MASTER THESIS

VISUAL CLASSIFICATION WITH A SIMULATED DISORDERED BORON DOPANT NETWORK

Jo-Yu (Kevin) Liu

FACULTY OF BEHAVIOURAL, MANAGEMENT AND SOCIAL SCIENCES

FIRST SUPERVISOR:

PROF. DR. FRANK VAN DER VELDE

SECOND SUPERVISOR:

PROF. DR. IR. BERNARD GEURTS

23.04.2021

UNIVERSITY OF TWENTE.

Abstract

As artificial intelligence slowly approaches human-like intelligence and capabilities, limitations to current computational technology and architecture surfaces, and creates barriers and challenges towards the scaling of neural systems that are inherently too large and complex to simulate. With millions of neurons in V1 alone, a prominent, and concurrent example of the affected domains is machine vision. By harnessing the inherent, in-material, complexity of a boron dopant atoms network, Chen et al's (2020) neuromorphic hardware research has brought about a promising solution to the ever increasing computational needs that machine vision artificial intelligence demands. The present paper sets out to simulate, and find out how feasible it is to use, the boron network to solve Boolean logic gates and to run current machine vision algorithms that mimic human visual perceptual architectures. The boron dopant cells are simulated as reservoir networks, using perceptron learning and later, a self-interpreted control node scheme. The results show that the boron-reservoir architecture introduces a form of complexity between the input and output, which, with its inherent randomness, reduces the reliability of all simulated networks in its ability to solve simple problems like the Boolean logic gates, but also provides enough complexity to sometimes solve the nonlinear logic gates. By altering the cell and network architecture, the inherent randomness can be reduced so that functional cells can be reliably created. The simulated architecture is then extended to simulate basis pattern recognition given by digit recognition and by simulating so-called simple cells as found in the first two layers of a model based on the visual cortex.

Table of Contents

Abstract 2

1 Introduction 4

1.1 Machine vision and intelligent systems 4

1.2 Goal of current research 5

2 A perceptual classification model 7

3 The need for Neuromorphic Computing 9

3.1 The computational capacity and efficiency problem 9

3.2 Current hardware solutions to the computational capacity problem 10

3.3 Chen et al. 2020 12

4. Reservoir computing 14

4.1 Comparison to boron system 15

4.3 Properties of an RC 15

5 Simulating the boron dopant cell 16

5.1 Testing network architecture on Perceptron logic gates 19

5.2 Splitting the cell into sub cells 23

5.3 Control nodes 27

5.4 Saturation cells 33

6 Digit recognition 35

7 Testing simple cell architecture using Gabor filter recognition 40

8 General Discussion 45

9 Conclusion 50

References 51

Appendix 55

A.1 Appendix 1 55

A.2 Appendix 2 57

A.3 Appendix 3 59

1 Introduction

1.1 Machine vision and intelligent systems

Machine learning and intelligent systems have enabled a myriad of profound, innovative solutions to increasingly complex problems, like self-driving cars, fault detection in manufacturing and even medical diagnosis (Vergara, 2014; Tian, Wang, Liu, Qiao, & Li, 2020). As machine learning's applicability and complexity broadens, so do their performance requirements and specifications; of which the capabilities of machine vision have been a topic of considerable interest. Whether driving, fault detection, or medical diagnosis, one of the most important senses that humans and other intelligent beings rely on to gain information about their environment is vision. Following the same line of thought, for an intelligent machine to perform tasks involving visual perception in a human-like manner, their visual perceptual capabilities should be comparable to that of humans.

In the past decades, boundless strides have been made in the world of machine learning and machine vision. Most notably, basic object and feature recognition has been vastly improved upon, solving problems like partial object scale and translational invariance through the use of convolutional neural networks (CNN), and deep convolutional networks (DCN) (Goodfellow, Yoshua & Courville, 2016). Much of the progress can be attributed to Hubel and Wiesel's (1962) discovery of the receptive fields and architecture of a feline visual cortex, and the simple to complex cell architecture in each layer of the visual perceptual system. Much like a convolutional network, the simple to complex cell architecture dictates that the layers of the visual cortex consist of simple and complex neurons. In each layer of the visual cortex, simple cells selectively respond to various pattern elements and their outputs are further fed into complex cells in the same layer, where responses to similar pattern elements are pooled together to create more scale and translational invariance (Goodfellow et al., 2016). The complex cells in a layer feed into the simple cells of the next layer, which allows these cells to respond to even more complex pattern elements. By exploiting the architecture and pooling operation of the system, Fukushima (1980) and LeCun, Bottou, Bengio, & Haffner (1998) were able to create intelligent machines that recognized number digits unaffected by shifts in translational position. In recent years, deeper understanding of the human visual perceptual system led to

improvements made in the simple to complex architecture, and heightened performances in both speed and accuracy of rapid object recognition to rival human performances (Serre, Olivia & Poggio, 2007).

Yet, while significant improvements have been made, there remain many differences between machine and human vision. First, conventional CNNs or other deep neural networks require large volumes of training data sets in order to gain scale and translational invariance to novel objects, whereas humans require very little (Han, Roig & Poggio, 2020). Second, while object scale and translational invariance has been solved using CNNs, humans are capable of learning 3-dimensional object rotational invariance. Third, while CNNs and DCNs are able to rival humans on basic tasks like rapid object and face recognition in a given context (usually the dataset), they do not generalize well to objects presented out of their usual context (Linsley, Eberhardt, Sharma, Gupta & Serre, 2018). Lastly, and perhaps most importantly, considering that a cubic millimeter of the primary visual cortex consists of 100,000 neurons (Alonso, 2002), it would take rather sizable computing power and computation time with conventional Complementary Metal Oxide Semiconductor (CMOS) computing machines to simulate the parallel processes performed by these neurons. These differences highlight a key problem in current machine vision research. If human level performances are the striven standards, the resulting complex, power and time consuming processes that need to be simulated would be far too inefficient using conventional computers. Computational limitations are a long standing problem in artificial intelligence research, and as conventional CMOS slowly approaches their computational limits, different steps have been taken to tackle these limits.

1.2 Goal of current research

In an effort to facilitate the development of neuromorphic hardware in its potential to surpass CMOS systems, the current research is focused firstly on developing a simplified simulation of a neuromorphic cell architecture; specifically, the disordered boron dopant cell architecture designed by Chen et al. (2020). The boron dopant architecture from Chen shows promising results for the future of neuromorphic hardware with its ability to solve all Boolean logic gates, and a simple digit recognition task without the need for updating connection weights like neural synapses. The simulated neural architectures should have the same input to output relationship, but do not need to perform the same exact same kinds of operations and algorithms as that of the

actual boron cell (aiming to avoid a simulation based on the complex underling quantum mechanical behaviour of a boron cell).

In current research, a boron cell is regarded as a 'reservoir' network. A reservoir network consists of a randomly organized set (or 'reservoir') of connection paths between neurons or nodes, that transfer activation from input to output. This appears to resemble the randomly organized way in which a boron cell conducts a current from the input to the output electrodes. Using the simulation of the boron cells as reservoirs, more information about the parameters of a cell, such as the role of the connection sparsity and their weights in the cells, could be obtained. Furthermore, to investigate the inherent randomness involved in a boron cell, different cell architectures will be tested. In this manner, the cell's partial capabilities under different architectures can be explored. Once a reasonable simulation of a boron cell as a reservoir has been achieved, that is, the architecture and input output behaviour of the cell is mimicked as best as possible, tests can be done to see whether such an architecture is capable of solving first, Boolean logic gates, then finally, fundamental machine vision problems. Specifically, the ultimate goal of the current research is to test the effectiveness of the simulated cell network's ability to solve visual recognition machine learning problems in a manner similar to our understanding of the human visual perceptual architecture, as described in Serre et al (2007). In the following section, the models of visual perception from Serre et al. (2007), the boron dopant architecture from Chen et al. (2020), and the use of reservoir computing will be further elaborated on.

