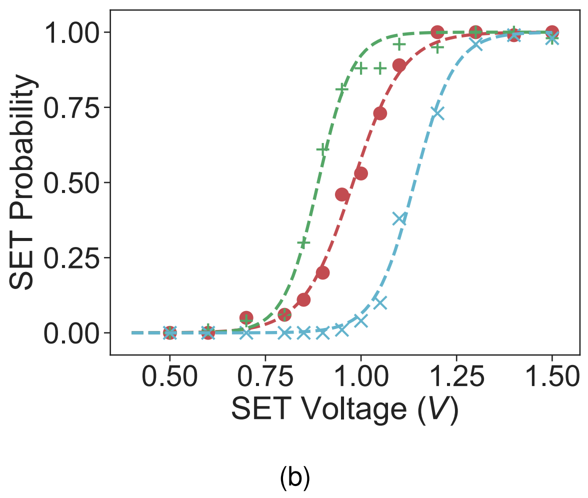
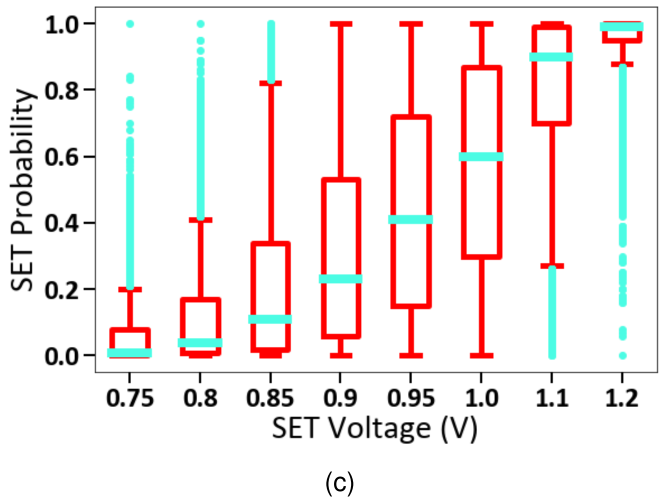


Figure 4.8: Cumulative distributions of device states after sub-threshold 100ns wide programming pulses for a sweep of SET voltages. Those assuming a resistance below a 20kΩ threshold (vertical red dashed line) are defined as SET in the low resistive state. An arrow indicates the CDFs resulting from larger SET voltages.

A SET probability is extracted for each value of V_{SET} by calculating the fraction of devices in the array which assume a resistance below the 20kΩ threshold after programming. The resulting relationship is plotted Fig. 4.9a which is seen to be well fit by a sigmoid function. In a further experiment, where each of the devices were RESET/SET cycled 100 times over a range of V_{SET}, a probability-voltage relationship was determined for each device by calculating the fraction of times (out of one hundred) the device was SET per voltage. The relationship for three devices is shown in Fig. 4.9b, whereby a significant device-to-device variability is observed in their sigmoidal characteristics. The device-to-device variability in this relationship over the full device population is summarised in Fig. 4.9c by plotting the SET probability in a series of boxplots over a range of SET voltages. Strikingly, for a $V_{SET} = 0.95V$, corresponding to a SET probability of 0.42, the box spans the entire SET probability range and there exists a 10% chance that an arbitrary device will SET every time or not at all under the same programming condition. The variability in the sigmoidal slope and shift, the two parameters that determine the shape of the function, is plotted for the full device population in Fig. 4.9d as a scatter plot with the probability density of each independent sigmoid parameter superimposed onto opposing axes. Both parameters are seen to be normally distributed and largely uncorrelated.







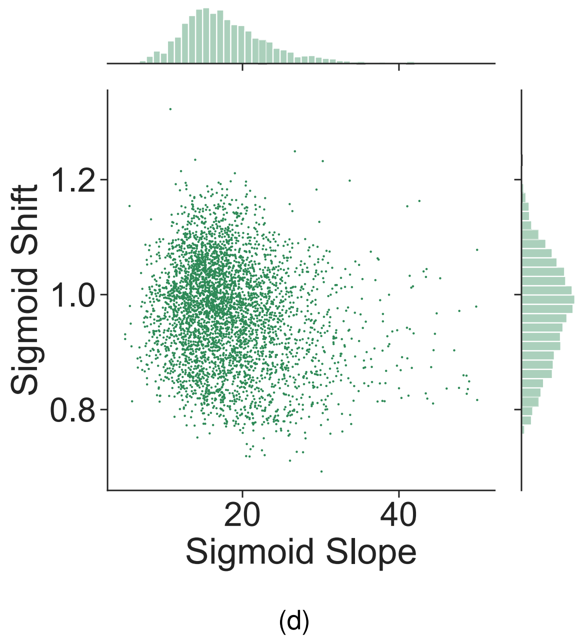


Figure 4.9: **The stochastic SET operation measured in oxide-based resistive memory** (a) The average relationship over an array of 4096 OxRAM devices between the probability of a device being SET into its low resistive state as a function of the SET voltage applied over the device. The relationship can be well approximated as a sigmoidal function. Two free parameters of the sigmoid function are the shift (in SET voltage for p=0.5) and the slope (the rate at which the probability changes with SET voltage) which are annotated in the plot. (b) Three single device relationships between SET probability and the applied SET voltage. Each device was cycled 100 times over a range of SET voltages and the fraction of times the device switched corresponds to the SET probability at that SET voltage for that device. (c) Boxplots showing the SET probability distribution over an array of 4096 devices for a range of SET voltages. A blue horizontal bar shows the SET probability median, limits of the box show a 25$\%$ deviation from the median, whiskers show a 45$\%$ deviation and blue points correspond to outlying devices. (d) A scatter plot where each point corresponds to the sigmoidal slope and shift fitted to a single device. In total 4096 devices are plotted. The resulting probability density function of the slope and shift are plotted on opposing axes. The device-to-device variability in slope and shift are both seen to be normally distributed and uncorrelated.

The deviation between the probability of a single device and the median probability (the median of all

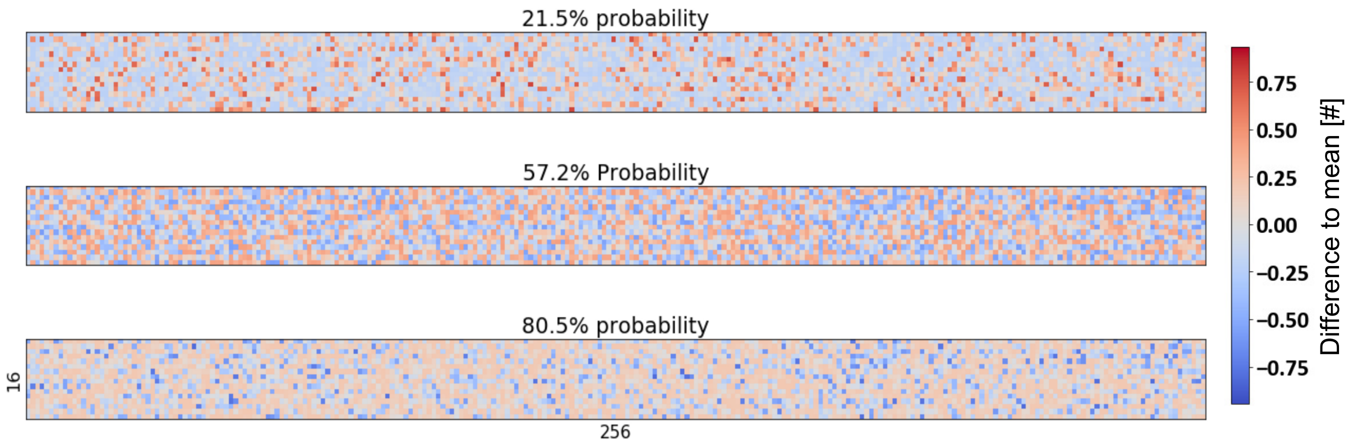


Figure 4.10: Switching probability for each device in a 16x256 cell 4k device array coloured based on its deviation from a the mean switching probability over a 4k 1T1R array. Strong reds and blues indicate a significant deviation while softer shades show cells close to the mean probability.

devices over 100 cycles) is plotted for three median probabilities in a heatmap in Fig. 4.10. The heatmaps respect the shape of the RRAM array composed of 16 wordlines and 256 source/bit lines. Soft reds and blues correspond to devices with switching probabilities equal to or close to the mean per applied SET voltage. Stronger reds and blues indicate, by contrast, devices which have a switching probability significantly less (blue) or greater (red) than the median value for the SET voltage. Consistent with Figs. 4.9b, 4.9c and 4.9d, its also clear from visual inspection that a substantial device-to-device variability in the switching probability is present. The dispersion is most pronounced at voltages corresponding to probabilities between 21.5$\%$ and 80.5$\%$ whereby the top and bottom heatmaps of Fig. 4.10 are peppered with divergent coloured cells. The spatial distribution of this device-to-device variability would also appear to be random-uniform. However, to properly assess the absence of spatial correlations, the SET probability variability distribution was evaluated according to the NIST test suite SP800-22 [351]. This test suite is commonly used to validate random number generators by running fifteen tests on the generator output - in particular searching for spatial correlations. The number walk, composed of the complete 4k device over the array and the 100 cycles for each SET voltage, passes the full suite of tests. According to these tests, the device-to-device spatial correlation can be confidently considered as non significant.

Specification of algorithm

The OxRAM based Markovian IP algorithm, which exploits both the HRS log-normal random variable and the Bernoulli SET random variable, is depicted in Fig. 4.11 which operates as follows. A neuron measures its own firing rate by leaky-integrating (low-pass filtering) its output spike train. Then, at fixed intervals (here 400ms), it compares this integrated rate with a target rate. Based on this firing-rate difference error, a SET/RESET cycle is performed on $1T1R_1$ and $1T1R_2$ of the hybrid neuron of Fig. 4.4a. These parameters control the input gain and input time constant and thus determine the neurons excitability. By performing a RESET operation on the devices, new parameter values for the neuron circuit model are effectively sampled from a log-normal random variable - changing it's excitability accordingly. However, continually re-sampling these parameter values in this fashion will not allow the neuron to converge towards a stable state. Therefore we propose to modulate the re-sampling using the SET Bernoulli random variable. By generating a voltage over each of the RRAM devices incorporated into the neuron circuit proportional to the firing-rate difference error (between the measured and target rates) the device will transition to the LRS with some probability that is a function of this error. If the device remains in the HRS (i.e., the device did not SET after application of the voltage), then the parameter is not re-sampled. If the device switches, however, the device is immediately RESET and therefore samples a new resistance value from its high resistive state cycle-to-cycle variability distribution. Crucially, in a similar vein to the ideas presented in section 3, this implements an in-memory Markov Chain locally within each neuron by leveraging the random variables within the OxRAM nano-devices that can be harnessed through the application of simple electrical pulses.

To regulate how sensitive the parameter re-sampling is to the measured firing-rate difference error, we introduce a 'tolerance'. This tolerance sets a minimum error that is tolerated before the re-sampling probability for a neuron becomes non-zero. The tolerance is an important quantity in the algorithm since a tolerance which is too small prevents convergence to a stable state since the neuron's configuration is too sensitive to small fluctuations in its input activity. At the other extreme, a tolerance which is excessively large prevents a neuron from self-organising at all. Furthermore, we propose that 1T1R$_1$, since it impacts only the gain, should re-sample from an HRS PDF with a median equal to its current resistance value while 1T1R$_2$ should re-sample from an HRS PDF with a median shifted by a constant learning rate from its previous value. The learning rate multiplied by the current resistance value can then be added to or subtracted from this value give the value of the new median. Since 1T1R$_2$ has a positive correlation with the firing rate (as it governs the input time constant) this mean shift should be positive for under-firing and negative for over-firing.

Accompanying circuits

Such an algorithm should be implemented locally, inside or near to the neuron circuit, with further analogue or digital CMOS circuits - thereby avoiding a von Neumann bottleneck. Here we provide examples of two CMOS circuits, which respectively detect the sign and magnitude of the error difference between the measured an target firing rates (Fig. 4.12), and then use these quantities to generate an appropriate SET programming voltage pulse over a given 1T1R structure (Fig. 4.13).

The circuit in Fig. 4.12 compares the constant bias voltage V_{target} with the voltage signal corresponding to the integrated output firing rate iV_{out}. We assume that this integration can be performed using a differential-pair integrator circuit [338]. Currents equal to the positive and negative difference between iV_{out} and V_{target} are copied into transistors M6 and M7. If iV_{out} is greater than V_{target}, M7 pulls the node at the input of the inverter down to ground and results in a high output signal. In the contrary case, M6 pulls the node up to VDD, and the output of the inverter is low. The output signal UP/DN therefore provides a logical signal corresponding to the sign of the firing-rate difference error. The voltages $V1$ and $V2$ vary with the logarithm of this error for negative and positive differences respectively. If the error is large and positive, for example, $V2$ would reflect this in assuming a large voltage relative to $V1$. Additionally, in the case where iV_{out} and V_{target} are very close, the 'bump' circuit [352] in the centre of the schematic results in a large $stop$ voltage.

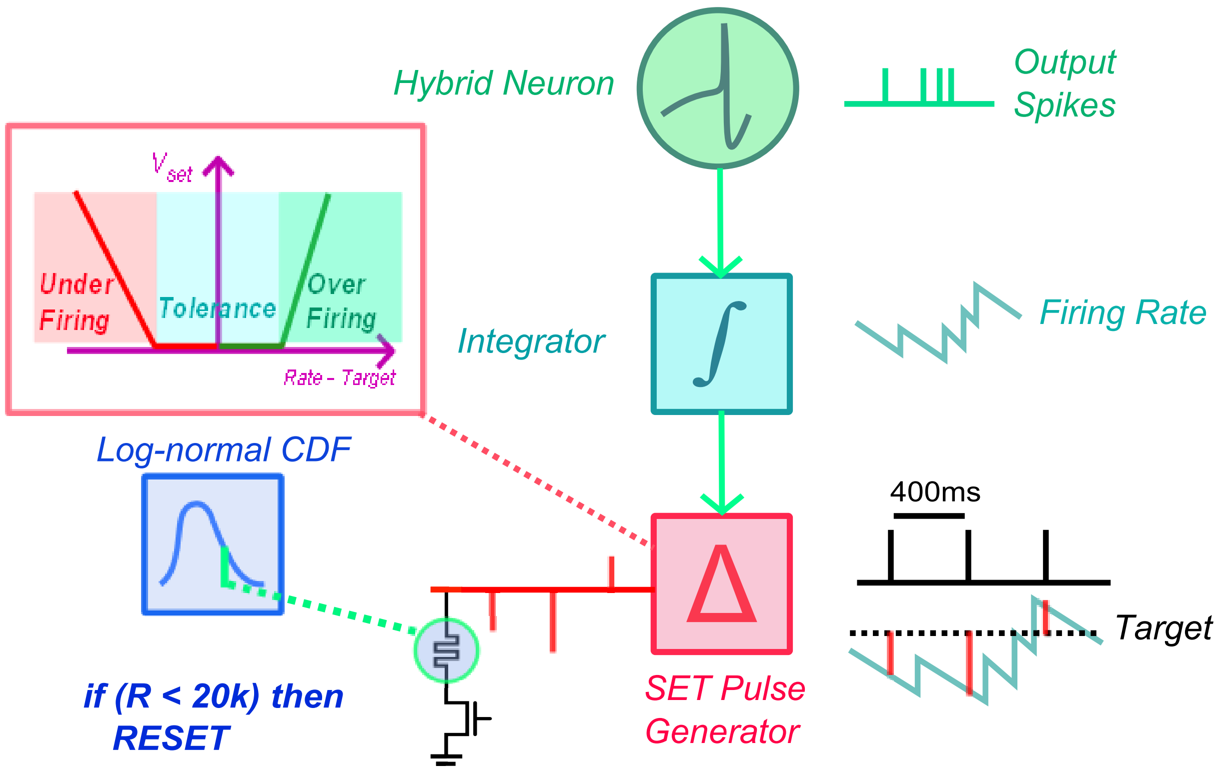


Figure 4.11: Diagram of the proposed OxRAM-based intrinsic plasticity algorithm. The hybrid neuron circuit has two 1T1R that set the properties of the neuron model (green circle). The neuron propagates a spike/pulse train to an integrator circuit (light blue block) which transforms the discrete voltage pulses into a continuous analogue voltage encoding it's activity. This signal (blue waveform) is compared with a target (black dashed line) and periodically (black pulse train) the error to this target is evaluated (red pulses). Based on these differences, SET voltage pulses are generated (red block) over the incorporated 1T1R structures in their high resistive states. This intrinsically makes use of the SET Bernoulli random variable to make a decision on whether the resistance value should be re-sampled. If the device is SET, such that the resistance after the SET operations is below 20kΩ, then it is immediately RESET at which point the resistance values of the neurons incorporated RRAM are re-sampled (green vertical bar) using a log-normal random variable (navy blue block). This random variable is accessed through the intrinsic probability density of a the HRS cycle-to-cycle variability.

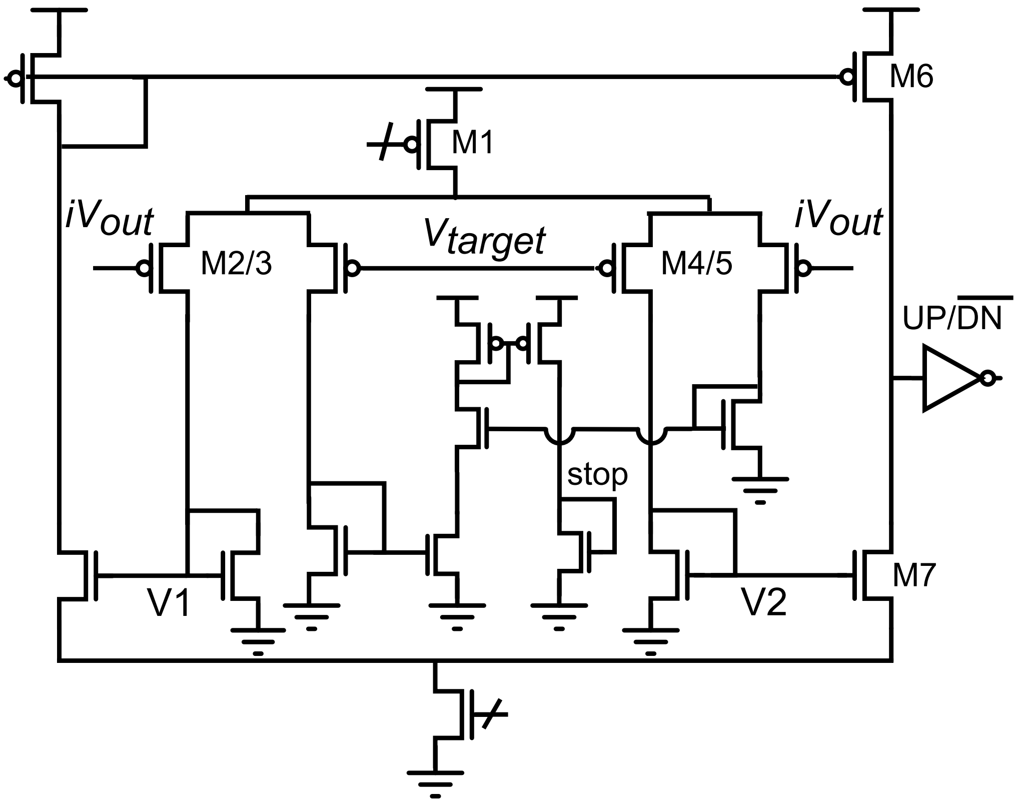


Figure 4.12: Schematic of the circuit responsible for computing the sign and magnitude of the firing-rate difference error. Text labels correspond to important node voltages besides labels beginning in M which name certain transistors.

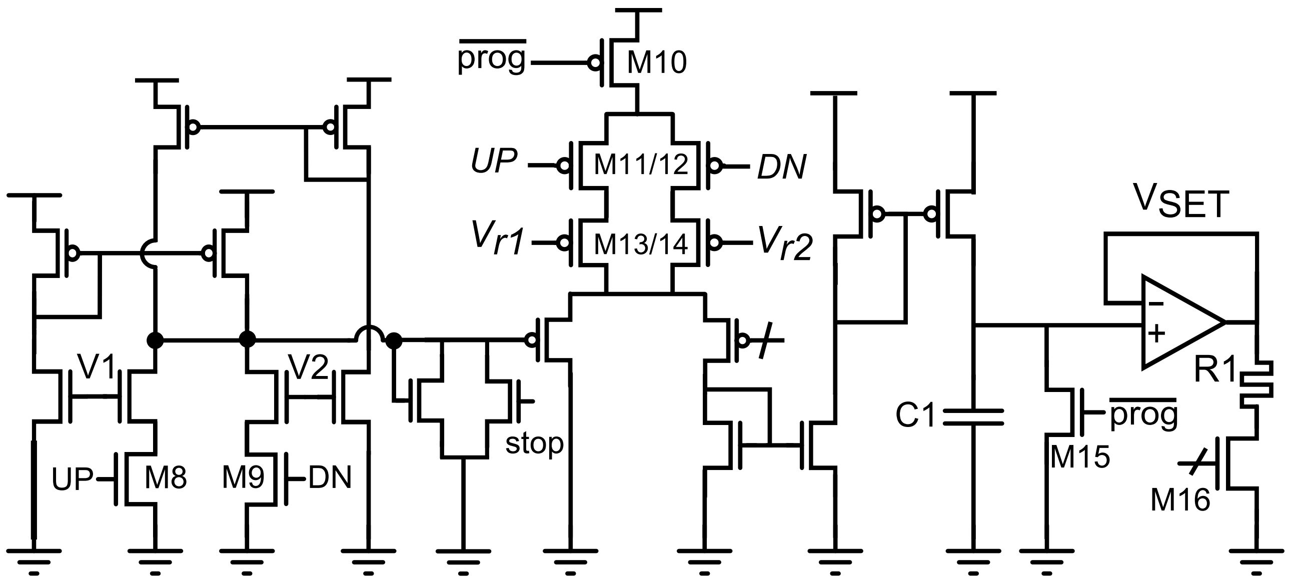


Figure 4.13: Schematic of the circuit responsible for applying a SET programming pulse to a 1T1R structure as a function of the outputs of the circuit in Fig. 4.12 and the biases V_{rl} and V_{r2}. Text labels correspond to important node voltages besides labels beginning in M which name certain transistors. The OP-AMP drawn at the right-hand side is a standard operational transconductance amplifier circuit.

These five signals are then input to the circuit in Fig. 4.13 along with one control signal $prog$ and two further bias parameters V_{r1} and V_{r2}. Each time that the active-low control signal $prog$ is pulsed, a current flows onto capacitor C1. The resulting voltage ramp is then buffered onto the eventual 1T1R structure - appearing as a ramped voltage pulse over the device. The peak of the voltage ramp corresponds to the applied V_{SET} voltage, and therefore determines the probability of the device being SET into its low resistance state. If the voltage $stop$ is greater than $V1$ or $V2$, then the current that flows onto the capacitor is negligible and the peak V_{SET} voltage will be zero. When this is not the case, a current proportional to $V1$ if UP/DN is low or $V2$ if UP/DN is high flows onto C1. This current is also modulated as a function of the sign of UP/DN and the bias voltages V_{r1} and V_{r2}. V_{r1} is used to influence the rate of change of the peak V_{SET} with a negative firing range difference, and V_{r2} for a positive firing rate difference. This effectively allows for an asymmetrical relationship between firing rate difference and SET probability (Fig. 4.11) which may be favourable, for example, in the situation where positive deviations from the target firing rate are punished more (i.e, there is a greater chance to re-sample parameters) than a negative one since elevated firing rates result in a higher power consumption of the circuit. The peak value of V_{SET} is plotted with the firing rate difference in Fig. 4.14 for three pairs of V_{r1} and V_{r2} - demonstrating how their values can be altered to obtain different symmetrical or asymmetrical relationships.

Spiking neural network simulation

We now assess the effectiveness of the algorithm through a set of spiking neural network simulations calibrated on the measured characteristics of the OxRAM devices. First, the case of a single neuron under stimulation from a Gaussian current source, whereby the DC current injected into the neuron re-samples from a Gaussian distribution every 20ms, is considered. As described, the difference between

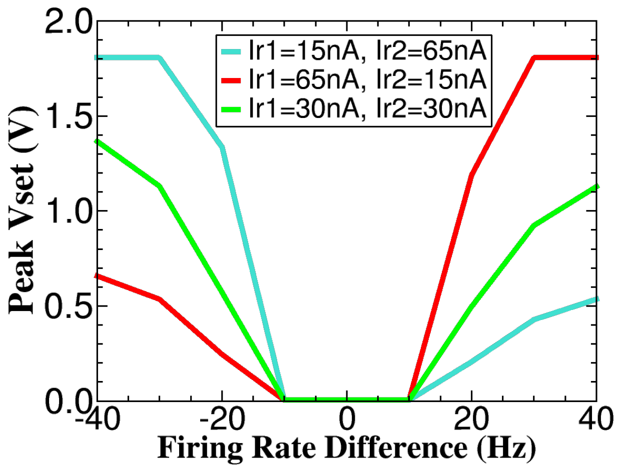
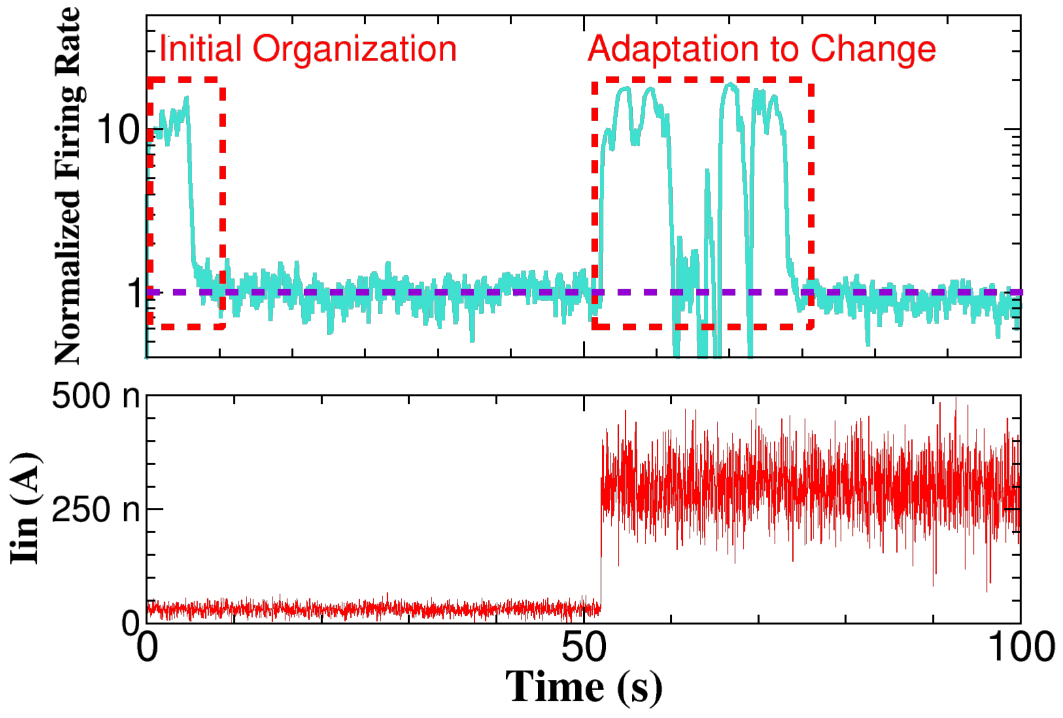


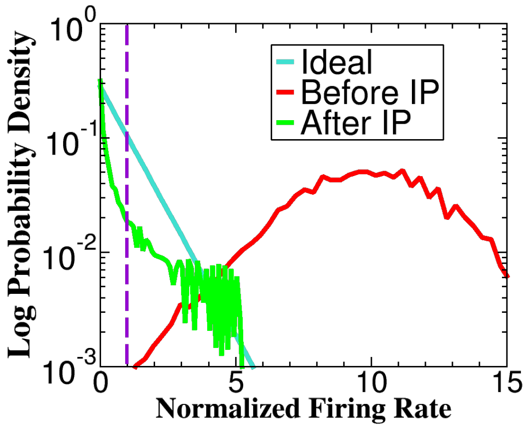
Figure 4.14: **Effect of the biases V_{rl} and V_{r2} on the SET pulse peak voltage value.** For three bias configurations, which are labelled in the legend with the currents which are mirrored into the circuit instead of the actual voltage applied to the transistor gates, the peak value of the V_{SET} ramp as a function of the firing rate difference is plotted. It should be noted that for uneven biases, an asymmetrical relationship can be obtained.

the measured and target rates is used to re-sample *R1* and *R2* from a log-normal HRS random variable (Fig. 4.3) with a sigmoidal probability as a function of the firing rate difference, computed by the SET Bernoulli random variable (Fig. 4.9a). In this case, the sigmoid for excessive firing is steeper than for the contrary - facilitated by the circuit of Fig. 4.13 - with the intuition that this should encourage a configuration favouring an exponential firing rate PDF. Resistances of the resistive memory devices are randomly initialised and the firing rate is initially in excess of the target in Fig. 4.15(a). However, in time the neuron adapts *R1* and *R2* to fire around the target rate. The properties of the input Gaussian current source are modified at around 50s and the neuron is able to once again find a new stable configuration. In addition, the neuron assumes a firing rate distribution similar to that of the ideal case of the exponential PDF, plotted in Fig. 4.15(b). This indicates that the neuron has minimised its power consumption around this target while maintaining its information capacity. To evaluate this adaptation in time, the Kullback-Leibler divergence (which computes the similarity between sets of distributions) between the evolving firing-rate of the hybrid neuron PDF and the ideal distribution is plotted in Fig. 4.15(c) - a value of zero corresponds to the optimum configuration. After an initial period of ten seconds which exhibit significant oscillations, the Kullback-Leibler divergence decreases from a maximum value of 10 to a value around 0.1

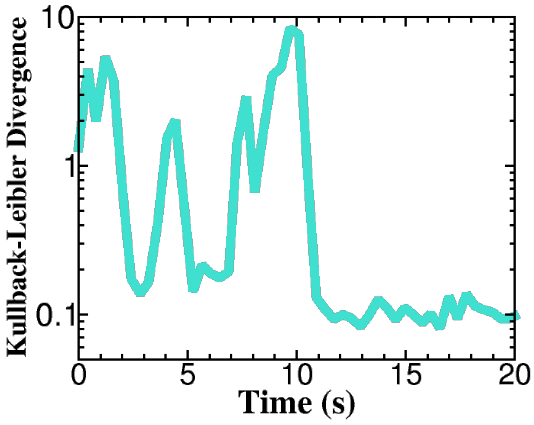
Next, we consider the application of OxRAM based IP in the case of a recurrently connected spiking neural network. The simulated topology consists of an input layer (blue) of 12 Poisson neurons [353] which feed-forward into a recurrently connected excitatory population of 35 neurons (green) with a connection probability of 0.75. Poisson neurons are neurons which fire at random intervals such their inter-spike time PDF is a decaying exponential function. The excitatory neurons have a 0.2 chance to connect recurrently amongst themselves and there is no spatial connectivity kernel as is the case of liquid-state machines [83]. In addition, the excitatory population excites an inhibitory population (red). The neurons in this population recurrently connect amongst themselves and also project inhibitory synapses to the excitatory population - putting on the brakes via negative feedback when the positive population is excessively excited. The



(a)



(b)



(c)

Figure 4.15: (a) The neuron adapts *R1* and *R2* such that it fires (blue) around a target rate (purple dashed) with a Gaussian current source (red). After a change in the properties of the current (50s) the neuron re-adapts *R1* and *R2* to fire around the target rate once again. The rate has been normalised to the target. (b) Target (purple dashed) normalised firing rate probability densities (PDF). The PDF after organisation (green) is closer to the ideal (blue) than that of a randomly initialised neuron (red). (c) The evolving Kullback-Leibler divergence plotted as the neuron organises *R1* and *R2*. A value of zero indicates the neuron has found the ideal configuration that maximises information and minimises power.

neurons in the excitatory population are equipped with the proposed intrinsic plasticity algorithm. The tolerance is set to 70Hz for both over- and under-firing. All synapses are the hybrid synapses of Fig. 4.6a and the neurons in the excitatory population are the hybrid neuron models of (Fig. 4.4a). The inhibitory population are simply LIF neuron models whose parameters are not subject to IP. The resistance values of the hybrid neurons are bounded within the order of measured values in Fig. 4.3.

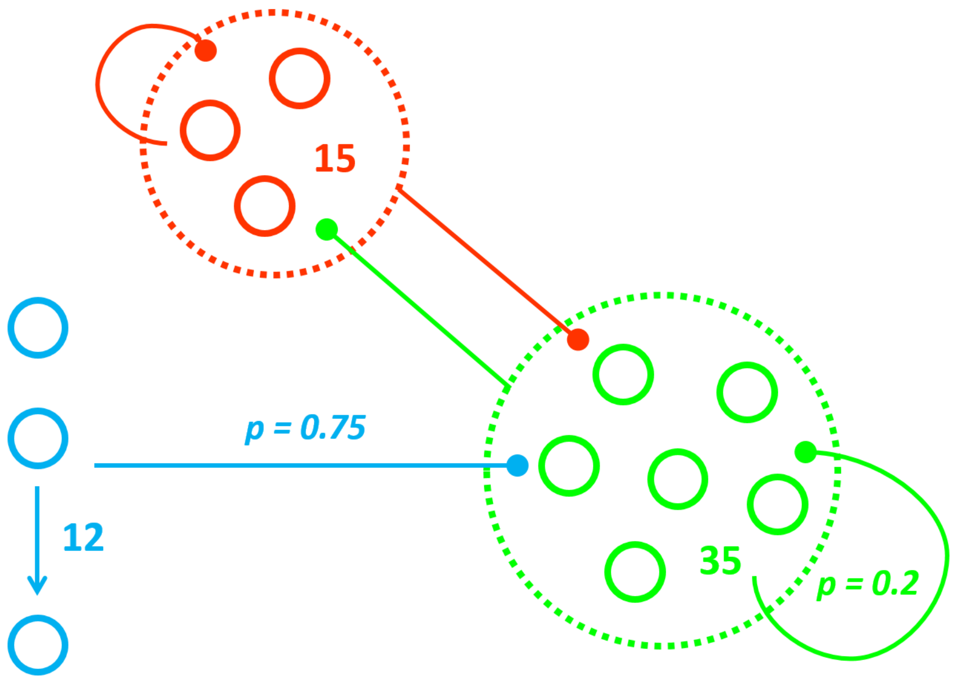
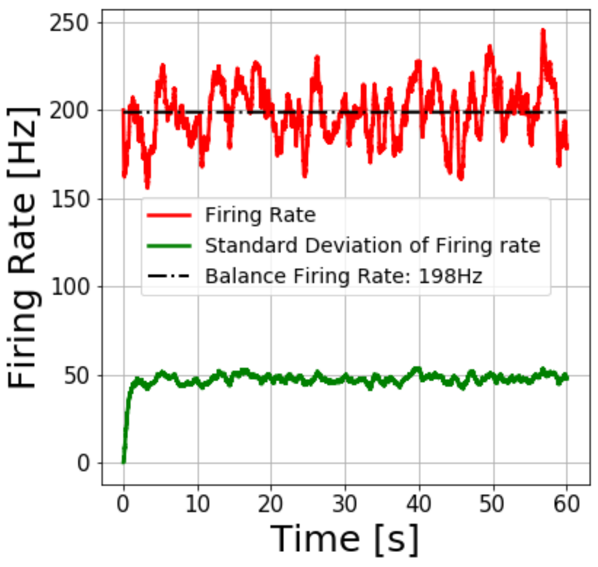
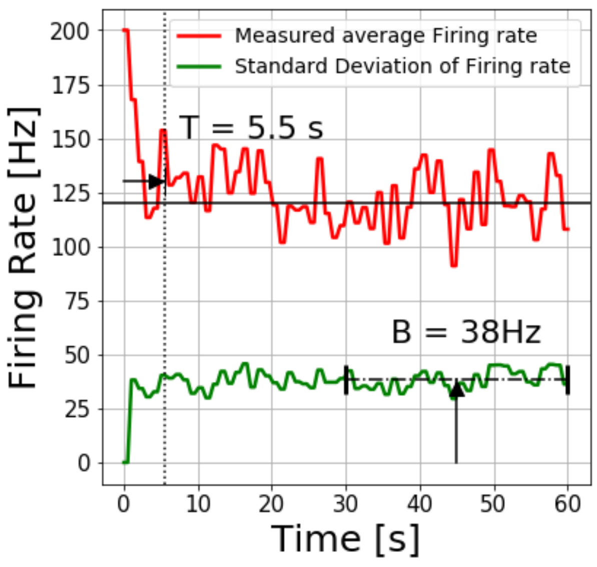


Figure 4.16: The recurrent spiking neural network topology used in simulation. An input Poisson group (blue) feeds forward to an excitatory neuron population which are hybrid DPI neurons with intrinsic plasticity (green). A population of inhibitory neurons (red) is excited by the excitatory population and feeds back with inhibiting synapses that impose an upper limit for the mean firing rate in the excitatory population.

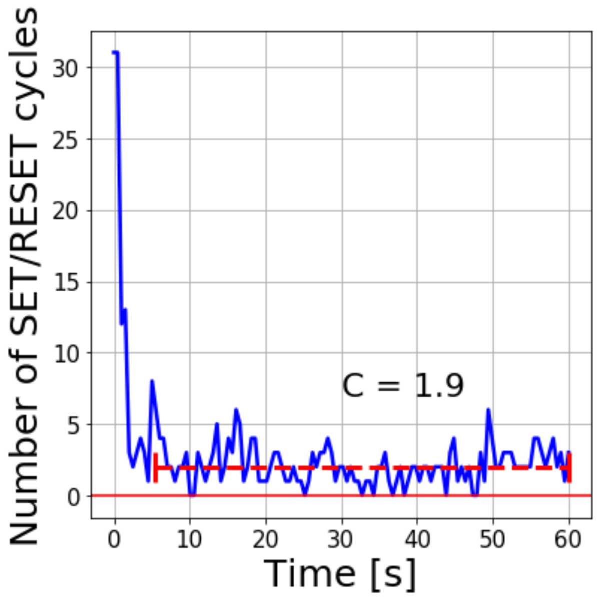
First, for illustrative purposes, the mean firing rate and standard deviation in the firing rate for the 35 excitatory neurons are plotted in the absence of an IP algorithm in Fig. 4.17a. The mean rate oscillates around a natural frequency of 200Hz while the standard deviation amongst firing rates within the population is 50Hz. By contrast, Fig. 4.17b plots the same metrics where the neurons in the excitatory population employ the proposed IP algorithm - given a target firing-rate of 120Hz. After an initial transient period of excessive over-firing the network quickly self-organises in 5.5 seconds and then settles in a configuration where the mean firing rate respects the stipulated target. Notably this, despite the increase in the number of neurons, this is faster than in the case of the single neurons in Fig.4.15a. The standard deviation amongst the firing rates is 38Hz. Lastly, in Fig. 4.17c, the number of SET/RESET programming cycles, per each each 400ms periodic refresh, drops from an initial count of 34 cycles to 2.1 cycles. Low RRAM switching activity is an equally important indication of convergence since not only should the network mean tend to the target, while maintaining an acceptable standard deviation amongst the individual rates in the population, but the switching activity should also cease (or become negligible).



(a)



(b)



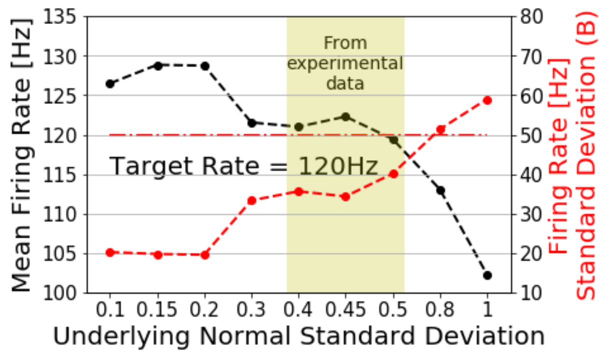
(c)

Figure 4.17: **Spiking neural network simulations employing intrinsic plasticity allows the recurrent network to self-organise to fire at a target rate.** Firing rates are plotted in red, standard deviations in firing rates amongst the population in green and the number of SET/RESET cycles per periodic update is plotted in blue. (a) A plot of the average firing rate and the standard deviation between firing rates of the excitatory population plotted without the application of IP. (b) A plot of the average firing rate and the standard deviation between firing rates of the excitatory population plotted in the presence of the proposed IP algorithm. The variable T denotes the time after initialisation for the network to converge and B the standard deviation amongst firing rates after convergence. (c) Plot of the reduction in the number of RESET/SET operations as a function of time per 400ms re-sampling period. The annotated variable C denotes the number of RESET/SET cycles per re-sample after convergence.