2 A perceptual classification model

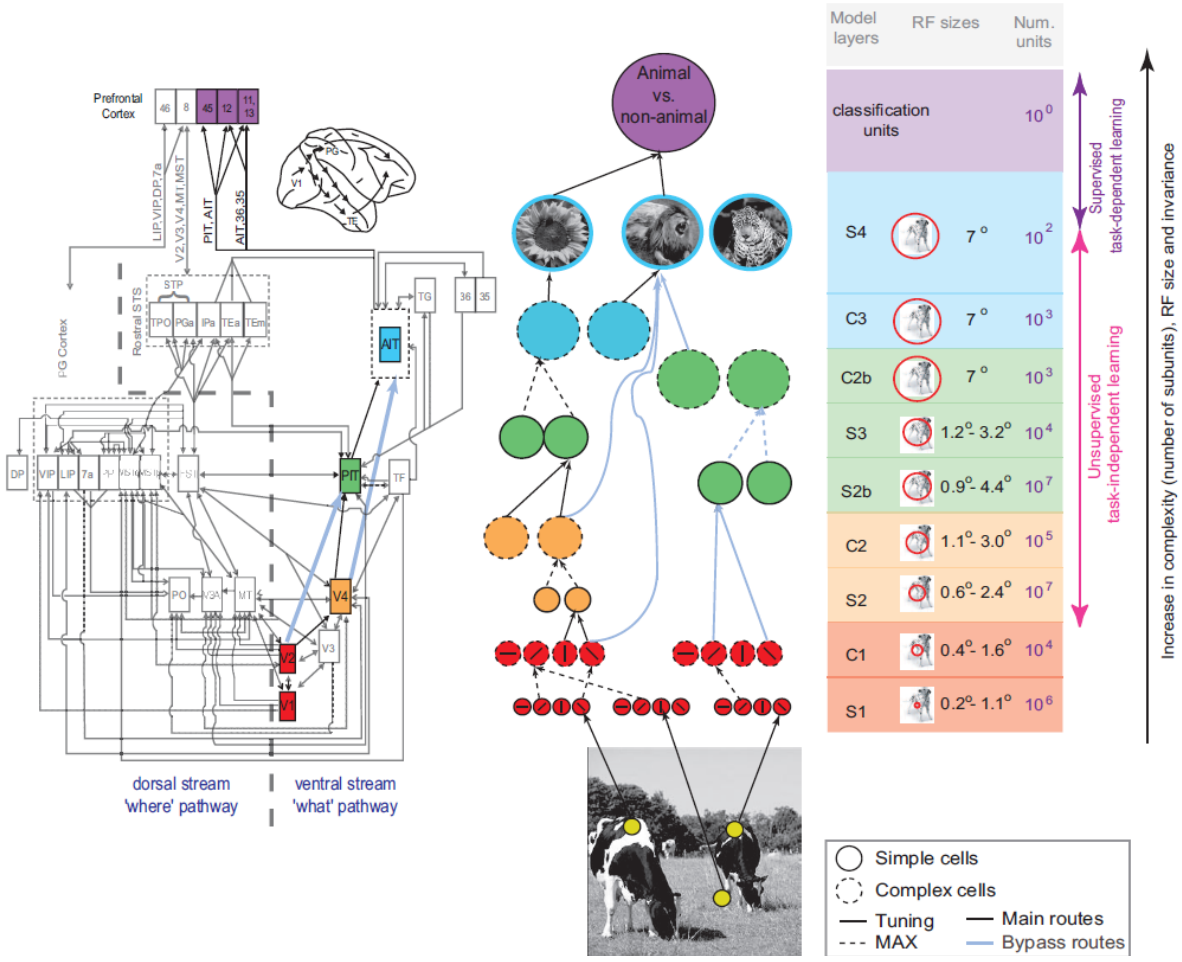


Figure 1. Model of Serre et al's (2007) simple to complex cell hierarchy in different layers of the visual cortex. The model consists of interlayered simple and complex cells, where complex cells are used to pool similar simple cell outputs to create invariance to scale and translational position. Simple cell layers like S2 are also able to combine C1 outputs to create more complex patterns. These functions are mapped to the underlying biological mechanisms behind human vision, namely, the ventral "what" pathway; accounting for V1, V2, V4, till inferotemporal cortices. Figure from Serre et al., (2007).

With nature and human physiology setting tremendous standards in computation volume, accuracy and speed, promising areas of development in domains of machine vision are found in the realm of biomimicry. Inspired by the simple to complex cell-hierarchy from Hubel and Wiesel (1962), Serre et. al (2007) proposed a feedforward model (figure 1) of machine vision learning which is capable of categorization tasks that are complex and rapid, correlating well with human performance and reflecting the well-known anatomy of the visual cortices. The simple to complex cell hierarchy, as described by Hubel and Wiesel states that visual input in the

primary visual cortex (or striate cortex, or V1) is first analyzed by a sublayer of simple S1 units, which, much like the cortical simple cells, respond to bars with different orientations of varying scales (known as Gabor filters) and densely covering the visual field. The subsequent sublayer of units consists of complex C1 units, which biologically correspond to the complex striate cells. The C1 units receive multi-phase inputs from the S1 units with similar stimulus preferences, and pool the inputs. As a result, the C1 units become more insensitive to the varying size or position of an object, which is an attribute of cortical complex cells.

Serre's model extends the simple to complex cell-hierarchy from the V1 layer to other (higher) layers in the visual cortex. In each layer, a set of pattern elements is learnt by the S nodes (figure 1 left, S1-S4) in that layer based on the input from the C cells (figure 1 left, C1-C3) in the previous layer, using unsupervised learning (imprinting). The C nodes in each layer pool responses of these S units to create (more) translation invariance. In the higher layers of the model, this results in, firstly, a 'pattern dictionary' that is universal in its recognition abilities, but more importantly, a model that not only performs similarly to human capabilities, but is also more similar to our understanding of biological human vision. While the accuracy and speed of their results are impressive, the scale at which their system operates is still far below that of the human visual system. Specifically, the primary visual cortex (V1) alone consists of hundreds of millions of neurons (Giannaris & Rosene, 2012; Leuba & Kraftsik, 1994), with overlapping, differently oriented retinotopic V1 simple cells responding to each position of the visual field. In contrast, the model devised by Serre consists of far less retinotopic neurons, processes images that are 256x256 pixels, rather than the much larger, and flexible, visual field of the human perceptual system and the filters represents only four line orientations. Actual Gabor filters used in texture analysis are sinusoidal waves multiplied by a Gaussian function to create linear contours of different orientations. These biologically set linear contours vary far more than just 4 different filters in rotational angle alone. In this sense, the Gabor filters used in Serre et al. (2007) are much more simplified, and fewer calculations are made at each position of the retinal field in comparison to the human visual system.

Much of Serre's work, as the majority of visual perceptual research is, is dedicated to improving the accuracy and speed of machine vision. However, their paradigm lacks the considerations of ever increasing computational volume and power consumption when their

model is scaled up towards levels closer to human capabilities. In the next section, developments and advances in current computing hardware, as well as the need for power efficient computing hardware are explored.

3 The need for Neuromorphic Computing

3.1 The computational capacity and efficiency problem

To effectively capture and operate on the vast amounts of information embedded in our environments, massive amounts of computations, both simple and complex, are required. These computations are made at every hierarchy of information processing, from the fundamentals of perception to higher level cognitive processes like reasoning or action deliberation. At the fundamental level, vision provides a plethora of information for humans, and machines alike, to explore for an understanding of the environment. Visual perception includes multiple scalable processes of organizing, identifying and interpreting visual information.

In recent years, artificial perception has been used in many different application domains (Yang et al., 2020). Most of its application happens in the automotive industry and other mass production industries; for example, to recognize parking space dimensions for assisted or automated parking, or faulty products on an assembly line, and even intelligent agricultural production (Vergara, 2014; Tian et al., 2020). While machine vision has seen successes in some basic, environment and task specific applications, achieving human level vision remains distant and difficult, with three challenges that need to be solved.

Firstly, the trade-off between speed and accuracy has always been a challenge in machine vision. Increasing the size, and therefore information acquisition as well as analysis capabilities, of neural networks inevitably increases the processing time (Huang et al., 2017). Capturing the variability of information in the real world requires sizable networks, and the processing speed in these networks is far from comparable to that of the human brain. For example, obstacle detection technology in automated driving is currently impractical, due to its lack of processing capacity and speed to achieve real-time computing in an accurate manner (Shi et al., 2017). While this could be solved by increasing hardware capacity and capabilities, traditional CMOS devices have slowly been approaching the limits of volume downscaling and raw performance.

Secondly, not only must devices be able to compute vast flows of information in real-time, they must also fit within a certain volume. After all, nobody would drive a car that is twice the size just to fit a computing device *potentially* capable of automatically avoiding obstacles. Additionally, the widely accessible, and most commonly owned computing devices of today tend to be pocket sized, like smartphones or smart watches (LeCun, 2019); thus, not only must computing devices perform at a much higher, and more efficient level, it must also stay under a certain size if its purpose is to benefit the average layman.

Finally, and perhaps most importantly, the power consumption of neural networks simulated on traditional CMOS architectures is far too inefficient for the massive scale at which human vision operates. The majority of machine learning research is predominantly focused on obtaining high levels of accuracy without consideration of computational restraints (Martina, 2019), and often, machine systems are pushed to their limits just to perform a fraction of the tasks that human vision is capable of. To this end, there exists a need for computational hardware that is far quicker, and more power efficient than the standard CMOS hardware of today.

3.2 Current hardware solutions to the computational capacity problem

While algorithmic developments can be used to optimize resource usage in neural networks (Li et al., 2016), hardware developments can potentially both increase, and optimize, the resource efficiency for neural networks, increasing processing power, decreasing processing speed and device volume simultaneously (Misra & Saha, 2010). An example of using hardware to improve efficiency and capacity, is the use of graphical processing units (GPU) in machine learning (Dsouza, 2020). In recent years, GPUs have been used in machine learning to significantly decrease the time to run the massive amounts of vector calculations required to train larger neural networks at the cost of more energy usage. Unlike the conventional Central Processing Units (CPUs), GPUs are able to perform multiple, simple tasks in parallel, rather than sequentially, on a much higher memory bandwidth (Sato, 2018). This allows the CPU to be used for longer, more complex, sequential tasks, while the GPU can perform tasks that have higher memory loads in parallel, like mathematical operations (addition, subtraction, matrix multiplication, etc.). Other examples of hardware advances that take advantage of the inherent parallelism of neural processes are for example, dedicated neural network chips also known as application specific integrated circuits or ASICs (Reddy, 2019; Misra & Saha, 2010).

ASICs are mostly CMOS based hardware designed for accelerating machine learning algorithm workloads, specifically, for neural networks (Nurvitadhi et al., 2016). Usually, an ASIC focuses on increasing efficiency in a certain part of the machine learning process, like training or inference where there are intensive parallel workloads. An example are Google's Tensor Processing Units (TPUs), which are designed simply as matrix processors, and save energy by removing the requirement for memory access while making massive calculations (Sato, 2018). Essentially, TPUs are more efficient parallel processing units in comparison to GPUs. Nevertheless, both GPUs and ASICs are still based on standard CMOS (Von Neumann) architecture, which heavily depends on Moore's law in order to continuously increase processing power, efficiency, and decrease in device volume. Unfortunately, the effect of Moore's law, which states that the number of transistors on a chip would double every 2 years, is slowly deteriorating, as the chip making industry finds difficulties in downsizing transistors approaching the atomic level (Schuller et al., 2015; Gupta, 2019).

According to Schuller et al. (2015), due to limitations in device packing density and the "lack of foreseeable limit in overall energy consumption", there exists a need for enhanced, or different, computing systems. In addition to the problem of Moore's law, traditional von Neumann architecture (of which both ASICs and GPUs are based on) consists of physically separated, rigid, functional units like the control processor (CPU), memory, arithmetic/logic and data paths. The classic von Neumann computing architecture differs in performance from neuromorphic architectures at both device and system level. At a device level, conventional silicon based systems, in comparison to the human brain, are much less power efficient and less dense in information capacity. At the systemic level, the traditional von Neumann architecture has temporal and energy bottlenecks since information must be constantly moved through different parts of the rigid system. In contrast, the human brain has co-located processor and memory abilities, due to the adaptability and flexibility of its constituents, and thus does not suffer from the same energy and temporal bottlenecks that are inherent to the von Neumann architecture (Schuller et al., 2015). By studying and understanding the architecture of the human brain, hardware systems with intrinsic complexity can be exploited to mimic neurons and synapses, without the need of using algorithms to simulate their behaviours. The advent of neuromorphic hardware brings promising ideas and results for neural networks to reach human-like performance.

3.3 Chen et al. 2020

Recently, encouraging results were published by research on disordered boron dopant atom networks Chen et al's. (2020). By injecting boron dopant atoms in a disordered, uncontrolled manner into silicon, Chen and his colleagues produced a cell of interconnected boron atoms. The cell can be electrically tuned to solve nonlinear Boolean logic gates like XOR and XNOR.