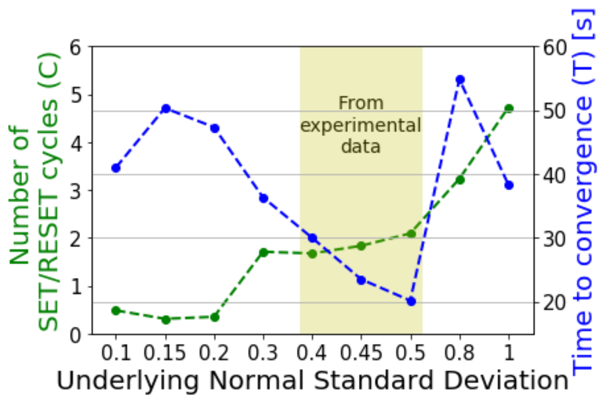
Impact of device variability

The properties of two OxRAM random variables, like any property of resistive memory devices, comes with of course an inherent variability. In this section, we investigate the contribution of two variability sources - the cycle-to-cycle variability of the HRS log-normal random variable and the device-to-device variability in the SET Bernoulli random variable. As plotted in Fig. 4.3b, cycle-to-cycle variability in the HRS after a RESET operation follows a log-normal distribution. One way of quantifying this variability is through the standard deviation of the 'underlying' normal distribution of the log-normal. The underlying normal is simply obtained by applying a logarithm function to the log-normal distribution. Across the devices measured in this study the underlying normal standard deviation of HRS log-normal random variable was approximately between 0.4 and 0.5. The device-to-device variation in the SET Bernoulli random variable is modelled by sampling the sigmoidal shift parameter from the measured normal distribution plotted in Fig. 4.9d. The sigmoidal slope was considered to be constant. We quantify the effects of these two variability sources using the defined performance metrics annotated in Figs. 4.17b and 4.17c as; time to convergence (T), standard deviation amongst firing rates after convergence (B) and count of SET/RESET cycles after convergence (C)) averaged over three independent runs from randomly initialised parameters. We first examine the impact of the cycle-to-cycle HRS variability on the network in Figs. 4.18a and 4.18b.

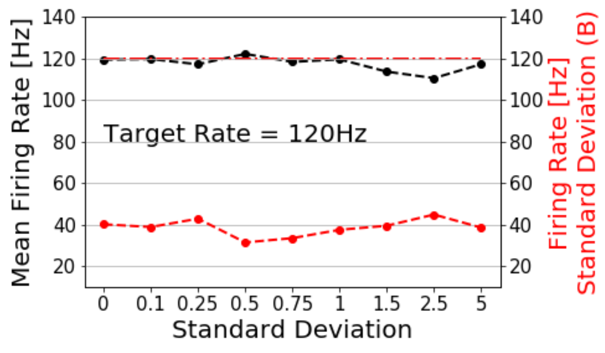
In Fig. 4.18a, it is seen for low values of standard deviation of the cycle-to-cycle HRS PDF the network struggles to settle to a mean precisely equal to the target - although there is a low standard deviation amongst firing rates and a low count of SET/RESET cycles after converging. This is likely linked to the fact that the unnaturally narrow log-normal distribution does not allow for a sufficient search of the parameter space. This is supported by the result in Fig. 4.18b, whereby the convergence time drops for a higher cycle-to-cycle HRS standard deviation up to 0.5. Due to the more progressive convergence, the neuron likely stops re-sampling parameters when it enters the fringes of its tolerated firing-rate difference error. In contrast however, for values larger than 0.5, the convergence time is then seen to increase again. By the same logic, this should be due to the fact that consecutive states of the Markov Chain are too distal and not sufficiently correlated. Within the range of experimentally measured cycle-to-cycle variability therefore the hyper-parameters of the algorithm have allowed it to find a sweet spot and converge faster than values higher or lower cycle-to-cycle variability. This leads to the conclusion that the intrinsic cycle-to-cycle HRS variability has a positive impact on performance. In Figs. 4.18c and 4.18d the same metrics are plotted but for the case of device-to-device SET variability. Here we sample undesired horizontal shifts in the probability-error sigmoid from a normal distribution for each device. Therefore each has a permanent offset from the desired value which impacts the effective tolerance. The time to convergence in Fig. 4.18d increases with greater standard deviation of the normally distributed shifts in the probability-error sigmoid. However, the standard deviation amongst the neurons firing rates, the mean distance from the target firing rate and count of SET/RESET cycles appear to be largely unaffected. This result is encouraging since it appears that, even in the presence of significant device-to-device variability, the IP algorithm allows the network to self-organise and find a configuration which can compensate for the non-ideal devices and fire around the target rate - at the expense of a longer period of self-organisation.



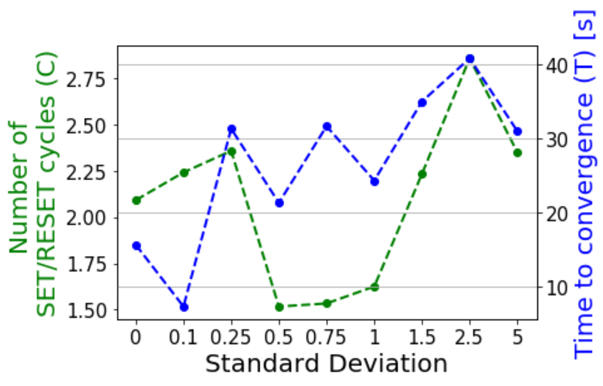
(a)



(b)



(c)



(d)

Figure 4.18: **The impact of cycle-to-cycle variability in the RESET and device-to-device variability in the sub-threshold SET on performance metrics of the intrinsic plasticity algorithm.** (a) Impact of the standard deviation (of the underlying normal) in the cycle-to-cycle high resistive state resistances log-normal probability density function (following a RESET) on mean firing rate and standard deviation in the firing. (b) Impact of the standard deviation (of the underlying normal) in the cycle-to-cycle high resistive state resistances log-normal probability density function (following a RESET) on convergence time and the number of SET/RESET cycles after convergence. (c) Impact of normally distributed device-to-device SET probability standard deviation on firing rate and standard deviation in firing rate. (d) Impact of normally distributed device-to-device SET probability standard deviation on convergence time and the number of SET/RESET cycles.

Power Consumption

A SET/RESET cycle, required to re-sample a parameter, incurs a fixed penalty in energy and therefore such an algorithm will consume an amount of energy proportional to the update rate (here 400ms) and the number of devices in a network which have undergone a SET/RESET cycle during this periodic update. Under standard programming operations (see Appendix 6.3) the 1T1R structures studied in this paper consume approximately 50pJ per SET/RESET cycle. Neurons also pay an energy penalty every time they spike (for the DPI neuron in 180nm CMOS this is 800pJ). This is therefore an order of magnitude more expensive than a SET/RESET cycle as well as being approximately two orders of magnitude more frequent. Clearly, as is the case in biology, it becomes advantageous to expend a small amount of energy to reduce the (comparatively) much greater energy consumed via excessive neural activity. As an illustrative example we plot the cumulative energy consumption of the two networks in Fig. 4.17b (one employing IP and the other firing at its natural rate) in Fig. 4.19. Since the target firing rate (120Hz) is significantly lower than the natural rate (200Hz) the energy consumed, in spite of the cost of initial organisation considering the SET/RESET cycling, is reduced by half. This demonstrates the opportunities in energy management of the algorithm in applications in which the system is not connected to a reliable source of power.

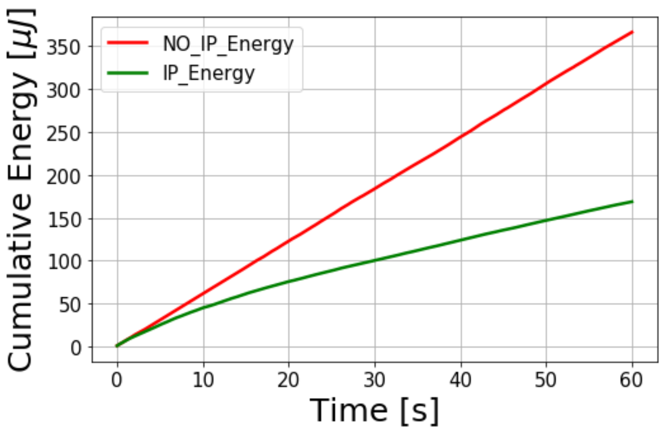


Figure 4.19: The cumulative energy consumed by a recurrent neural network firing at its natural frequency (red) and the same network implementing the described intrinsic plasticity algorithm (green).

4.3 Neuromorphic event routing architecture

Due to the recent success of connectionist approaches to artificial intelligence [35], namely various feats in deep learning [72, 74, 77] whereby large general purpose models are parameterised through backpropagation [37, 51], the capabilities of graphical models - a collection of nodes (neurons) networked together by edges (synapses) - has been well recognised. The fact that animal nervous systems, given hundreds of millions of years of evolution to figure out an effective strategy, have also opted to compute using graphs provides a strong argument for the potential of the approach. Aspects of this were seen in chapter 2 which concerned itself with the development of bio-inspired neural network models. A variety of hardware systems have been proposed in order to run and parameterise different types of neural networks. Graphical [70, 99] and tensor [100] processing units (GPU/TPUs), which are application-specific-integrated-circuits (ASICs) that are optimised to perform large-scale matrix operations, act as the predominant substrate for various types of large-scale deep learning models. Neuromorphic processors, based on analogue [85, 86, 339] or digital [87, 88] circuit models of biologically-plausible neurons and synapses, facilitate the implementation of neural networks with bio-inspired dynamics and topologies: reservoir computing [82, 83] and winner-take-all [224, 225, 346] models for example. Neuromorphic processors are composed of multiple neuron ‘cores’, that feature arrays of synapse and neuron circuit models like those considered in the first section of this chapter. Various implementations of an ‘event routing’ protocol called address event representation (AER) are used as means of realising specific neural network topologies on neuromorphic processors [354]. In AER, each neuron is assigned an address and also has some form of memory that stores the address the neurons which pre-synaptic to it. Whenever a neuron spikes, its address is broadcast on some form of global bus. Each of the neurons which contain this address in their pre-synaptic neuron memory generate, locally, a pre-synaptic spike. So-called ‘flat’ AER uses a global look-up table that sequentially reads and broadcasts address events from several neuromorphic cores, one at a time, onto a global AER bus [355, 354]. Two similar approaches, called broadcast-mesh and hierarchical-fractal AER, extend flat AER by allowing events to ‘hop’ between neighbouring AER nodes [356] and to ‘climb’ up and down a hierarchy of routing nodes [357] respectively. Such a hierarchical approach gains efficiency by exploiting the observation that, in the biological neural networks that neuromorphic processors aim to emulate, neurons connect most heavily amongst their neighbours and very sparsely with distal cells [358, 359]. Router-mesh [360] and multicasting-mesh [106] AER, instead of using a global look-up table, incorporate distributed routers with local memory and digital control circuits which, based on the address of the event input to the router, steer it between neighbour neighbouring routers until it arrives at its intended destination.

As an alternative to such ‘virtual’ AER-based implementations of neural networks whereby digital circuits link up nodes on the fly using volatile memories, hardware exploiting the non-volatility of resistive memory arrays have also been proposed to realise physically connected silicon-based neural network models. Further to storing information on parameters in their non-volatile conductive states, RRAM devices are able serve as a physical, analogue, link between nodes and execute neural network models in-memory as discussed in chapter 3. Arrays of RRAM have been used to implement feed-forward [145, 146] and long-short-term-memory [147, 148] neural network models as well as spiking neural networks [151, 152]. The technology has also been used to realise Hopfield networks [156] and to perform in-memory inference on Bayesian networks [158, 160].

These existing approaches to graphical hardware, however, are each subject to various sets of problems. GPUs suffer from a von Neumann bottleneck between on-chip memory centres and processing units, and therefore expend large sums of energy in the repeated transportation and storage of data. To train and perform inference with deep learning models, a substantial power consumption on the order of hundreds of watts is required [101]. That makes GPUs unsuitable for certain applications, such as those at the edge. Neuromorphic processors consume many orders of magnitude less power, on the order of hundreds of milliwatts [339, 87], but require digital circuitry and volatile memory to manage the routing of address events in an asynchronous manner and are subject to deadlock issues as well as upper-limits on fan-out and event bandwidth [112, 361]. Both GPUs and neuromorphic processors entail a static power draw required by, potentially, large blocks of volatile memory and digital circuits. The use non-volatile resistive memory arrays partly solve these issues but, based on current approaches, come at a heavy price. To implement the dot-product operation which describes how nodes inter-connect with each other through edges, analogue voltages are required to be generated using a digital-to-analog converter (DAC). Then, since the result of this dot-product operation is a current, it must be converted back into a voltage using a current sense amplifier. Then, it must once again be digitised using an analog-to-digital converter (ADC) to provide input for the DACs of the next layer of neurons. The energy penalties incurred due to the DAC, sense amplifier and, in particular, the ADC are generally prohibitive [131]. In RRAM-based neural network inference chips, even those opting for very low-precision conversions, ADCs have been observed to consume up to 80$\%$ to the total system energy budget and require 70$\%$ of the silicon area [132].

This section proposes an alternative RRAM-based non-von Neumann computing fabric for the implementation of neural networks based on distributed systolic matrix of analogue-domain circuits, called 'tiles', which, in a similar fashion to section 4.2, incorporate local resistive memory arrays. The proposed hardware does not require volatile memory, digital routing circuits nor ADCs. One class of tile, the neuron tile, contains analogue neuron and synapse circuit models. Each of these tiles contains a local resistive memory array which determines the pre-synaptic connection weights to each neuron. Neuron tiles propagate output voltage pulses to neighbouring 'routing tiles' which use a further resistive memory array and analogue circuits to determine how these pulses are then propagated to its own neighbouring routing and neuron tiles, and so on. The resulting mosaic of computational and routing tiles not only succeeds in continuously distributing memory and computing homogeneously throughout the hardware, eliminating any von Neumann bottleneck, but it also avoids the use of volatile memories which necessitate a static power draw as well as costly analog-to-digital signal conversion. This is because all computation is performed in the analogue-domain, leveraging the local non-volatile conductance states of RRAM devices.

We first present the circuit elements and system level composition of what we call the 'neuromorphic memory mosaic', or simply 'the mosaic' for short. We then demonstrate how this substrate can be applied to create small-world neural network, or graphical, models [362, 358]. Finally, we demonstrate through spiking neural network simulation how a small-world neural network, based on the neuromorphic memory mosaic, can be applied as reservoir computing model [82, 83] to engineer a static feature space from a input time-series for a simple linear classification model in the heartbeat arrhythmia detection task.

4.3.1 Column circuits

The neuromorphic memory mosaic is based on a fundamental building block referred to here as a 'column circuit'. The purpose of these circuits is to transform input voltage pulses into read currents that depend on the non-volatile conductance states programmed into a plurality of resistive memory devices. These currents can then be used for asynchronous, analogue-domain computation. In the mosaic architecture, these currents are used to update the membrane voltages of neuron circuit models in 'neuron columns' and also to determine if incoming voltage pulses should be blocked or passed through 'routing columns'.

Neuron column

The neuron column, depicted in Fig. 4.20, has at its input N parallel one-transistor-one-resistor (1T1R) structures. Pre-synaptic rectangular voltage pulses, here with a pulse-width of one microsecond, arrive at the selector transistor gate of the 1T1Rs in an asynchronous fashion. The common bottom electrode has a constant DC voltage V_{bot} applied to it and the common top electrode is driven to the voltage V_x by an operational amplifier (OPAMP) circuit. The OPAMP output is connected in negative feedback to its non-inverting input (due to the 90°phase-shift between the gate and drain of transistor M_1 in Fig. 4.20) and has the constant DC bias voltage V_{top} applied to its inverting input. As a result, the output of the OPAMP will modulate the gate voltage of transistor M_1 such that the current it sources onto the node V_x will maintain its voltage as close as possible to the DC bias V_{top}. Whenever an input pulse $V_{in} < n >$ arrives, a current i_{in} equal to $(V_x - V_{bot})G_n$ will flow out of the bottom electrode. The negative feedback of the OPAMP will then act to ensure that $V_x = V_{top}$, by sourcing an equal current from transistor M_1. By connecting the OPAMP output to the gate of transistor M_2, a current equal to i_{in}, will therefore also be buffered, as i_{buff}, into the branch composed of transistors M_2 and M_3 in series. A transformation is then be applied to i_{buff} by a synaptic circuit model (here, the circuit in Appendix 6.10a). In this case, like the synapse circuit of Fig. 4.6a, it generates an exponentially decaying current that is in turn in injected onto the membrane capacitor of a leaky-integrate and fire neuron model (here, the circuit in Appendix 6.9a). In Fig. 4.20, the results of a SPICE simulation accompany the circuit schematic as three insets where all three devices have a conductance of 250μS - representative of an OxRAM device in the HCS. It can be seen that, resulting from the three input pulses, the membrane voltage, V_{mem}, of the neuron circuit then increases twice after two synaptic current injections before firing upon the arrival of a final third pulse. Note that inhibitory pre-synaptic weights are possible by using an additional column structure feeding into a synapse circuit which, instead of injecting a current via PMOS current mirror into the neuron circuit, sinks current from the neuron by connecting the output of the synapse circuit to an NMOS current mirror. Such 'differential' column structures have been designed and laid out for fabrication.

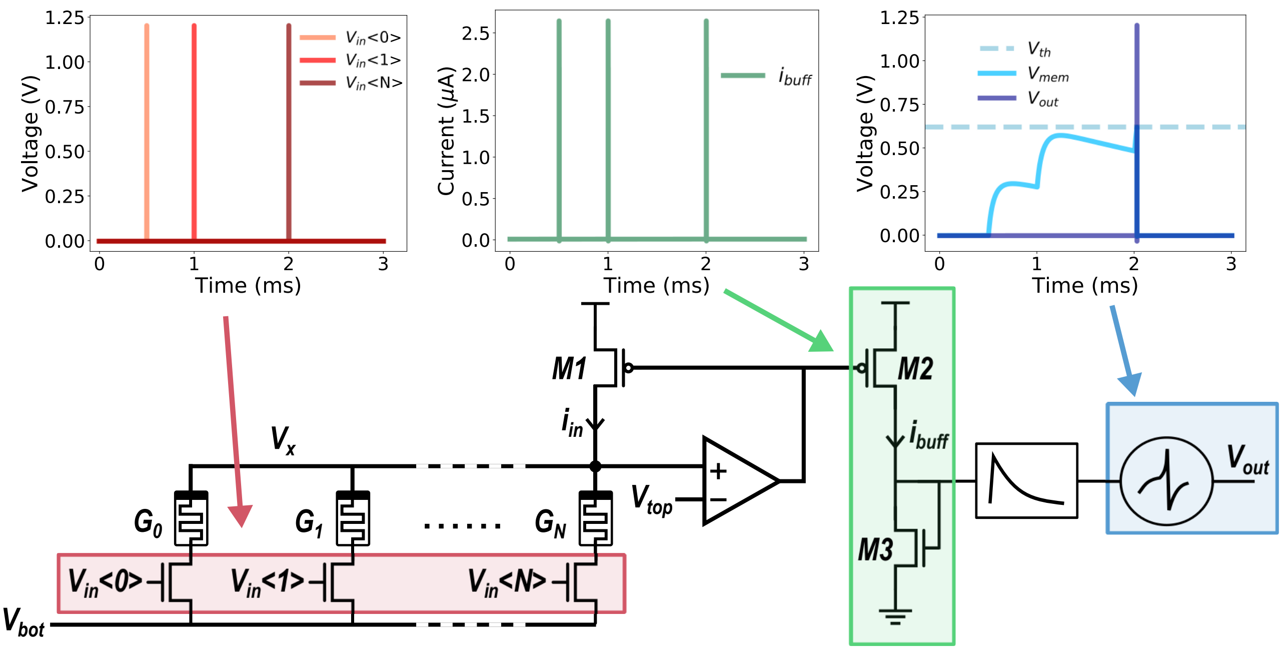


Figure 4.20: **The neuron column circuit with example waveforms.** Input (red, left) voltage pulses, V_{in}, draw a current i_{in} proportional to the conductance state, G_n, of the read 1T1R structures. This current is buffered (green, centre), i_{buff}, into a synapse circuit model which applies a transform on it and in turn injects it into a neuron circuit model. The neuron circuit integrates this current into a membrane voltage (blue, right), V_{mem} which causes the neuron to fire at the output after exceeding a threshold V_{th}.

The configurable conductance state of the resistive memory devices, G, in the column determines magnitude of i_{buff} that is mirrored into the synapse circuit model. As discussed throughout chapter 3, the conductance of OxRAM devices in the high conductive state (HCS) is determined by the SET programming current. As another example of this, the cumulative probability distribution of all of the devices in the 4k device array (see Appendix 6.3) shown due to a single SET operation under five sets of programming conditions is plotted in Fig. 4.21a. The largest values of the high conductance state in this array are larger than those of the 16k device array, used throughout chapter 3, reflecting the larger selector transistor of the devices in the 4k device array which allow larger SET programming currents to be obtained. The spread of these CDFs indicate the single cycle device-to-device conductance variability that exists within a population.

In a fabricated circuit implementation of the neuron column in Fig.4.20, the device G_0 in the column was programmed over a sweep of compliance currents that resulted in a series of conductances. After each device had been programmed, this conductance value was recorded and then an input pulse was applied to $V_{in} < 0 >$ resulting in a current waveform with a peak proportional to the programmed conductance state being injected onto the neuron membrane capacitor. The resulting voltage waveform was measured (using the OPAMP shown in Fig.6.4a in a voltage buffer configuration that had been integrated with the circuit). Each of these voltage waveforms, due to six respective conductances, are superimposed and plotted in Fig.4.21b. It can be clearly seen as the value of conductance increases, so too does the size of the integrated voltage waveform - serving well as a programmable synaptic weight parameter. The layout of a column circuit can be seen in Appendix 6.13a.

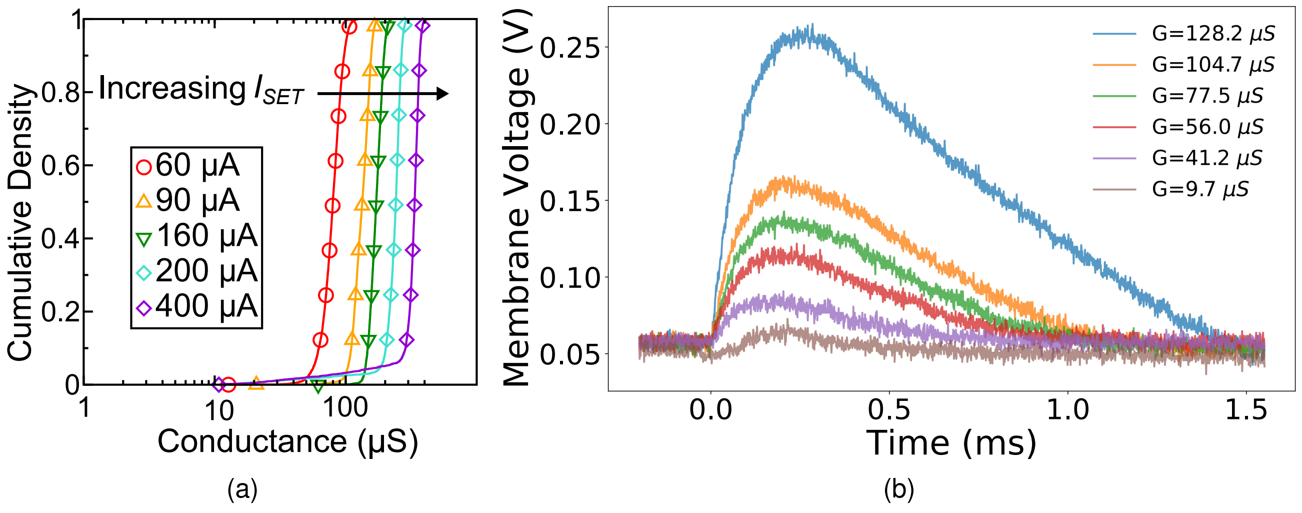


Figure 4.21: (a) Five cumulative distributions resulting from the application of a single SET programming pulse on each device in an array of 4096 OxRAM devices over a range of SET programming currents, I_{SET}. (b) Measured voltage traces from a fabricated neuron column circuit due to an input voltage pulse to the column. From an initial resting membrane voltage of 0.05V, the membrane voltage waveform obtained due to increasing series of conductance values in the neuron column circuit is shown.

Routing column

The routing column, depicted in Fig. 4.22, works largely on the same set of principles as the neuron column. An OPAMP connected in the same negative feedback configuration mirrors the current i_{buff} into the circuit drawn at the right-hand side of Fig. 4.22 upon the arrival on an input pulse to one of the 1T1R structure gates. This circuit compares i_{buff} with a DC bias current i_{ref} using a pair of ‘back-to-back’ current mirrors. Intuitively the two current mirrors compete to either pull the common node, that at the input of the inverter, up to vdd (due to the two PMOS) or down to gnd (due to the two NMOS). If i_{ref} is greater than i_{buff}, the inverter will output a low voltage and, on the contrary, a high voltage. A set of graphs in the inset of Fig.4.22 plot the results of a SPICE simulation of the routing column. The devices read by the first and final input voltage pulses are assigned an HCS conductance of 250μS while the remaining device, that is read upon the arrival of the second pulse, has an LCS conductance of 2μS. The magnitude of the corresponding buffered current due to each pulse, the green pulses in the central figure inset, changes accordingly whereby the buffered current due to the second pulse is negligible relative to the first and final ones. This results in only the first and final input pulses passing through the routing column, therein generating a voltage pulse at its output, while the second pulse is blocked.

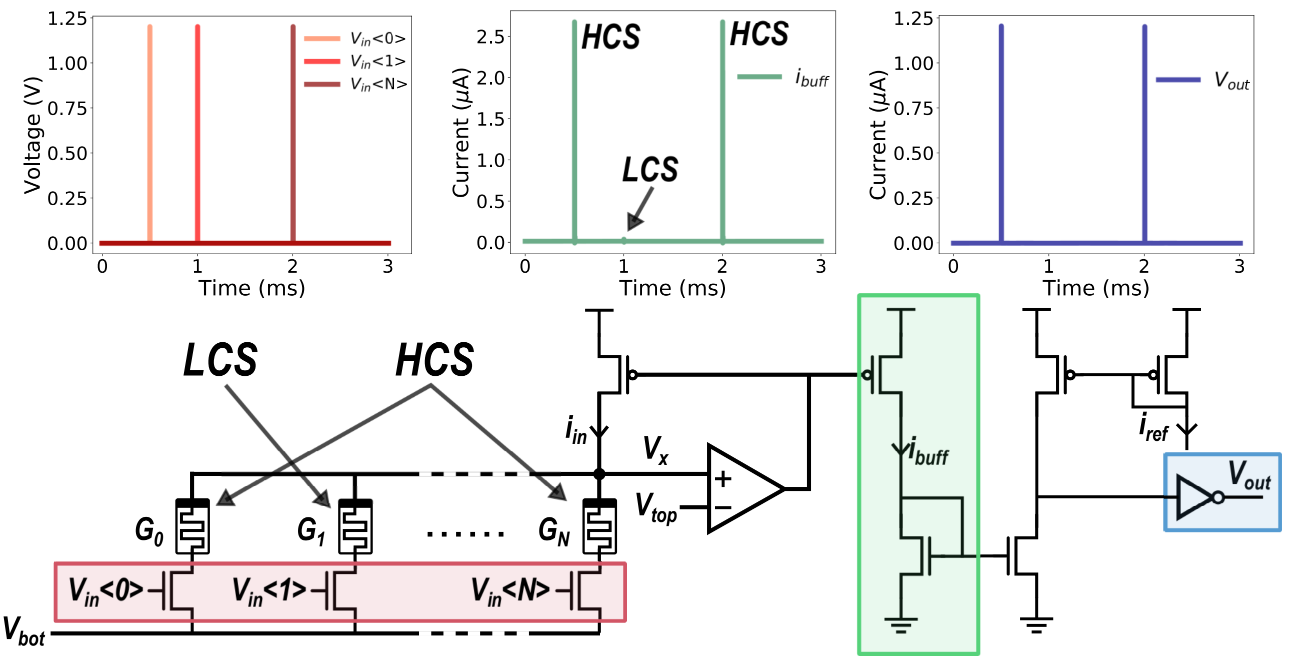
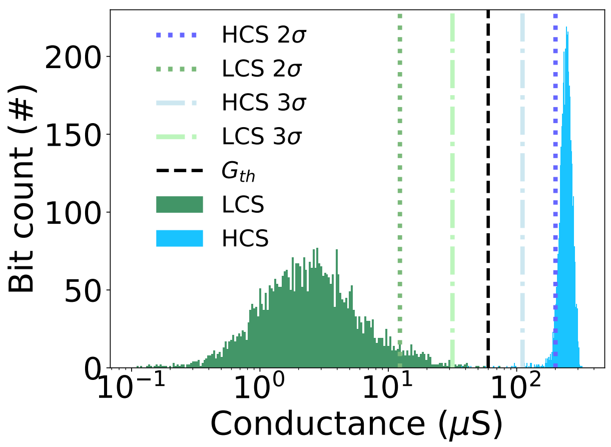
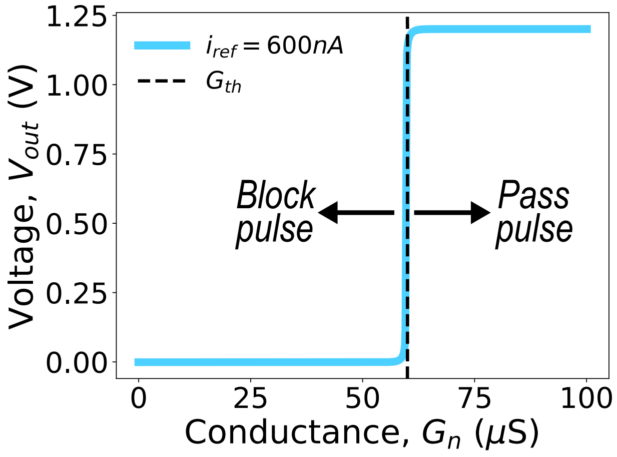


Figure 4.22: **The routing column circuit with example waveforms.** Input (red, left) voltage pulses, V_{in}, draw a current i_{in} proportional to the conductance state, G_n, of the read 1T1R structures. Two devices are labelled with HCS, indicating that they have been programmed with a conductance corresponding to the high conductance state, and one is labelled LCS in reference to the low conductance state. This resulting currents are buffered (green, centre), i_{buff}, into a current comparator circuit where it is compared with a reference current i_{ref}. When the buffered current exceeds the reference current a voltage pulse is generated at the column output (blue, right).

Since the conductance state of a device that is read upon the arrival of an input pulse determines i_{buff}, it can then be used to store in a non-volatile fashion information on what pre-synaptic voltage pulses should propagate through a routing column, and which should not. The HCS and LCS bit count distributions from the 4k device array (see Appendix 6.3), resulting from single SET and RESET operations on each device in the array are plotted in the histogram in Fig. 4.23a. Each distribution is discretised in one hundred conductance bins and the number of bits in each bin is plotted on the y-axis. This allows both the HCS and LCS distributions to viewed on the same scale which would not be possible if the distributions were plotted in terms of probability density. Vertical dashed lines in Fig. 4.23a show the large 'memory window' between the LCS and HCS distributions at two and three standard deviations. This thereby demonstrates that by programming all devices under the same programming conditions, devices in the HCS and LCS can be well distinguished within a population with a negligible bit error rate. An additional vertical dashed line shows a conductance approximately half-way between the LCS and HCS distributions - a value that could be used as a conductance threshold to separate HCS and LCS bits which define whether a pulse should be respectively passed or blocked. In Fig. 4.23b, a SPICE simulation of a DC sweep of the conductance state of an RRAM device at the input of a routing column which is subject to an input pulse, is swept from 1μS to 100μS. This conductance range spans the upper and lower tails of the LCS and HCS distributions plotted in Fig. 4.23a. The threshold bias current, i_{ref}, is set equal to 600nA - corresponding to the half-way conductance denoted by the vertical line in Fig. 4.23a. It can be seen that as the conductance approaches this threshold, the output voltage of the comparator abruptly swings up from gnd to vdd and therefore the non-volatile LCS and HCS conductance states, even in the presence of measured device-to-device variability, can be used to reliably block or pass incoming voltage pulses.



(a)



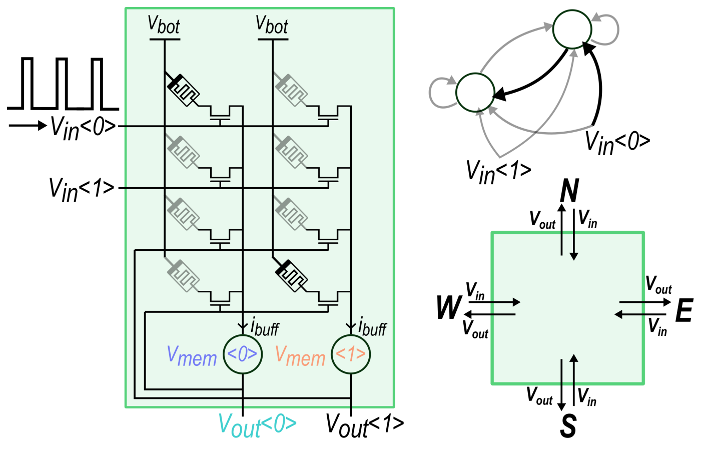
(b)

Figure 4.23: (a) The bit count histograms of the low (green) and high (blue) resistive states of 4096 devices. Vertical dashed lines show the memory window at two (darker green and blue) and three (lighter green and blue) standard deviations (labelled in the figure legend as 2/3σ). A black vertical dashed line shows the conductance approximately half-way between the two distributions. (b) The results of a DC SPICE simulation whereby the conductance of a resistive memory device at the input of a routing column circuit was swept between 1μS to 100μS. A black dashed vertical shows the threshold at which the output of routing column switches from low to high due to a threshold reference current, $i_{ref} = 600nA$. This reference current is set equal to the current that flows due column device conductance equal to that of the vertical black line drawn in part (a).

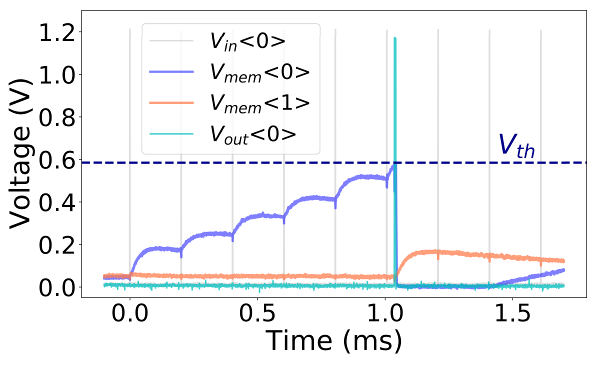
4.3.2 Neuron and routing tiles

The presented neuron column realises a bio-inspired spiking neuron model and the routing column offers a means of blocking or passing the voltage pulses generated at the output of this neuron model. These columns can be respectively agglomerated into neuron and routing 'tiles' by stacking consecutive columns side-by-side and connecting their gates, row-wise, to common input lines. A simple neuron tile, composed of only two neuron columns and receiving two inputs, is shown for means of illustration in Fig. 4.24a. This neuron tile includes two additional rows of RRAM which receive input from the output of each of the neurons in the tile - thereby permitting local recurrent synaptic connections within the tile. Effectively the conductive states of the devices define the synaptic weights of a feed-forward layer of synapses from $V_{in} < 0 >$ and $V_{in} < 1 >$ while the extra two rows define the weights of recurrent connections within the tile as well as the weights of any self-loops that neurons make with themselves. Echoing ideas from FPGAs [95] and mesh AER schemes [360, 106], each of these input and output voltage pulses can enter from, and exit towards, the neighbouring north (N), south (S), east (E) and west (W).

Experimental results measured from a fabricated implementation of the neuron tile circuit are plotted in Fig.4.24b. In the experiment the two devices coloured in black in Fig.4.24a were SET into the high conductive state whilst the grey shaded devices were programmed in the low conductive state - resulting in the simple network depicted in the top right panel of Fig.4.24a. A train of input voltage pulses were applied to $V_{in} < 0 >$ which is observed to periodically increase the membrane voltage. After six input pulses this membrane voltage exceeds the threshold, V_{th}, and subsequently fires an output pulse. Since a recurrent feedback connection exists in the programmed topology (from neuron 0 to neuron 1) the neuron in the other column also integrates a voltage on its membrane - despite having received no external input signal.



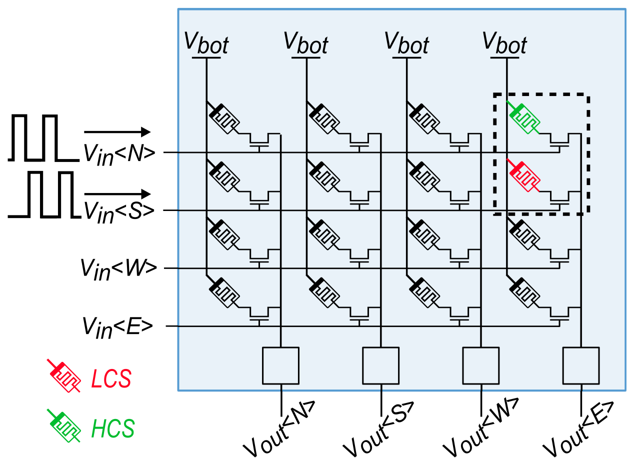
(a)



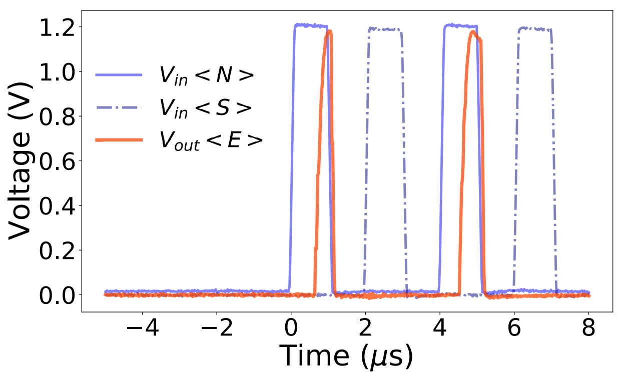
(b)

Figure 4.24: (a) **Diagrams of the neuron tile**. (left) Two stacked neuron columns realise a neuron tile. Four of the devices (top of the array) define the synaptic connections from inputs $V_{in} < 0 >$ and $V_{in} < 1 >$ to the two neurons and an additional four devices (bottom of the array) define the recurrent connections between neurons and neuron self-loops. Devices are coloured in black or grey to indicate respectively whether they were in the high or low conductive state respectively during the experimental results plotted in part (b). V_{bot} corresponds to the voltage of the same name in Fig. 4.20 and i_{buff} shows the direction of the buffered current that is mirrored into the synapse and neuron circuits which are represented by open circles (top right). This tile circuit permits the configuration of the two neuron, two input network shown here. (bottom right) These input and output voltage pulses can come from and be propagated to four neighbouring tiles to the north (N), south (S), east (E) and west (W). (b) Voltage traces measured from a fabricated neuron tile circuit. Due to an input pulse train (grey pulses) at $V_{in} < 0 >$ the membrane of the zeroth neuron column in the tile integrates an increasing amount of voltage (purple trace) until, after six pulses, the neuron fires (light blue trace). As a result of the feedback connection to the other neuron column, it then also exhibits an increase in membrane voltage.

In order to route these pulses generated by the neurons to their neighbouring neuron tiles we require a further tile, based on the routing column circuit, whose constituent devices will direct the flow of the pulses. Such a routing tile, capable of passing and blocking pulses coming from two neurons to and from the N, S, W and E neighbours is shown in Fig. 4.25a. An experiment was performed using a fabricated version of this routing tile where two devices (coloured in green and red in Fig.4.25a) were programmed in respective high and low conductive states. The other devices were left in the pristine state. This has the effect of allowing incoming pulses from the North to propagate out to the East, while blocking pulses coming from the South. Pairs of pulses were applied to the North and South tile inputs and, as plotted in Fig.4.25b, an output pulse of observed at the East when input pulses arrive to the North while it remains clamped at gnd for the equivalent pulses arriving from the South. Note that the output pulse does not appear as rectangular due the fact that the output of the circuit is required to charge a pad on the probing station, through which the experiment is conducted, that has a capacitance of some tens of pico Farads. If this output were connected to the gate of another 1T1R structure on a subsequent routing or neuron tile the rising and falling edges of the pulse would be sharper.



(a)



(b)

Figure 4.25: (a) Circuit schematic of a routing tile. Two devices coloured green and red denote respectively the devices programmed in the high and low conductance states in the experiment of part (b). Rectangular pulse waveforms depicted the left-hand side indicate where the input voltage pulses were applied during this experiment. (b) Experimental results from a fabricated version of the routing tile shown in part (a). Continuous and dashed blue traces show the waveforms applied to the North and South inputs while the orange trace shows the response of the output towards the East. The Eastern output follows the Northern input resulting from the device programmed into the high conductance state in part (a).

The routing tile shown in Fig.4.25a can pass only one pulse per direction. However, in order to enable greater configurability it is desirable to propagate different pulses that originate from difference sources but travel in the same direction differently. This can be achieved by adding additional ‘routing channels’ per direction. An example of a routing tile with two channels per direction is shown in Fig. 4.26. It is based on eight stacked routing columns, culminating in an 8×8 array of RRAM-devices. Whether input pulses propagate or not between all possible combinations of neighbouring tiles is determined by the conductive state of a corresponding device in the array. An example of how it can be configured is shown in Fig. 4.26, whereby a voltage pulse originating from the tile's southerly neighbour, $V_{in} < 1, S >$, is blocked by devices in the LCS in six of the routing columns and passed in two columns which have their devices in the HCS - generating output voltage pulses $V_{out} < 1, N >$ and $V_{out} < 1, W >$ that propagate pulses to the respective north and westerly neighbours through the additional channel. Evidently, it is possible to use any number of routing channels per direction to increase configurability as desired; although the memory requirements double for each additional routing column that is used per direction (requiring a 12×12 array for three columns per direction and a $16\times\ 16$ array for four). The number of neuron and routing columns per tile determines the trade-off between configurability and efficiency as well as the prevalence of two potential issues of the mosaic approach - 'path sharing' and 'lost pulses'. The former results when too few routing columns are used per direction and neurons on the same tile are obliged to share the same set of post-synaptic weights and connections. The latter results when pulses originating from two different pre-synaptic sources overlap at the inputs of routing or neuron columns and effectively merge into a single pulse instead of arriving to the destination neuron as two pulses.

These two tile types can be pieced together in the pattern depicted in Fig. 4.27a which gives rise to a continuous mosaic of neuromorphic computation and memory. Each of the green squares correspond to a neuron tile and each of the blue ones to a routing tile. The neuron tiles receive input from their neighbouring four routing tiles, update their membrane voltages using the synaptic currents that are generated as a function of the conductive states of the devices in their columns and then, upon exceeding their spiking threshold voltages, emit voltage pulses at their outputs. These pulses are then picked up by their neighbouring routing tiles, those that delivered their input, and then steered onward to their destinations as a function of the non-volatile conductance states of the devices in the blue sea of routing tiles that link up neurons in the graph. An example neural network topology, obtained by randomly programming devices in a computer model of the mosaic to be in the HCS with probabilities $p_r = 0.075$ and $p_n = 0.4$, in the routing and neuron tiles respectively, is shown in Fig. 4.27b. The resulting graph exhibits an intriguing set of connection patterns that strike resemblances to many of the graphical motifs observed in animal nervous systems such as central 'hub-like' neurons that send and receive synapses from numerous nodes, reciprocal connections between pairs of nodes reminiscent of winner-take-all mechanisms, as well a number of heavily connected local neural clusters [363]. The sixteen (green) neuron tiles in this mosaic model contain four neuron columns each and the forty-eight (blue) routing tiles each comport sixteen routing columns.

The mosaic is evidently better suited for neural network architectures where neurons connect most frequently with physically nearby neurons and sparsely with distal ones. Models such as feed-forward neural networks for example would be impossible or require a large extent of router path sharing. Rather, since the hardware shares the same set of spatial constraints as biological nervous systems, it is much more amenable to implement bio-inspired topologies like those considered in chapter 2 - those that naturally exhibit 'small-world' properties [362, 363]. The use of locally-connected and modular small world networks is frequently employed by biological nervous systems as an adaptation in light of the similar spatial constraints faced by this silicon-based hardware [358, 359].

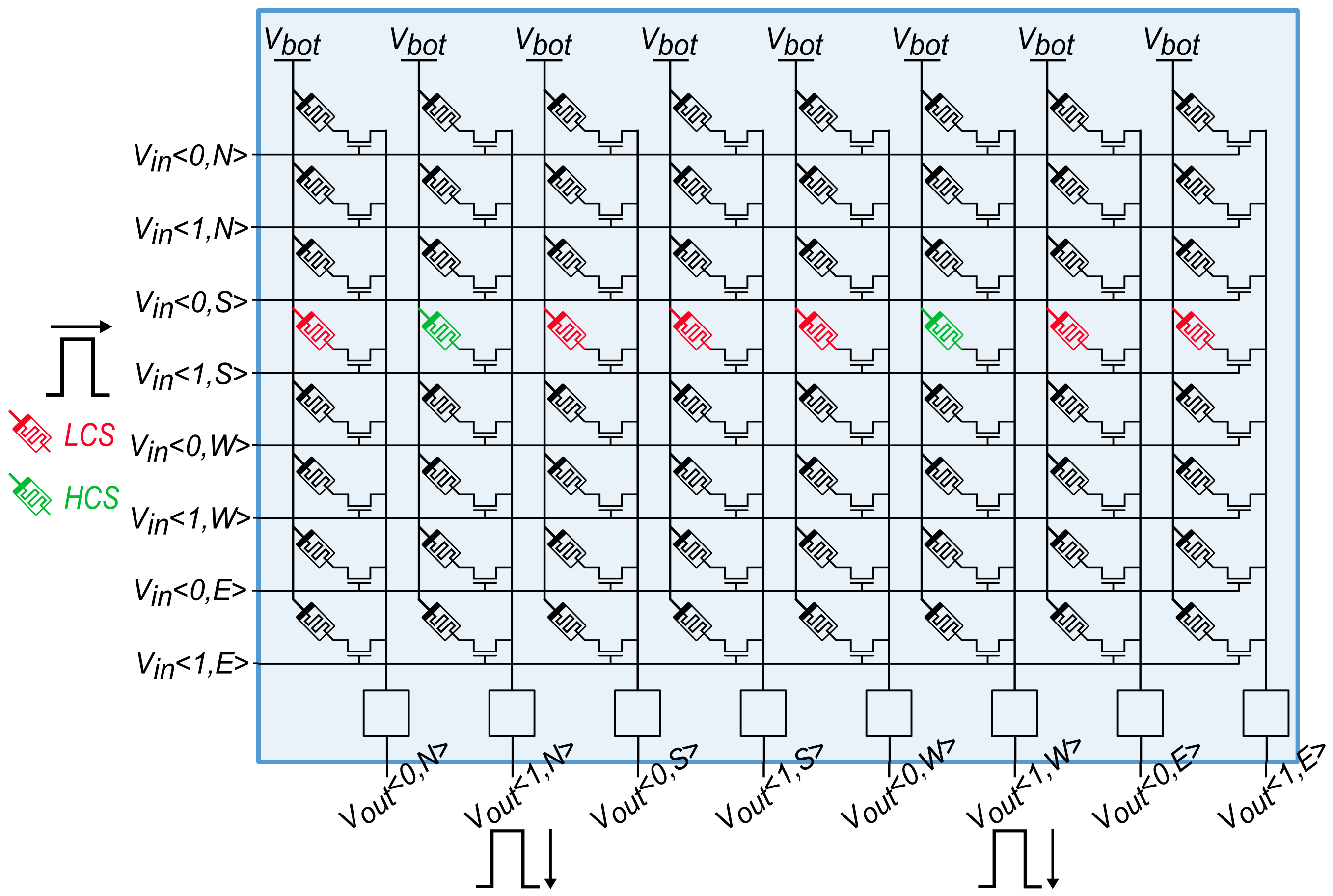


Figure 4.26: Schematic for a routing tile composed of eight routing columns. The routing tile receives eight inputs, comprising two pulse channels per direction, labelled as $< 0 >$ or $< 1 >$, from the neighbouring tiles to the north (N), south (S), east (E) and west (W), and provides complimentary outputs. An example is shown of an input pulse arriving to the common gate of the fourth row of memory. Devices are coloured green or red to denote whether they are in the HCS or LCS. It it shown that, due to this input pulse, output pulses are produced by the routing columns containing the (green) devices programmed in the HCS. V_{bot} denotes the DC bias of the same name in Fig. 4.22 and open rectangles are used to denote the current comparator circuit at the end of the routing column.

4.3.3 Application to reservoir computing

The field of reservoir computing [82, 83] is an appealing use case for neuromorphic processors since it requires 'static' randomly connected small-world neural networks whose synapses are not required to be modified by on-chip adaptation circuits which can be costly in terms of memory, silicon area and energy [86, 339, 361]. Reservoir computing models are characterised by the random connection of neurons amongst their neighbours with an exponentially decaying spatial connectivity probability kernel - meaning that neighbouring neurons have a higher probability of synapsing onto each other than neurons which are further apart [83]. Reservoir computing models are understood to perform a temporal projection that integrates information of an input time-series into the collective instantaneous activations of the reservoirs neurons. At the end of an input time-series, the collective low-pass filtered spike-trains of the neurons, corresponding to the calcium concentration that encodes instantaneous firing-rate in biological neurons

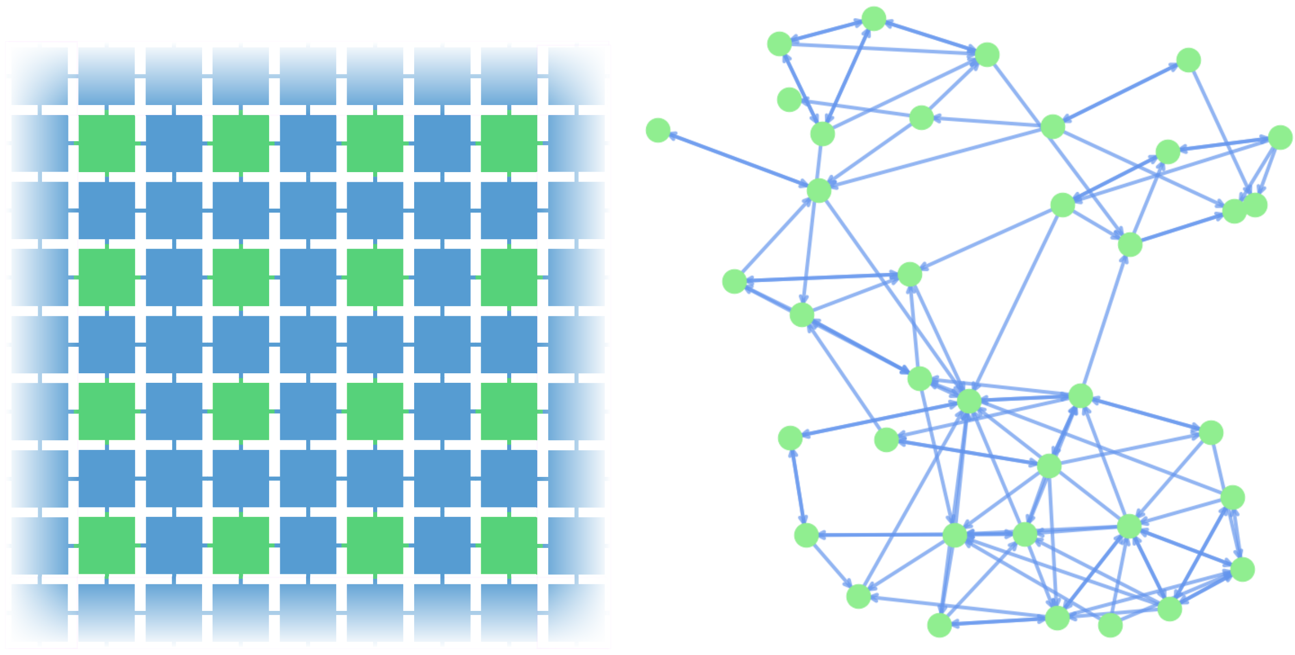

(a) (b)

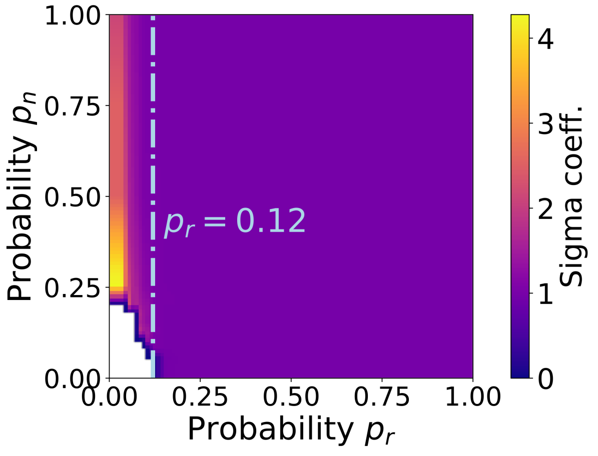
Figure 4.27: (a) The neuromorphic memory mosaic. Green squares correspond to neuron tiles and blue squares to routing tiles. The bridges drawn between tiles correspond to the north, south, east, west signal buses carrying the V_{in} and V_{out} voltage pulses. (b) An example graph resulting from the random programming of devices in each of the tiles in the mosaic pictured in part (a). The green circles correspond to neurons which exist in the neuron tiles and the blue edges are defined by the resulting paths that are formed between neuron tiles through the routing tiles. The arrows on the end of each of the edges denote their direction.

[364, 365], can be used as input features for a machine learning classification model. In neuromorphic processors, limitations in the fan-in/out and configurability of the reservoir computing model often impose certain constraints on the small-world graph properties that can be achieved [112, 361]. Here we demonstrate an application of the mosaic to reservoir computing by (as in Fig. 4.27b), randomly programming the devices in routing tiles and neuron tiles with probabilities p_r and p_n.

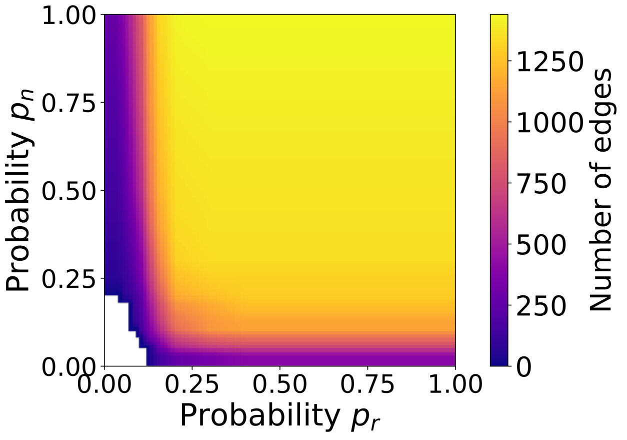
4.3.4 Small world graph properties

Small-world neural networks are characterised by two graphical properties; the clustering coefficient and the average path length that respectively describe the tendency for nodes to inter-connect in localised clusters and how few intermediate nodes must be traversed on average to exchange information between a pair of nodes [363]. Typically, a high clustering coefficient and a short average path length are required for a graph to be considered as small-world. The so-called 'sigma' [366, 367] coefficient is often to used a means of determining whether a graph is a small-world or not. The sigma coefficient is a number proportional to the ratio of the clustering coefficient of the graph and the clustering coefficient of a random network or equivalent size divided by the ratio of the average shortest path length of the graph and of the same random network. The greater the value of sigma above one, the more the graph resembles a small-world.

To investigate the impact of the conductance states (i.e., LCS or HCS) of devices in the neuron and routing tiles on these small-world properties, we randomly program devices in a mosaic model of sixteen neuron tiles, of four neuron columns each, and thirty-eight routing tiles, of sixteen columns each, over a sweep of p_n and p_r. In Fig. 4.28a a heatmap is plotted that shows how the sigma coefficient varies as these two probabilities are varied. What is apparent is that the most important parameter to obtain a graph that exhibits strong small-world characteristics is the probability of routing tile devices being in the HCS, p_r. It is seen that for any value above 0.12 the resulting mosaic-based neural network is not a small-world. This should not be wholly surprising however, given the large number of possible paths that can exist between neurons through a multitude of different routing tile combinations. Therefore, given a moderate fraction of routing tile devices in the HCS, here between 3% and 12%, a variety of complex networks with rich graphical properties can be realised. The number of edges, or synapses between neurons, over the same range of probabilities is plotted in Fig. 4.28b where the probability of the routing tile devices being in the HCS is seen to have a similar effect. For a p_r greater than 0.12, the number of edges in the graph rapidly explodes and tends towards a maximum value of 1296 edges.



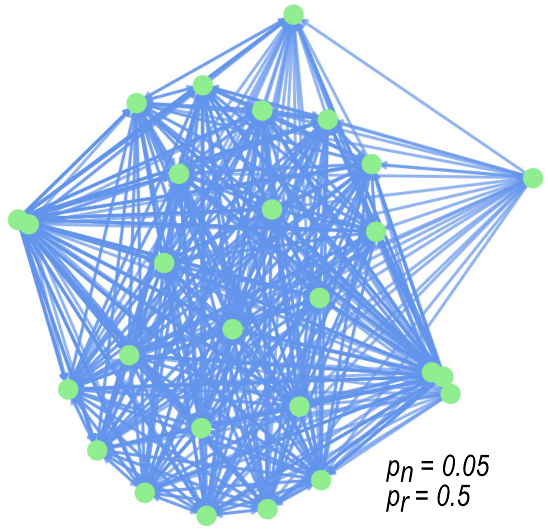
(a)



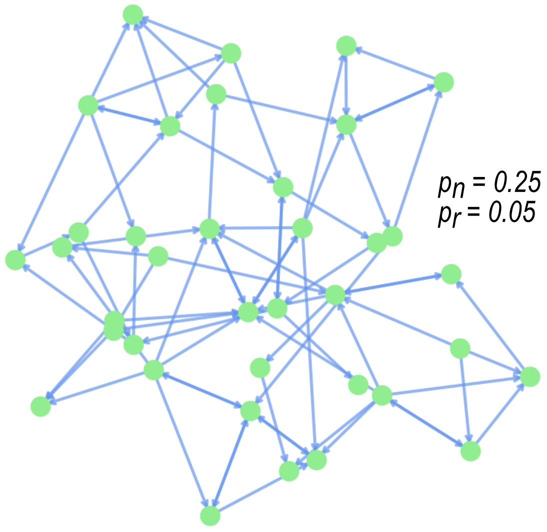
(b)

Figure 4.28: **Heatmaps of a two-dimensional sweep of p_n and p_r** White sections of the heatmaps correspond to graphs where there existed isolated, unconnected neurons and therefore it was not possible to calculate a small-world coefficient. (a) Heatmap of a sigma coefficient that describes the small-worldness of the resulting network. A vertical blue line shows the point at which point ($p_r = 0.12$) the graph stops being small-world. (b) Heatmap showing the number of edges that exist in a graph.

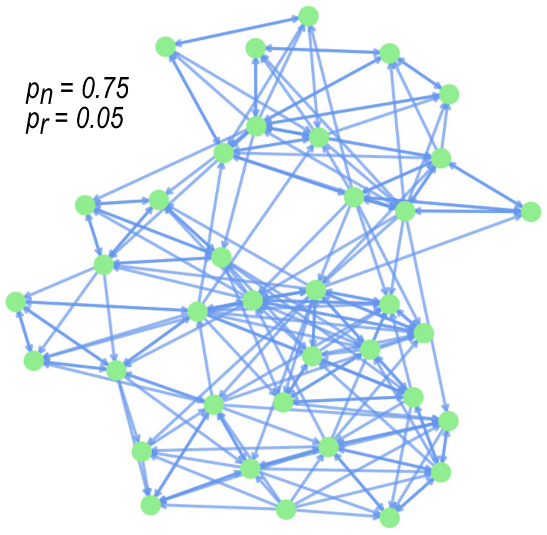
In Fig. 4.29, four example neural networks over a range of HCS probabilities for the devices in the neuron and routing tiles being in the HCS are shown. The network in Fig. 4.29a is formed in the mosaic with a high value of p_r and, even with a low value of p_n, the resulting graph resembles an all-to-all recurrent neural network topology. The other three neural networks shown in Figs. 4.29b, 4.29c and 4.29d all use a value of p_r below the small-world threshold of 0.12 and all therefore exhibit different visual characteristics of small-world graphs.



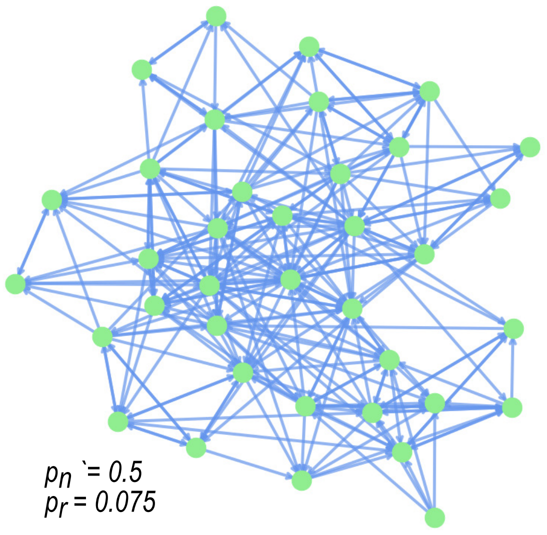
(a)



(b)



(c)



(d)

Figure 4.29: **Four examples of graphs resulting from different combinations of p_r and p_n.** (a) Graph resulting from $p_r = 0.5$ and $p_n = 0.05$. (b) Graph resulting from $p_r = 0.05$ and $p_n = 0.25$. (c) Graph resulting from $p_r = 0.05$ and $p_n = 0.75$. (d) Graph resulting from $p_r = 0.075$ and $p_n = 0.5$.

4.3.5 Heartbeat arrhythmia detection

We now apply a mosaic-based small-world spiking neural network (similar to that in Fig. 4.29d), therein a reservoir computing model, to engineer an input feature space for a perceptron that is trained to detect arrhythmic heartbeats. Since the neuron columns require voltage pulses as inputs, we propose to 'delta-modulate' the two-channel electrocardiogram recordings of recorded heartbeats (see Appendix 6.6) into a series of 'UP' and 'DN' voltage pulses using a delta-modulator circuit [368, 369]. The delta-modulation circuit designed in this work is that shown in Fig. 4.30. For an input voltage waveform, V_{in}, which in Fig. 4.30 is a sinusoid, capacitor $C1$ acts as a high-pass filter and removes the DC component of the waveform. The AC component of the waveform is superimposed upon a bias voltage V_{ref} that charges the non-inverting terminal of the OPAMP A_1, whenever transistor M_1 is switched on. Since A_1 is connected in a negative feedback configuration, the OPAMP acts to buffer the voltage waveform at the non-inverting terminal to its output V_x while unloading the input. Two OPAMPS, A_2 and A_3, configured as open-loop voltage comparators compare this buffered voltage with the DC thresholds V_{hi} and V_{lo}. Whenever V_x is greater than V_{hi} an 'UP' event voltage pulse is generated at V_{up} and the contrary for V_{dn}. These UP and DN pulses feed back, on the left-hand side of the schematic in Fig. 4.30, to the input of a logical OR gate that, whenever an event is generated, discharges capacitor C_2 and thereby switches the inverter that turns on transistor M_1 and resets the voltage at the non-inverting terminal of the OPAMP to V_{ref}. This brings to an end the output event pulse. The non-inverting terminal of A_1 remains pinned to V_{ref} for as long as it takes capacitor C_2 to charge beyond the switching voltage of the inverter through the transistor that is biased with the DC voltage V_{freq} at its gate. The voltage V_{freq} therefore controls the maximum rate of event generation - somewhat similar to the refractory period in a neuron circuit model. The layout of this delta-modulator circuit can be seen in Appendix 6.11a.

The results of a SPICE simulation accompany the circuit schematic as insets in Fig. 4.30. At the input it can be seen how, as V_{in} varies, the voltage V_x follows it on the same trajectory although being repeatedly, and abruptly, reset to V_{ref} upon exceeding the threshold biases V_{hi} or V_{lo} - resulting in the 'chopped' dark red waveform in the left-most inset in Fig. 4.30. This results in successive waves of UP and DN events at V_{in} and V_{out} whose inter-pulse timing describes the rate at which the envelope of the input sinusoid changes. In this SPICE simulation, these event pulses are input to the gates of two selector transistors in a neuron column. The RRAM device that is read upon UP events has a conductance of 100μS and that on DN events, 67μS. In Fig 4.30 it can be seen that as the synapse circuit model injects proportional synaptic currents onto the membrane capacitor of the neuron circuit model, V_{mem} increases until it reaches the threshold voltage of the neuron and triggers an output spike.

This demonstrates how such an interfacing scheme can be used to feed a mosaic based reservoir computing model with electrocardiogram waveforms of heartbeats. In this fashion, heartbeats, such that shown in Fig. 4.31, are converted into trains of voltage pulses that describe upward and downward changes of the signal that can then be fed to neurons within the mosaic-based model.

A simulation of a reservoir computing model realised using the mosaic architecture is then used to generate 2700 static data-points for each of 2700 ECG recordings from a single patient (patient 200) from the MIT-BIH heart arrhythmia database [296]. The patient exhibits approximately half normal and half arrhythmic beats and therefore offers a balanced dataset. The mosaic (as in Fig. 4.27a) is realised using sixteen neuron tiles with four neuron columns and forty-eight routing tiles with sixteen columns. Devices in neuron tiles are programmed in the HCS with $p_n = 0.35$ and in the routing tiles with $p_r = 0.07$. Assuming a differential neuron column is used, the device in the excitatory column has a 0.4 probability of being in the LCS and the corresponding device in the inhibitory column a 0.6 probability of being in the LCS - therefore 40% of synapses in the model are inhibitory and 60% are excitatory. Each of the neurons is equipped with a low-pass filter circuit [338] which gives rise to thirty-six continuously evolving variables which, together, integrate information during each 700ms long ECG recording. The thirty-six instantaneous low-pass filter

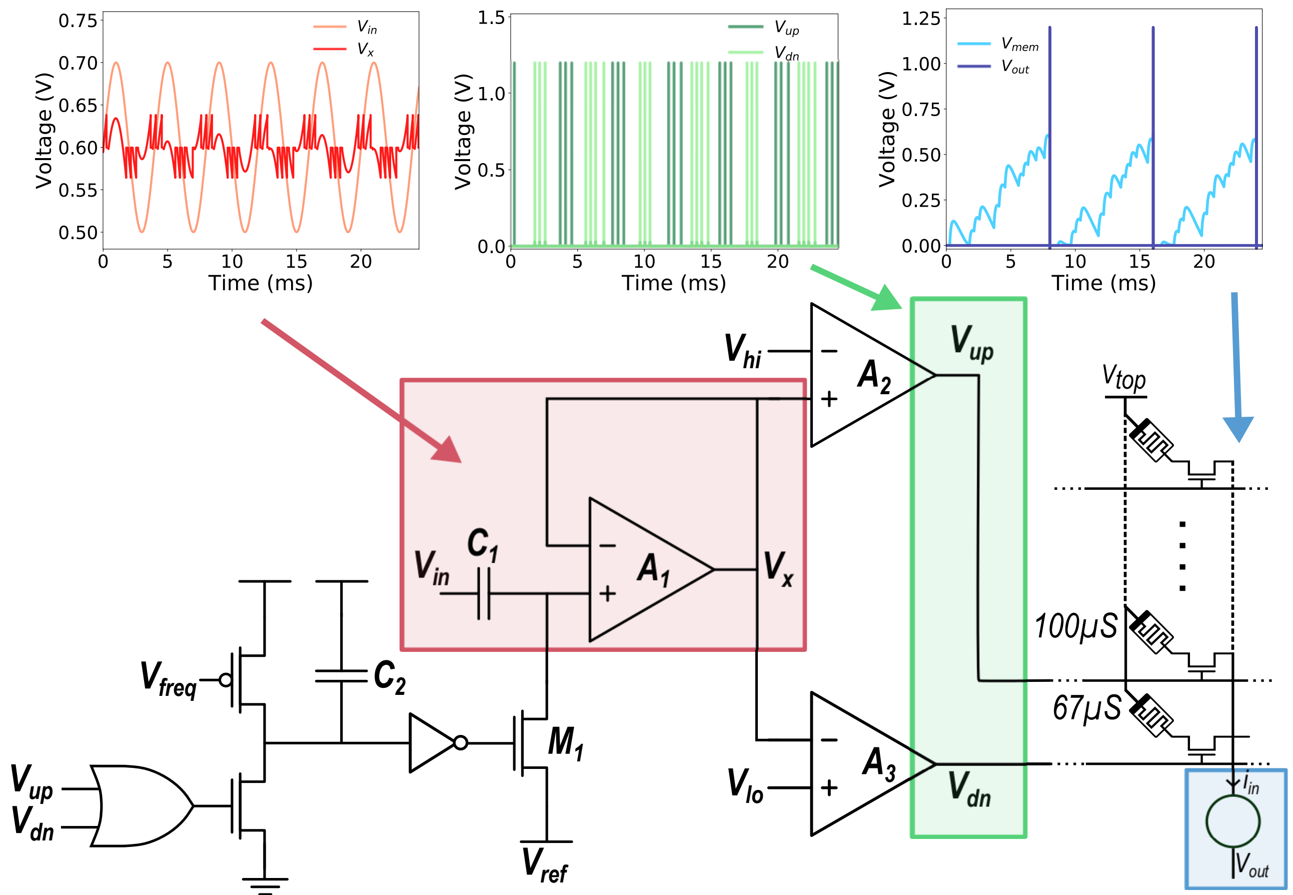


Figure 4.30: Schematic and corresponding SPICE waveforms of a delta-modulator circuit interfaced to a neuron column. Insets show (red) the input sinusoidal waveform and the resulting 'chopped' waveform at node V_x, (green) the resulting UP and DN events at nodes V_{up} and V_{dn} and (blue) the membrane and spiking output voltage resulting from the events generated by the input sinusoid.

output voltages at the end of the simulation are taken as a static data-point describing each heartbeat. The full dataset is therefore composed of 2700 data-points of thirty-six features.

The first two principal components [370] for each data-point in the resulting data-set are plotted in Fig. 4.32a. Normal heartbeats are labelled as green points and arrhythmic ones as orange points. It is encouraging that normal and arrhythmic heartbeats form distinct green and orange clouds of points indicating that, at the end of the simulation, the integrated representation of the time-series manifests in two distinctive types of activation patterns that could potentially be well distinguished by a linear classifier. We train a perceptron model, effectively linking up each up each of the thirty-six input features with two output class encoding neurons (i.e., normal or arrhythmic), with a categorical cross-entropy loss and a training split of 67$\%$ of the full dataset. The resulting model is then tested on the remaining 33$\%$. As plotted in the green boxplot in Fig. 4.32b, on average, over 100 train/test iterations where the train and test sets are randomly shuffled on each iteration, the perceptron classifies 94.1$\%$ of the test points correctly using the mosaic engineered features as inputs on average. To understand if these integrated, mosaic engineered features, are more useful than using simply the count of pulses in each event channel alone, thereby discarding temporal information in the original signal, we train a perceptron model using the same

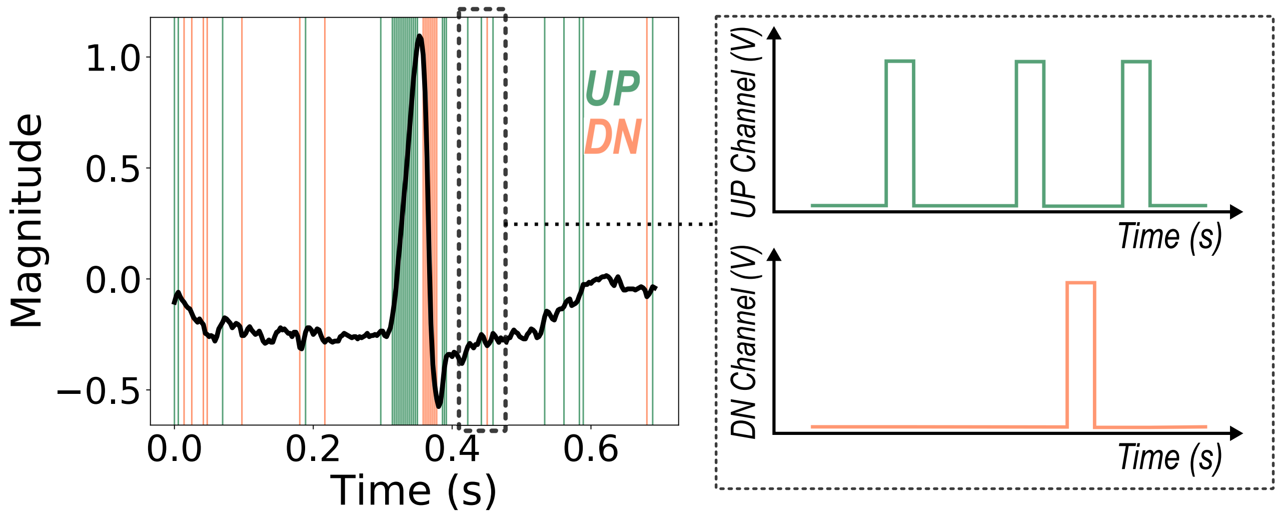
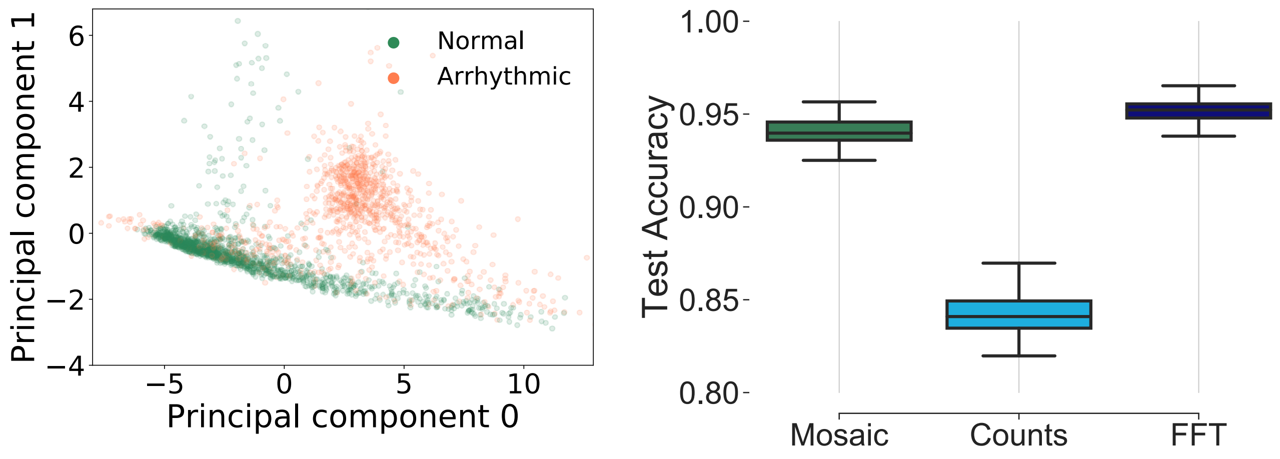


Figure 4.31: (left) A delta-modulated electrocardiogram of a heartbeat. (left) The black waveform shows the original ECG waveform from a single channel of a healthy heartbeat. Superimposed green and orange vertical lines show the UP and DN that have been extracted by the delta-modulator circuit. (right) The resulting timestamped UP and DN events are transduced as voltage pulses in UP and DN channels in the circuit of Fig. 4.30.

train-test splits over 100 iterations. With these features, labelled as 'Counts' in Fig. 4.32b, a reduced average test accuracy of 84.2% results. Based on this drop in performance, of approximately 10%, it is fair to conclude that the mosaic-based reservoir computing model is able to apply a useful transformation on the delta-modulated pulse trains and that the resulting features are more linearly separable. To compare with a more traditional approach to time-series processing, a fast-Fourier transform (FFT) is applied to each 700ms ECG waveform - resulting in 125 frequency components in each of the two ECG channels. From these 250 frequency components, a feature selection algorithm was used to extract the 36 best [294], which were then used to train the same perceptron model with the same train-test split once more. Labelled as 'FFT' in Fig. 4.32b, these features allow the resulting perceptron model to obtain an average accuracy of 95.2$\%$, ultimately 1% better than was possible using the same number of mosaic engineered features. However, perhaps the features generated by the mosaic could be improved through better hyper-parameter tuning or through the use different neuron and synapse circuit models. The incorporation of spike frequency adaptation into neuron models, for example, is known to improve the performance of reservoir computing models [371]. Furthermore, beyond random programming of the memory devices, surrogate gradient-based learning algorithms [372, 252] could be used to learn small-world architectures and weight matrices through backpropagation in an automatic and deterministic fashion that could be transferred into the conductance states of the devices in the mosaic.

4.4 Chapter discussion

During the course of the thesis all of the circuits presented in this chapter were designed and laid out for fabrication in a hybrid CMOS-RRAM process - in which resistive memory devices are integrated between metal layers four and five [290] of a standard 130nm CMOS process. This includes the hybrid neuron



(a) (b)

Figure 4.32: (a) The first two principal components resulting from the application of the principal components analysis [370] algorithm on the original thirty-six low-pass filtered spike trains of each neuron into a two dimensional representation allowing the two classes to be visualised. Green dots correspond to each electrocardiogram labelled as a healthy heartbeat and orange dots as heartbeats which have been labelled as arrhythmic. (b) The accuracy distribution of a perceptron classifier using (left, green) the thirty-six neuron firing rates, (centre, light-blue) the sum of UP and DN events in the two ECG channels and (right, dark-blue) the thirty-six best features resulting from the fast Fourier transform of the signals.

and synapse circuits (Figs. 4.4a and 4.6a), neuron and routing column circuits (Figs. 4.20 and 4.22), neuron and routing tile circuits (Figs. 4.24a and 4.25a) as well as the column interfaced delta-modulator circuits (Fig. 4.30). At the time of writing, the wafers containing the circuits have only just completed fabrication and so only a subset have undergone basic verification that could could be presented in this thesis . However, the test of the rest of these structures will be an important step on the path to realising a fully-integrated neuromorphic memory mosaic.

Something that became apparent during the layout of these circuits was the large area consumed by the transistors used to realise the circuits in programming path of the RRAM devices relative to the area consumed by the rest of the analogue circuits. Resistive memory technologies, including the OxRAM technology considered in this thesis, require large programming voltages - well in excess of the voltages tolerated by standard transistors in most scaled CMOS nodes. The 130nm CMOS node used to design the circuits in this work has a nominal supply voltage of 1.2V for example. However, the voltages required in the SET, RESET and, in particular, the forming programming operations exceed this limit. As a result of this two types of transistors are required to in order to arrive at a reliable design. One standard 'thin' gate oxide transistor and another high-voltage transistor with a 'thick' gate oxide. The thick gate oxide transistor exhibits less ideal characteristics and also requires larger dimensions to satisfy a more constrained set of layout design rules. The thin oxide transistor, which tolerates a nominal supply voltage of 1.2V, was used to design all of the analogue circuits such as the neurons, synapses, OPAMPS and logic blocks (see Appendix 6.1). The thick-oxide transistor, with a nominal supply voltage of 4.8V, was used to design all of the circuits in the programming path of the devices such as the the 1T1R structures themselves and the multiplexers, transmission gates and level-shifters which were used to deliver programming signals to the devices (see Appendix 6.1).

Two circuits showing such additional multiplexing and level-shifter circuits, required in each of the columns and around each of the 1T1R structures in the hybrid neurons and synapses, are shown respectively in Figs. 4.33a and 4.33b. A column requires a multiplexer at each end of the parallel set of 1T1R structures in order to switch the common top and bottom nodes between programming waveforms and the functional nodes of the analogue circuit. To select devices for programming, a digital control signal, S, is used to multiplex the nodes between these two types of signal. Furthermore, in order to interface the output 1.2V voltage pulses, originating from the output of the neuron and routing columns, additional level-shift circuits are required in the column circuit such that a 4.8V pulse can fully turn on the thick oxide selector transistors as in Fig. 4.33a (see Appendix 6.1).