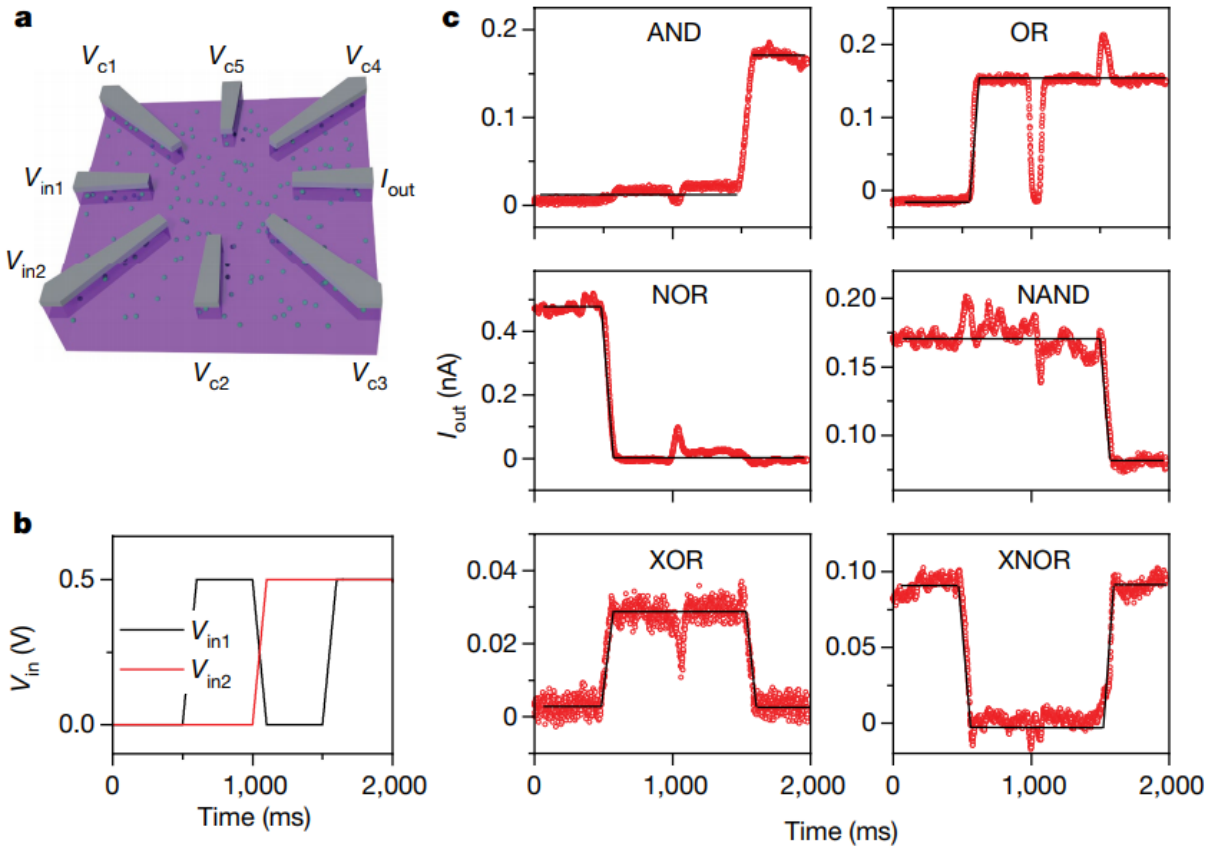


Figure 2. Boolean logic gate output voltages from figure 3 in Chen et al. (2020). (a) The boron cell architecture, with 2 inputs, 3 controls and 1 output, the blue dots represent the boron dopant atoms. (b) Input waveforms used in the cell, logic of 0 and 1 are represented as 0 and 0.5 volts. (c) Boolean logic gate output voltages at 77K. The outputs for XOR and XNOR are significantly lower than outputs from AND, NAND, OR and NOR. From Chen et al., (2020)

The intrinsic complexity of the system is realized by taking advantage of the charge transport regimes induced on boron dopant atoms at lower temperatures. With two input electrodes, one output and five control electrodes (illustrated in figure 2a), diverse nonlinear operational capabilities can be realized through the manipulation of the control electrode voltages (illustrated in figure 2b), which manipulate the overall reservoir voltage to tune the

output characteristics. The use of control nodes to manipulate reservoir activation and input to output relationship effectively solves the problem of simulating synapses through connection weight changes. While the cell suffers from certain glaring issues, like its inability to function past certain temperatures, Chen and his colleagues were able to perform basic visual recognition tasks using the cells at room temperature. Additionally, they were also able to solve all Boolean logic gates using a boron cell, as illustrated in figure 2c. More testing needs to be done to find the limits of the boron dopant cell. However, not only is its production complex, inaccessible (and expensive), one of the most fundamental problems of testing with the cell is the lack of control over the structure of the boron dopant network during manufacturing; in other words, every cell (reservoir) that is created is unique and different, and may have slightly different properties. In their report, one of the cells that Chen et al. have created could solve all logic gates with a certain control setting. But it is not guaranteed that subsequent cells would.

As Hirjibehedin (2020) commented, the dopant cell's "..reproducibility of control parameters for different devices, improved reliability of operation at higher temperatures.." are all problems that need to be addressed in the next developmental steps of Chen et al's. dopant cell. Additionally, shown in figure 2c, when further examining the output characteristics of their Boolean logic gates results, a significant difference in output voltages can be noticed between the linear and nonlinear logic gate outputs (0.03 for XOR, 0.15 for AND and OR). As previously mentioned, practical applications of neural networks are usually large, interconnected, multi-layered networks. The output characteristics from figure 2c pose as a roadblock for upscaling, as larger networks using interconnected layers of boron cells could have difficulties distinguishing between positive outputs with such a large range of activation. Without the possibility of upscaling the boron cell into interlayered networks, it would seem that there are no practical, real world problems that a single layer of boron cells could solve. Therefore, the purpose of the remainder of the thesis is as follows. Firstly, to simulate and test the boron dopant cell in a simplified manner. Secondly to find a potential architecture that is more invariant to the randomness inherent in creating a boron cell. Thirdly, explore the output characteristics of a simulated boron cell, and to see if its output can be rectified in a way that enables it to be used in multilayered networks. And lastly, to test the compatibility of such an architecture with current machine vision architectures. The fundamental structures of the boron cell, with its random randomly formed conduction path can be compared to that of a reservoir network, which forms a

system of randomly connected nodes. In the next section, the concepts of a reservoir computing system are explored.

4. Reservoir computing

Inspired by recurrent neural networks (RNNs) and echo state networks (ESNs), reservoir computing (RC) is a black-box method of mapping input signals to higher dimensional spaces through the dynamics of a fixed, nonlinear system. The reservoir itself consists of nodes with fixed, randomly formed connectivity paths (figure 3a). Since the connection within the reservoir are fixed, only the readout weights are updated with simple and fast training processes like linear regression, drastically reducing the computational costs of learning compared to standard RNNs in which all weights are adjusted through gradient descent methods. Its low computational costs, as well as its temporal dependence makes RC an effective architecture for low cost real-time computations (Jaeger, 2002). Practical real world problems like rapid, real-time language processing, where prior words in a sentence influences the processing of a current word in the timescale of hundreds of milliseconds have been simulated using an RC (Hinaut & Dominey, 2013).

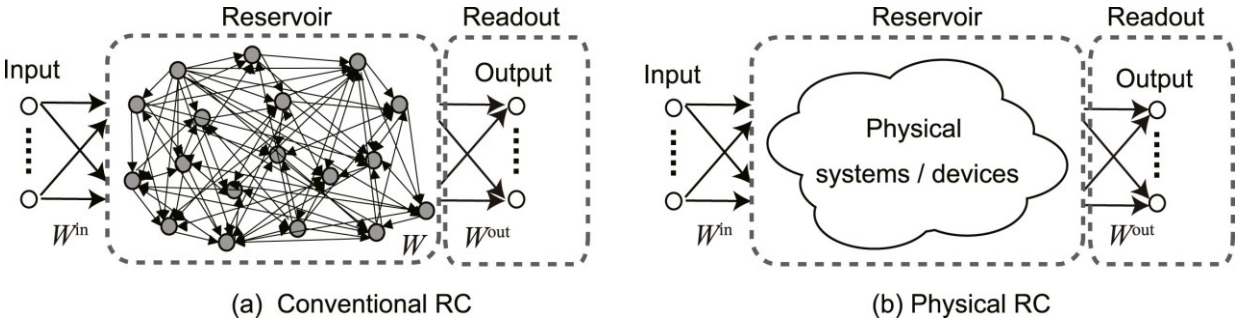


Figure 3 Fig. 1 from Tanaka et al. (2013). RC framework, the reservoir is fixed and only the readout weights are altered. (a) A standard RC system with an RNN-based reservoir, where nodes are randomly connected with recurrent pathways. (b) An RC system in hardware, where the reservoir is realized using a physical system or device.

With its attractive low learning costs and fast information processing power of real-time problems, RCs potentially provide a paradigm for physical hardware that is more efficient in training time and costs in comparison to current machine learning computing hardware. According to Tanaka et al. (2019), physical RC has potential as an unconventional computing

paradigm based on novel hardware (figure 3b). Coincidentally, the current boron atom based hardware in question appears to be architecturally similar to the architecture of an RC.

4.1 Comparison to boron system

Like a boron cell system, RCs are a black-box filled with nodes that form random connectivity paths, and moreover, the weights within the cell are fixed, as are the conduction paths between boron atoms. The similarities between an RC and a boron cell are striking at first glance, however, unlike an RC, the boron cell system is capable of altering the fixed dynamics within the cell by adjusting voltages of the control electrodes.

These control electrodes can alter the nature of the reservoir, creating dynamic solution spaces instead of a fixed one. To effectively capture and test an RC based boron cell simulation, a basic RC system should be first simulated and tested using basic linear learning algorithms like perceptron. Finally, the properties, and ultimately, effects of the control nodes can be implemented once a basic RC system is working, and capable of solving standard computational problems like the Boolean logic gates. This would not only address the issue of how RC can be used to simulate boron cells, but it also enhances the RC framework itself, as an RC in combination with the effect of control nodes has not been simulated yet.

4.3 Properties of an RC

As Tanaka et al. (2019) suggests, there are several requirements for a physical reservoir to effectively solve problems. Firstly, high dimensionality is necessary for the high dimensional mapping of inputs, and is related to the number of independent input signals obtained from the reservoir. In practice, this can be difficult to achieve, as randomly formed connections forms random amounts of independent input signals. Secondly, nonlinearity is also necessary for the reservoir to perform nonlinear mapping, which transforms inputs that are not linearly separable to ones that are in classification tasks. Finally, the responses of a reservoir to distinct input signals must separable into different classes, but should also be insensitive to unessential, small fluctuations so that similar inputs can be grouped as the same class. Conclusively, it is recommended that reservoir system parameters (amount of nodes, sparsity of connections and connection weights) are set close to the transition point between highly nonlinear and more linear regimes so that expanding and contracting transformations are limited.

5 Simulating the boron dopant cell

The initial goal was to simulate the architecture and input, output behaviours of the boron cell as a reservoir network. Python 3.7 was chosen as the preferred programming language due to its accessibility and learnability. In accordance with the descriptions in Chen et al. (2020), the boron dopant cell (figure 4) architecture consists of two input (V_{in1} to V_{in5}), one output (I_{out}), 5 control electrodes (V_{c1} to V_{c5}) and a reservoir of disordered boron dopant atoms.

Without considering the controls, the cell can be simulated as a neural network, through vectors and matrices that represent the network layers (input, reservoir and output) and connections. Three vectors are used to represent node charges (activations) of the input, output and reservoir layer of the cell; four matrices are used to represent the connections between and within the layers: input *x* to reservoir, input *y* to reservoir, reservoir to reservoir, and reservoir to output. The connection matrices represent the connection strength values between input, reservoir, output layers and between cell nodes within the reservoir. The architecture is shown and described below in figure 4.

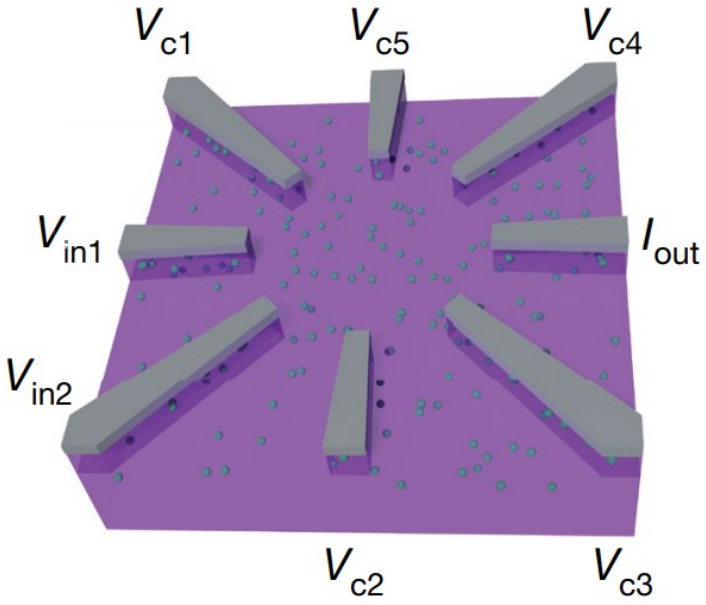


Figure 4. Boron dopant cell from Chen et al. (2020). The turquoise coloured dots are the injected, disordered boron dopant atoms ontop of silicon (purple). V_{in1} to V_{in2} are the input electrodes, V_{c1} to V_{c5} are the control electrodes, and I_{out} is the output electrode. From Chen et al. (2020).

There are several properties that interactively alter the dynamics of the network, namely, the solution spaces that are generated:

1. The random number generator (given by seeding numbers in Python) used to determine connection potentials for all layers
2. The size of the reservoir, or, 'cell size'
3. The connection density/sparsity within the reservoir
4. The connection density/sparsity between input, output and reservoir
5. The connection strength (weights) between all layers.

Figure 5 illustrates the cell architecture. Much like the actual boron cell, all connections between input, reservoir and output are randomly generated, its densities controlled by a value between 0 and 1, indicating the sparsity of connections in or between given layers.

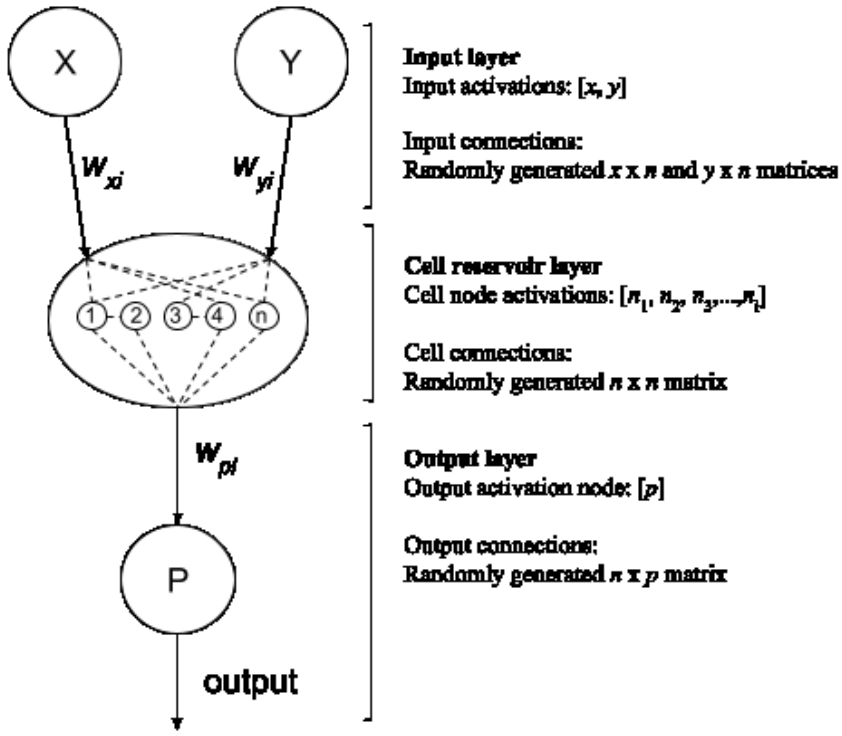


Figure 5. Basic architecture of the simulated boron cell. The input nodes, a matrix holding 2 (x, y) input values, are connected to the reservoir (x x n and y x n connection matrices represented as W_{xi} and W_{yi}) via randomly generated connections that are controlled by the sparsity parameter (0 – 1, 1 being no connections and 0 being fully connected). The randomness of the connections can be controlled and reproduced via randomizing seed values, that ensure the same random values are generated. The cell reservoir consists of n boron atoms, and are also randomly connected to one another based on sparsity. Finally, the output connection and node is simulated in the same manner as the input connections using sparsity