For example in Fig. 4.33a, the devices see $V_x = V_{top}$ and V_{bot} on their respective top and bottom electrodes and receive 4.8V pulses at their gates as a function of the their pre-synaptic activity. However, when S is high, each device sees V_{src} and V_{bit} at these same terminals (in reference to the names 'source' and 'bit' lines that are normally given to these lines in a standard RRAM array) and $Vword < n >$ at the gate of the n_th device. Similarly, for the 1T1R structures incorporated into the hybrid neuron and synapse circuits (Fig. 4.33b), a multiplexer is required at each node. These multiplexers share a common selection signal S and each RRAM device requires an independent select signal. Each of the selector transistor gates are connected to the 4.8V supply voltage such that, when not selected for programming, the resistance of the 1T1R is determined predominantly by conductance state of the RRAM device.

From inspection of the layout of the column circuits, pictured in Appendix 6.13a, while the resistive memory devices and the analogue circuit models respectively consume 0.05$\%$ and 15$\%$ of the silicon area per column (Fig. 4.33a), the selector transistors, level-shifters and multiplexers consume 4$\%$, 36$\%$ and 45$\%$ of the total area respectively. Similarly, in the case of the hybrid neuron and synapse circuit layouts (see Appendices 6.15a and 6.14a), where three multiplexers are required per 1T1R structure as in Fig. 4.33b, 80$\%$ and 4$\%$ of the total circuit area is consumed by the multiplexers and selector transistors.

While there is likely room for considerable reduction in these ratios through transistor sizing and layout optimisation, what is clear is that high device density, normally listed as one of the principal benefits of RRAM technologies, is not possible due to two practical issues. First, the large minimum dimensions imposed by the layout design rules pertaining to the thick oxide transistors that are required, in this technology, to be used in the RRAM programming path. Second, due to the degraded properties of these thick oxide transistors, they are required to sized with large width over length ratios (or be composed of multiple fingers) to allow for the series voltage drop over the multiplexers, relative to the minimum resistance of the RRAM devices in the HCS (on the order of a few kΩ) to be minimal. Therefore, the most straightforward path to reducing the ratio between the area of the circuit consumed by the functional and programming elements, would be through device level engineering in two respects. First a reduction in the programming voltages to allow the use of thin-oxide transistors in the programming path and, secondly, for the minimum resistance (conductance) of devices to be increased (decreased) which would ease the constraints on the series resistance in the design of the multiplexing circuits.

In spite of the poor device density resulting from these practical considerations, this chapter presented two key ideas as to how a truly non-von Neumann neuromorphic computing fabric could be realised by leveraging the overarching advantage of resistive memory devices - conductance non-volatility.

Firstly, it was demonstrated and discussed how resistive memory devices can be integrated within

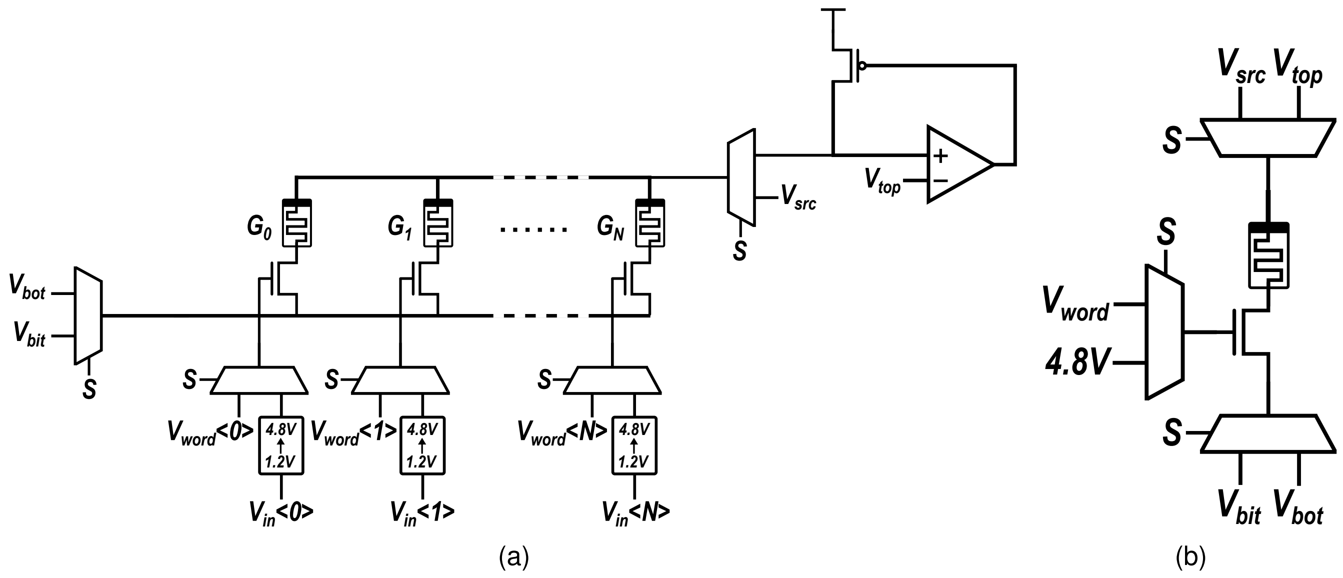


Figure 4.33: (a) Schematic of the column circuit with the additional multiplexers, level-shifters (the rectangle containing the text 1.2V - 4.8V) and programming and control signals required to program the conductance states of the resistive memory devices. The control signal S, selects the column for programming whereby the voltages V_{bit}, V_{src} and the vector V_{word} are applied to the common top, bottom and selector gates respectively. (b) Schematic of the 1T1R structure required in the hybrid neuron and synapse circuits. The control signal, S, switches the incorporated 1T1Rs between existing inside the analogue circuit or having its terminals connected to programming voltages.

neuron and synapse circuit models such that their parameters could be, instead of stored in a centralised volatile memory centre, distributed throughout the substrate and exist physically within the computational elements themselves. This avoids burning energy to maintain information in volatile memory circuits and in the generation and distribution of a potentially large number of DC biases. Furthermore it allows for each neuron and synapse to have their own independent set of parameters and the ability to determine these parameters locally using local analogue or digital circuits - an example of this was presented in the implementation of a local intrinsic plasticity Markov chain algorithm. Normally on a neuromorphic processor, groups of some thousands of neurons and synapses that exist on each core are forced to share the same sets of parameters stored in the central bias generator. This is largely imposed due to energy considerations and metal area constraints imposed by the number of metal levels and available silicon area. Considering a state of the art analogue-domain neuromorphic processor [339], each bias parameter on average consumes about 4μW of power. If the chip were to have unique parameters for each neuron, assuming only three parameters were required per neuron (time constant, gain and refractory period like in Fig.4.4a), each circuit would burn 12μW of power. This power consumption means, with only 1000 individually parameterised neurons, an undesirable 12mW of static power is required. Moreover, perhaps the least practical aspect of this would be the physical metal area required to route the three biases from the bias generator to each neuron. Assuming a four metal layer 180 nm technology [339], 1.5μm^2 of area would required per neuron. For one thousand neurons, this number grows to 1.5mm^2 - the equivalent of half a standard silicon chip.

In the second section of this chapter, a system-level means of interconnecting neuron circuits together through a continuous mosaic of resistive memory augmented tiles, what we called the neuromorphic memory mosaic, was proposed. Just as the hybrid neuron and synapse circuits distributed memory regarding the dynamical properties of neuron and synapse circuits, the mosaic distributes information regarding the neural network topology that interconnects such neuron models. This architecture was seen to be particularly well suited for bio-inspired small-world neural network architectures and a use case of a mosaic-based small-world neural network applied as a reservoir computing feature extractor was demonstrated in the context of a heart arrhythmia detection task.

Unlike the alternative hardware means of realising neural networks, GPUs, neuromorphic processors or other RRAM array based graphical hardware, the mosaic does not require volatile memory, digital control circuitry nor digital-to-analog and analog-to-digital converters. Therefore it succeeds in realising a non-von Neumann computer on which custom neural network topologies can be implemented and configured using non-volatile RRAM devices, but without the drawbacks that normally curtail the efficiency of other RRAM-based approaches - principally the use of ADCs.

Although they were considered and presented separately, ultimately these two ideas can be combined by placing hybrid CMOS-RRAM neuromorphic circuit models at the end of the neuron columns (instead of CMOS circuit models). In this fashion, a full silicon full non-von Neumann hardware will be possible where all computation is performed locally, using the memory elements themselves, and information is transported through brief asynchronous voltage pulses. Overall the mosaic architecture arrives at something that resonates strongly with the organisational and communicative principles observed in animal nervous systems. This is perhaps due to the fact they both arise from the consideration of similar sets of spatial connectivity and energy constraints.

In reflecting on the parallels between this silicon architecture and biology, an interesting question arises concerning the spike-based nature of computation and communicating information. On one hand, biological neurons have been proposed to spike because they use inter-spike timing as a means of encoding information [373, 374, 375] and to learn [336, 337]. On the other, it could be supposed that the spiking behaviour of neurons is an adaptation permitting the fast transfer of information via action potentials over large distances [376, 377] - despite the fact that this conversion from analogue (voltage integration on a neuronal membrane) to digital (spiking) is understood to result in a significant loss of information [378]. This second idea is supported by the observation that nervous systems often opt to use non-spiking, graded-potential, neurons [379, 380, 209], that transmit information in an analogue, diffusive, fashion when neurons can be inter-connected over short distances. From this perspective then, in a neuromorphic computing system, should silicon neurons spike, given the fact that pulses propagate equally as fast as continuous analogue signals when sent through a metal wire?

The question of whether or not spiking is computationally important is an open one. However, from the stand-point of practicality and energy-efficiency in electronic circuits the answer is yes, absolutely! In neuromorphic processors for example [85, 86, 88], neurons communicate via brief address events [112, 361]. If it were required to transmit continuous analogue signals between neurons, the bandwidth of an AER-style approach to routing would most likely plummet and necessitate a rethink of how neuromorphic processors might realise neural networks. Equally in the mosaic, not only would it require a re-design in the columns circuits such that they could transmit continuous signals but, most crucially, would mean that RRAM devices must be read continuously. Due to the large conductance values in the HCS (low resistances of the LRS) of most RRAM technologies, this would entail a significant undesirable power consumption. By implementing spiking neuron models in the mosaic, regardless if they are of computational advantage or not, it allows information to be transmitted in brief pulses, obviating the need for DACs and ADCs. In addition, due to their asynchronous nature, energy expensive RRAM device reading operations occur only when the information they store is needed. Overall this underlines the importance of event-based computation in the development of energy efficient neuromorphic computing hardware - in particular for those which leverage resistive memory devices.

Chapter 5

Conclusion

While the three chapters of this thesis were presented largely independently of one another, they are together intended to paint overlapping regions of a larger picture of a full Bayesian neuromorphic computing system. Chapter 2 showed how biological nervous systems could be used as a source of inspiration in the development of memory efficient model architectures in the proposal of a model for the air-current evoked escape response of the cricket cercal system. When compared to a universal approximator (i.e, a generic deep learning approach), the bio-inspired architecture was found to allow for a reduction of up to two orders of magnitude in the required number of synaptic parameters when applied to the detection of simulated attacking predators. Additionally, the specific directed topology consisting of neurons with well defined roles and logical relationships was seen to permit interpretation of the model. It was also shown that neurons and synapses with dynamical properties were important in addressing tasks based on temporal information in a model inspired by the elementary motion detection circuits in the early visual system of *Drosophila*.

Chapter 3 addressed the question of how the parameters of such neural network models could be determined if they were based on oxide-based resistive memory devices which, if it were possible to overcome their myriad of non-ideal properties, could greatly reduce the energy required in connectionist AI approaches. A solution was presented leveraging the framework of Bayesian machine learning. Bayesian machine learning provides the tools that allow certain non-ideal properties of RRAM, namely their variability, to be harnessed - turning the crippling drawback of the technology into an indispensable, core computational primitive. In-situ and ex-situ realisations of Markov chain Monte Carlo sampling, leveraging RRAM as random variables from the perspective of their variability, were proposed to train RRAM-based Bayesian models. In the in-situ setting (section 3.2) it was observed in experimental implementations of the algorithm that RRAM-based MCMC sampling was capable of training models with five orders of magnitude less energy than a state of the art CMOS alternatives. This result reflected the potential of this approach in bringing adaptive learning capabilities to the edge where they cannot currently be implemented due to energy constraints. Another key result was observed in the ex-situ setting whereby a Bayesian neural network model, transferred onto an RRAM-based hardware, was able to leverage the uncertainty in its output predictions in order to say ‘I don’t know’ in a safety-critical application. By gating decisions using uncertainty thresholding the RRAM-based Bayesian neural network was able to take zero erroneous, potentially harmful, actions. Despite its widespread use across countless domains, one hurdle faced by Metropolis-Hastings MCMC sampling algorithms applied to the training of neural network models is their drop in efficiency as the number of model parameters and the size of datasets increase. This makes it challenging to apply the algorithm to extremely large models and big data datasets like those used in the deep learning setting. A potential solution for this limitation was discovered in section 2.2, where the proposed cricket terminal abdominal ganglion escape response model was able to achieve an equivalent accuracy to model with two orders of magnitude more parameters. Such memory efficient bio-inspired directed neural network architectures offer a means of maintaining model sizes at a reasonable level where Metropolis-Hastings MCMC sampling can be at its most effective, while tackling higher complexity tasks. Approaches using MCMC with mini-batches were also discussed as a means of scaling to bigger datasets.

Chapter 4 completes the picture by detailing a non-von Neumann computing fabric on which RRAM-based bio-inspired models can be realised. By incorporating resistive memory into neuron and synapse circuit models in section 4.2, inspired by the organisation of animal nervous systems, centralised volatile memory centres were avoided - alleviating the von Neumann bottleneck and eliminating static power consumption for parameter storage. This was seen to be of benefit in the implementation of local, massively parallelised, plasticity mechanisms which would be impossible through existing approaches based on centralised volatile memories. An example was presented whereby the high resistive state of an oxide-based RRAM was used to implement a Markov chain mimicking a homeostatic intrinsic plasticity algorithm observed in animal nervous systems. Then in section 4.3, by introducing the spiking neuron column circuit and the routing column circuit, a non-von Neumann means of inter-connecting neurons in a custom topology was proposed, a neuromorphic memory mosaic; based on a continuous array of tiles of such column circuits using RRAM to store information. Crucially, the prohibitive cost of analogue to digital conversion, which dominated the energy budget in the simulation of the RRAM-based MCMC sampling system designed in section 3.2, could be avoided since all computation and routing is performed in the analogue-domain. It was also discussed in section 4.4 that the asynchronous spiking, or event-based, nature of computation employed by the mosaic was also key in avoiding energy expensive continuous reading of RRAM devices. While this mosaic of neuromorphic memory is not amenable to models such as feed-forward networks, it was discussed how biological nervous systems, which face similar spatial connectivity constraints to the silicon-based mosaic, could once again provide a solution to this potential limitation by taking inspiration from the small-world organisation of their neural network architectures. An example of a small-world reservoir computing model, with random local dense inter-connectivity, was implemented in a simulation of the neuromorphic memory mosaic and applied as a means of engineering a feature space for a linear machine learning classifier.

The motivation of carrying out this work was that, by exploring the interface between machine learning, biology and technology and understanding the constraints imposed and the opportunities offered by each, a new hardware-focused approach to AI which could reduce the energy requirements of intelligent, locally-adaptive, edge systems could be proposed. The three chapters of this thesis put forward ideas for how this could be achieved and, along the way, solutions for problems in the ethical application of AI, namely enabling model interpretability through the use of bio-inspired architectures and the quantification of prediction uncertainty through the Bayesian modelling approach, were encountered. Given then that various issues faced by AI can be solved independently through the ideas developed in each of the thesis chapters; what can a full Bayesian neuromorphic computing system, integrating the presented elements, look like?

5.1 A vision of a Bayesian neuromorphic computing system

A future vision of a Bayesian neuromorphic computing system based on resistive memory is depicted in Fig. 5.1. It is based on an evolution of the mosaic architecture of chapter 4, where four neuron tiles, labelled as A, B, C and D, are shown. While in the third chapter the neurons in each mosaic tile were considered as single entities, here we go Bayesian, and, in the same fashion as the hardware proposed section 3.3, each of the neuron tiles in the mosaic becomes a single 'Bayesian spiking neuron'. Each of the neuron columns in a tile correspond then to different samples of that neuron. An example of a directed Bayesian neural network is shown in the middle of Fig.5.1. Such a topology could be realised by programming the appropriate set of interconnections in the intermediate routing tiles - which, in addition to routing events, sends information pertaining to the sample from which the event originated. Since this example contains only four neurons, the network that is drawn is largely arbitrary. However, as discussed previously, one could imagine a larger bio-inspired, small-world, neural network architecture amenable to a mosaic-based implementation featuring a greater number of Bayesian spiking neurons. The inset on the right-hand side of Fig. 5.1 shows how the Bayesian spiking neuron D is realised using a neuron tile. Flipped with respect to the arrays presented in chapter 3, each row of the tile corresponds to one pre-synaptic distribution. The synapse g_0 is labelled in a blue rectangle as an example. Each of the columns is one sample of the spiking neuron D. The joint activation of all of the spiking neurons in the tile is the output distribution of each tile - indicated by the green rectangular box in Fig. 5.1. The output distribution of each tile-based Bayesian spiking neuron, $p(D)$, could correspond to a distribution of spike-times, average rates or some other collective property of the circuit models in that neuron tile. By using the mosaic here as a substrate for a spiking Bayesian neural network, the sequential row-wise reading strategy required to perform inference using the mutli-layer Bayesian neural network hardware presented in section 3.3 can be avoided. Instead the stored samples will be read in an asynchronous, parallelised, fashion - potentially reducing inference latency and energy. This proposed Bayesian neuromorphic computing system can be equipped with circuits, like those designed in section 3.2, to train the conductance parameters of each column through in-situ RRAM-based MCMC sampling. This would therefore enable it to adapt and learn locally, on-chip. The interpretability of the bio-inspired architecture and the uncertainty encapsulated in its output prediction distributions, that of neuron D in Fig. 5.1 for example, also facilitates the responsible and ethical application of such a system to safety-critical edge learning applications.

The interactions and co-dependencies our these four key elements - resistive memory, Bayesian machine learning, bio-inspiration and spiking neurons - are summarised in the directed graph depicted in Fig.5.2. Each arrow denotes how one aspect relies on the other. At the point of entry to this graph, there is the core principle that runs thought this thesis that RRAM arrays could allow dot-product operations to be implemented with a high efficiency, owing to device non-volatility. Secondly, if we then want to use an RRAM-based hardware to learn at the edge, then to do so we must resort to Bayesian machine learning; because it is the only framework based on the manipulation of random variables that can be used

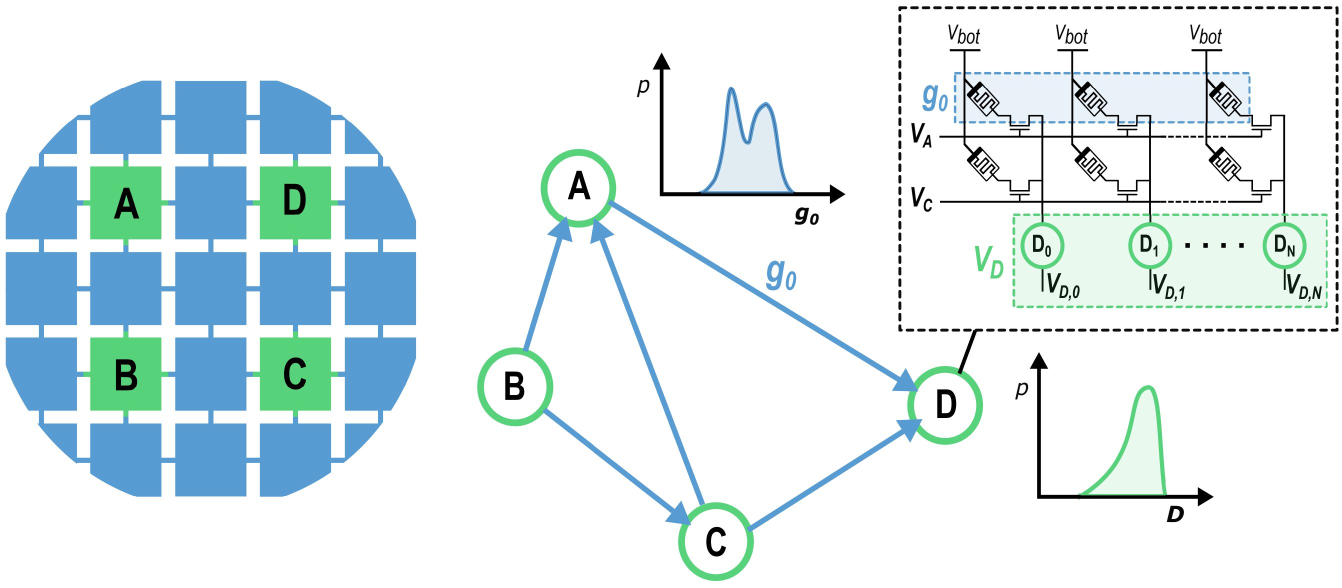


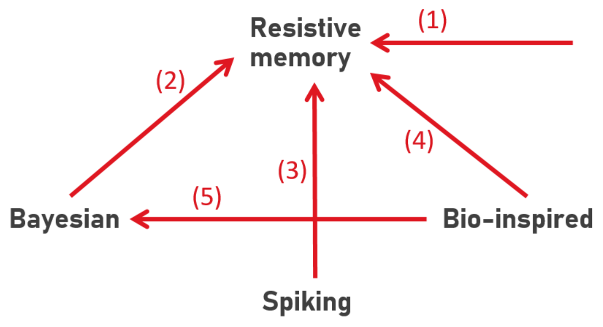
Figure 5.1: (left) A neuromorphic memory mosaic architecture composed of four (green) neuron tiles and twenty five (blue) routing tiles. Each of the neuron tiles is labelled as A, B, C and D. (centre) A generic directed graph topology with neurons (open green circles) A, B, C and D interconnected by directed synapses (blue arrows). (right) Each neuron in this network corresponds to a single mosaic neuron tile. The tile has a number of rows equal to the number of pre-synaptic connections and a column for each of the sampled spiking neuron models. The blue dashed box drawn along the top row of the tile shows the synaptic distribution for the synapse labelled as g_0 and the green box around the individual neuron models, the activation distribution of neuron D.

to describe resistive memory characteristics. Third, if we use resistive memory devices then we want to minimise as much as possible how often they are read. One means to do this is to construct a neural network with a spiking activation function - since memory devices will be read sparsely and in an asynchronous fashion. Finally, looking towards biological nervous systems is important for two reasons. One is that bio-inspired architectures are compatible with the proposed mosaic architecture due their small-world connectivity patterns. The other is that, as shown in the first chapter, bio-inspired neural architectures allow model sizes to be reduced substantially. Given that Metropolis-Hastings MCMC sampling performs better in lower dimensional model spaces, bio-inspiration also offers a means of scaling up the algorithm to more complex tasks.

5.2 Perspectives and future work

In addition to the chapter specific perspectives, detailed in each of the chapter discussions, there are some final and more general perspectives for future work listed briefly here;

- **Develop the hardware**: Starting with the circuits designed, laid out and fabricated during this thesis, it will be important to fully test and validate each of the structures - in particular those which could not be tested in time for the conclusion of this thesis. These circuits can serve as a basis on which to realise a full memory mosaic, as proposed in section 4.3, in silicon that also incorporates the hybrid neuromorphic circuits presented in section 4.2. Secondly, the peripheral control and



1) **RRAM** because of the potential of the low-energy dot-product operation
2) **Bayesian**, because RRAM-based models must be trained with Markov chain Monte Carlo sampling
3) **Spiking**, because we must avoid continous device read and analog-to-digital conversion
4) **Bio-inspired**, to apply small-world topologies compatible with the mosaic architecture
5) **Bio-inspired**, to reduce the number of parameters to be trained through MCMC sampling

Figure 5.2: (left) Four labels, corresponding to the four axes of the envisaged Bayesian neuromorphic computing approach, and the dependencies between them (red arrows). (right) A written list summarising the reasoning behind each red arrow. The number in the list corresponds to the number denoted alongside each arrow.

programming circuits developed in chapter 3, required to implement RRAM-based MCMC on a self-contained chip, should also designed, laid out, fabricated and validated. A long term view would see such RRAM-based MCMC sampling circuits incorporated into the memory mosaic architecture as a step towards the Bayesian neuromorphic computing system depicted in Fig. 5.1. Hardware development also extends to engineering of the properties of the resistive memory technology. Besides the importance of reducing the programming voltages and reducing the maximum conductivity in the high conductive state, as discussed in section 4.4, the ideal device properties for RRAM-based Bayesian machine learning methods are not yet known. Devices could be engineered, for example, to have more variability or a flatter conductance vs. standard deviation relationship in the HCS if these characteristics were deemed desirable.

- **Further bio-inspired architectures**: Bio-inspired neural network models, because of their memory efficiency and small-world graphical properties, were understood to be key in addressing the scaling issue of the MCMC sampling algorithm as well as easing the spatial connectivity constraint of the neuromorphic memory mosaic architecture. While, admittedly, the models presented in chapter 2 are small and application specific, future work should understand how bio-inspired modules like these can be interconnected and what other mechanisms might be required to glue everything together and realise an artificial nervous system model. Close attention should be paid to research dedicated to the mapping of full brain connectomes in insects like *Drosophila* [381, 382, 383, 253], that can inspire further architectural motifs, as well as to brain structures which are becoming increasingly well understood such as insect mushroom bodies [381, 382, 253, 384] and central complexes [385, 383] that seem important in linking up sensory processing with memory and output motor responses. Further to this, bio-inspired neural networks may help to further justify the cause for the use of RRAM as a memory element in computer systems. Von Neumann implementations of neural networks, despite their poor memory efficiency, are extremely flexible - owing to the endless possibilities in how the instructions that execute in their central processing units can be recursively organised. On the other hand, RRAM-based neural networks, although potentially requiring less energy in their execution, may be much more restricted. For example one would have to work within the architectural possibilities of routing and neuron tile dimensions had been fabricated onto a neuromorphic memory mosaic chip. Furthermore, RRAM is also not amenable to models and applications where the memory might have to be re-configured a large number of times - since RRAM technologies are often very limited in terms endurance as was seen in section 3.2. This could be problematic, for example, in the case of dynamic neural networks where the graphical structure of the model changes continuously during its execution [386]. By contrast, the volatile memory circuit used in the memory hierarchy of von Neumann computers can exchange bits almost endlessly. Foregoing notions of biological synaptic structural plasticity [387], RRAM-based bio-inspired neural networks may simply be more compatible with RRAM-based computers because, just as each synapse is hardwired in biology, physical RRAM devices can have a one-to-one correspondence with each memory element in the neural network that is being modelled.

- **Dynamical Bayesian SNNs**: The spiking neuron activation was observed to be key to the potential efficiency of the mosaic architecture since it offered a means of leveraging resistive memory without conversions between analogue and digital domains and permitted asynchronous event-based computation which avoids continuous device reading. In the context of a Bayesian neural network model, it was also discussed how event-based computation could having to read each sample of each Bayesian neuron in a sequential manner. Furthermore, the models to which Markov chain Monte Carlo sampling was applied in chapter 3 were 'static'; in that training and inference were performed in sequential and discrete steps and the neuron models did not incorporate dynamical properties. However, neural networks that incorporate dynamical synapse and spiking neuron models, such as those considered in section 2.3 and chapter 4, were seen to be advantageous in tasks containing temporal information. Exploration is required in order to understand what properties of Bayesian spiking neural networks that would be based on such models, such as inter-spike time or average spike rate distributions, can be leveraged for inference and for MCMC sampling based training. Some recent works have started to address questions of how networks of spiking neurons can be viewed as Bayesian models - particularly in how their collective dynamics might perform sampling-based inference [388, 389, 390, 391]. Although this offers a starting point for such an investigation, further work is required to understand how these two disciplines can merged effectively.

- **Useful edge learning**: Throughout the thesis, an overwhelming focus was directed towards supervised learning approaches - with a brief flutter into the reinforcement learning domain in section 3.2. Repeated emphasis was also put on the potential for, and the importance of facilitating, edge learning. However, it is important to pose the question of whether or not local supervised learning at the edge makes sense. While large supervised labelled datasets can be used to train neural network models in an ex-situ fashion at the cloud, edge learning systems will likely have no means of acquiring such a pre-labelled never mind store it in what will likely be a constrained memory. An end user, or a kind citizen, could perhaps take time to label data locally for the system, something like a teacher, but that seems rather impractical. Reinforcement learning in many applications could be a viable solution, but that also entails considerable trial and error and so would be limited to applications that are not safety-critical. Humans could not learn to cross a busy road by reinforcement learning for example, and autonomous vehicles, except in situations where a few million spare cars are at your disposal, could not realistically learn how to drive through a reinforcement learning approach at the edge. The third domain of machine learning, unsupervised learning, would certainly be the most appropriate. An edge system could observe sensory information and learn representations and latent variables related to the environment in which it has been placed. However, for many applications at the edge, such a passive system may not be useful. For example, considering once again the use case of the implantable cardioverter defibrillator, an unsupervised learning algorithm might learn the underlying characteristics that distinguish a normal heartbeat from an arrhythmic one, but may not have a concept that an arrhythmic heartbeat was dangerous and requires intervention. A solution to this problem might be what is broadly referred to as 'self-supervised' learning - a sub-domain of unsupervised learning where representations can be given significance using ideas borrowed from the supervised and reinforcement domains [392, 393, 394]. Self-supervised learning is characterised by the local generation of labels and teacher signals which can be provided to a model that has formed useful representations of its environment through local unsupervised learning. In many respects animals are also obliged to learn in this self-supervised fashion too. Take as an example the olfactory learning process that occurs in the mushroom body of insects [381, 253, 384, 395]. The mushroom body receives input from olfactory sensory neurons and, in a similar fashion to a reservoir computing model, generates a higher dimensional representation of these inputs in the Kenyon cells of the mushroom body calyx. Based on feedback from the environment, or the gastric system within its own body, dopaminergic and octopaminergic neurons modulate the strength of synaptic connections from the Kenyon cells to mushroom body output neurons that encode the learned valence of certain olfactory stimuli [384, 395]. Similar principles, using the mathematical toolbox that is under development in the self-supervised learning field should be explored and adapted to the RRAM-based MCMC sampling approach to understand if this is a more viable approach to learning at the edge. Similarly ideas from continuous learning and active learning could also be explored. Alternatively, perhaps edge learning needs to propose a new set of modelling and algorithmic tools and become a domain in its own right that takes a fresh look at what a learning artificially intelligent isolated from external labels and biases can be.

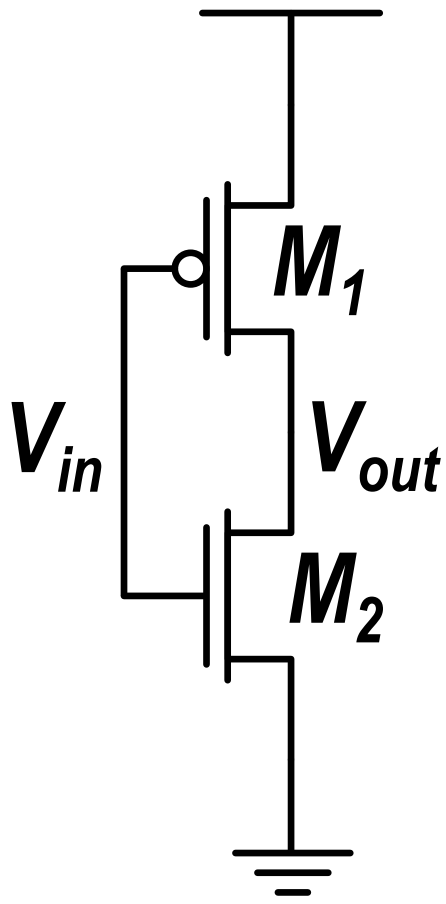
Chapter 6

Appendices

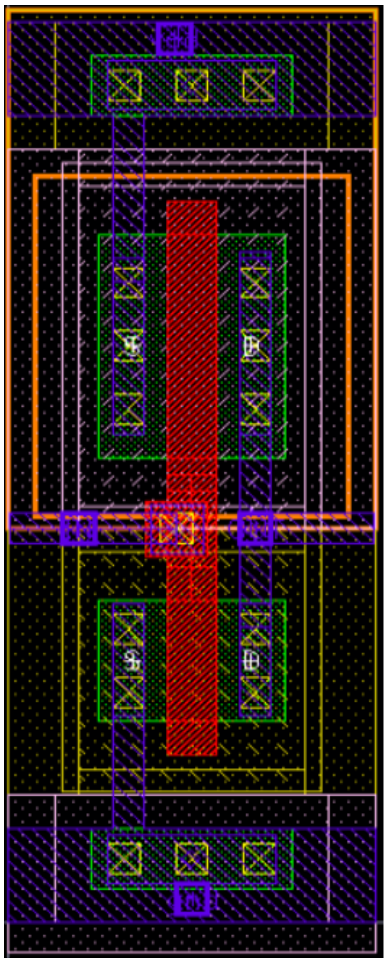
6.1 Appendix - Additional circuits and layouts

The circuit schematics and layouts for the other circuits and layouts not discussed or presented in the main text. The thin-oxide transistors have the dimensions: PMOS width=1.2μm, length=250nm. NMOS width=650nm, length=250nm unless otherwise stated. The thick-oxide transistors have the dimensions: PMOS width=3.3μm, length=500nm. NMOS width=2.07μm, length=500nm unless otherwise stated. All transistor bulks connect to vdd is they are a PMOS and gnd if they are an NMOS through taps at the top and bottom of each structure. A bar symbol and ground symbol are used to refer to vdd and gnd which are connected to 1.2V and 0V supply voltages respectively unless otherwise specified for the thin-oxide transistors and 4.8V and 0V for the thick-oxide transistors. Each layout figure is the layout image as it exists in the Cadence Layout XL that was used to perform the layout. Layers pertaining to polysilicon, metal, transistor active regions, nwells, and n+ and p+ doped regions are shown with different colours.

6.1.1 Inverter unit cell



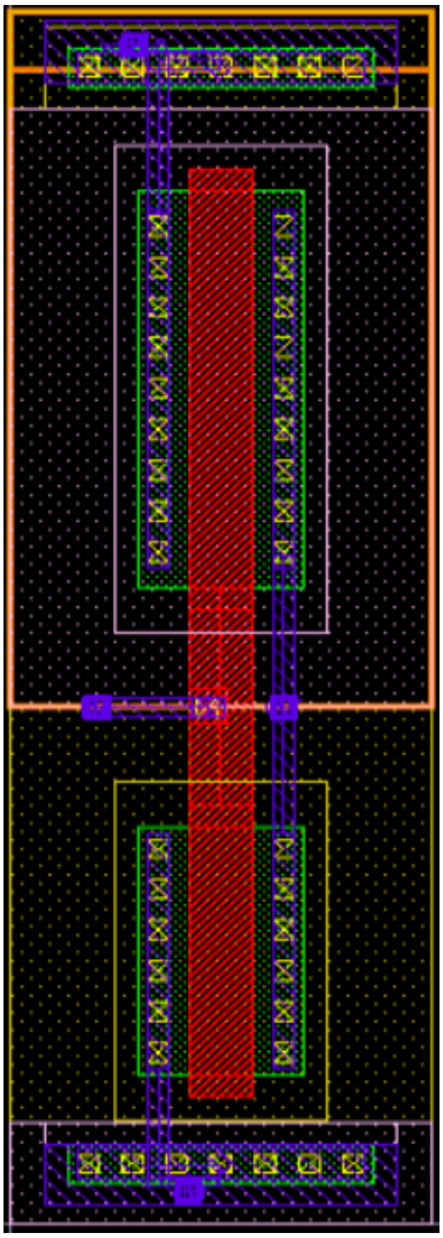
(a)



(b)

Figure 6.1: (a) The inverter unit cell used throughout the design and layout. PMOS (M_1) width=1.2μm, length=250nm. NMOS (M_2) width=650nm, length=250nm. These transistor sizes are assumed to be constant for each of the circuits unless otherwise specified. b) The inverter unit cell layout. The cell rectangular 5.05μm in height and 1.82μm across.

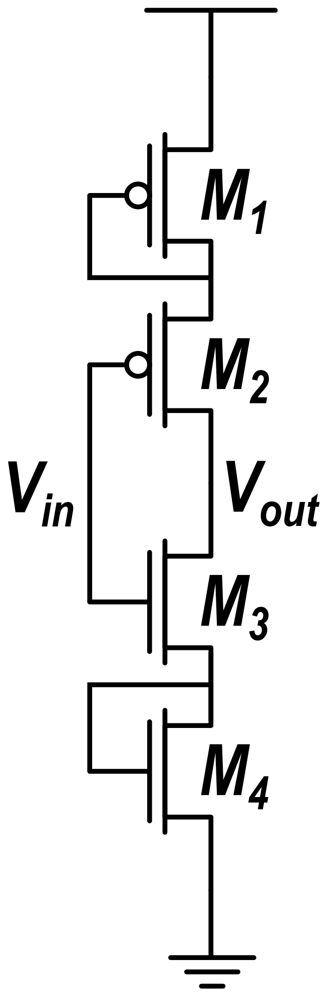
6.1.2 Thick oxide inverter unit cell



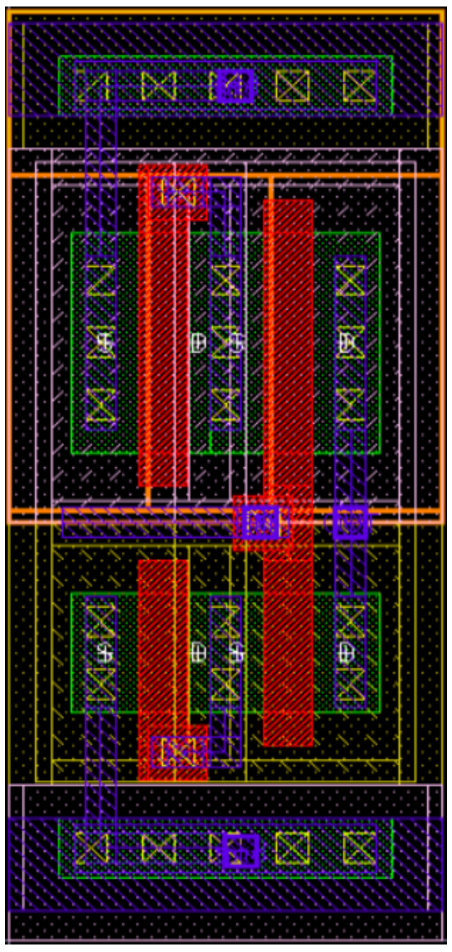
(a)

Figure 6.2: (a) The layout of unit cell of the inverter, which has the same circuit as in Fig. 6.1a but using the thick-oxide transistor. PMOS (M_1) width=3.3μm, length=50nm. NMOS (M_2) width=2.07μm, length=500nm The cell rectangular 10.1μm in height and 3.26μm across - consuming three and half times more area than the thin-oxide unit cell. It requires a nominal vdd of 4.8V.

6.1.3 Starved inverter



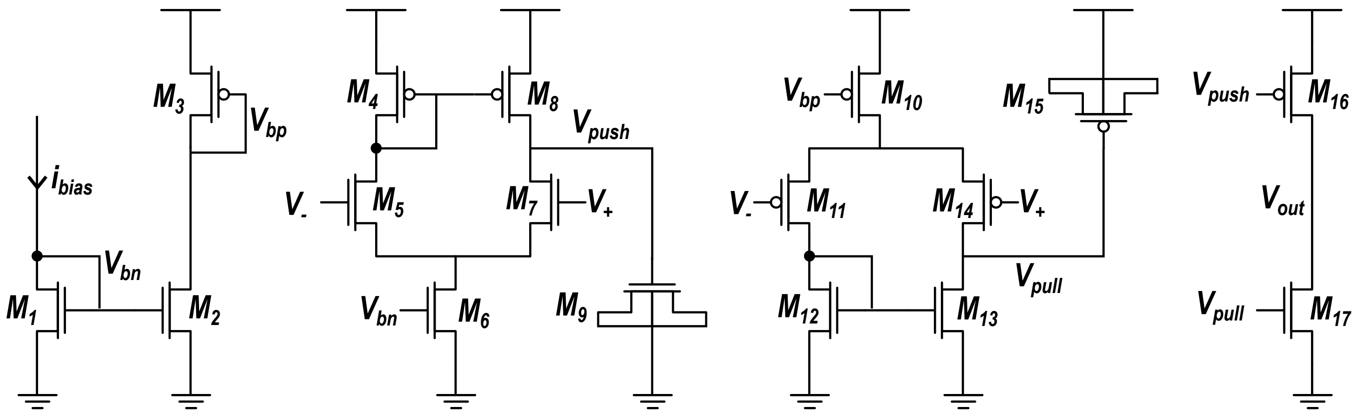
(a)



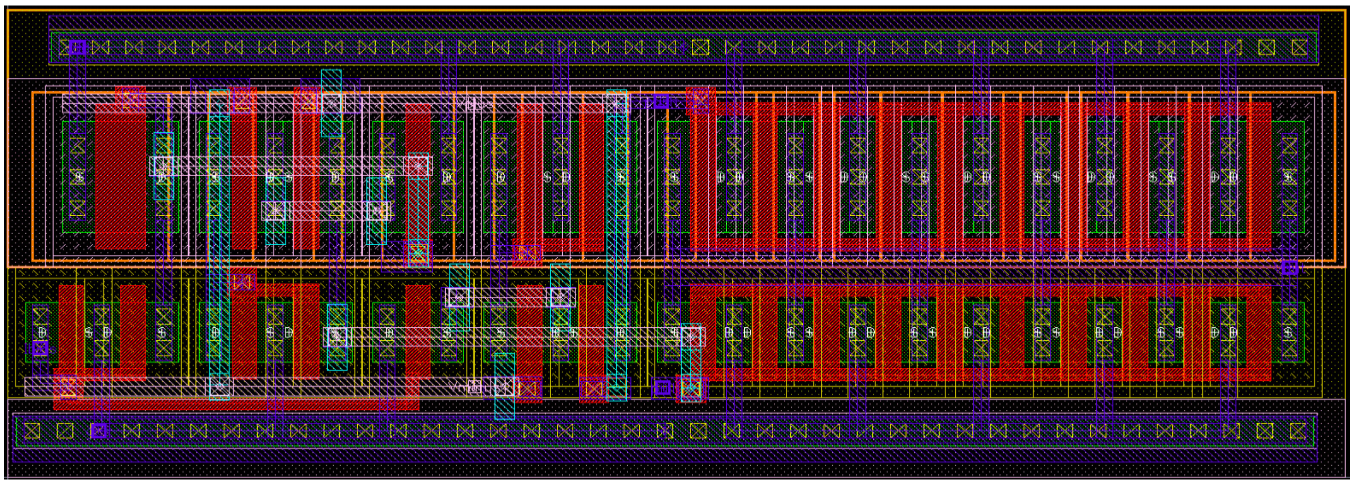
(b)

Figure 6.3: (a) The starved inverter circuit schematic which is used to induce small delays in the pulse propagation paths. b) The layout of the starved inverter circuit.

6.1.4 Operational amplifier



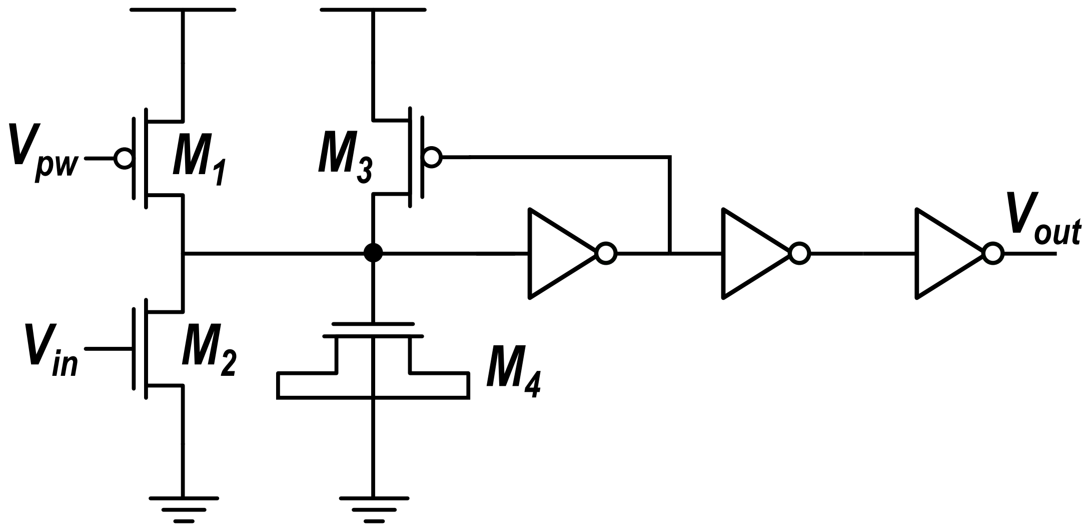
(a)



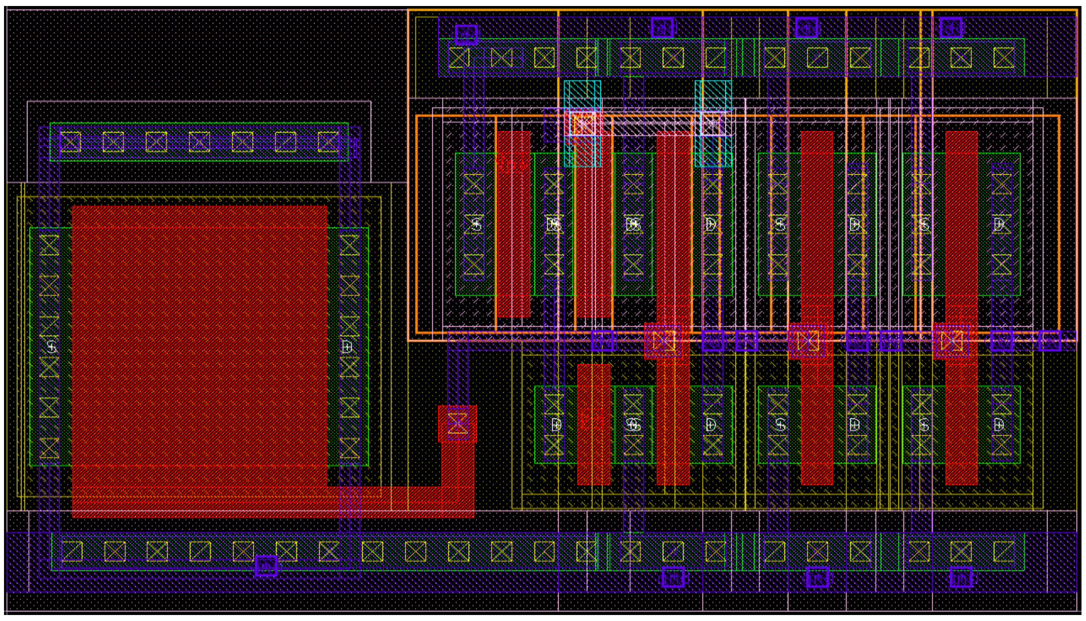
(b)

Figure 6.4: (a) The rail-to-rail operational transconductance amplifier circuit used throughout the design and layout. The OPAMP is based on two differential-pairs - one which conducts in the upper cycle of an input waveform ($M_4 - > M_8$) and another which conducts in the lower cycle ($M_{10} - > M_{14}$). The output stage of the OPAMP ($M_{16} and M_{17}$) is a push-pull amplifier that acts to drive a load. Transistors the transistors in the push-pull output stage are composed each of ten fingers each. The current mirror circuit ($M_1 - > M_3$) receives a DC bias current i_{bias} and generates voltages V_{bn} and V_{bp} which mirror this bias into the tails of the respective differential-pairs. Transistor M_3 has a gate length of 500nm. MOS capacitors M_9 and M_{15} are placed at each of the differential-pair single ended outputs in order to compensate the circuit. b) The layout of the OPAMP circuit. The MOS capacitors M_9 and M_{15} are not shown.

6.1.5 Pulse extender



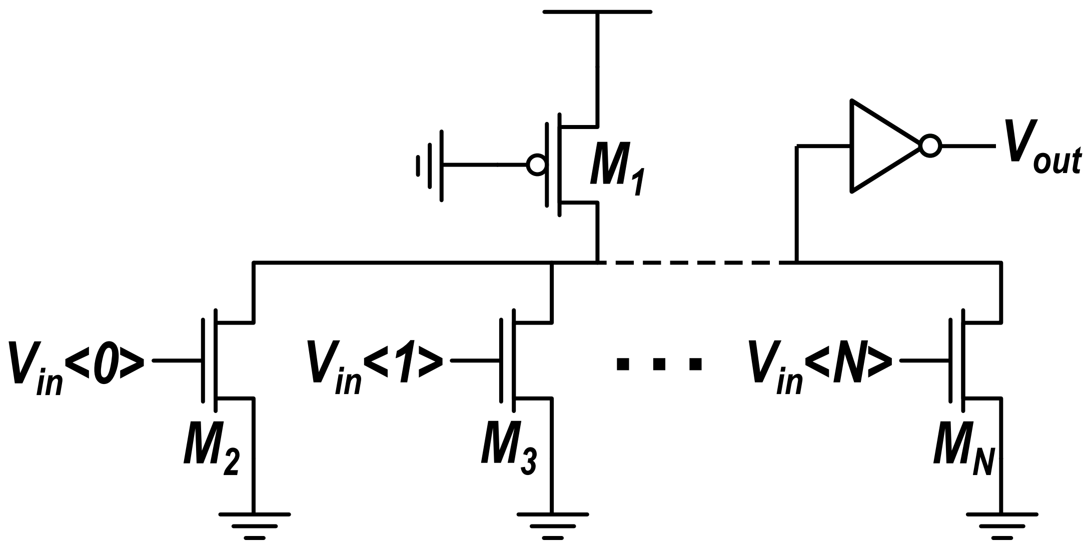
(a)



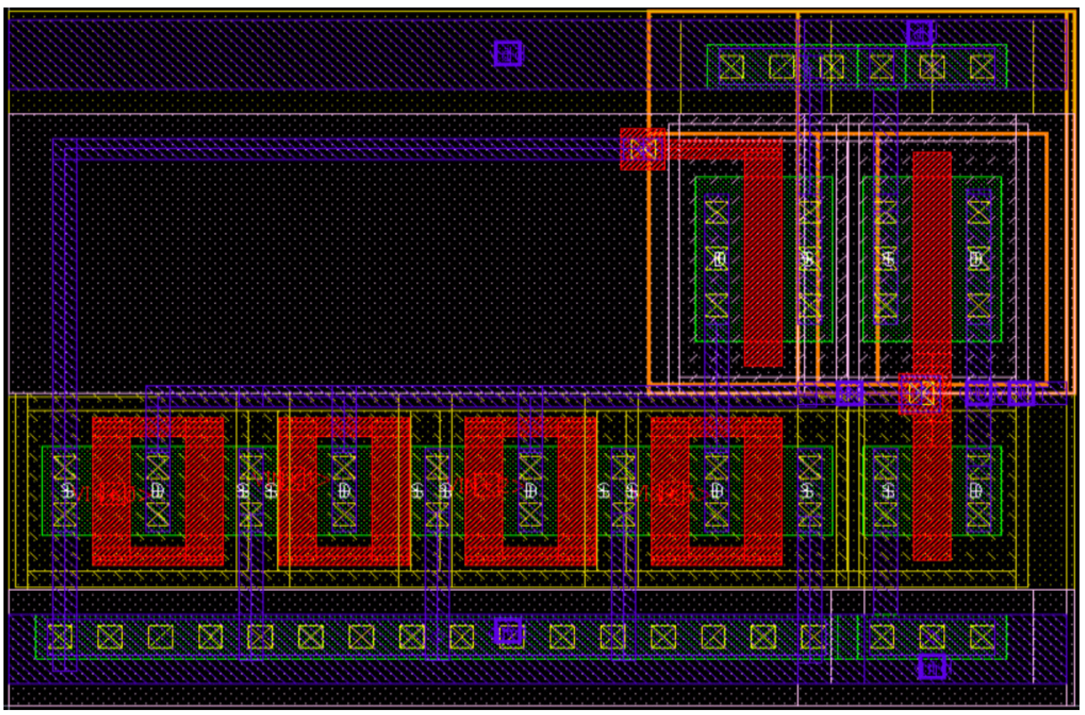
(b)

Figure 6.5: (a) The pulse extender circuit schematic that is used to turn an input spike (V_{in}) into a rectangular pulse of a desired width at the output (V_{out}) - determined by the bias V_{pw}. (b) The pulse extender layout. The structure at the left-hand side of the layout is the MOS capacitor M_4 of the schematic in part (a).

6.1.6 Half OR



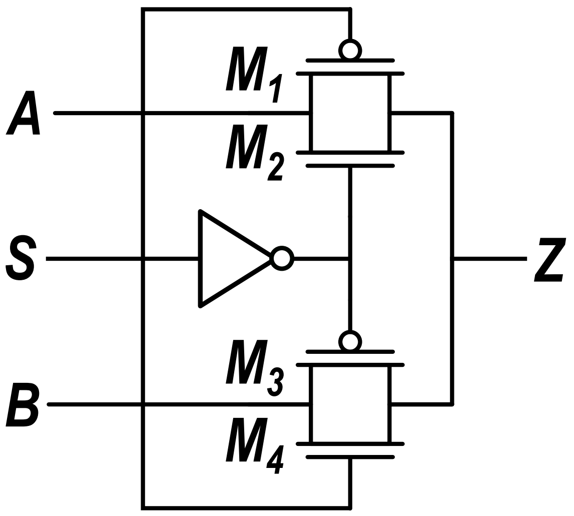
(a)



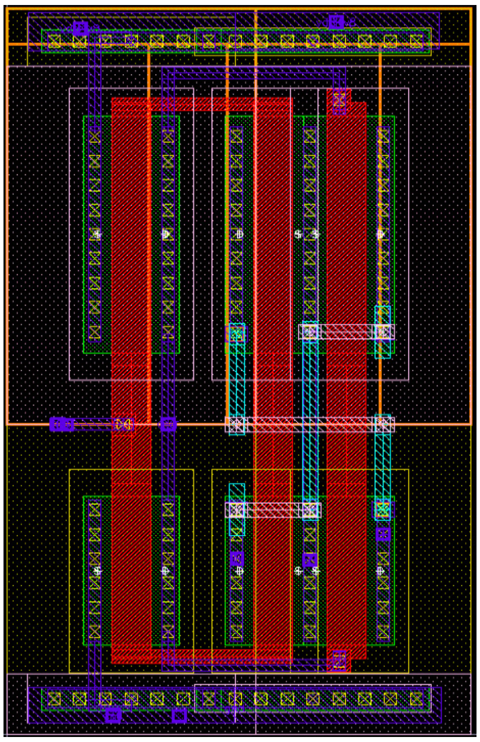
(b)

Figure 6.6: (a) The circuit for the half OR circuit that is used at the input of the DPI-synapse circuit as well as in the delta-modulator circuit. It is called ‘half’ OR since there is no PMOS complement to the logic of the NMOS transistors. This is because, for an OR gate of N inputs, it would be required to stack N PMOS in series - and they would therefore not be saturated for a large number of devices. Instead a single PMOS (M_1) is placed between the inverter input noe and vdd and has its gate tied to gnd - such that it is always conducting. Each of the NMOS transistors is designed with fingers each such that, when an input signal arrives, they are able to sink more current than the PMOS can source (even though they are both turned on) and pull the input node of the inverter down to 0V. (b) The layout of the half OR circuit.

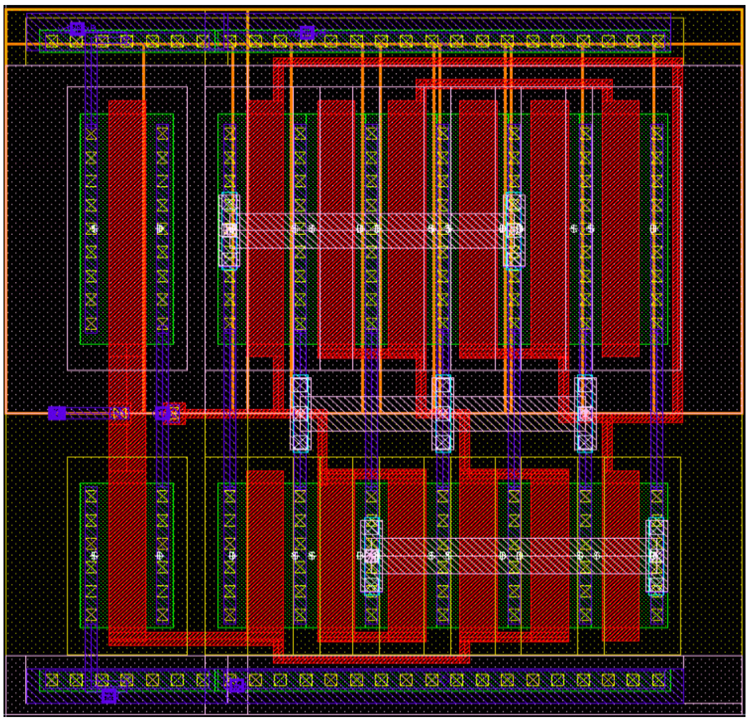
6.1.7 Multiplexer



(a)



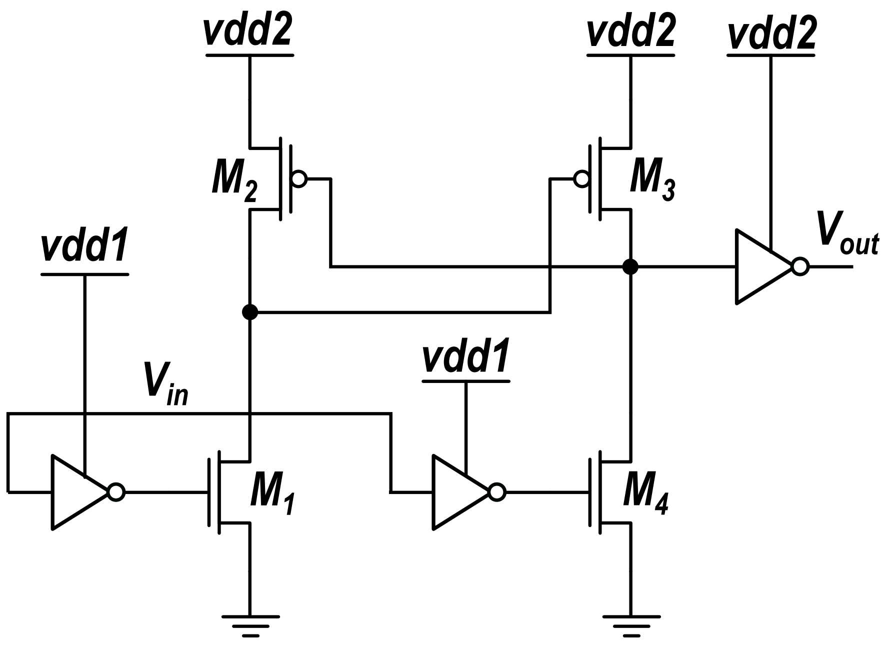
(b)



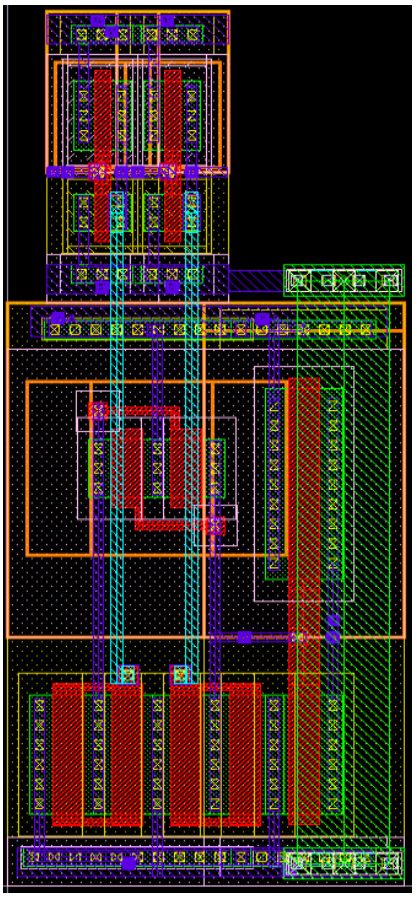
(c)

Figure 6.7: (a) The circuit used to multiplex analogue signals A or B onto Z as a function of the control signal S. The circuit is bidirectional and can pass voltages and currents. The vdd of the circuit is 4.8V. (b) One version of the layout of the circuit of part (a) that is used to drive high impedance loads - namely transistor gates. As a result each transistor has only a single finger. (c) Another version of the layout of the circuit of part (a) that is used to drive low impedance loads - namely programming the 1T1R. As a result each transistor has three fingers.

6.1.8 Level shifter



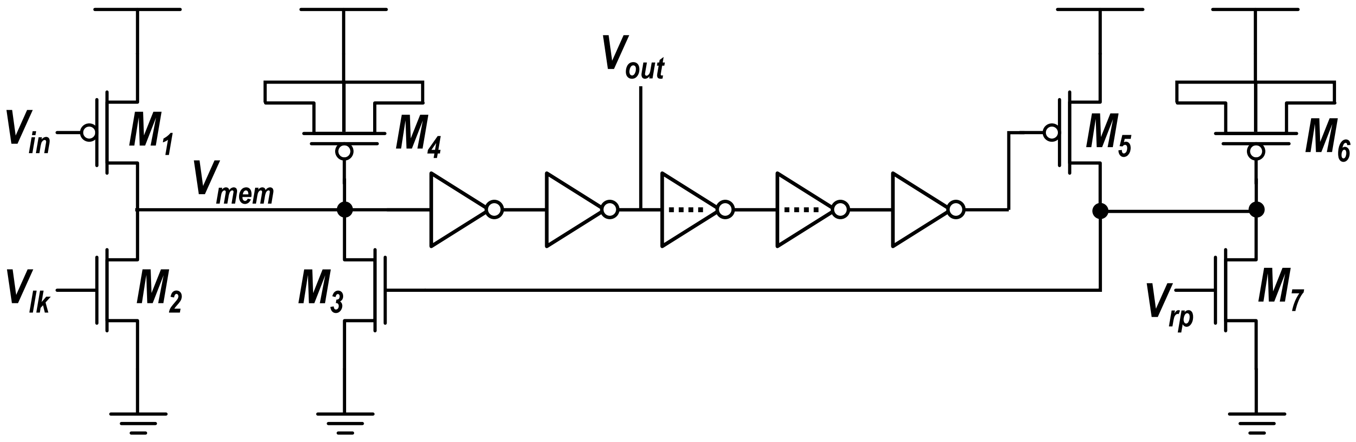
(a)



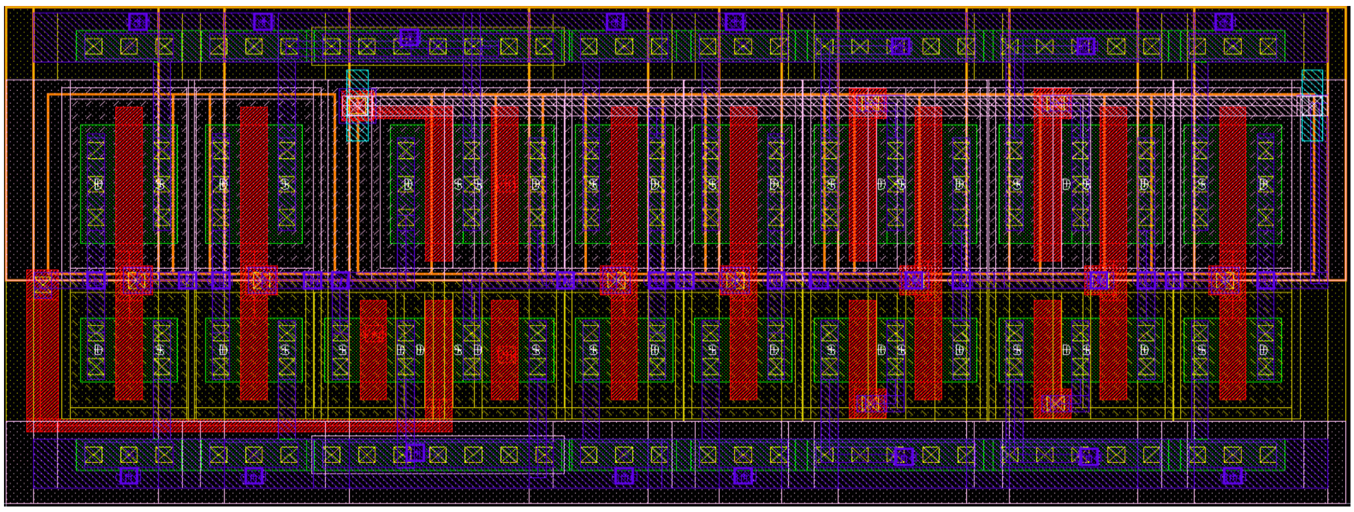
(b)

Figure 6.8: (a) The level shift circuit used to boost 1.2V pulses up to 4.8V pulses. It requires two thin-oxide inverters operating at $vdd1$=1.2V and then two thick-oxide transistor branches and a final thick=oxide inverter with $vdd2$=4.8V. (b) The layout of the level shift circuit. The 1.2V circuits are stacked on top of the 4.8V circuits and their ground are connected together using a vertical running metal4 (green) layer. This layout is a good example of the difference in area consumed by simple thin-oxide (top) and thick-oxide (bottom) circuits.

6.1.9 CMOS neuron



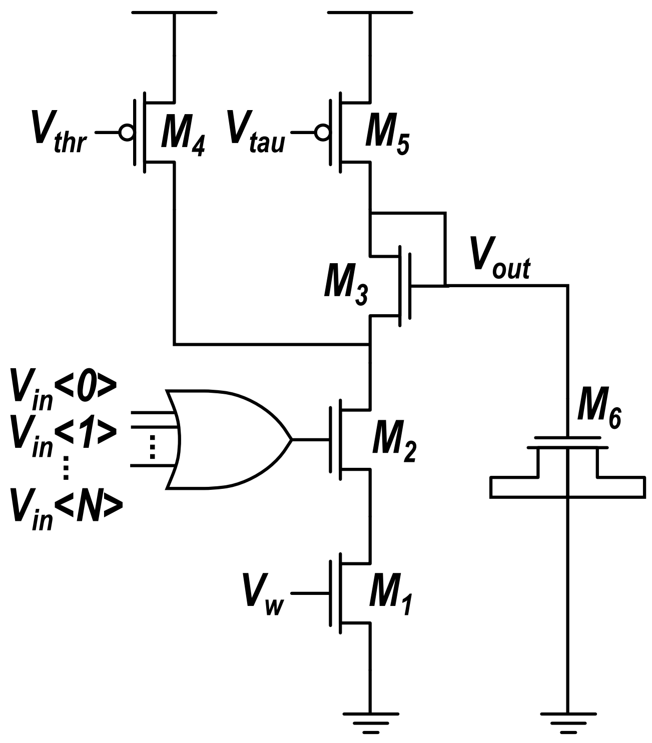
(a)



(b)

Figure 6.9: (a) The schematic of the CMOS neuron circuit used throughout the design and layout. The inverter symbols with the horizontal dashed lines correspond to the starved inverter circuits depicted in Fig. 6.3a. These starved inverter add a small delay between the rising edge of V_{out} and the shunting feedback to transistor M_3 such that is has time to rise to vdd. MOS capacitors M_4 and M_6 are used as the membrane and refractory period capacitors. (b) The layout of the CMOS neuron circuit where the MOS capacitors shown in part (a) are not shown.

6.1.10 CMOS synapse



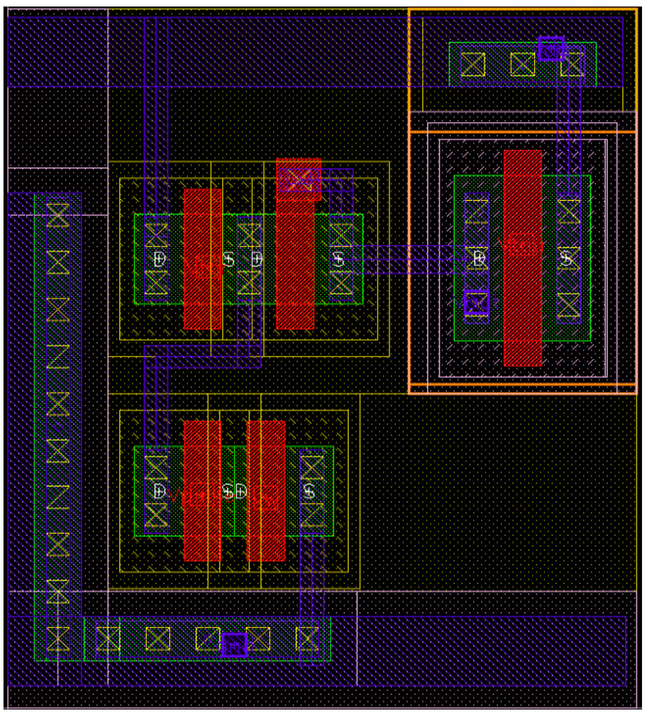
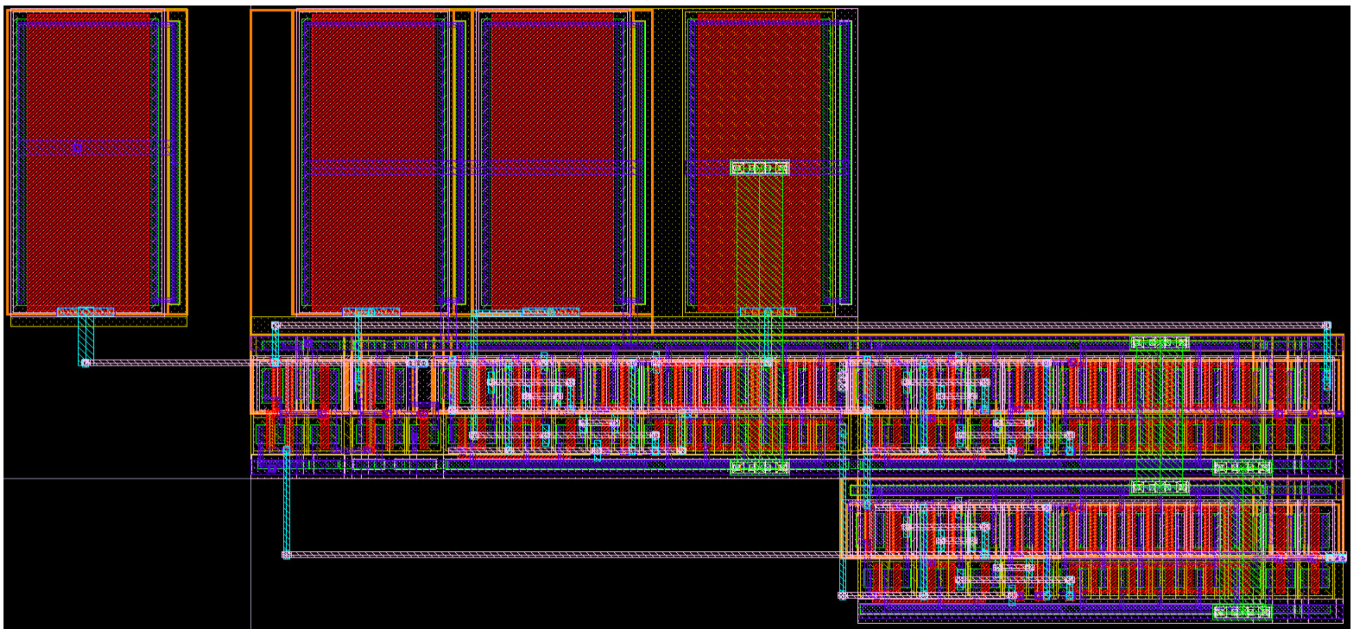


Figure 6.10: (a) Schematic of the CMOS synapse circuit used throughout the design and layout. The pre-synaptic input of the synapse (transistor M_2) is driven by the output of the half OR gate circuit depicted in Fig. 6.6a. In the column circuit, Fig. 6.13a, used throughout the design, the buffered current is mirrored in through M_1 and gated by the same pulse that reads the 1T1R that caused the current by connecting the same signals (V_{in}) as inputs to the half OR gate (see Appendix 6.6a). (b) The layout of the CMOS synapse circuit. MOS capacitor M_6 in part (a) is not shown.

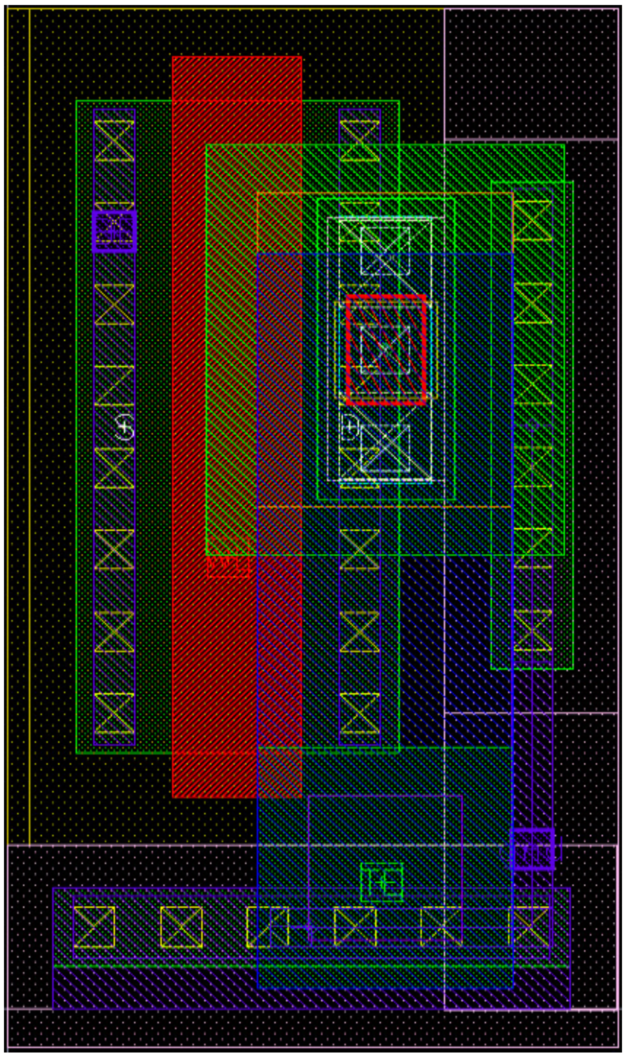
6.1.11 Delta modulator layout



(a)

Figure 6.11: The layout of the delta-modulator circuit discussed in the paper. This image shows the MOS capacitors, characterised by large red rectangular polysilicon gates, that are used to de-couple the input from its DC bias, compensate the voltage follower and determine the maximum event generation rate.

6.1.12 1T1R cell


(a)

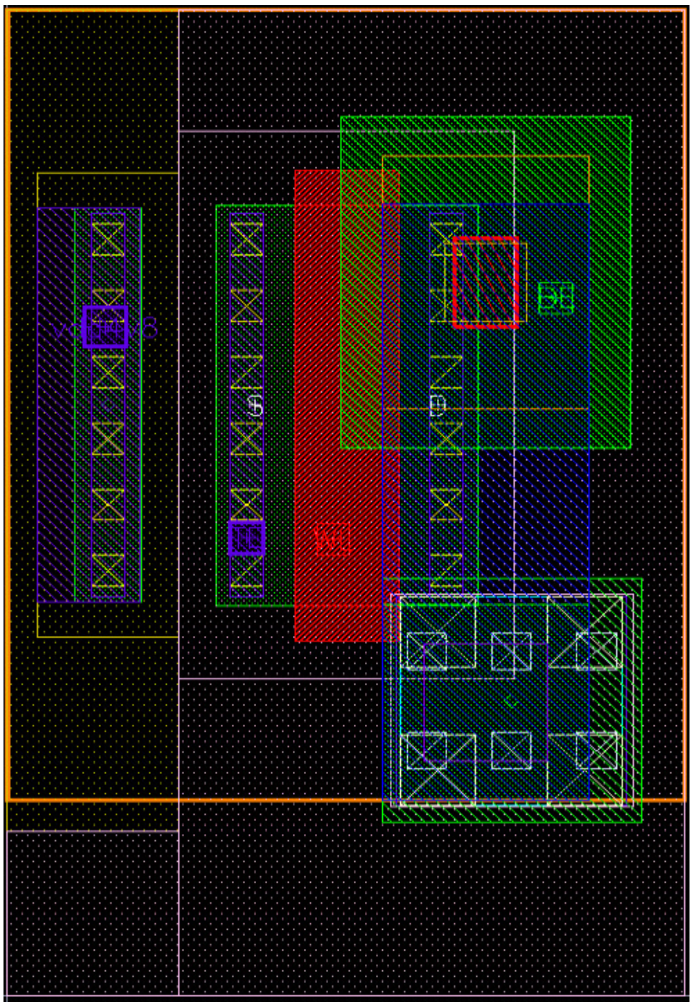
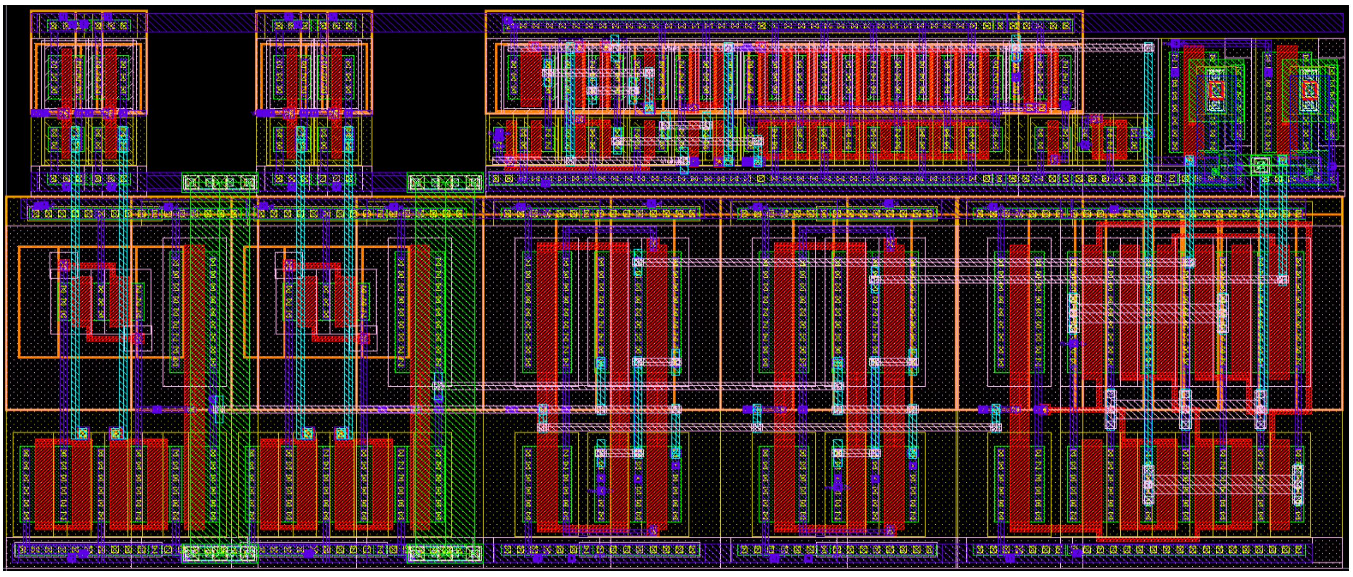

(b)

Figure 6.12: (a) Layout of the series one-transistor-one-resistor structure where the selector transistor is an NMOS that is used for all resistive memory points in the paper. The resistive memory device corresponds to the small red, open rectangle on the upper right of the layout. The structure uses a thick-oxide transistor with dimensions: width=2.72μm, length=500nm. The transistor bulk tap is connected to the common gnd. (b) Layout of the series one-transistor-one-resistor structure where the selector transistor is an PMOS. Although initially design to be used as RRAM devices incorporated into the leak bias of the hybrid synapse circuit it was eventually not used due to issues with biasing its bulk at vdd=4.8V while existing in a vdd=1.2V circuit. The resistive memory device corresponds to the small red, open rectangle on the upper right of the layout. The structure uses a thick-oxide transistor with dimensions: width=2.05μm, length=500nm. The transistor bulk tap is connected to vdd=4.8V.