Within the reservoir, each boron atom cell node is modeled with its own activation function (hyperbolic tangent), which squashes the incoming activations to a value between 1 and -1. The process of squashing could be comparable to the tunneling properties of electrons in the dopant cell, but also avoids over inflated levels of activation, and given its nonlinear nature, can be used to induce further nonlinearity in the reservoir. For now, the functions of the controls can be ignored, as our interest lies mainly in the behaviour of input activation when it goes through a reservoir like the boron dopant cell. Like all gradient based machine learning architecture, the present architecture suffers from potentially vanishing gradients of activation when activation within the cell is too high. This can occur from an interplay between the parameters: cell size (n), connection sparsity, and connection strength (w), as increasing either one of the parameters will increase the overall activation of the cell. However, unlike sparsity or connection strength, the effects of cell size are not always present, since it is possible that connections are not made with newly added cell nodes. But in general, increasing cell size will increase the potential of complex connection structures, and more independent connection streams, which increases the amount of complex problems it can potentially solve through varying connection seeds. Nevertheless, the increased complexity comes with an increased activation, as more cell nodes and connections will lead to more overall activation. To create more complex solution spaces while avoiding the loss of information gradient may be difficult, but adjusting cell size while balancing sparsity and connection strength will help increase the complexity, and therefore, adaptability of the cell while avoiding losing gradient from over amplifying input activation.

To test the basic capabilities and the effects of and interactions between various parameters of the network architecture, the settings mentioned above will be varied while the network attempts to solve perceptron problems like the Boolean logic gates (AND, OR, NAND, NOR, XOR and XNOR). Once that has been achieved, the function of the control nodes and the architecture to support it will be determined. The perceptron will be used as the initial benchmark, as it is capable of reliably solve all Boolean logic gates. Once the cell(s) and architecture is capable of solving all known Boolean logic gates using control nodes, it can then further be tested with simplified digit recognition test similar to that from Chen et al. (2020). Lastly, if the simulation of the Boron dopant cell has managed to mimic what Chen et al's Boron dopant cell has accomplished (namely, solving all Boolean logic gates and performing simple

digit recognition task), the use of boron-reservoir cells can be tested with simple cells as used by Serre et al. (2007).

5.1 Testing network architecture on Perceptron logic gates

In this initial explorative phase, the cell configuration parameters were randomly selected. The following cell configurations were tested in all possible combinations: Cell sizes 10, 20, 50 and 100; randomizer seed combinations ([input connection seed, cell connection seed, output connection seed]) [1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]; connection strengths 0.2, 0.5, 0.8, random (between 0 and 1); sparsity value of 0.8, 0.85, 0.88, 0.92, 0.95, 0.98. These parameters are tested with input and output values 1 and 0.

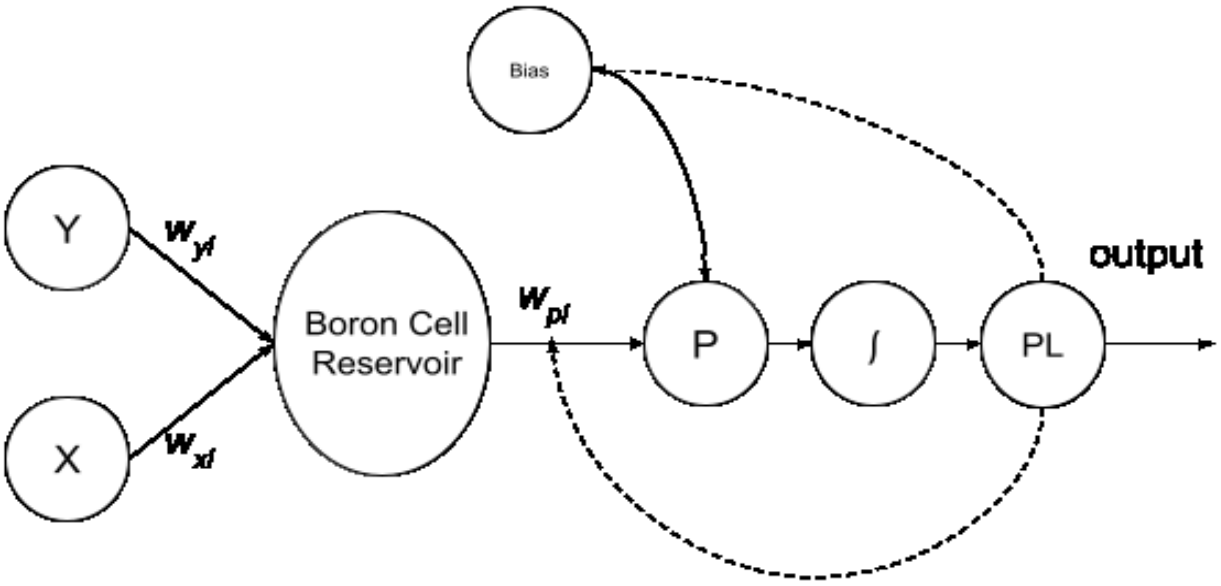


Figure 6. The boron-reservoir cell architecture with perceptron learning rule. After output node P, bias is added to an output copy, which is fed into a step function. The step functions purpose is to make the error checking process easier, and checks if the total output with bias exceeds 0, if it does, the stand-in output is set to 1, if not, it is set to 0. The output is further fed into the perceptron learning algorithm, where the error is calculated with stand-in output subtracted by desired output. This means that the error can be used to decrease or increase the output connection weights (error = -1, error = 1 respectively), or to verify that the output is correct (error = 0). The error is then further used to update the bias accordingly. Adjustments are made only in the output connections and bias as is conventional for perceptron learning. The connection weights are as illustrated in figure 5.

To solve all Boolean logic gates, the basic boron-reservoir architecture is combined with the perceptron learning algorithm, and uses input values of 1 and 0. Illustrated in figure 6, the algorithm takes the difference between an expected and actual output as an error measurement to adjust the bias and the connection weights between the reservoir and the output node P. The size of each correction is dependent on the learning rate, and the amount of corrections is determined

by the amount of learning runs. If the learning rate was too large, there is a chance that a functional network setting does not solve the logic gates due to over-learning. As such, it was also important to explore the effects of differentiating slightly between learning rate and runs. Thus, in addition to the variables mentioned above, they were also tested with learning rates of 0.1, 0.01 and 0.001 at 100 learning runs, and 1000 learning runs.