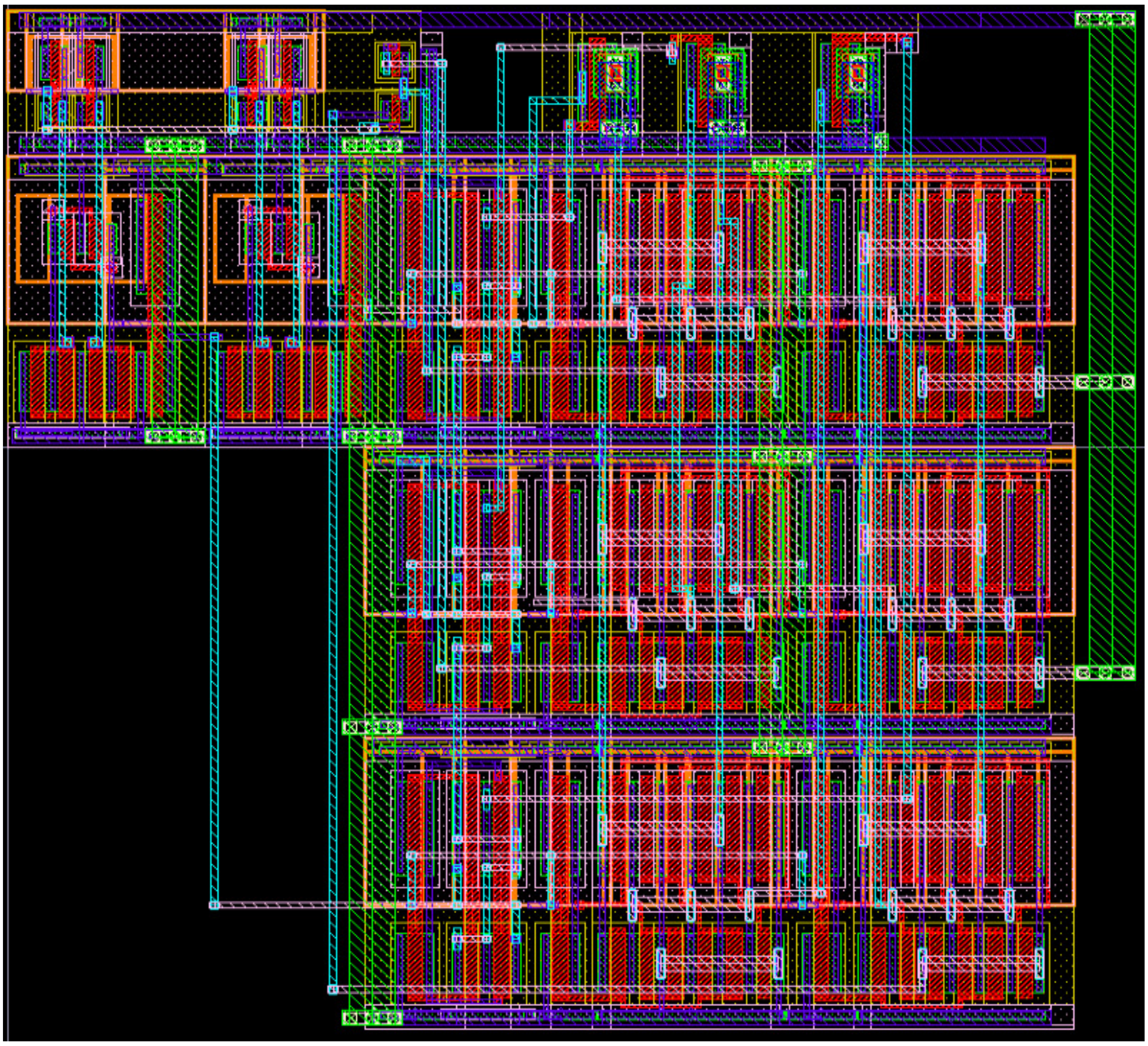
6.1.13 Column circuit layout



(a)

Figure 6.13: The layout of the column circuit used throughout the design and layout that was presented and discussed in the thesis. The top row of circuits operate off the 1.2V supply and the bottom, much larger row, operate of the 4.8V supply. The circuit contains two NMOS 1T1R structures in its top right corner and requires two gate driving multiplexers for each gate and one larger multiplexer that switches the connections of the top electrode.

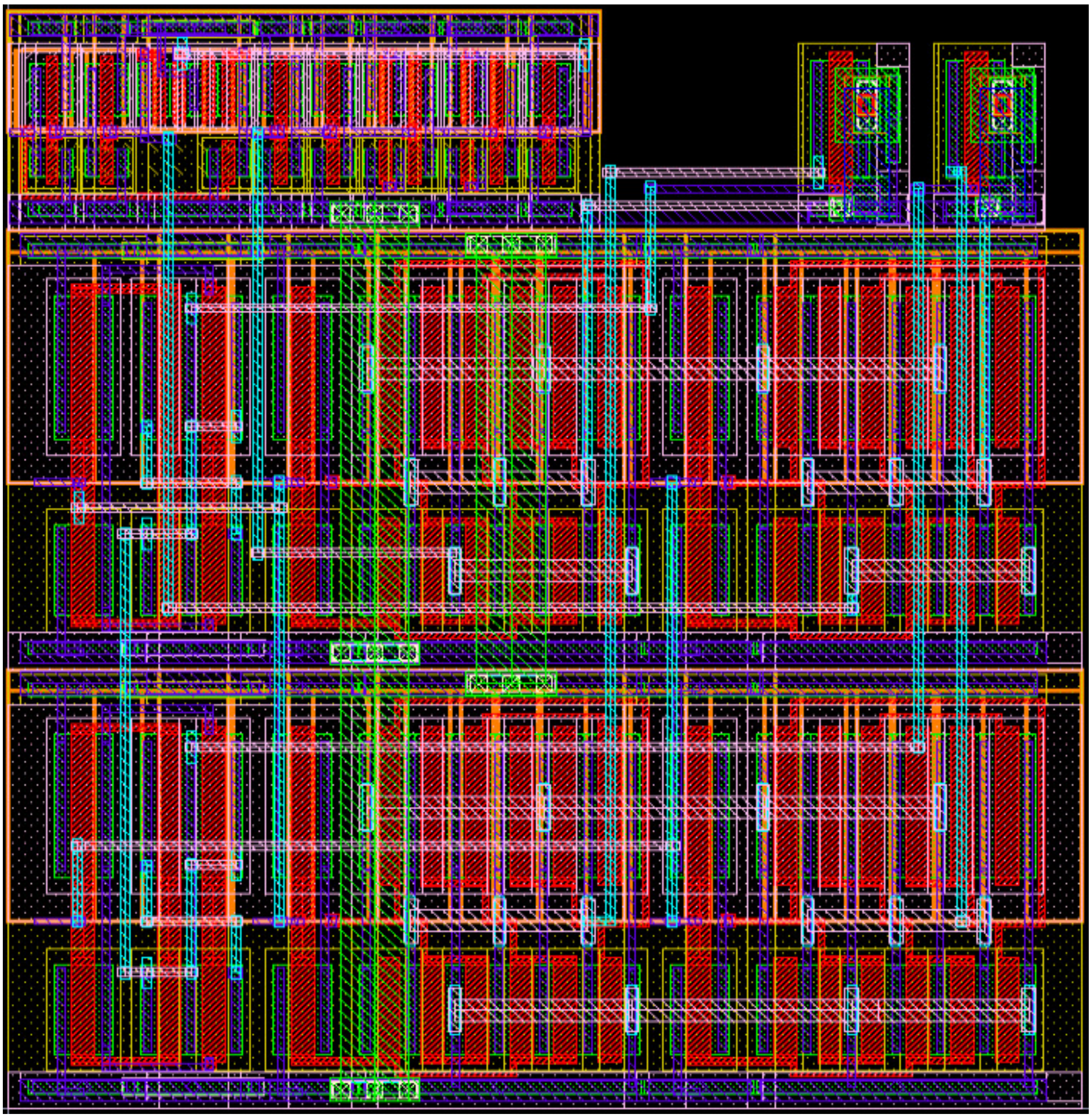
6.1.14 Hybrid synapse layout



(a)

Figure 6.14: The hybrid synapse layout containing three NMOS 1T1R structures in the top right corner of the images and three rows of gate, top and bottom electrode multiplexers. The transistors of the synapse circuit itself are found in the centre of the top 1.2V row of the circuit and constitutes a relatively tiny area compared with the area consumed by the multiplexing and level shifting circuits. The MOS capacitor circuits are not shown.

6.1.15 Hybrid neuron layout



(a)

Figure 6.15: The hybrid neuron circuit containing two NMOS 1T1R devices and two rows of multiplexing circuits to switch these structures in and out of the neuron circuit which is found in the top left corner of the layout. The MOS capacitor circuits are not shown.)

6.2 Appendix - Note on MOS capacitors

The hybrid CMOS-RRAM process used to design and layout the circuits of this thesis consists of a standard 130nm CMOS layer which is processed as normal up to metal3, finished off by depositing resistive memory device stacks between metal4 and metal5. Normally the process should have capacitors available between metal5a and metal6 but in this case they are not added. Therefore, in order to obtain capacitors it is required to use either metal-insulator-metal capacitors, realised with large overlapping metal layers, or MOS capacitors. MOS capacitors are more practical since they can be realised by shorting the source, drain and bulk of a MOS transistors. The transistor can then be sized to give a desired capacitance. However, MOS capacitance is highly non-linear as a function of the applied bias, V_{gs}. For a V_{gs} lower than the threshold voltage of the transistor, the top and bottom capacitor plates are the gate electrode and the bulk of the transistor. However, when V_{gs} exceeds this threshold, the channel of the transistor begins to conduct and acts as the new dominant capacitor plate - reducing the effective plate separation distance considerably. Therefore as V_{gs} is increased, and channel conduction becomes more dominant, the capacitor increases non-linearly as seen in the green curve Fig. 6.16a obtained through SPICE simulation of an NMOS capacitor. As V_{gs} is made increasingly negative a similar effect is also seen in the orange curve of Fig. 6.16a, most likely due to the accumulation of minority carriers in the channel. A number of curves are shown as the MOS area is increased. The effect of MOS capacitor area on the capacitance value is summarised in Fig. 6.16b under two V_{gs} biases.

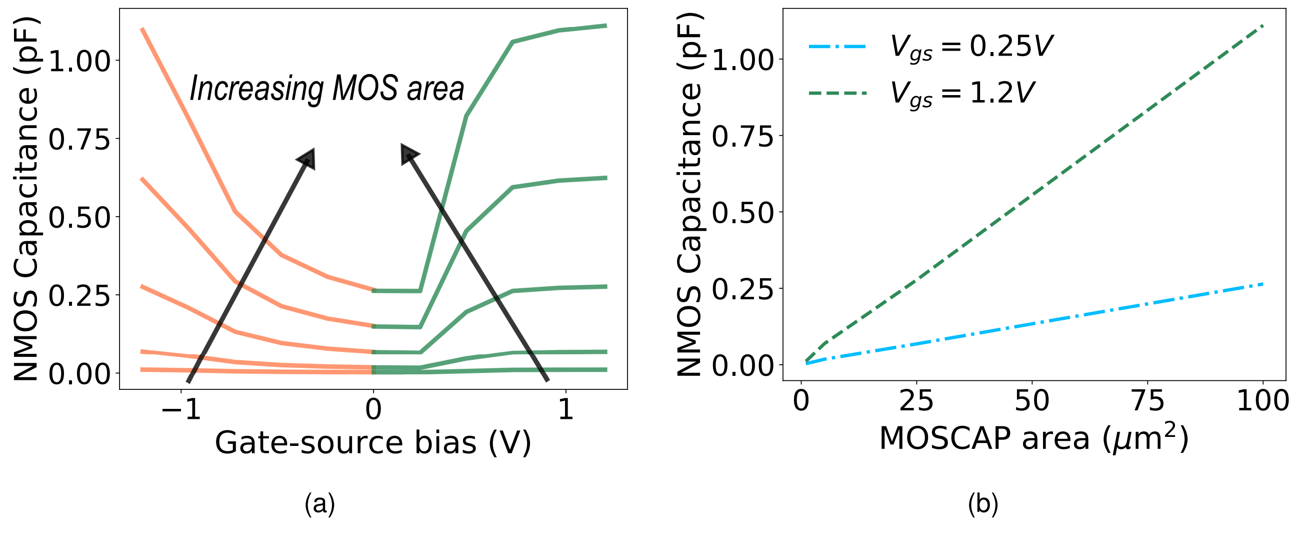


Figure 6.16: (a) Five curves showing the relationship between simulated NMOS capacitance and the gate-source bias (V_{gs}). Green curves show the relationship for positive bias and orange ones for negative bias. The MOS cap areas are increased in the following range: 1μm^2, 5μm^2, 25μm^2, 56μm^2 and 100μm^2. (b) The relationship between MOSCAP area and the resulting capacitance value under two bias conditions (green $V_{gs} = 0.25$V and blue $V_{gs} = 1.2$V) for an NMOS device.

The SPICE simulations consisted of applying a DC bias (V_{gs}) over a NMOS transistor with a shorted drain, source and bulk and then injected a current pulse of known magnitude onto the capacitor. Using the equation;

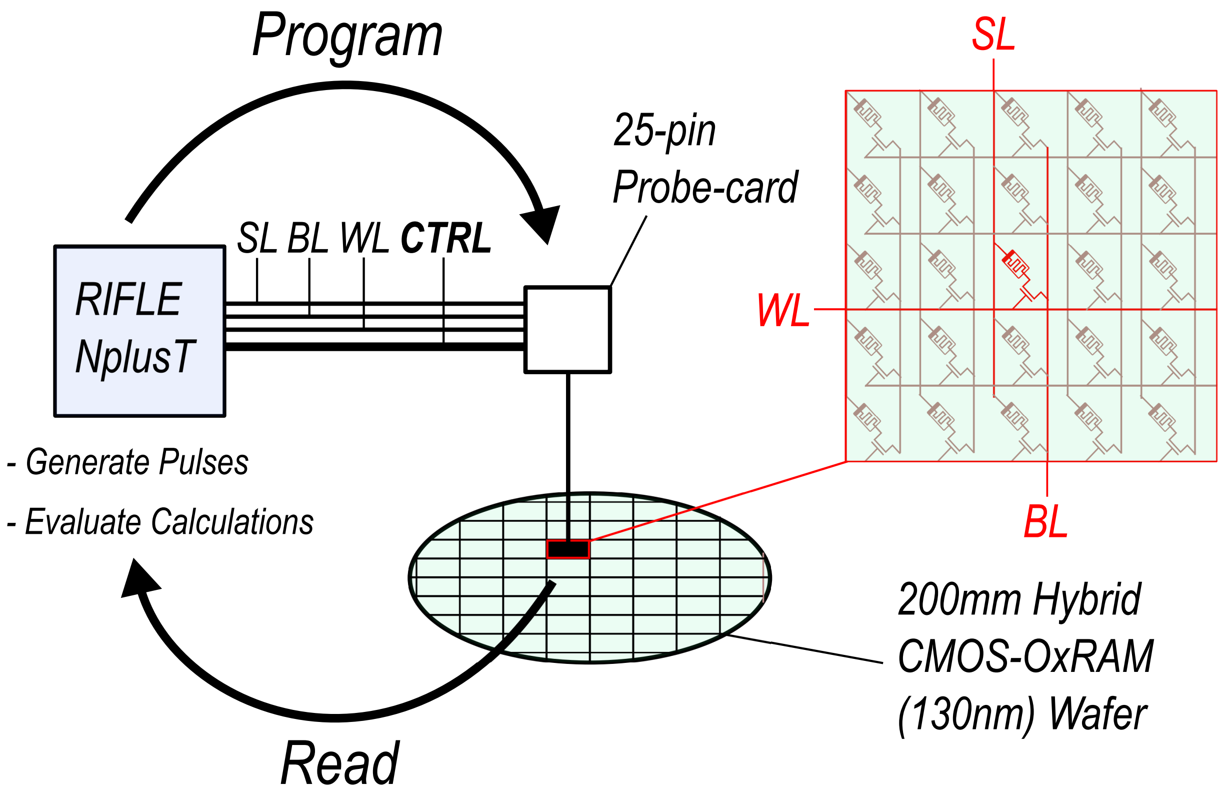
$$i = C\frac{dV}{dt}, \tag{6.1}$$

where i is the magnitude of the current pulse, C is the capacitance of the MOS capacitor and dV/dt is the rate of change of voltage over the capacitor. By measuring the slope dV/dt, resulting from an input current pulse, the capacitance is given by the ratio of the two quantities.

6.3 Appendix - Resistive memory experiments

Two versions of fabricated OxRAM memory arrays are used in the presentation of this thesis. The first is a $4,096$ (4k) device array (16×256 devices) of 1T1R structures. The second chip is a $16,384$ (16k) device array (128×128 devices) of 1T1R structures. In each array, the OxRAM cell consists of a HfO_2 thin-film sandwiched in a TiN/HfO_2/Ti/TiN stack. The HfO_2 and Ti layers are 5 nm tick and have a mesa structure 300 nm in diameter. The OxRAM stack is integrated into the back-end-of-line of a commercial 130 nm CMOS process. In the 4k device array the n-type selector transistors are 6.7μm wide. In the 16k array the n-type selector transistors are 650nm wide. Voltage pulses, generated off chip, can be applied across specific source (SL), bit (BL) and word lines (WL) which contact the OxRAM top electrodes, transistor sources and transistor gates respectively. External control signals determine to what compliment of SL, BL and WL the voltage pulses are applied over by interfacing with CMOS circuits integrated with the arrays. Signals for the 4k device array are generated using the Keysight B1530 module and those for the 16k device array are generated by the RIFLE NplusT engineering test system. The RIFLE NplusT system can also run C++ programs that allow the system to act as a computer-in-the-loop with the 16k device array as depicted in Fig. 6.17a.

Before either chip can be used, it is required to form all the devices in the array. In the forming process oxygen vacancies are introduced into the HfO_2 thin-film through a voltage-induced dielectric breakdown. This is achieved by selecting devices in the array one at a time, in raster scan fashion, and applying a voltage between the source and bit lines. At the same time, the current is limited to the order of μAs by simultaneously applying an appropriate V_{WL} (transistor gate) voltage. A form operation consists of the following conditions; 4k device array - V_{SL}=4V, V_{BL}=0V, V_{WL}=0.85V, 16k device array - V_{SL}=4V, V_{BL}=0V, V_{WL}=1.3V. After the devices have been formed, they are conditioned by cycling each device in the array between the LCS and the HCS one hundred times. Unless otherwise specified, the standard RESET conditions throughout the paper were; 4k device array - V_{SL}=0V, V_{BL}=2.5V, V_{WL}=3V, 16k device array - V_{SL}= 0V, V_{BL}= 2.6V, V_{WL}=4.0V. Unless otherwise specified, the standard SET conditions used were; 4k device array - V_{SL}=2V, V_{BL}=0V, V_{WL}=1.2V, 16k device array - V_{SL}=2.0V, V_{BL}=0V, V_{WL}=1.6V. The device conductances are determined by measuring the voltage drop over a known low-side shunt resistance connected in series with the selected SL in a read operation. Devices are read according to the following conditions; 4k device array - V_{SL}=0.1V, V_{BL}=0V, V_{WL}=4.8V, 16k device array - V_{SL}=0.4V, V_{BL}=0V, V_{WL}=4.0V. All off-chip generated voltage pulses for programming and reading have a pulse-width of 1μs.

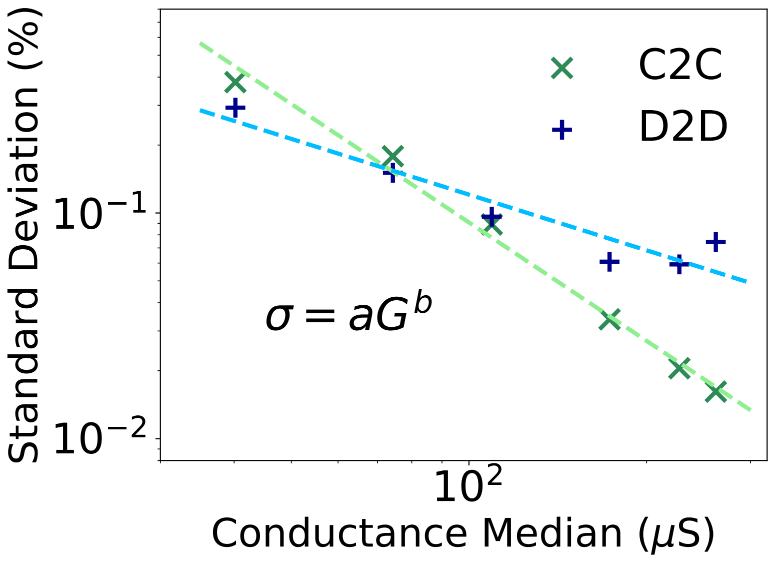


(a)

Figure 6.17: **Diagram of the computer-in-the-loop with the resistive memory array.** The computer-in-the-loop experiment using the RIFLE NplusT engineering test system, which incorporates a digital sequencer, 100 MHz arbitrary waveform generators, 70Msample/sec cell current measurement capability, as well as a C++ programmable computer. The computer can configure pulses applied by the arbitrary waveform generator to source (SL), bit (BL), and word lines (WL) signals. The CTRL bus controls periphery circuitry integrated within the hybrid CMOS-OxRAM wafer and determines to which device in the array the SL, BL and WL signals will be applied to (here the red coloured 1T1R structure). All these signals are interfaced to our 200 mm (8-inch) wafer through a 25-pin probe-card, which contacts to 25 metal pads integrated on top of the back-end-of-line of the wafer. Using this setup, the computer is therefore able to program the conductance states of devices integrated in the array, read the resulting states, and then, based on these results, re-program the devices in the array in a continuous program-read loop. In this fashion, we can implement RRAM-based algorithms, whereby programming operations can be determined based on the feedback of the result of the previous programming steps. In the case of the RRAM-based MCMC sampling algorithm implemented in the paper, after reading programmed device conductance values, the computer is then responsible for calculating the dot-product between the data points and the conductance model, evaluating the likelihood of the conductance model and finally performing the acceptance ratio calculation. The ultimate aim of such a system is to experimentally verify RRAM-based algorithms, before a full system can be designed and integrated on a standalone chip.

6.4 Appendix - Implementation of RRAM-based MCMC simulator

To characterise properties of the HfO_2 physical random variable, as plotted in the second chapter, the 4k device array chip was used. This allows use of a larger selector transistor which offers a greater range of the SET programming currents. A 4k device array was formed, conditioned and then RESET/SET cycled one hundred times over a range of nine word line voltages (V_{WL}), corresponding to the range of SET programming currents. Each device was read after each SET operation. Between each step in V_{WL}, the devices were additionally RESET/SET cycled 100 times under standard programming conditions ensuring that, for each of the hundred cycles at different V_{WL}, the initial conditions were the same. The conductance was read after each SET operation and recorded. This conductance data was then processed using the python library NumPy and then using the curve fit tool from SciPy giving rise to the relationships between cycle-to-cycle and device-to-device variability plotted in Fig. 6.18a. The relationships between the cycle-to-cycle and device-to-device variability, σ, and the conductance median of the respective distributions were fit using a power law - the equation in Fig. 6.18a, giving parameters a and b.



(a)

Figure 6.18: The relationship between the standard deviation, σ, and the conductance median for the cycle-to-cycle (blue crosses) and device-to-device (green plus symbols) variability distributions. Each can be fit with a power law, $\sigma = aG^b$, where a and b can be used to model this physical relationship in a simulation.

A custom behavioural simulation of the computer-in-the-loop experiment was developed in python implementing the presented Metropolis-Hastings Markov Chain Monte Carlo sampling algorithm using these two power law fits. The proposal distributions were assumed to be normal distributions with a standard deviation equal to that of the cycle-to-cycle variability fit. Each time the simulator moves between array rows it assigns a mean to each of the proposal distributions of each of the devices in the new row using the device-to-device variability power law. Upon each new proposal each device samples from a normal distribution around this mean. The sampling was performed using the random function suite from the NumPy library. The likelihood and acceptance ratio calculations were performed in the log-domain as described in chapter 2 of the thesis.

6.5 Appendix - Note on implementation of neural network models

A variety of tools were used to realise the different types of neural network models applied across the chapters. The PyTorch [218] deep learning framework was used in chapter 1 to implement the TAG model and the benchmark models. In chapter 2, TensorFlow [396] was used to implement the benchmark neural networks that the RRAM-based MCMC experimental models were compared to. In chapter 2, the PyMC3 framework was used to train and implement the Bayesian neural network models [397]. In the first section of chapter 3, the spiking neural network simulation library Brian2 [398] was used to implement the intrinsic plasticity models while a custom spiking neural network simulator was developed based on the odeint library, available from the SciPy python package, to perform the simulations in the second section of chapter 1 and the second section of chapter 3.

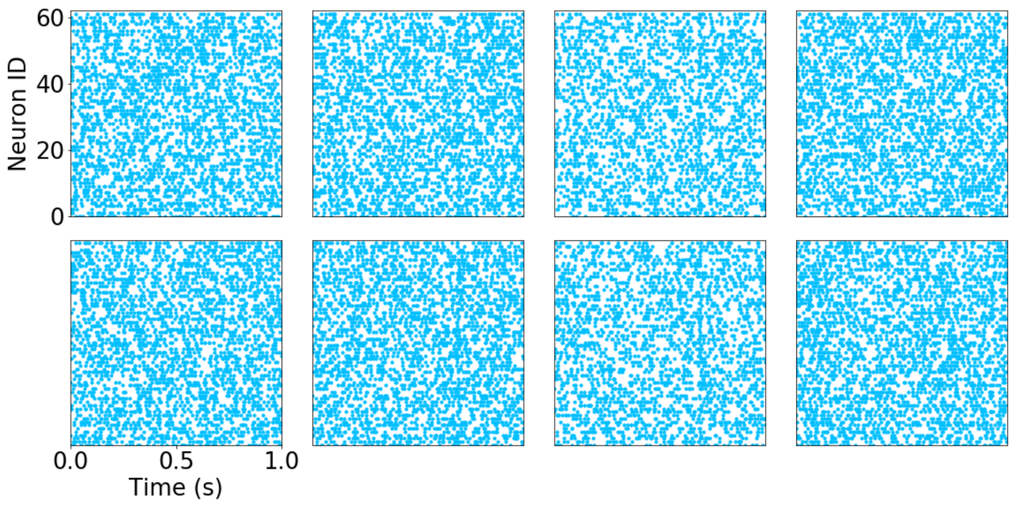
6.6 Appendix - Pre-processing of the electrocardiogram dataset

For certain demonstrations in the second and third chapter addressing arrhythmic heartbeat detection, the MIT-BIH heart arrhythmia database [296], composed of half hour dual-channel electrocardiogram recordings from 48 subjects, was downloaded. Single heartbeats were isolated in each recording into a 700 ms time-series and centred on the R-wave peak of the beat. The fast Fourier transform function from NumPy was then used to generate a frequency spectrum of each time-series. In the first section of chapter two, the ten lowest frequency components were used as the input features for the RRAM-based Bayesian neural network that was trained experimentally using RRAM-based MCMC sampling. Each heartbeat data-point was then either labelled as either a normal, healthy, heartbeat or as a heartbeat exhibiting signs of arrhythmia. The dataset contains observations of tens of arrhythmia types. A subset of 250 data-points were taken randomly from 47 of the subjects and then used to train the models. Then, to test the models, all of the data-points from one, previously unseen, subject were used - subject 200.

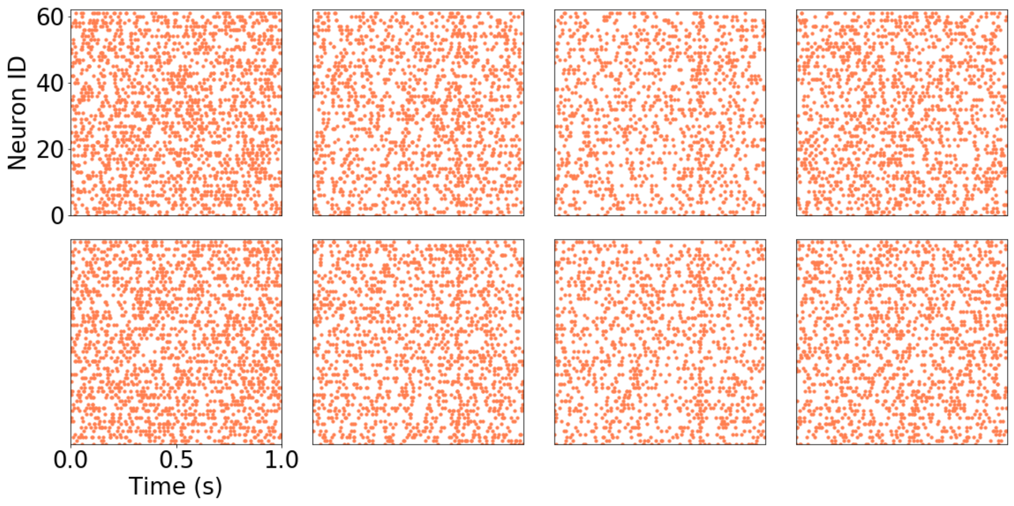
In the second section of chapter two, the dataset was used again to demonstrate that an ex-situ trained RRAM-based Bayesian neural network could say ‘I don’t know’. In this case, normal heartbeats and the four most classes of arrhythmia were used - left and right blocked branches, atrial premature beats and premature ventricular contractions [296]. The sixty-four ‘best’ frequency components over the full dataset were selected using the Chi2 best-K feature selection algorithm [294]. Four hundred examples of each heartbeat were randomly sampled from all forty-eight patients allowing a test and training set of one thousand data points each.

In the final section of chapter three, delta-modulated time-series were generated using all of the heartbeats of subject 200 - the same patient used as the test case for the RRAM-based Bayesian neural network trained using RRAM-based MCMC sampling. Each of the 700ms time-series was delta-modulated by measuring, at what timestamps, the signal deviated up or down by a threshold value. When a new UP or DN event was registered, the magnitude of the signal is recorded and the each new subsequent magnitude value is compared with this.

6.7 Appendix - Cercal system statistical model response

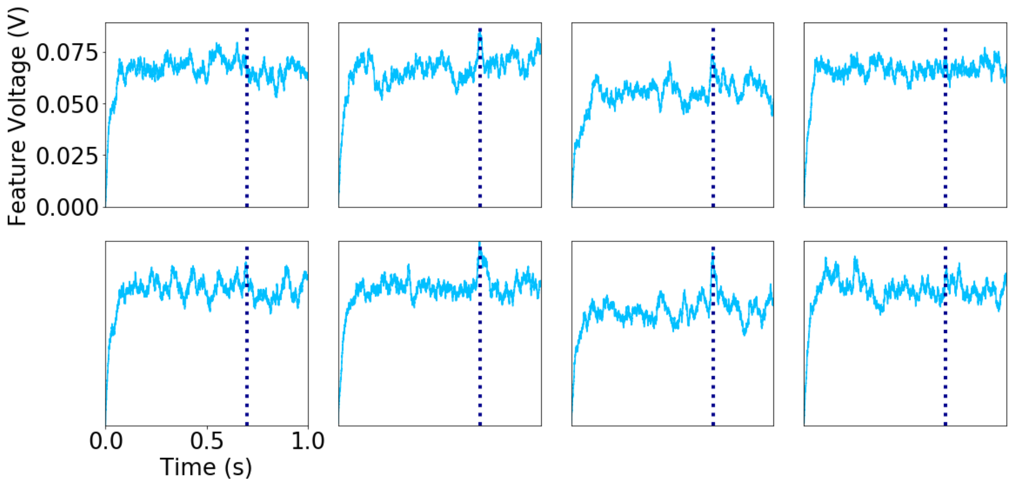


(a)

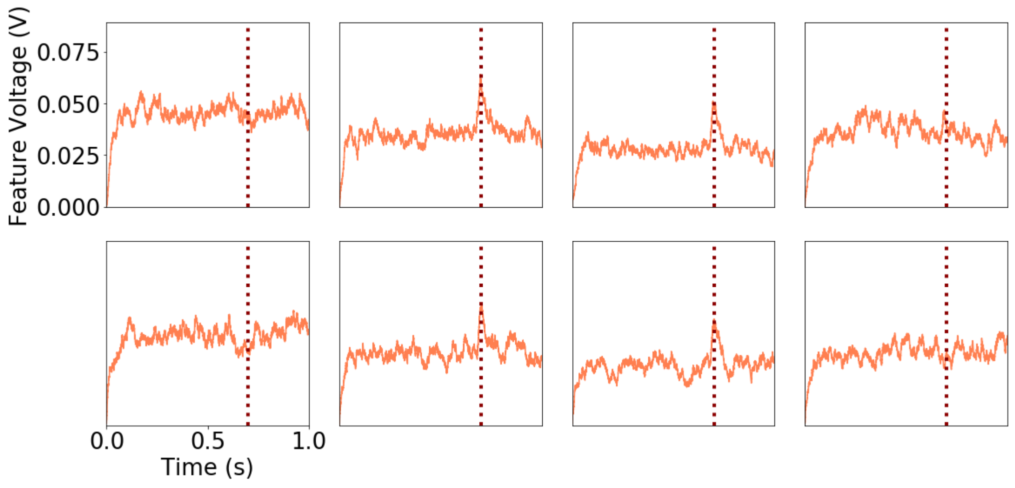


(b)

Figure 6.19: The raster plots of all of the sensory neurons, corresponding to those nested below the hairs, due an ‘easy’ to detect simulated attack with low background air intensity and a large fast attacker. From left to right the columns show the activity of the hairs oriented to 45°, 135°, 225°and 315°. The top row correspond to the hairs on the left cerci and the bottom row to the hairs on the right cerci. (a) As blue points, the spike-times of all of the slow (low frequency) hair sensory neurons are plotted. (b) As orange points, the spike-times of all of the fast (high frequency) hair sensory neurons are plotted.

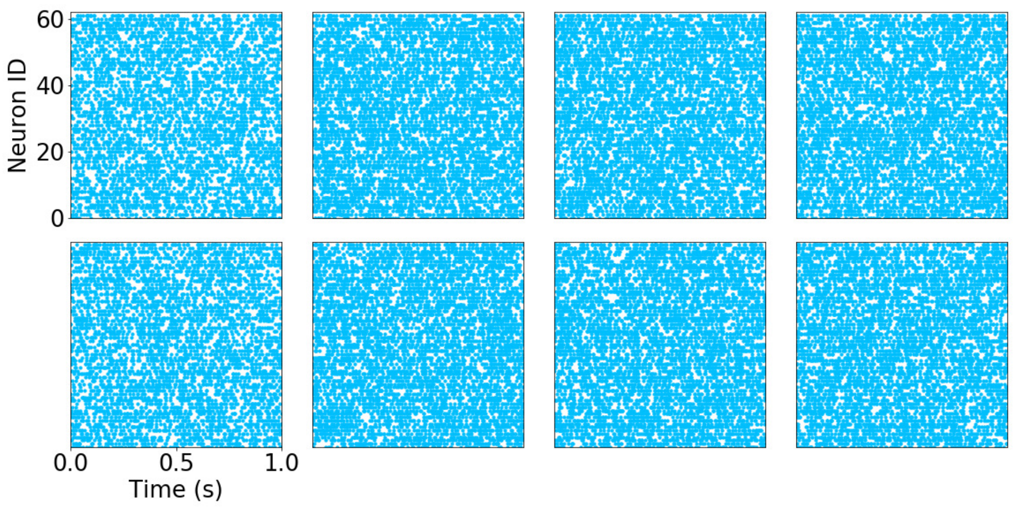


(a)

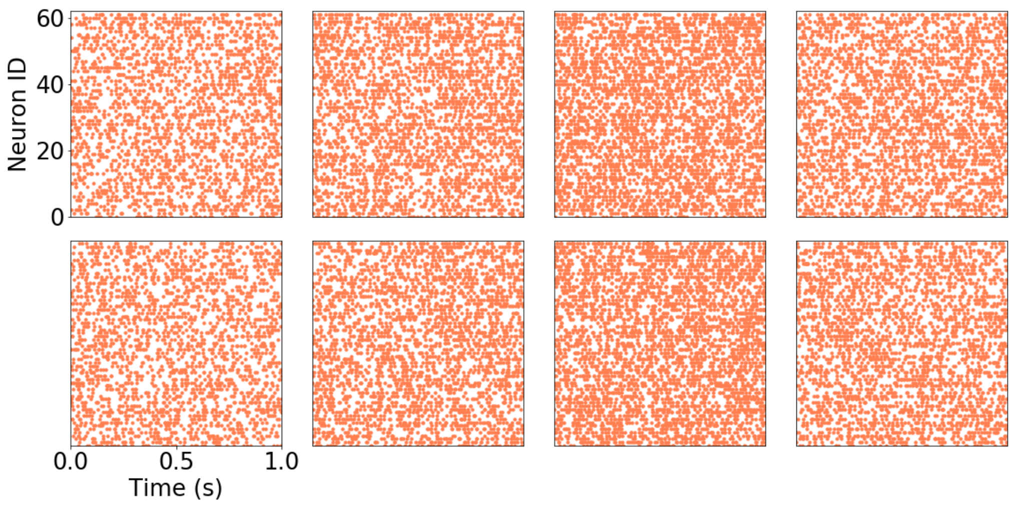


(b)

Figure 6.20: The membrane voltage plots of all of the sensory neurons, corresponding to those nested below the hairs, due the same ‘easy’ to detect simulated attack, with low background air intensity and a large fast attacker, that generated the raster plot in Appendix 6.19a. From left to right the columns show output voltages of the input layer neurons being excited by the hair populations oriented to 45°, 135°, 225°and 315°. The top row correspond to the input layer neurons receiving input from the hair populations on the left cerci and the bottom row to the populations on the right cerci. The vertical line drawn at 700ms denotes the attack time. The output voltages at this time, of all sixteen neurons, are used as the features that describe each data point in the training and testing of the TAG model in chapter 2.2. (a) As a continuous blue signal, the output voltages, denoted as feature voltage in the plot to denote the fact they are used as input features to the TAG model, of all of the slow (low frequency) input neurons are plotted. (b) As a continuous orange signal, the output voltages, denoted as feature voltage in the plot to denote the fact they are used as input features to the TAG model, of all of the fast (high frequency) input neurons are plotted.

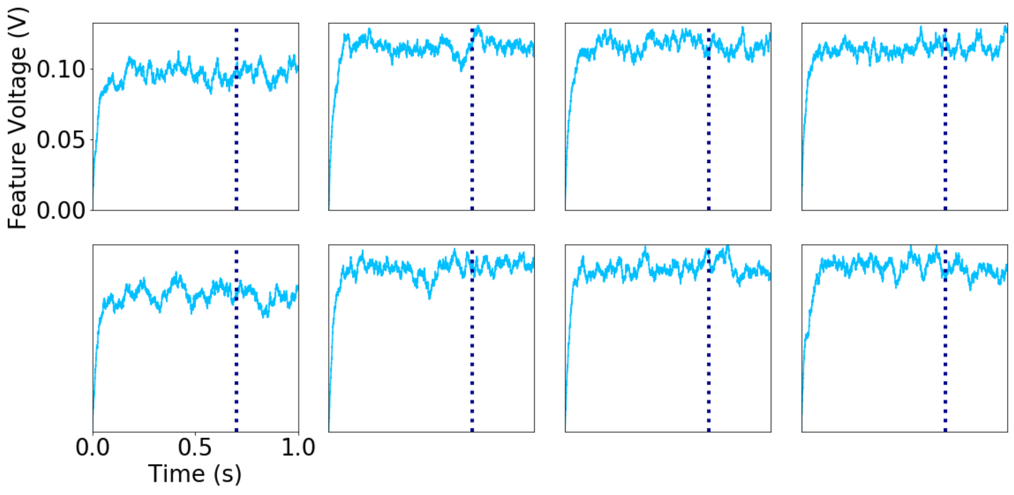


(a)

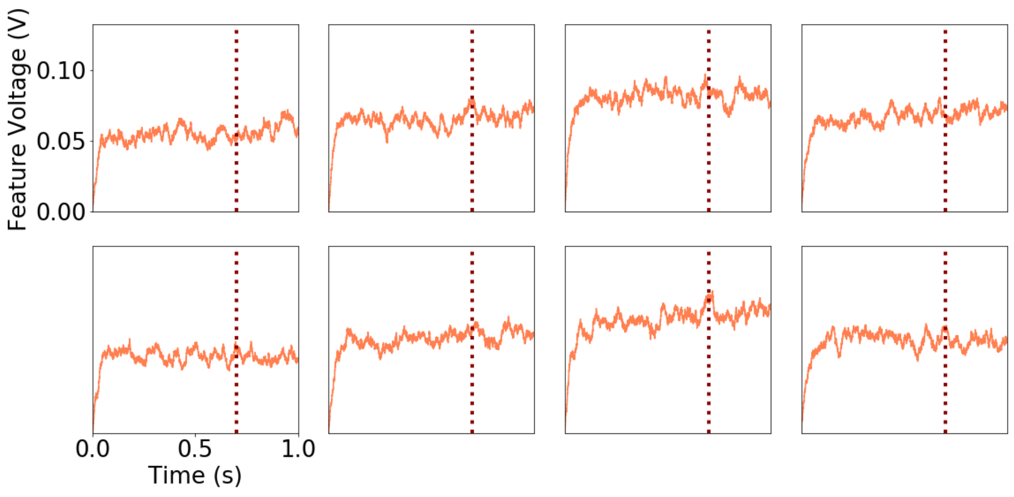


(b)

Figure 6.21: The raster plots of all of the sensory neurons, corresponding to those nested below the hairs, due an 'hard' to detect simulated attack with high background air intensity and a smaller slow attacker. From left to right the columns show the activity of the hairs oriented to 45°, 135°, 225°and 315°. The top row correspond to the hairs on the left cerci and the bottom row to the hairs on the right cerci. (a) As blue points, the spike-times of all of the slow (low frequency) hair sensory neurons are plotted. (b) As orange points, the spike-times of all of the fast (high frequency) hair sensory neurons are plotted.



(a)



(b)

Figure 6.22: The membrane voltage plots of all of the sensory neurons, corresponding to those nested below the hairs, due the same 'hard' to detect simulated attack, with low background air intensity and a large fast attacker, that generated the raster plot in Appendix 6.21b. From left to right the columns show output voltages of the input layer neurons being excited by the hair populations oriented to 45°, 135°, 225°and 315°. The top row correspond to the input layer neurons receiving input from the hair populations on the left cerci and the bottom row to the populations on the right cerci. The vertical line drawn at 700ms denotes the attack time. The output voltages at this time, of all sixteen neurons, are used as the features that describe each data point in the training and testing of the TAG model in chapter 2.2. (a) As a continuous blue signal, the output voltages, denoted as feature voltage in the plot to denote the fact they are used as input features to the TAG model, of all of the slow (low frequency) input neurons are plotted. (b) As a continuous orange signal, the output voltages, denoted as feature voltage in the plot to denote the fact they are used as input features to the TAG model, of all of the fast (high frequency) input neurons are plotted.

6.8 Appendix - Observed shift of the OxRAM normal random variable

An interesting device behaviour not previously documented in the literature was observed under continual RESET/SET cycling whereby the OxRAM normal random variable, leveraged in chapter two, was seen to shift, expand shift and contract. Each subsequent set of one hundred reads in the high conductance states were fit with a normal distribution. In the plots of Fig. 6.23 ten sets of one hundred cycles, coloured from red (first one hundred cycles) to violet (last one hundred cycles), from two separate devices in an array are plotted in order to show how the OxRAM random variable changes in time. While in Fig. 6.23a this shift is minimal, the properties of the random variable of the device plotted in Fig.6.23b exhibit considerable, although gradual, change: the median shifts towards a lower conductance and the standard deviation of the distribution becomes larger. While not discussed in the thesis, this 'random walk' behaviour of the OxRAM random variable could well be exploited within the context of Markov Chain Monte Carlo sampling where, if characterised properly, could also harnessed as a computational resource.

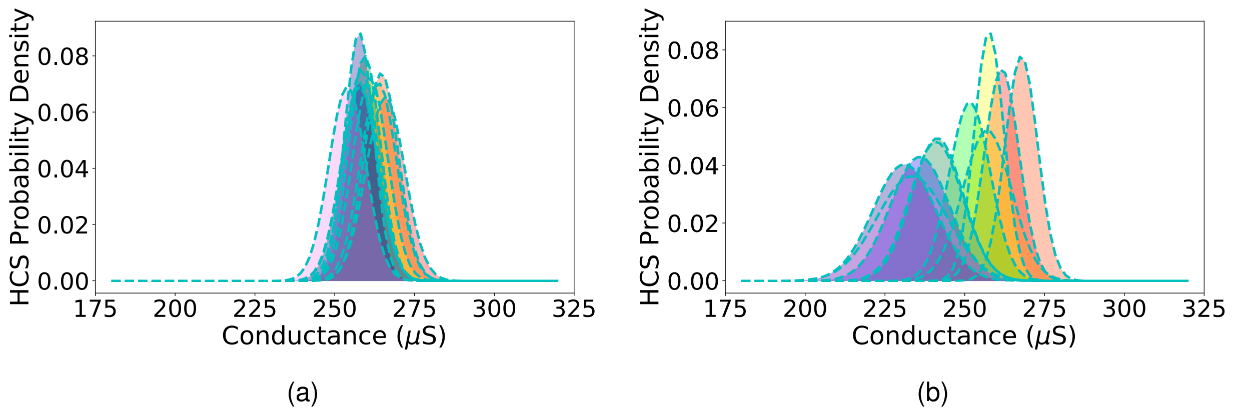


Figure 6.23: **The change in the random variable of a single OxRAM device as it undergoes RESET/SET cycling.** Examples of two devices are shown whereby the conductance in the HCS, for sets of one hundred sequential cycles are fit with a normal distribution. The colour of the normal distribution progresses through the colours in the order of the visible spectrum - from red to violet. Therefore the random variable of the first one hundred cycles is red and that of the last one hundred cycles is violet. (a) An example of a device which does not change significantly as it is cycled and all of the fitted normal distributions more or less overlap. (b) An example of a device where the physical normal random variable is seen to shift significantly as it is cycled. The normal distribution fitted from the first one hundred cycles (red) is centred around 260μS and has a low standard deviation. However, as the device is cycled, the median of the fitted normal moves, progressively, towards a lower conductance and its standard deviation increases.

6.9 Appendix - Cercal system model input neuron circuit implementation

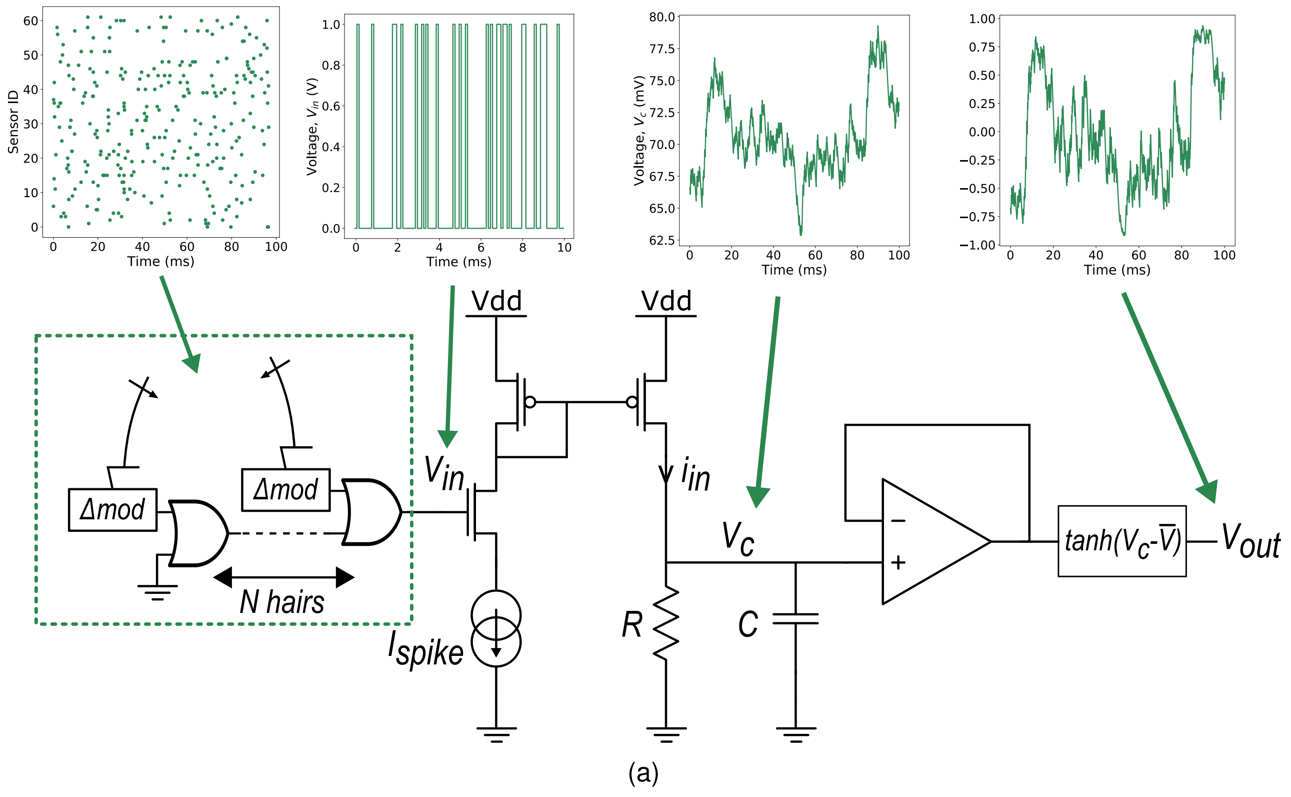
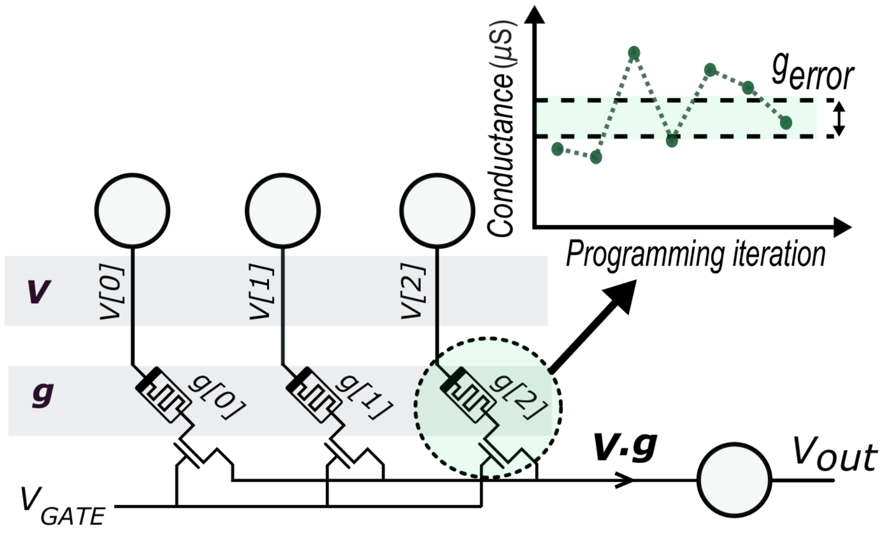


Figure 6.24: Circuit implementation of an input neuron. In order to integrate numerous events into a single continuous voltage, a chain of OR gates can be used to drive a single current mirror that injects the current pulse, i_{in} on the parallel resistor-capacitor circuit. An OPAMP connected as a voltage follower can then buffer this voltage into a hyperbolic tangent function circuit whose output can be used as an input feature. This voltage contains information on the recent spiking activity from a population of event-based sensors.

6.10 Appendix - Robustness of TAG model to random parameter permutations

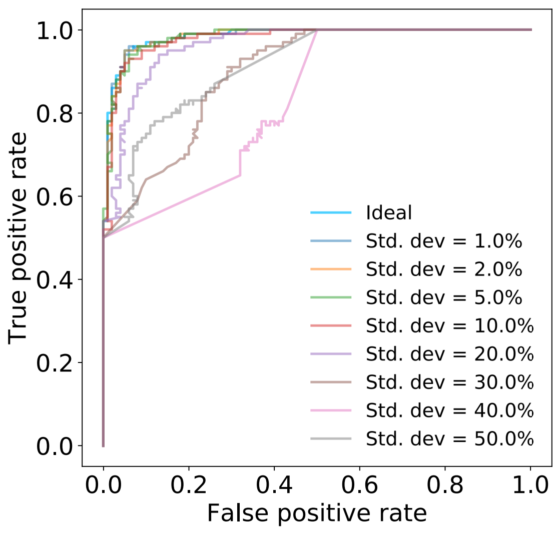
Neuromorphic implementations of neural network models also necessitate that the models take into account technological constraints such that they can be implemented in an efficient manner on a silicon substrate. This is at odds with deep learning whereby graphic processing units are typically employed but without a great deal of consideration of the underlying hardware. In contrast to the von Neumann architecture of GPUs which separate processing and memory centres on the chip, neuromorphic computing models are increasingly turning to resistive memory. The pre-synaptic weights to a neuron are defined using a row or RRAM as shown in Fig.6.25a. However, resistive memory devices are intrinsically random and, as such, it is extremely challenging to program a device conductance state with high-precision [129]. Therefore, when programming RRAM, a closed-loop program and verify scheme is typically employed. In such schemes, a device is iteratively programmed and read until the resulting conductance value obtained after programming falls within a tolerated error margin [264, 307].



(a)

Figure 6.25: Scheme indicating how a row of three resistive memory devices realise three synaptic connections between input activations **V** and a post-synaptic neuron. By applying these activations over the columns of the row of resistive memory the current that flows out of the rows will be equal to the dot product **V** $\cdot$ g, where **g** is the conductance vector of the row. In the inset of device an arrow indicates the closed-loop programming routine that is used in order to program the device conductance state, $g[2]$, whereby the memory is iteratively programmed until its conductance falls within a pre-determined range of error.

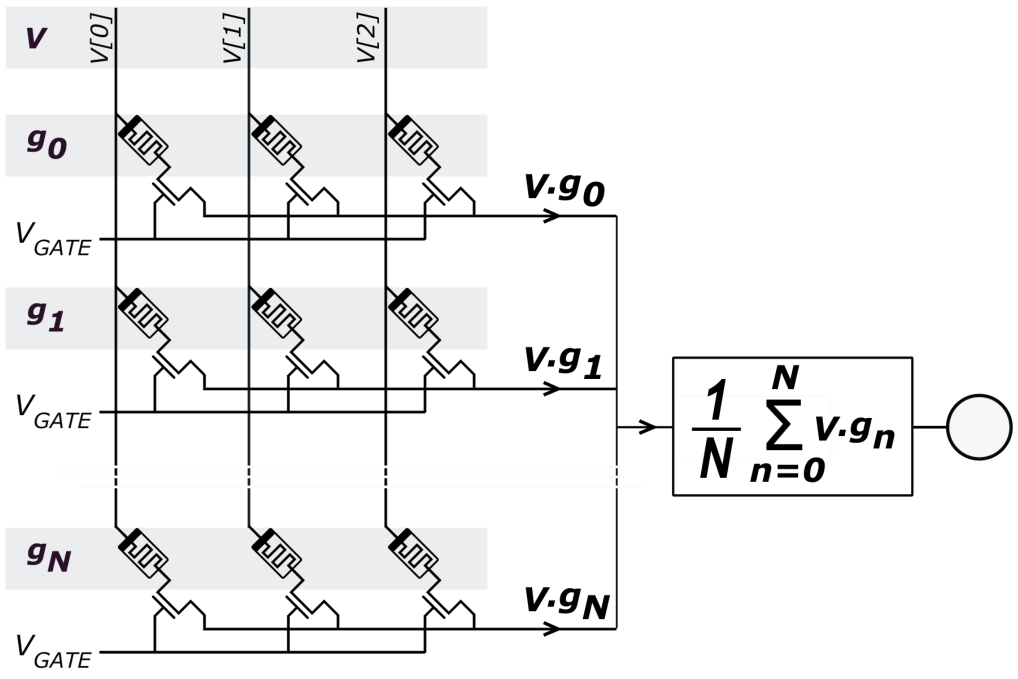
Therefore it is instructive to understand the robustness of the TAG model to random fluctuations to the synaptic parameters that correspond to different sizes of tolerated error margins. We do this by means of re-sampling each parameter from a normal distribution, centred on the optimised value, over a range of distribution standard deviations. The ROC curves resulting from this sweep are plotted in Fig. 6.26a. The ROC curves are observed to be minimally degraded up to a standard deviation of 10% and, thereafter, somewhat collapse.



(a)

Figure 6.26: he receiver operating characteristics (ROC) curves resulting from the model where the parameter values have been re-sampled from a normal distribution centred on the optimal value over a range of percentage standard deviations.

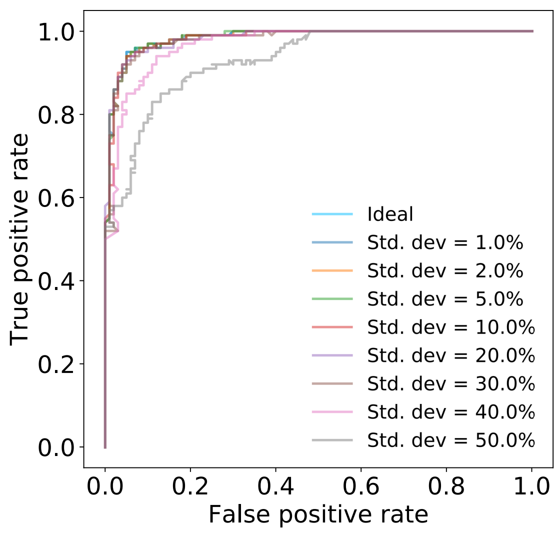
Like resistive memory devices, the neural components of biological systems are also observed to be very noisy and random. In order to cope with this, nervous systems have been widely understood to employ signal averaging strategies, for example sending multiple synaptic connections between pairs of neurons and averaging out the error [399]. This averaging strategy can be easily mimicked in the neuromorphic model by simply adding more rows of resistive memory per post-synaptic neuron and then dividing the current that flows out of the row by the number of rows used, using for example a simple current divider circuit [400] (depicted in Fig 6.27a).



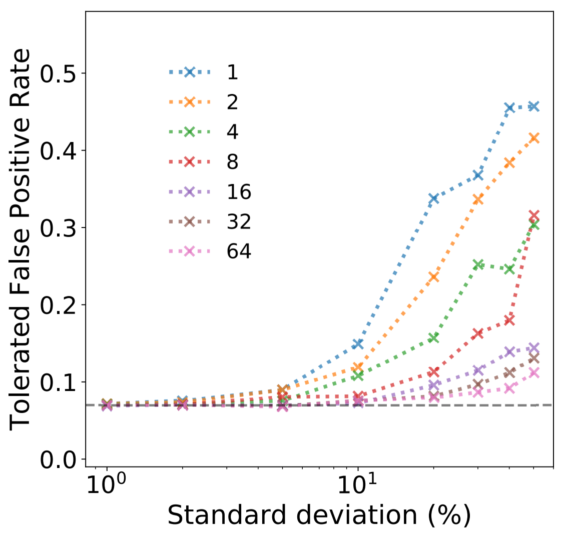
(a)

Figure 6.27: A scheme for performing row averaging, whereby multiple rows of resistive memory are used together. The current corresponding to the dot-product between the input voltage vector and each of the conductance vectors sums at the common output node. This summed current is then divided by N to compute the average, where N is equal to the number of rows being averaged over.

In Fig. 6.28a, we evaluate ROC curves over the same range of normal distribution standard deviations but this time employing the proposed averaging technique over sixteen rows per post-synaptic neuron. In this case, the ROC curves appear not to be degraded even up to 30% standard deviation demonstrating the effectiveness of this strategy - albeit increasing the memory requirements as a linear function of the number of rows used for averaging. The trade-off between the number of rows averaged over and the standard deviation of the normal distribution that re-samples the optimal parameter values are summarised in the plot of Fig. 6.28b.



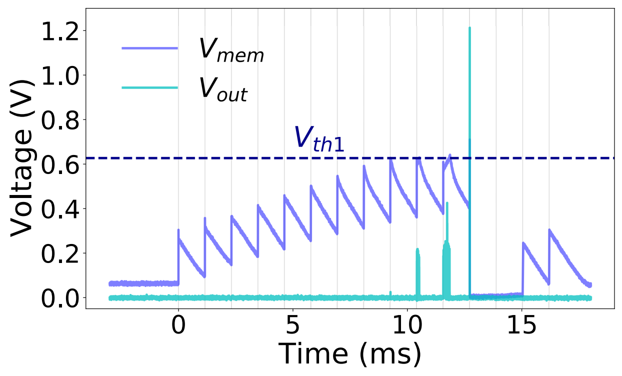
(a)



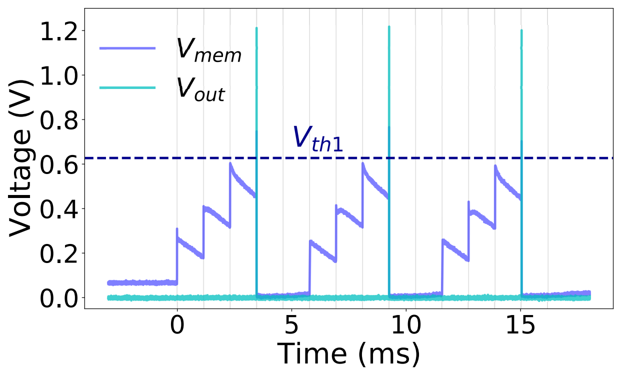
(b)

Figure 6.28: (a) The receiver operating characteristics (ROC) curves resulting from the model where the parameter values have been re-sampled from a normal distribution centred on the optimal value over a range of percentage standard deviations but averaging over $N = 16$ rows. (b) The tolerated false positive rate as a function of the standard deviation of the normal distributions that are used to re-sample the parameter values. Each point has been averaged over 10 re-sampling iterations.

6.11 Appendix - Measurements of fabricated CMOS neuron circuits



(a)



(b)

Figure 6.29: **Experimental measurements of fabricated CMOS neuron circuits using two different sets of bias conditions.** (a) A neuron integrates rectangular current pulses and eventually spikes. After it spikes it enters a refractory period during which incoming current pulses are no longer integrated. Note that when the membrane voltage approaches the spiking threshold the output almost switches twice. This is an issue present in this circuit since it does not incorporate a positive feedback connection onto the membrane capacitor which helps the device switch as it approaches the threshold. In addition it can be seen that the resting potential of the membrane is not zero but some tens of micro volts. (b) The same circuit as in part (a) but with a different bias configuration. Namely, the bias current governing the input time constant has been reduced resulting a greater input time constant and therefore more output spikes.

Bibliography

[1] Deep Learning Market Size Worth 10.2 Billion By 2025. $https://www.grandviewresearch.com/press-release/global-deep-learning-market$, 2017.

[2] Edge AI Chipsets: Technology Outlook and Use Cases. Technical report, 09 2019.

[3] Emma Strubell, Ananya Ganesh, and Andrew Mccallum. Energy and policy considerations for deep learning in nlp. In *In the 57th Annual Meeting of the Association for Computational Linguistics (ACL). Florence, Italy. July 2019*, 06 2019.

[4] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *arXiv preprint arXiv:1907.10597*, 2019.

[5] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[6] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[7] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[8] Edmon Begoli, Tanmoy Bhattacharya, and Dimitri Kusnezov. The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1(1):20–23, 2019.

[9] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[10] Ylva Fernaeus, Martin Jonsson, and Jakob Tholander. Revisiting the jacquard loom: threads of history and current patterns in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1593–1602, 2012.

[11] John Michael Dubbey and John Michael Dubbey. *The mathematical work of Charles Babbage*. Cambridge University Press, 2004.

[12] George Boole. *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Dover Publications, 1854.

[13] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[14] John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

[15] Adrien-Marie Legendre and DE Smith. On the method of least squares. *A Source Book in Mathematics, Ed. DE Smith (originally published in 1805)*, pages 576–579, 1959.

[16] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251:254, 2012.

[17] Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

[18] Pierre Simon Laplace. *Théorie analytique des probabilités*. Courcier, 1820.

[19] Andrey Andreyevich Markov. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. *Dynamic probabilistic systems*, 1:552–577, 1971.

[20] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[21] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[22] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[23] Edward I George, UE Makov, and AFM Smith. Conjugate likelihood distributions. *Scandinavian Journal of Statistics*, pages 147–156, 1993.

[24] Barry A Cipra. The best of the 20th century: Editors name top 10 algorithms. *SIAM news*, 33(4):1–2, 2000.

[25] John Zachary Young. The functioning of the giant nerve fibres of the squid. *Journal of Experimental Biology*, 15(2):170–185, 1938.

[26] Norbert Wiener. *Cybernetics or control and communication in the animal and the machine*. Technology Press, 1948.

[27] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[28] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[29] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[30] Bernhard Hassenstein and Werner Reichardt. Systemtheoretische analyse der zeit-, reihenfolgen- und vorzeichenauswertung bei der bewegungsperzeption des rüsselkäfers chlorophanus. *Zeitschrift für Naturforschung B*, 11(9-10):513–524, 1956.

[31] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[32] David P Watson and David H Scheidt. Autonomous systems. *Johns Hopkins APL technical digest*, 26(4):368–376, 2005.

[33] Yutaka Naitoh and Roger Eckert. Ionic mechanisms controlling behavioral responses of paramecium to mechanical stimulation. *Science*, 164(3882):963–965, 1969.

[34] Siming Li, Christopher M Armstrong, Nicolas Bertin, Hui Ge, Stuart Milstein, Mike Boxem, Pierre-Olivier Vidalain, Jing-Dong J Han, Alban Chesneau, Tong Hao, et al. A map of the interactome network of the metazoan c. elegans. *Science*, 303(5657):540–543, 2004.

[35] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[36] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[37] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[38] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[39] Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[40] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[41] James R Slagle. A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the ACM (JACM)*, 10(4):507–520, 1963.

[42] Gerald J Sussman. A computational model of skill acquisition. 1973.

[43] M Ross Quillan. Semantic memory. Technical report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA, 1966.

[44] Patrick H Winston. Learning structural descriptions from examples. 1970.

[45] Adolfo Guzmán. Decomposition of a visual scene into three-dimensional bodies. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 291–304, 1968.

[46] David Waltz. Understanding line drawings of scenes with shadows. In *The psychology of computer vision*. Citeseer, 1975.

[47] Donald Waterman. A guide to expert systems. 1986.

[48] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[49] Donald Michie. Experiments on the mechanization of game-learning part i. characterization of the model and its parameters. *The Computer Journal*, 6(3):232–236, 1963.

[50] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[51] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[52] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[53] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[54] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[55] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[56] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[57] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[58] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[59] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[60] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[61] John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

[62] Wolfgang Maass and Thomas Natschläger. Networks of spiking neurons can emulate arbitrary hopfield nets in temporal coding. *Network: Computation in Neural Systems*, 8(4):355–371, 1997.

[63] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[64] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[65] Vladimir Vapnik and AJ Lerner. Generalized portrait method for pattern recognition. *Automation and Remote Control*, 24(6):774–780, 1963.

[66] Mark A Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[67] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[68] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[69] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[70] NVIDIA Launches the World's First Graphics Processing Unit: GeForce 256. $https : //www.nvidia.com/object/IO_2 0020111_5 424.html$, 1999.

[71] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[72] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[73] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Graves Alex, Antonoglou Ioannis, W Daan, and R Martin. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[74] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[75] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[76] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[78] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[79] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pages 2528–2535. IEEE, 2010.

[80] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[81] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[82] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[83] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[84] Ethan Farquhar, Christal Gordon, and Paul Hasler. A field programmable neural array. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE, 2006.

[85] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[86] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141, 2015.

[87] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015.

[88] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[89] Jeffrey R Yost. *The IBM century: Creating the IT revolution*. IEEE Computer Society Press, 2011.

[90] James R Goodman. Using cache memory to reduce processor-memory traffic. In *Proceedings of the 10th annual international symposium on Computer architecture*, pages 124–131, 1983.

[91] Dean M Tullsen, Susan J Eggers, and Henry M Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22nd annual international symposium on Computer architecture*, pages 392–403, 1995.

[92] John Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[93] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.

[94] Richard D Greenblatt, Thomas F Knight, John T Holloway, and David A Moon. A lisp machine. In *Proceedings of the fifth workshop on Computer architecture for non-numeric processing*, pages 137–138, 1980.

[95] Peter Alfke, Ivo Bolsens, Bill Carter, Mike Santarini, and Steve Trimberger. It's an fpga! *IEEE Solid-State Circuits Magazine*, 3(4):15–20, 2011.

[96] Amos R Omondi and Jagath Chandana Rajapakse. *FPGA implementations of neural networks*, volume 365. Springer, 2006.

[97] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. Origami: A convolutional network accelerator. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 199–204, 2015.

[98] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 26(2):203–215, 2007.

[99] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2):39–55, 2008.

[100] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[101] Song Huang, Shucai Xiao, and Wu-chun Feng. On the energy efficiency of graphics processing units for scientific computing. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009.

[102] Misha Mahowald and Rodney Douglas. A silicon neuron. *Nature*, 354(6354):515–518, 1991.

[103] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André Van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73, 2011.

[104] Chiara Bartolozzi and Giacomo Indiveri. Synaptic dynamics in analog vlsi. *Neural computation*, 19(10):2581–2603, 2007.

[105] Kostas Pagiamtzis and Ali Sheikholeslami. Content-addressable memory (cam) circuits and architectures: A tutorial and survey. *IEEE journal of solid-state circuits*, 41(3):712–727, 2006.

[106] Carlos Zamarreño-Ramos, Alejandro Linares-Barranco, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. Multicasting mesh aer: A scalable assembly approach for reconfigurable neuromorphic structured aer systems. application to convnets. *IEEE transactions on biomedical circuits and systems*, 7(1):82–102, 2012.

[107] A-J Annema. Analog circuit performance and process scaling. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(6):711–725, 1999.

[108] Ning Qiao and Giacomo Indiveri. Scaling mixed-signal neuromorphic processors to 28 nm fd-soi technologies. In *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 552–555. IEEE, 2016.

[109] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. Spike-timing-dependent plasticity: a comprehensive overview. *Frontiers in synaptic neuroscience*, 4:2, 2012.

[110] Diasynou Fioravante and Wade G Regehr. Short-term forms of presynaptic plasticity. *Current opinion in neurobiology*, 21(2):269–274, 2011.

[111] Jan Benda and Andreas VM Herz. A universal model for spike-frequency adaptation. *Neural computation*, 15(11):2523–2564, 2003.

[112] Saber Moradi and Rajit Manohar. The impact of on-chip communication on memory technologies for neuromorphic systems. *Journal of Physics D: Applied Physics*, 52(1):014003, 2018.

[113] Gustav Tauschek. Reading machine. *US Pat*, 2026329, 1935.

[114] Isaac L Auerbach, John Presper Eckert, Robert F Shaw, and C Bradford Sheppard. Mercury delay line memory using a pulse rate of several megacycles. *Proceedings of the IRE*, 37(8):855–861, 1949.

[115] Arnold S Farber and Eugene S Schlig. Nondestructive memory array, November 21 1967. US Patent 3,354,440.

[116] Robert H Dennard. Field-effect transistor memory, June 4 1968. US Patent 3,387,286.

[117] Fujio Masuoka, Masamichi Asano, Hiroshi Iwahashi, Teisuke Komuro, and Shinichi Tanaka. A new flash e 2 prom cell using triple polysilicon technology. In *1984 International Electron Devices Meeting*, pages 464–467. IEEE, 1984.

[118] Fujio Masuoka, Masaki Momodomi, Yoshihisa Iwata, and Riichiro Shirota. New ultra high density eprom and flash eeprom with nand structure cell. In *1987 International Electron Devices Meeting*, pages 552–555. IEEE, 1987.

[119] A. Beck, J. G. Bednorz, Ch. Gerber, C. Rossel, and D. Widmer. Reproducible switching effect in thin oxide films for memory applications. *Applied Physics Letters*, 77(1):139–141, 2000.

[120] Qi Liu, Jun Sun, Hangbing Lv, Shibing Long, Kuibo Yin, Neng Wan, Yingtao Li, Litao Sun, and Ming Liu. Resistive switching: Real-time observation on dynamic growth/dissolution of conductive filaments in oxide-electrolyte-based reram (adv. mater. 14/2012). *Advanced Materials*, 24(14):1774–1774, 2012.

[121] H. . P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, Dec 2010.

[122] Claude Chappert, Albert Fert, and Frédéric Dau. The emergence of spin electronics in data storage. *Nature materials*, 6:813–23, 12 2007.

[123] Thomas Mikolajick, Christine Dehm, Walter Hartner, Ivan Kasko, MJ Kastner, Nicolas Nagel, Manfred Moert, and Carlos Mazure. Feram technology for high density applications. *Microelectronics Reliability*, 41(7):947–950, 2001.

[124] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[125] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[126] Sascha Vongehr and Xiangkang Meng. The missing memristor has not been found. *Scientific reports*, 5:11657, 2015.

[127] IG Baek, MS Lee, S Seo, MJ Lee, DH Seo, D-S Suh, JC Park, SO Park, HS Kim, IK Yoo, et al. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses. In *IEDM Technical Digest. IEEE International Electron Devices Meeting, 2004.*, pages 587–590. IEEE, 2004.

[128] Michael Kund, Gerhard Beitel, C-U Pinnow, Thomas Rohr, Jorg Schumann, Ralf Symanczyk, K Ufert, and Gerhard Muller. Conductive bridging ram (cbram): An emerging non-volatile memory technology scalable to sub 20nm. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pages 754–757. IEEE, 2005.

[129] Stefano Ambrogio, Simone Balatti, Antonio Cubeta, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. Statistical fluctuations in hfo x resistive-switching memory: part i-set/reset variability. *IEEE Transactions on electron devices*, 61(8):2912–2919, 2014.

[130] Yong Chen, Gun-Young Jung, Douglas AA Ohlberg, Xuema Li, Duncan R Stewart, Jan O Jeppesen, Kent A Nielsen, J Fraser Stoddart, and R Stanley Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14(4):462, 2003.

[131] Indranil Chakraborty, Mustafa Ali, Aayush Ankit, Shubham Jain, Sourjya Roy, Shrihari Sridharan, Amogh Agrawal, Anand Raghunathan, and Kaushik Roy. Resistive crossbars as approximate hardware building blocks for machine learning: Opportunities and challenges. *Proceedings of the IEEE*, 2020.

[132] Leibin Ni, Yuhao Wang, Hao Yu, Wei Yang, Chuliang Weng, and Junfeng Zhao. An energy-efficient matrix multiplication accelerator by distributed in-memory computing on binary rram crossbar. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 280–285. IEEE, 2016.

[133] Alexandre Levisse, B Giraud, JP Noel, M Moreau, JM Portal, et al. Architecture, design and technology guidelines for crosspoint memories. In *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 55–60. IEEE, 2017.

[134] Xiaoyu Sun, Xiaochen Peng, Pai-Yu Chen, Rui Liu, Jae-sun Seo, and Shimeng Yu. Fully parallel rram synaptic array for implementing binary neural network with (+ 1,- 1) weights and (+ 1, 0) neurons. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 574–579. IEEE, 2018.

[135] Alessandro Grossi, Elisa Vianello, Cristian Zambelli, Pablo Royer, Jean-Philippe Noel, Bastien Giraud, Luca Perniola, Piero Olivo, and Etienne Nowak. Experimental investigation of 4-kb rram arrays programming conditions suitable for tcam. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12):2599–2607, 2018.

[136] BE Boser and Eduard Sackinger. An analog neural network processor with programmable network topology. In *1991 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 184–314. IEEE, 1991.

[137] Hiroshi Ishiwara. Proposal of adaptive-learning neuron circuits with ferroelectric analog-memory weights. *Japanese journal of applied physics*, 32(1S):442, 1993.

[138] Osamu Fujita and Yoshihito Amemiya. A floating-gate analog memory device for neural networks. *IEEE transactions on electron devices*, 40(11):2029–2035, 1993.

[139] Hyejung Choi, Heesoo Jung, Joonmyoung Lee, Jaesik Yoon, Jubong Park, Dong-jun Seong, Wootae Lee, Musarrat Hasan, Gun-Young Jung, and Hyunsang Hwang. An electrically modifiable synapse array of resistive switching memory. *Nanotechnology*, 20(34):345201, 2009.

[140] Alex Pappachen James and Sima Dimitrijev. Cognitive memory network. *Electronics Letters*, 46(10):677–678, 2010.

[141] Sven Möller, Craig Perlov, Warren Jackson, Carl Taussig, and Stephen R Forrest. A polymer/semiconductor write-once read-many-times memory. *Nature*, 426(6963):166–169, 2003.

[142] Tae-Wook Kim, Hyejung Choi, Seung-Hwan Oh, Gunuk Wang, Dong-Yu Kim, Hyunsang Hwang, and Takhee Lee. One transistor–one resistor devices for polymer non-volatile memory applications. *Advanced Materials*, 21(24):2497–2500, 2009.

[143] G Navarro, A Verdy, N Castellani, G Bourgeois, V Sousa, G Molas, M Bernard, C Sabbione, P Noé, J Garrione, et al. Innovative pcm+ ots device with high sub-threshold non-linearity for non-switching reading operations and higher endurance performance. In *2017 Symposium on VLSI Technology*, pages T94–T95. IEEE, 2017.

[144] Fabien Alibart, Elham Zamanidoost, and Dmitri B Strukov. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nature communications*, 4(1):1–7, 2013.

[145] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M Shelby, Irem Boybat, Carmelo di Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan CP Farinha, et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60–67, 2018.

[146] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature communications*, 9(1):1–8, 2018.

[147] Can Li, Zhongrui Wang, Mingyi Rao, Daniel Belkin, Wenhao Song, Hao Jiang, Peng Yan, Yunning Li, Peng Lin, Miao Hu, et al. Long short-term memory networks in memristor crossbar arrays. *Nature Machine Intelligence*, 1(1):49–57, 2019.

[148] Zhongrui Wang, Can Li, Peng Lin, Mingyi Rao, Yongyang Nie, Wenhao Song, Qinru Qiu, Yunning Li, Peng Yan, John Paul Strachan, et al. In situ training of feed-forward and recurrent convolutional memristor networks. *Nature Machine Intelligence*, 1(9):434–442, 2019.

[149] Greg S Snider. Spike-timing-dependent learning in memristive nanodevices. In *2008 IEEE international symposium on nanoscale architectures*, pages 85–92. Ieee, 2008.

[150] Damien Querlioz, Olivier Bichler, and Christian Gamrat. Simulation of a memristor-based spiking neural network immune to device variations. In *The 2011 International Joint Conference on Neural Networks*, pages 1775–1781. IEEE, 2011.

[151] Mirko Prezioso, Farnood Merrikh-Bayat, BD Hoskins, Gina C Adam, Konstantin K Likharev, and Dmitri B Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64, 2015.

[152] Stanisław Woźniak, Angeliki Pantazi, Thomas Bohnstingl, and Evangelos Eleftheriou. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6):325–336, 2020.

[153] Miao Hu, Catherine E Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Davila, Hao Jiang, R Stanley Williams, J Joshua Yang, et al. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018.

[154] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Magic—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[155] An Chen. Utilizing the variability of resistive random access memory to implement reconfigurable physical unclonable functions. *IEEE Electron Device Letters*, 36(2):138–140, 2014.

[156] Fuxi Cai, Suhas Kumar, Thomas Van Vaerenbergh, Xia Sheng, Rui Liu, Can Li, Zhan Liu, Martin Foltin, Shimeng Yu, Qiangfei Xia, et al. Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks. *Nature Electronics*, 3(7):409–418, 2020.

[157] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[158] Joseph S Friedman, Laurie E Calvet, Pierre Bessière, Jacques Droulez, and Damien Querlioz. Bayesian inference with muller c-elements. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(6):895–904, 2016.

[159] Damir Vodenicarevic, Nicolas Locatelli, Alice Mizrahi, Joseph S Friedman, Adrien F Vincent, Miguel Romera, Akio Fukushima, Kay Yakushiji, Hitoshi Kubota, Shinji Yuasa, et al. Low-energy truly random number generation with superparamagnetic tunnel junctions for unconventional computing. *Physical Review Applied*, 8(5):054045, 2017.

[160] Rafatul Faria, Kerem Y Camsari, and Supriyo Datta. Implementing bayesian networks with embedded stochastic mram. *AIP Advances*, 8(4):045101, 2018.

[161] Stuart SP Parkin, Masamitsu Hayashi, and Luc Thomas. Magnetic domain-wall racetrack memory. *Science*, 320(5873):190–194, 2008.