Regardless of whether the cell is capable of creating a valid solution space or not, when the ratio of learning runs to learning rate is too low, it may fail to fully adjust either the bias or the connection weights. Additionally, the learning rate itself can also be too high, as some cells may require finer learning intervals to solve logic gates with its given activation values. Thus, the effective learning rate to run combination is examined first. Between all possible learning rate to run combinations, 1000 learning runs with 0.1 and 0.01 learning rate had the highest success rate by far (Appendix 1, table A). Looking further into the data (Appendix 1, figure 2 and 3), the majority of successful cases happen within the first 100 runs for learning rate 0.01, and 10 runs for learning rate 0.1. Since the rounding function that the learning algorithm employs rounds only up to four decimals and the learning rate is significantly higher than the differences between outputs, it is likely that solutions that occur past 200 learning runs are due to floating-point hardware limitations, where decimals are only ever approximations of fractions, and will therefore never be exactly the same each time it is called (Peters, 2020). Consequently, the 'convergences' that occur after 100 (learning rate 0.01) and 10 (learning rate 0.1) learning runs happen very randomly, and the differences in output are always so small that it is impossible to attribute to the learning algorithm given the learning rate. Still, within the first learning run intervals of both learning rates, a learning rate of 0.01 had far more successes (595 successes for 0.01, as opposed to 490 for 0.1). Given that they have similar overall success rates, using 0.01 as the default learning rate would reduce the chances of over learning.

The results of the 1000 learning run, 0.01 learning rate test to solve all Boolean logic gates are displayed in table 1a and 1b. A total of 20 out of 480 cell configurations were able to solve all Boolean logic gates. A cell configuration is considered a success when it is able to solve all of the nonlinear Boolean logic gates, a partial success when it is able to solve at least one, and a failure if it was not able to solve any. As expected, by inserting a reservoir into a perceptron learning algorithm, a level of complexity is induced which reduces the success rate of solving all

linear logic gates, but also introduces the possibility of solving nonlinear logic gates. These results imply that the randomly generated connections of the reservoirs are sometimes capable of generating something similar to that of a multilayered perceptron network, but that they can also prevent simple linear logic gates from being solved as a result of the randomness.

Table 1a.
Overall successes of all cell configurations

	Success	Partial	Failure
OVERALL	20	251	209

Table 1b.
Successes of each logic gate in all cell configurations

Logic gate	Success	Failure	% S
OR	271	209	56.5
NOR	247	233	51.5
AND	113	367	23.5
NAND	88	392	18.3
XOR	23	457	4.8
XNOR	37	443	7.7

With the remaining results of the test runs, the capabilities and function of the network can be analyzed at each setting combination. To understand how the parameter affects the success of solving linear logic gates, each parameter is examined on its own to see how it individually affects success rate; additionally, the effects of interactions between parameters are also examined. The results are split into tables A, B, C, D, E, F, G in appendix 2.

First, appendix 2 table A displays the rate of success of the different random number generator seed values used to determine the network connections. Since every seed combination is tested with every other parameter combination, table A can be used to determine the overall most successful seed combination to ensure that future networks have a higher chance of generating usable network structures. From the table, it appears that the seed combination [1, 2, 3], [4, 5, 6] and [7, 8, 9] had the most success with 5 cells solving all logic gates, while the seed combinations [10, 11, 12] and [13, 14, 15] performed slightly worse, with 3 and 2 successful cells respectively. Nevertheless, it appears that there are no large differences between seeds, as most of them were able to generate 5 successful cells, with two exceptions that were still fairly similar.

Appendix 2, tables B, C, D display the success rate while varying connection strength, sparsity and cell size. From the range of parameters that were chosen, a connection strength of 0.2 had overwhelmingly most amount of successes (15 compared to the next highest at 3), and

the same could be said about cell size 50 (11 successes compared to the next highest at 5). There was not a discernable pattern in the results of the sparsity table (table D), however, it seems that success rate drastically lowers to 0 past a sparsity value of 0.92. Ultimately, while these values mean fairly little by themselves, they provide valuable information about the successfulness of cell configurations within the predetermined ranges. For example, that a connection strength of 0.8 is likely far too large for cell sizes ranging from 10-100, when connection sparsity ranges from 0.8 to 0.98, and results in 0 successes. Likewise, sparsity values above 0.92 may induce too little connections within boron cells of size 10-100, leading to no successes. Interestingly, as cell size increases, the amount of complete failures reduces, and partials increases. This matches the initial expectations of how increasing cell size affects the cell, increasing complexity and potential to solve problems. Nevertheless, perhaps this increased solution space complexity comes at a cost of problem solving capabilities, as the amount of successes is also much lower than that of cell size 50. Conclusively, a cell size of 50, with a connection sparsity of between 0.8-0.92 and a connection strength of 0.2 are the configurations with the highest likelihood of success.

Table 1c.

Example of outputs from cell configuration: cell size 50, sparsity 0.8, connection strength 0.2, seed [1, 2, 3]

Logic Gate	Learning Run	Output per input: [(1,1), (1,0), (0,1), (0,0)]
AND	17	[0.00147314 -0.00391926 -0.00421121 -0.29]
NAND	251	[-0.00100592 0.0019038 0.00183239 0.03]
OR	0	[2.12109019 2.0975272 2.08868218 -0.1]
NOR	34	[-0.12185593 -0.12281558 -0.12138665 0.01]
XOR	394	[-7.73845112e-04 2.12869259e-03 1.98061737e-04 -2.60000000e-01]
XNOR	136	[6.63445324e-05 -4.65378071e-03 -3.04653326e-03 2.00000000e-02]

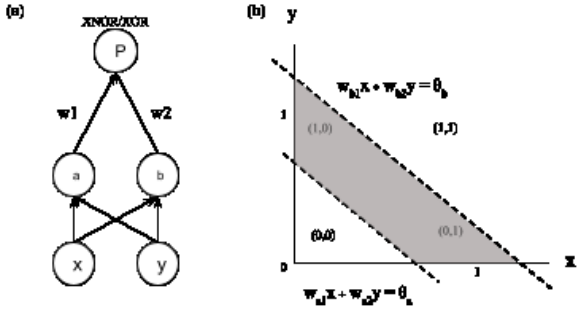
Table 1c displays the outputs from a sample cell that has 50 nodes, connection sparsity of 0.8 and connection strength 0.2. Similar to figure 2c from Chen et al., a large difference between outputs can be observed. Like in figure 2c, the nonlinear logic gates XOR and XNOR produce a much smaller output in comparison to the linear logic gates. Moreover, the OR gate produces significantly higher output compared to the other logic gates. While these results do not pose a problem currently, just like Chen's cell, the difference between outputs will eventually pose a problem for up-scaled, multilayer networks. Nevertheless, for now, the reliability of solving each logic gate remains the main focus.

At this stage of development, the algorithm behaves similarly to a standard reservoir computing system, where only the weights between reservoir and output are adjusted during

learning. To introduce the control nodes in the system, the assumption was made that control nodes would have one of three potential effects. Firstly, control nodes that would be able to reduce or increase the activation of cell nodes, by directly inputting positive or negative activation. Secondly, control nodes that would reverse the sign of the input connections from positive to negative, or the other way around. Finally, control nodes that would reduce or increases activation within the output node. When compared to the boron cell in figure 2 from Chen et al., inputs 1 and 2 would have their connections reversed by controls Vc1 and Vc2, and the output would be adjusted by control Vc4. The boron atom nodes in the cell would be adjusted by controls Vc3 and Vc5, each influencing a different sub-part of the cell, and thus potentially two different patches of boron atoms.

To build an architecture that supports the use of these control nodes, the boron-reservoir cell as simulated above will be split into two sub cells, so that controls 3 and 5 would be able to influence two different boron atom patches (sub cells). First, however, in the following section, the boron-reservoir cell is split into two sub cells without the use of control nodes. This architecture is tested again on the perceptron algorithm used above to see how the new architecture alters its ability to solve Boolean logic gates.