[162] M Radosavljević, M Freitag, KV Thadani, and AT Johnson. Nonvolatile molecular memory elements based on ambipolar nanotube field effect transistors. *Nano Letters*, 2(7):761–764, 2002.

[163] Tsutomu Tezuka and Atsushi Kurobe. Quantum dot memory cell, July 13 1999. US Patent 5,923,046.

[164] Brian Julsgaard, Jacob Sherson, J Ignacio Cirac, Jaromír Fiurášek, and Eugene S Polzik. Experimental demonstration of quantum memory for light. *Nature*, 432(7016):482–486, 2004.

[165] Open data, reséaux énergies. $https://opendata.reseaux-energies.frl$, 2020.

[166] Lambda. Openai's gpt-3 language model: A technical overview, 2020.

[167] Brian Hayes. Cloud computing, 2008.

[168] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *Proceedings of the 2015 international workshop on internet of things towards applications*, pages 7–12, 2015.

[169] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[170] Xiaowei Xu, Yukun Ding, Sharon Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1, 04 2018.

[171] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[172] Barry J Maron, Win-Kuang Shen, Mark S Link, Andrew E Epstein, Adrian K Almquist, James P Daubert, Gust H Bardy, Stefano Favale, Robert F Rea, Giuseppe Boriani, et al. Efficacy of implantable cardioverter–defibrillators for the prevention of sudden death in patients with hypertrophic cardiomyopathy. *New England Journal of Medicine*, 342(6):365–373, 2000.

[173] Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 1(3), 2016.

[174] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[175] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[176] M Mirowski. The automatic implantable cardioverter-defibrillator: an overview. *Journal of the American College of Cardiology*, 6(2):461–466, 1985.

[177] Stuart J Connolly, AP Hallstrom, Riccardo Cappato, Eleanor B Schron, K-H Kuck, Douglas P Zipes, H Leon Greene, S Boczor, M Domanski, D Follmann, et al. Meta-analysis of the implantable cardioverter defibrillator secondary prevention trials. *European heart journal*, 21(24):2071–2078, 2000.

[178] James P Daubert, Wojciech Zareba, David S Cannom, Scott McNitt, Spencer Z Rosero, Paul Wang, Claudio Schuger, Jonathan S Steinberg, Steven L Higgins, David J Wilber, et al. Inappropriate implantable cardioverter-defibrillator shocks in madit ii: frequency, mechanisms, predictors, and survival impact. *Journal of the American College of Cardiology*, 51(14):1357–1365, 2008.

[179] Kara E Yopak, Thomas J Lisney, and Shaun P Collin. Not all sharks are “swimming noses”: variation in olfactory bulb size in cartilaginous fishes. *Brain Structure and Function*, 220(2):1127–1143, 2015.

[180] Fabienne Dupuy, Thomas Steinmann, Dominique Pierre, Jean-Philippe Christidès, Graham Cummins, Claudio Lazzari, John Miller, and Jérôme Casas. Responses of cricket cercal interneurons to realistic naturalistic stimuli in the field. *Journal of Experimental Biology*, 215(14):2382–2389, 2012.

[181] Melis Yilmaz and Markus Meister. Rapid innate defensive responses of mice to looming visual stimuli. *Current Biology*, 23(20):2011–2015, 2013.

[182] Nathan C. Klapoetke, Aljoscha Nern, Martin Y. Peek, Edward M. Rogers, Patrick Breads, Gerald M. Rubin, Michael B. Reiser, and Gwyneth M. Card. Ultra-selective looming detection from radial motion opponency. *Nature*, 551:237, Nov 2017.

[183] Hiroto Ogawa and John P. Miller. *Cercal System*, pages 1–6. Springer New York, New York, NY, 2019.

[184] Jan M Ache, Jason Polsky, Shada Alghailani, Ruchi Parekh, Patrick Breads, Martin Y Peek, Davi D Bock, Catherine R von Reyn, and Gwyneth M Card. Neural basis for looming size and velocity encoding in the drosophila giant fiber escape pathway. *Current Biology*, 29(6):1073–1081, 2019.

[185] Niko Tinbergen. The study of instinct. 1951.

[186] Jesse N Weber and Hopi E Hoekstra. The evolution of burrowing behaviour in deer mice (genus peromyscus). *Animal Behaviour*, 77(3):603–609, 2009.

[187] Yi Wei, Dmitry Tsigankov, and Alexei Koulakov. The molecular basis for the development of neural maps. *Annals of the New York Academy of Sciences*, 1305(1):44–60, 2013.

[188] Alexander Borst and Moritz Helmstaedter. Common circuit design in fly and mammalian motion vision. *Nature neuroscience*, 18(8):1067, 2015.

[189] Anthony M Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):1–7, 2019.

[190] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 1969.

[191] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204, 2015.

[192] Thomas Dalgaty, Elisa Vianello, Denys Ly, Giacomo Indiveri, Barbara De Salvo, Etienne Nowak, and Jerome Casas. Insect-inspired elementary motion detection embracing resistive memory and spiking neural networks. In *Conference on Biomimetic and Biohybrid Systems*, pages 115–128. Springer, 2018.

[193] Michael Schmuker, Thomas Pfeil, and Martin Paul Nawrot. A neuromorphic network for generic multivariate data classification. *Proceedings of the National Academy of Sciences*, 111(6):2081–2086, 2014.

[194] Fredrik Sandin and Mattias Nilsson. Synaptic delays for insect-inspired temporal feature detection in dynamic neuromorphic processors. *Frontiers in Neuroscience*, 14:150, 2020.

[195] Randall D Beer, Hillel J Chiel, Roger D Quinn, Kenneth S Espenschied, and Patrik Larsson. A distributed neural network architecture for hexapod robot locomotion. *Neural Computation*, 4(3):356–365, 1992.

[196] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[197] Gwen A Jacobs, John P Miller, and Zane Aldworth. Computational mechanisms of mechanosensory processing in the cricket. *Journal of Experimental Biology*, 211(11):1819–1828, 2008.

[198] Teresita C Insausti, Claudio R Lazzari, and Jérôme Casas. The terminal abdominal ganglion of the wood cricket nemobius sylvestris. *Journal of morphology*, 269(12):1539–1551, 2008.

[199] TC Insausti, CR Lazzari, and Jérôme Casas. The morphology and fine structure of the giant interneurons of the wood cricket nemobius sylvestris. *Tissue and Cell*, 43(1):52–65, 2011.

[200] Christelle Magal, Olivier Dangles, Philippe Caparroy, and Jérôme Casas. Hair canopy of cricket sensory system tuned to predator signals. *Journal of theoretical biology*, 241(3):459–466, 2006.

[201] John Miller, Susan Krueger, Jeff Heys, and Tomáš Gedeon. Quantitative characterization of the filiform mechanosensory hair array on the cricket cercus. *PloS one*, 6:e27873, 11 2011.

[202] Jeff Heys, Prathish Kumar Rajaraman, Tomáš Gedeon, and John Miller. A model of filiform hair distribution on the cricket cercus. *PloS one*, 7:e46588, 10 2012.

[203] JP Bacon and RK Murphey. Receptive fields of cricket giant interneurones are related to their dendritic structure. *The Journal of physiology*, 352(1):601–623, 1984.

[204] Gwen A Jacobs and Frederic E Theunissen. Functional organization of a neural map in the cricket cercal sensory system. *Journal of Neuroscience*, 16(2):769–784, 1996.

[205] Sussan Paydar, Caitlin A Doan, and Gwen A Jacobs. Neural mapping of direction and frequency in the cricket cercal sensory system. *Journal of Neuroscience*, 19(5):1771–1781, 1999.

[206] Gwen A Jacobs and Frederic E Theunissen. Extraction of sensory parameters from a neural map by primary sensory interneurons. *Journal of Neuroscience*, 20(8):2934–2943, 2000.

[207] Gwen A Jacobs and RK Murphey. Segmental origins of the cricket giant interneuron system. *Journal of Comparative Neurology*, 265(1):145–157, 1987.

[208] DeanaA Bodnar, JohnP Miller, and GwenA Jacobs. Anatomy and physiology of identified wind-sensitive local interneurons in the cricket cercal sensory system. *Journal of comparative physiology. A, Sensory, neural, and behavioral physiology*, 168(5):553–564, 1991.

[209] Y Baba, K Hirota, T Yamaguchi, and T Shimozawa. Differing afferent connections of spiking and nonspiking wind-sensitive local interneurons in the terminal abdominal ganglion of the cricket gryllus bimaculatus. *Journal of Comparative Physiology A*, 176(1):17–30, 1995.

[210] GA Jacobs, JP Miller, and RK Murphey. Integrative mechanisms controlling directional sensitivity of an identified sensory interneuron. *The Journal of Neuroscience*, 6(8):2298–2311, 1986.

[211] T Shimozawa, T Kumagai, and Y Baba. Structural scaling and functional design of the cercal wind-receptor hairs of cricket. *Journal of Comparative Physiology A*, 183(2):171–186, 1998.

[212] MA Landolfa and JP Miller. Stimulus-response properties of cricket cereal filiform receptors. *Journal of Comparative Physiology A*, 177(6):749–757, 1995.

[213] MA Landolfa and GA Jacobs. Direction sensitivity of the filiform hair population of the cricket cereal system. *Journal of Comparative Physiology A*, 177(6):759–766, 1995.

[214] Thomas Steinmann and Jérôme Casas. The morphological heterogeneity of cricket flow-sensing hairs conveys the complex flow signature of predator attacks. *Journal of The Royal Society Interface*, 14(131):20170324, 2017.

[215] John P Miller, Gwen A Jacobs, and Frédéric E Theunissen. Representation of sensory information in the cricket cercal sensory system. i. response properties of the primary interneurons. *Journal of Neurophysiology*, 66(5):1680–1689, 1991.

[216] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[217] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[218] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[219] HC Bennet-Clark. The energetics of the jump of the locust schistocerca gregaria. *Journal of Experimental Biology*, 63(1):53–83, 1975.

[220] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[221] Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike*, pages 199–213. Springer, 1998.

[222] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2:598–605, 1989.

[223] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[224] John Lazzaro, Sylvie Ryckebusch, Misha Anne Mahowald, and Caver A Mead. Winner-take-all networks of o (n) complexity. In *Advances in neural information processing systems*, pages 703–711, 1989.

[225] Wolfgang Maass. On the computational power of winner-take-all. *Neural computation*, 12(11):2519–2535, 2000.

[226] Shin-ya Takemura, Arjun Bharioke, Zhiyuan Lu, Aljoscha Nern, Shiv Vitaladevuni, Patricia K. Rivlin, William T. Katz, Donald J. Olbris, Stephen M. Plaza, Philip Winston, Ting Zhao, Jane Anne Horne, Richard D. Fetter, Satoko Takemura, Katerina Blazek, Lei-Ann Chang, Omotara Ogundeyi, Mathew A. Saunders, Victor Shapiro, Christopher Sigmund, Gerald M. Rubin, Louis K. Scheffer, Ian A. Meinertzhagen, and Dmitri B. Chklovskii. A visual motion detection circuit suggested by drosophila connectomics. *Nature*, 500:175, Aug 2013.

[227] Matthew S. Maisak, Juergen Haag, Georg Ammer, Etienne Serbe, Matthias Meier, Aljoscha Leonhardt, Tabea Schilling, Armin Bahl, Gerald M. Rubin, Aljoscha Nern, Barry J. Dickson, Dierk F. Reiff, Elisabeth Hopp, and Alexander Borst. A directional tuning map of drosophila elementary motion detectors. *Nature*, 500:212, Aug 2013.

[228] Juergen Haag, Alexander Arenz, Etienne Serbe, Fabrizio Gabbiani, and Alexander Borst. Complementary mechanisms create direction selectivity in the fly. *eLife*, 5:e17421, aug 2016.

[229] Rudy Behnia, Damon A. Clark, Adam G. Carter, Thomas R. Clandinin, and Claude Desplan. Processing properties of on and off pathways for drosophila motion detection. *Nature*, 512:427, Jul 2014.

[230] Sarah Nicola Jung, Alexander Borst, and Juergen Haag. Flight activity alters velocity tuning of fly motion-sensitive neurons. *Journal of Neuroscience*, 31(25):9231–9237, 2011.

[231] Marie P. Suver, Akira Mamiya, and Michael H. Dickinson. Octopamine neurons mediate flight-induced modulation of visual processing in drosophila. *Current Biology*, 22(24):2294 – 2302, 2012.

[232] Alexander Arenz, Michael S. Drews, Florian G. Richter, Georg Ammer, and Alexander Borst. The Temporal Tuning of the Drosophila Motion Detectors Is Determined by the Dynamics of Their Input Elements. *Current Biology*, 27(7):929 – 944, 2017.

[233] Hassenstein B. and Reichardt W. Systemtheoretische Analyse der Zeit-, Reihenfolgen- und Vorzeichenauswertung bei der Bewegungsperzeption des Rüsselkäfers Chlorophanus. 11:513, 1956. 9-10.

[234] Reid R. Harrison and Christof Koch. An Analog VLSI Model of the Fly Elementary Motion Detector. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 880–886. MIT Press, 1998.

[235] Shih-Chi Liu. A neuromorphic aVLSI model of global motion processing in the fly. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(12):1458–1467, Dec 2000.

[236] R. R. Harrison. A biologically inspired analog IC for visual collision detection. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(11):2308–2318, Nov 2005.

[237] Johannes Plett, Armin Bahl, Martin Buss, Kolja Kühnlenz, and Alexander Borst. Bio-inspired visual ego-rotation sensor for MAVs. *Biological Cybernetics*, 106(1):51–63, Jan 2012.

[238] J. kramer. Compact integrated motion sensor with three-pixel interaction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(2):86–101, March 1996.

[239] J. kramer and C. Koch. Pulse-based analog VLSI velocity sensors. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44(2):86–101, Feb 1997.

[240] R. Sarpeshkar, J. Kramer, G. Indiveri, and C. Koch. Analog VLSI architectures for motion processing: from fundamental limits to system applications. *Proceedings of the IEEE*, 84(7):969–987, Jul 1996.

[241] P. A. Shoemaker. Implementation of Visual Motion Detection in Analog Neuromorphic Circuitry - A Case Study of the Issue of Circuit Precision. *Proceedings of the IEEE*, 102(10):1557–1570, Oct 2014.

[242] Moritz B Milde, Olivier JN Bertrand, Harshawardhan Ramachandran, Martin Egelhaaf, and Elisabetta Chicca. Spiking elementary motion detector in neuromorphic systems. *Neural computation*, 30(9):2384–2417, 2018.

[243] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128$\times$128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, Feb 2008.

[244] T. Serrano-Gotarredona and B. Linares-Barranco. A 128$\times$128 1.5% Contrast Sensitivity 0.9% FPN 3μs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers. *IEEE Journal of Solid-State Circuits*, 48(3):827–838, March 2013.

[245] Sadique Sheik, Elisabetta Chicca, and Giacomo Indiveri. Exploiting device mismatch in neuromorphic vlsi systems to implement axonal delays. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2012.

[246] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[247] Dan Simon. *Evolutionary Optimization Algorithms*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2013.

[248] David J. Montana and Lawrence Davis. Training Feedforward Neural Networks Using Genetic Algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, pages 762–767, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[249] Matthew S Maisak, Juergen Haag, Georg Ammer, Etienne Serbe, Matthias Meier, Aljoscha Leonhardt, Tabea Schilling, Armin Bahl, Gerald M Rubin, Aljoscha Nern, et al. A directional tuning map of drosophila elementary motion detectors. *Nature*, 500(7461):212–216, 2013.

[250] Ming Wu, Aljoscha Nern, W Ryan Williamson, Mai M Morimoto, Michael B Reiser, Gwyneth M Card, and Gerald M Rubin. Visual projection neurons in the drosophila lobula link feature detection to distinct behavioral programs. *Elife*, 5:e21022, 2016.

[251] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[252] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[253] Shin-ya Takemura, Yoshinori Aso, Toshihide Hige, Allan Wong, Zhiyuan Lu, C Shan Xu, Patricia K Rivlin, Harald Hess, Ting Zhao, Toufiq Parag, et al. A connectome of a learning and memory center in the adult drosophila brain. *Elife*, 6:e26975, 2017.

[254] Karl Deisseroth. Optogenetics. *Nature methods*, 8(1):26–29, 2011.

[255] J. von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

[256] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[257] D. Li, X. Chen, M. Becchi, and Z. Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 477–484, Oct 2016.

[258] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, Sep. 1971.

[259] Dmitri Strukov, Gregory Snider, Duncan Stewart, and Stan Williams. The missing memristor found. *Nature*, 453:80–3, 06 2008.

[260] Qiangfei Xia and Jianhua Joshua Yang. Memristive crossbar arrays for brain-inspired computing. *Nature Materials*, 18:309–323, 04 2019.

[261] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert Shelby, Irem Boybat, Carmelo Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan Farinha, Benjamin Killeen, Christina Cheng, Yassine Jaoudi, and Geoffrey Burr. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558, 06 2018.

[262] Mirko Prezioso, Farnood Merrikh-Bayat, Brian Hoskins, Gina Adam, Konstantin Likharev, and Dmitri Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521, 12 2014.

[263] Zhongrui Wang, Can Li, Wenhao Song, Mingyi Rao, Daniel Belkin, Yunning Li, Peng Yan, Hao Jiang, Peng Lin, Miao Hu, John William Strachan, Ning Ge, Mark Barnell, Qing wu, Andrew Barto, Qinru Qiu, Stan Williams, Qiangfei Xia, and Jianhua Joshua Yang. Reinforcement learning with analogue memristor arrays. *Nature Electronics*, 2, 03 2019.

[264] Miao Hu, Catherine E. Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Davila, Hao Jiang, R. Stanley Williams, J. Joshua Yang, Qiangfei Xia, and John Paul Strachan. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018.

[265] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, Wenhao Song, John William Strachan, Mark Barnell, Qing wu, Stan Williams, Jianhua Joshua Yang, and Qiangfei Xia. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications*, 9, 12 2018.

[266] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, Jianhua Joshua Yang, and he Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577:641–646, 01 2020.

[267] Tayfun Gokmen, Murat Onen, and Wilfried Haensch. Training deep convolutional neural networks with resistive cross-point devices. *Frontiers in Neuroscience*, 11:538, 2017.

[268] G. W. Burr, R. M. Shelby, S. Sidler, C. di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi, and H. Hwang. Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Transactions on Electron Devices*, 62(11):3498–3507, Nov 2015.

[269] D. Garbin, E. Vianello, O. Bichler, Q. Rafhay, C. Gamrat, G. Ghibaudo, B. DeSalvo, and L. Perniola. HfO_2-Based OxRAM Devices as Synapses for Convolutional Neural Networks. *IEEE Transactions on Electron Devices*, 62(8):2494–2501, Aug 2015.

[270] Abu Sebastian, Daniel Krebs, Manuel Le Gallo, Haralampos Pozidis, and Evangelos Eleftheriou. A collective relaxation model for resistance drift in phase change memory cells. *2015 IEEE International Reliability Physics Symposium*, pages MY.5.1–MY.5.6, 2015.

[271] X. Guan, S. Yu, and H. . P. Wong. On the switching parameter variation of metal-oxide rram—part i: Physical modeling and simulation methodology. *IEEE Transactions on Electron Devices*, 59(4):1172–1182, April 2012.

[272] S. Yu, X. Guan, and H. . P. Wong. On the switching parameter variation of metal oxide rram—part ii: Model corroboration and device design strategy. *IEEE Transactions on Electron Devices*, 59(4):1183–1188, April 2012.

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

[274] C. H. Bennett, D. Garland, R. B. Jacobs-Gedrim, S. Agarwal, and M. J. Marinella. Wafer-scale taox device variability and implications for neuromorphic computing applications. In *2019 IEEE International Reliability Physics Symposium (IRPS)*, pages 1–4, March 2019.

[275] S. Agarwal, S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James, and M. J. Marinella. Resistive memory device requirements for a neural algorithm accelerator. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 929–938, July 2016.

[276] S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian, and E. Eleftheriou. Mixed-precision architecture based on computational memory for training deep neural networks. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2018.

[277] Irem Boybat, Manuel Gallo, Nandakumar S.R., Timoleon Moraitis, Thomas Parnell, Tomas Tuma, Bipin Rajendran, Yusuf Leblebici, Abu Sebastian, and Evangelos Eleftheriou. Neuromorphic computing with multi-memristive synapses. *Nature Communications*, 9, 11 2017.

[278] Alexander Serb, Johannes Bill, Ali Khiat, Radu Berdan, Robert Legenstein, and Themis Prodromakis. Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses. *Nature Communications*, 7:12611, 09 2016.

[279] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat. Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Transactions on Nanotechnology*, 12(3):288–295, May 2013.

[280] Thomas Dalgaty, Melika Payvand, Filippo Moro, Denys R. B. Ly, Florian Pebay-Peyroula, Jerome Casas, Giacomo Indiveri, and Elisa Vianello. Hybrid neuromorphic circuits exploiting non-conventional properties of rram for massively parallel local plasticity mechanisms. *APL Materials*, 7(8):081125, 2019.

[281] A. Chen. Utilizing the variability of resistive random access memory to implement reconfigurable physical unclonable functions. *IEEE Electron Device Letters*, 36(2):138–140, Feb 2015.

[282] S. Balatti, S. Ambrogio, Z. Wang, and D. Ielmini. True random number generation by variability of resistive switching in oxide-based devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):214–221, June 2015.

[283] Damir Vodenicarevic, Nicolas Locatelli, Alice Mizrahi, Joseph S Friedman, Adrien F Vincent, Miguel Romera, Akio Fukushima, Kay Yakushiji, Hitoshi Kubota, Shinji Yuasa, et al. Low-energy truly random number generation with superparamagnetic tunnel junctions for unconventional computing. *Physical Review Applied*, 8(5):054045, 2017.

[284] Rafatul Faria, Kerem Y Camsari, and Supriyo Datta. Implementing bayesian networks with embedded stochastic mram. *AIP Advances*, 8(4):045101, 2018.

[285] Alice Mizrahi, Tifenn Hirtzlin, Akio Fukushima, Hitoshi Kubota, Shinji Yuasa, Julie Grollier, and Damien Querlioz. Neural-like computing with populations of superparamagnetic basis functions. *Nature communications*, 9(1):1533, 2018.

[286] Kerem Yunus Camsari, Rafatul Faria, Brian M Sutton, and Supriyo Datta. Stochastic p-bits for invertible logic. *Physical Review X*, 7(3):031014, 2017.

[287] William A Borders, Ahmed Z Pervaiz, Shunsuke Fukami, Kerem Y Camsari, Hideo Ohno, and Supriyo Datta. Integer factorization using stochastic magnetic tunnel junctions. *Nature*, 573(7774):390–393, 2019.

[288] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[289] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521:452–9, 05 2015.

[290] A. Grossi, E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. E. Hajjam, R. Crochemore, J. F. Nodin, P. Olivo, and L. Perniola. Fundamental variability limits of filament-based rram. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 4.7.1–4.7.4, Dec 2016.

[291] D. Ielmini. Modeling the universal set/reset characteristics of bipolar rram by field- and temperature-driven filament growth. *IEEE Transactions on Electron Devices*, 58(12):4309–4317, Dec 2011.

[292] Simon Rogers and Mark Girolami. *A first course in machine learning*. CRC Press, 2016.

[293] W H Wolberg and O L Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, 87(23):9193–9196, 1990.

[294] Huan Liu and Rudy Setiono. Chi2: feature selection and discretization of numeric attributes. *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 388–391, 1995.

[295] Alessandro Grossi, Elisa Vianello, Mohamed M Sabry, Marios Barlas, Laurent Grenouillet, Jean Coignus, Edith Beigne, Tony Wu, Binh Q Le, Mary K Wootters, et al. Resistive ram endurance: Array-level characterization and correction techniques targeting deep learning applications. *IEEE Transactions on Electron Devices*, 66(3):1281–1288, 2019.

[296] George B Moody and Roger G Mark. The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[297] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[298] Matt Hoffman, Arnaud Doucet, Nando de Freitas, and Ajay Jasra. Trans-dimensional mcmc for bayesian policy learning. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, page 665–672, Red Hook, NY, USA, 2007. Curran Associates Inc.

[299] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, Sep. 1983.

[300] Radu Berdan, Takao Marukame, Shoichi Kabuyanagi, Kensuke Ota, Masumi Saitoh, Shosuke Fujii, Jun Deguchi, and Yoshifumi Nishi. In-memory reinforcement learning with moderately-stochastic conductance switching of ferroelectric tunnel junctions. In *2019 Symposium on VLSI Technology*, pages T22–T23. IEEE, 2019.

[301] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[302] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. Metal–oxide rram. *Proceedings of the IEEE*, 100(6):1951–1970, 2012.

[303] Hao-Chiao Hong and Guo-Ming Lee. A 65-fj/conversion-step 0.9-v 200-ks/s rail-to-rail 8-bit successive approximation adc. *IEEE Journal of Solid-State Circuits*, 42(10):2161–2168, 2007.

[304] Yifan Sun, Nicolas Bohm Agostini, Shi Dong, and David Kaeli. Summarizing cpu and gpu design trends with product data. *arXiv preprint arXiv:1911.11313*, 2019.

[305] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[306] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 10 2019.

[307] R. Mochida, K. Kouno, Y. Hayata, M. Nakayama, T. Ono, H. Suwa, R. Yasuhara, K. Katayama, T. Mikawa, and Y. Gohou. A 4m synapses integrated analog reram based 66.5 tops/w neural-network processor with cell current controlled writing and flexible network architecture. In *2018 IEEE Symposium on VLSI Technology*, pages 175–176, June 2018.

[308] Charles Mackin, Hsinyu Tsai, Stefano Ambrogio, Pritish Narayanan, An Chen, and Geoffrey Burr. Weight programming in dnn analog hardware accelerators in the presence of nvm variability. *Advanced Electronic Materials*, page 1900026, 04 2019.

[309] A. Valentian, F. Rummens, E. Vianello, T. Mesquida, C. Lacat-Mathieu de Boissac, and C. Reita. Fully integrated spiking neural network with analog neurons and rram synapses. In *International Electron Device Meeting*, 2019.

[310] D. Querlioz, O. Bichler, A. F. Vincent, and C. Gamrat. Bioinspired programming of memory devices for implementing an inference engine. *Proceedings of the IEEE*, 103(8):1398–1416, Aug 2015.

[311] Marc Bocquet, Tifenn Hirztlin, J-O Klein, Etienne Nowak, Elisa Vianello, J-M Portal, and Damien Querlioz. In-memory and error-immune differential rram implementation of binarized deep neural networks. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 20–6. IEEE, 2018.

[312] Tifenn Hirtzlin, Marc Bocquet, Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Frontiers in Neuroscience*, 13:1383, 2020.

[313] Chetan Singh Thakur, Saeed Afshar, Runchun M Wang, Tara J Hamilton, Jonathan Tapson, and André Van Schaik. Bayesian estimation and inference using stochastic electronics. *Frontiers in neuroscience*, 10:104, 2016.

[314] Przemyslaw Mroszczyk and Piotr Dudek. The accuracy and scalability of continuous-time bayesian inference in analogue cmos circuits. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1576–1579. IEEE, 2014.

[315] Thomas Dalgaty, Nicollo Castellani, Damien Querlioz, and Elisa Vianello. In-situ learning harnessing intrinsic resistive memory variability through markov chain monte carlo sampling. *arXiv:2001.11426*, 2020.

[316] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, Nov 1996.

[317] Alessandro Grossi, Elisa Vianello, Mohamed M Sabry, Marios Barlas, Laurent Grenouillet, Jean Coignus, Edith Beigne, Tony Wu, Binh Q Le, Mary K Wootters, et al. Resistive ram endurance: Array-level characterization and correction techniques targeting deep learning applications. *IEEE Transactions on Electron Devices*, 66(3):1281–1288, 2019.

[318] P.L. Hsu and Herbert Robbins. Complete convergence and the law of large numbers. *Proc Natl Acad Sci USA*, 1947.

[319] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

[320] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

[321] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.

[322] Matthew Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15:1351–1381, 04 2014.

[323] Thomas Wiecki and Maxim Kochurov. Bayesian deep learning, 2016.

[324] David D Fan, Jennifer Nguyen, Rohan Thakker, Nikhilesh Alatur, Ali-akbar Agha-mohammadi, and Evangelos A Theodorou. Bayesian learning-based adaptive control for safety critical systems. *arXiv preprint arXiv:1910.02325*, 2019.

[325] Yihong Wang, Rubin Wang, and Xuying Xu. Neural energy supply-consumption properties based on hodgkin-huxley model. *Neural plasticity*, 2017, 2017.

[326] Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

[327] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[328] Dougal Maclaurin and Ryan P. Adams. Firefly monte carlo: Exact mcmc with subsets of data. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, page 543–552, Arlington, Virginia, USA, 2014. AUAI Press.

[329] Anoop Korattikara, Yutian Chen, and Max Welling. Austerity in mcmc land: Cutting the metropolis-hastings budget. In *International Conference on Machine Learning*, pages 181–189, 2014.

[330] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[331] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[332] Sebastian Farquhar, Lewis Smith, and Yarin Gal. Liberty or depth: Deep bayesian neural nets do not need complex weight posterior approximations. *arXiv e-prints*, pages arXiv–2002, 2020.

[333] Declan A Doyle, Joao Morais Cabral, Richard A Pfuetzner, Anling Kuo, Jacqueline M Gulbis, Steven L Cohen, Brian T Chait, and Roderick MacKinnon. The structure of the potassium channel: molecular basis of k+ conduction and selectivity. *science*, 280(5360):69–77, 1998.

[334] Martin Stemmler and Christof Koch. How voltage-dependent conductances can adapt to maximize the information encoded by neuronal firing rate. *Nature Neuroscience*, 2:521–527, 1999.

[335] Gina G Turrigiano and Sacha B Nelson. Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, 5:97, 2004.

[336] WB Levy and O Steward. Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. *Neuroscience*, 8(4):791–797, 1983.

[337] Yang Dan and Mu-ming Poo. Hebbian depression of isolated neuromuscular synapses in vitro. *Science*, 256(5063):1570–1573, 1992.

[338] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE*, 102(9):1367–1388, Sept 2014.

[339] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017.

[340] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, 12:891, 2018.

[341] Matthew D. Pickett, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. A scalable neuristor built with Mott memristors. *Nature Materials*, 12(2):114–117, 2013.

[342] Tomas Tuma, Angeliki Pantazi, Manuel Le Gallo, Abu Sebastian, and Evangelos Eleftheriou. Stochastic phase-change neurons. *Nature nanotechnology*, 11(8):693, 2016.

[343] Robert Berdan. Scanning electron microscopy photography, 2017.

[344] E. Vianello, O. Thomas, G. Molas, O. Turkyilmaz, N. Jovanović, D. Garbin, G. Palma, M. Alayan, C. Nguyen, J. Coignus, B. Giraud, T. Benoist, M. Reyboz, A. Toffoli, C. Charpin, F. Clermidy, and L. Perniola. Resistive memories for ultra-low-power embedded computing design. In *2014 IEEE International Electron Devices Meeting*, pages 6.3.1–6.3.4, Dec 2014.

[345] Wulfram Gerstner. Time structure of the activity in neural network models. *Physical review E*, 51(1):738, 1995.

[346] Giacomo Indiveri and Yulia Sandamirskaya. The importance of space and time for signal processing in neuromorphic agents: the challenge of developing low-power, autonomous agents that interact with the environment. *IEEE Signal Processing Magazine*, 36(6):16–28, 2019.

[347] Andreea Lazar, Pipa Gordon, and Jochen Triesch. Sorn: a self-organizing recurrent neural network. *Frontiers in computational neuroscience*, 3:23, 2009.

[348] R Baddeley, L F Abbott, M C Booth, F Sengpiel, T Freeman, E A Wakeman, and E T Rolls. Responses of neurons in primary and inferior temporal visual cortices to natural scenes. *Proceedings of the Royal Society B: Biological Sciences*, 264:1775–83, 1997.

[349] David Heeger. *Markov chains*. Cambrdige University Press, 2000.

[350] Yoshifumi Nishi, Ulrich Bottger, Rainer Waser, and Stephan Menzel. Crossover from deterministic to stochastic nature of resistive-switching statistics in a tantalum oxide thin film. *IEEE Transactions on Electron Devices*, PP:1–6, 09 2018.

[351] A Rukhin, J Soto, J C Nechvatal, M Smid, E Barker, L Stefan, and S Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *NIST: National Institute of Standards and Technology*, 2010.

[352] Tobias Delbrueck and C Mead. Bump circuits. In *Proceedings of International Joint Conference on Neural Networks*, volume 1, pages 475–479, 1993.

[353] J R Norris. *Poisson Model of Spike Generation*. 1998.

[354] Kwabena A Boahen. Point-to-point connectivity between neuromorphic chips using address events. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(5):416–434, 2000.

[355] Alessandro Mortara, Eric A Vittoz, and Philippe Venier. A communication scheme for analog vlsi perceptive systems. *IEEE Journal of Solid-State Circuits*, 30(6):660–669, 1995.

[356] Joseph Lin, Paul Merolla, John Arthur, and Kwabena Boahen. Programmable connections in neuromorphic grids. In *2006 49th IEEE International Midwest Symposium on Circuits and Systems*, volume 1, pages 80–84. IEEE, 2006.

[357] Siddharth Joshi, Steve Deiss, Mike Arnold, Jongkil Park, Theodore Yu, and Gert Cauwenberghs. Scalable event routing in hierarchical neural array architecture with global synaptic connectivity. In *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, pages 1–6. IEEE, 2010.

[358] Olaf Sporns and Jonathan D Zwi. The small world of the cerebral cortex. *Neuroinformatics*, 2(2):145–162, 2004.

[359] Rodrigo Perin, Thomas K Berger, and Henry Markram. A synaptic organizing principle for cortical neuronal groups. *Proceedings of the National Academy of Sciences*, 108(13):5419–5424, 2011.

[360] Muhammad Mukaram Khan, David R Lester, Luis A Plana, A Rast, Xin Jin, Eustace Painkras, and Stephen B Furber. Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2849–2856. Ieee, 2008.

[361] Vladimir Kornijcuk and Doo Seok Jeong. Recent progress in real-time adaptable digital neuromorphic hardware. *Advanced Intelligent Systems*, 1(6):1900030, 2019.

[362] Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’networks. *nature*, 393(6684):440–442, 1998.

[363] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186–198, 2009.

[364] Christoph Stosiek, Olga Garaschuk, Knut Holthoff, and Arthur Konnerth. In vivo two-photon calcium imaging of neuronal networks. *Proceedings of the National Academy of Sciences*, 100(12):7319–7324, 2003.

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

[366] Mark D Humphries, Kevin Gurney, and Tony J Prescott. The brainstem reticular formation is a small-world, not scale-free, network. *Proceedings of the Royal Society B: Biological Sciences*, 273(1585):503–511, 2006.

[367] Mark D Humphries and Kevin Gurney. Network ‘small-world-ness’: a quantitative method for determining canonical network equivalence. *PloS one*, 3(4):e0002051, 2008.

[368] Federico Corradi and Giacomo Indiveri. A neuromorphic event-based neural recording system for smart brain-machine-interfaces. *IEEE transactions on biomedical circuits and systems*, 9(5):699–709, 2015.

[369] Reid R Harrison. The design of integrated circuits to observe brain activity. *Proceedings of the IEEE*, 96(7):1203–1216, 2008.

[370] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[371] Wilten Nicola and Claudia Clopath. Supervised learning in spiking neural networks with force training. *Nature communications*, 8(1):1–15, 2017.

[372] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

[373] Fred Rieke, David Warland, Rob De Ruyter Van Steveninck, William S Bialek, et al. *Spikes: exploring the neural code*, volume 7. MIT press Cambridge, 1999.

[374] Alexander Borst and Frédéric E Theunissen. Information theory and neural coding. *Nature neuroscience*, 2(11):947–957, 1999.

[375] Rufin VanRullen, Rudy Guyonneau, and Simon J Thorpe. Spike times make sense. *Trends in neurosciences*, 28(1):1–4, 2005.

[376] Staffan Cullheim. Relations between cell body size, axon diameter and axon conduction velocity of cat sciatic α-motoneurons stained with horseradish peroxidase. *Neuroscience letters*, 8(1):17–20, 1978.

[377] MJ Gillespie and RB Stein. The relationship between axon diameter, myelin thickness and conduction velocity during atrophy of mammalian peripheral nerves. *Brain research*, 259(1):41–56, 1983.

[378] Biswa Sengupta, Simon Barry Laughlin, and Jeremy Edward Niven. Consequences of converting graded to action potentials upon neural information coding and energy efficiency. *PLoS Comput Biol*, 10(1):e1003439, 2014.

[379] KG Pearson and CR Fourtner. Nonspiking interneurons in walking system of the cockroach. *Journal of neurophysiology*, 38(1):33–52, 1975.

[380] KATHERINE Graubard. Synaptic transmission without action potentials: input-output properties of a nonspiking presynaptic neuron. *Journal of Neurophysiology*, 41(4):1014–1025, 1978.

[381] Katharina Eichler, Feng Li, Ashok Litwin-Kumar, Youngser Park, Ingrid Andrade, Casey M Schneider-Mizell, Timo Saumweber, Annina Huser, Claire Eschbach, Bertram Gerber, et al. The complete connectome of a learning and memory centre in an insect brain. *Nature*, 548(7666):175–182, 2017.

[382] C Shan Xu, Michal Januszewski, Zhiyuan Lu, Shin-ya Takemura, Kenneth Hayworth, Gary Huang, Kazunori Shinomiya, Jeremy Maitin-Shepard, David Ackerman, Stuart Berg, et al. A connectome of the adult drosophila central brain. *BioRxiv*, 2020.

[383] Romain Franconville, Celia Beron, and Vivek Jayaraman. Building a functional connectome of the drosophila central complex. *Elife*, 7:e37017, 2018.

[384] Yoshinori Aso, Divya Sitaraman, Toshiharu Ichinose, Karla R Kaun, Katrin Vogt, Ghislain Belliart-Guérin, Pierre-Yves Plaçais, Alice A Robie, Nobuhiro Yamagata, Christopher Schnaitmann, et al. Mushroom body output neurons encode valence and guide memory-based action selection in drosophila. *Elife*, 3:e04580, 2014.

[385] Thomas Stone, Barbara Webb, Andrea Adden, Nicolai Ben Weddig, Anna Honkanen, Rachel Templin, William Wcislo, Luca Scimeca, Eric Warrant, and Stanley Heinze. An anatomically constrained model for path integration in the bee brain. *Current Biology*, 27(20):3069–3085, 2017.

[386] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[387] Pico Caroni, Flavio Donato, and Dominique Muller. Structural plasticity upon learning: regulation and functions. *Nature Reviews Neuroscience*, 13(7):478–490, 2012.

[388] Sophie Deneve. Bayesian inference in spiking neurons. In *Advances in neural information processing systems*, pages 353–360, 2005.

[389] Yanping Huang and Rajesh PN Rao. Neurons as monte carlo samplers: Bayesian inference and learning in spiking networks. In *Advances in neural information processing systems*, pages 1943–1951, 2014.

[390] Hesham Mostafa, Lorenz K Müller, and Giacomo Indiveri. Rhythmic inhibition allows neural networks to search for maximally consistent states. *Neural computation*, 27(12):2510–2547, 2015.

[391] Rodrigo Echeveste, Laurence Aitchison, Guillaume Hennequin, and Máté Lengyel. Cortical-like dynamics in recurrent circuits optimized for sampling-based probabilistic inference. *Nature neuroscience*, pages 1138–1149, 2020.

[392] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019.

[393] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[394] Eric Jang, Coline Devin, Vincent Vanhoucke, and Sergey Levine. Grasp2vec: Learning object representations from self-supervised grasping. *arXiv preprint arXiv:1811.06964*, 2018.

[395] Yoshinori Aso, Daisuke Hattori, Yang Yu, Rebecca M Johnston, Nirmala A Iyer, Teri-TB Ngo, Heather Dionne, LF Abbott, Richard Axel, Hiromu Tanimoto, et al. The neuronal architecture of the mushroom body provides a logic for associative learning. *elife*, 3:e04577, 2014.

[396] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[397] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

[398] Dan FM Goodman and Romain Brette. The brian simulator. *Frontiers in neuroscience*, 3:26, 2009.

[399] A Aldo Faisal, Luc PJ Selen, and Daniel M Wolpert. Noise in the nervous system. *Nature reviews neuroscience*, 9(4):292–303, 2008.

[400] J Robert, PH Deval, and G Wegmann. Very accurate current divider. *Electronics Letters*, 25(14):912–913, 1989.