5.2 Splitting the cell into sub cells



(c)

x	y	a	b	a + b	=	aXORb
1	1	1	0	1		0
1	0	1	1	2	Reduce with bias	1
0	1	1	1	2		1
0	0	0	1	1		0

(d)

x	y	a	b	a + b	=	aXNORb
1	1	1	0	1		1
1	0	0	0	0		0
0	1	0	0	0		0
0	0	0	1	1		1

Figure 7. Multilayer perceptron solving XOR and XNOR. (a) an extra layer of nodes a and b are necessary to create the nonlinear projection needed for solving XOR and XNOR. (b) the solution space required to solve XOR, two parallel linear functions of the perceptron weights (w1 and w2) are used to create the solution boundaries. (c) Truth table showing the schematic for solving XOR and XNOR, for the former, sub cell A solves for OR while sub cell B solves for NAND, resulting in XOR. For the latter, sub cell A solves for AND while B solves for NOR, resulting in XNOR.

Figure 7 shows the truth table and diagram schematics for solving XOR and XNOR using a standard multilayer perceptron. In a multilayer perceptron, two additional nodes, a and b (figure 7a) are introduced. By adjusting the weights w_1 and w_2, a solution boundary can be set to solve nonlinear problems like XOR (figure 7b, the darkened area marks the solution space). If a boron-reservoir cell is split into two sub cells, following Chen et al. (2020), a genetic algorithm could be used to evolve the sub cells to learn the two required linear logic gates used for either XOR or XNOR.

Due to time and resource constraints, the present program will not be using the same method. Instead, since the purpose is to demonstrate the capabilities of the network architecture illustrated in figure 8, the sub cells are not simulated to solve each logic gate separately. This is not needed either in this approach because even if either, or neither, sub cells converge, the learning algorithm will still attempt to solve the final logic gate. In this manner the architecture in figure 8 behaves more similarly to a genuine reservoir than to a multilayer network illustrated in figure 7, as in a reservoir there are no sub cells to be solved either. Below, the application of perceptron learning to a sub cell boron-reservoir architecture is explored.

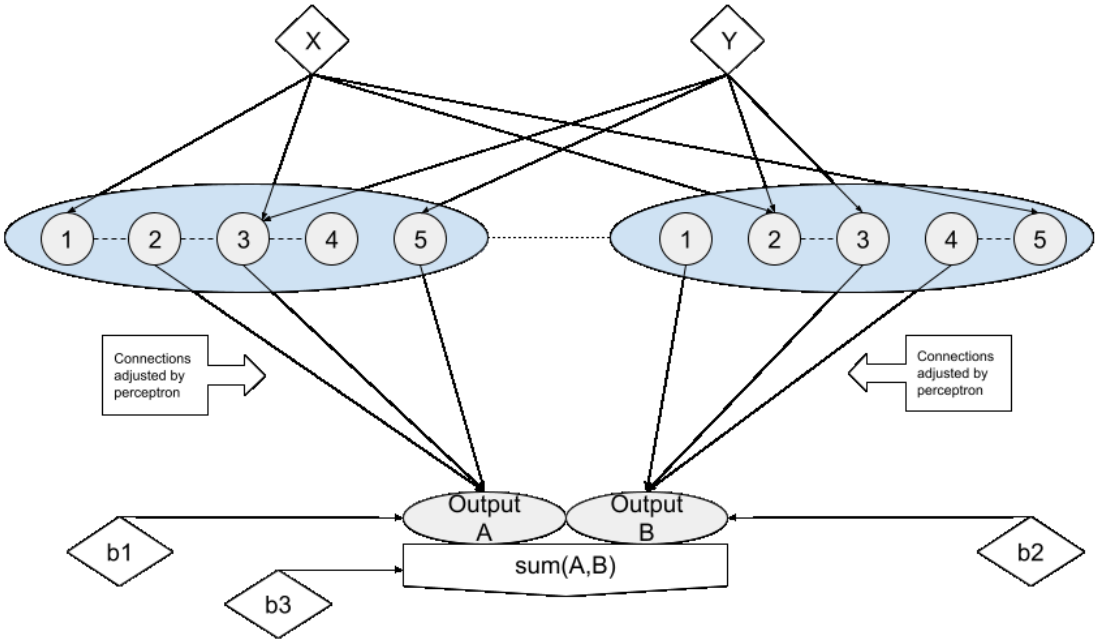


Figure 8. Boron-reservoir architecture with sub cells. Functionally, there are no differences between the single or sub cell iterations of the boron-reservoir architecture. However, the connections can now be controlled in a finer manner as connections between boron sub cells can now be controlled via external sparsity. The architecture still uses perceptron learning rule, where the output weights are adjusted, and bias b is applied to the final output.

Figure 8 displays the new sub cell architecture of the boron-reservoir cell. The cell is split into two sub cells that are connected internally, and to one another through separate connection matrices. The internal connections within a sub cell, represented by the dotted line between the numbered nodes of the (blue) sub cells of figure 8, remain controlled by the internal sparsity (1 being no connection, 0 being full connection). The connections between sub cell nodes, represented by the dotted line between the two sub cells in figure 8 are now separately dictated by external sparsity.

Tables 2a and 2b display the amount of successes in the sub cell test results. Table 2b compares the results obtained with the sub cell approach in figure 8 (column 'sub cell' in table 2b) with the results obtained with the single cell approach given in table 1b (copied in column 'single cell' in table 2b). In total, there were 0 cases of full successes.

.

Table 2a.
Overall successes of all cell configurations with sub cell architecture

	Success	Partial	Failure
OVERALL	0	6548	2668

Table 2b.
Comparison of- successes of each logic gate in all cell configurations in both single (3rd column) and sub cell (2nd column) simulations.

Logic gate	sub cell(%)	single cell(%)
OR	6548(71.1)	271(56.5)
NOR	6446(69.9)	247(51.5)
AND	1364(14.8)	113(23.5)
NAND	1060(11.5)	88(18.3)
XOR	11(0.1)	23(4.8)
XNOR	13(0.1)	37(7.7)

There were a total of 13, and 11 successfully solved cases of XNOR and XOR respectively; however, they were never solved simultaneously by the same cell settings for the sub cells. Compared to the single cell iteration of the algorithm, the OR and NOR have a much higher success rate in the sub cell iteration as a result of the added combinations of two seeds in each sub cell within a cell (e.g., single cell iteration has [1,2,3], and [5,6,7] as seeds; sub cell

iteration has [1,2,3,4] + [1,2,3,4] and [1,2,3,4] + [5,6,7,8] and more). On the other hand, AND and NAND have a much lower success rate in the sub cell iteration. Since the AND and NAND gates already had difficulty solving in the single cell version, having two cells that need to simultaneously solve AND or NAND together creates an additional barrier to solving; thus, leading to a lower success rate in the sub cells version.

The large amount of partially solved cells can be explained as follows. Since two separate cells are now used when solving linear logic gates in the sub cell iteration, what previously would have been solved in the single cell iteration will now only solve again when combined with another successful cell in the sub cell iteration. Thus, overall, there will be a lot more cases of partials, and a lot less cases of success when comparing the sub cell to the single cell results. The increase in partials and decrease in failures can be attributed to the increase in successful ORs and NORs and unsuccessful ANDs and NANDs, as well as the seed combination effect of sub cells. Since OR and NOR only need to differentiate any activation with no activation, it is not as affected by the sub cell configuration as AND and NAND are. In this sense, the ability to solve linear logic gates differed slightly.

The effects of seed, cell size, internal (within sub cell) sparsity and external (between sub cells) sparsity are displayed in tables in appendix 3. With a lot more cell configurations, the difference in effect of varying seeds are now practically neutralized, as all seeds result in similar ratios of success, partials and failures. Much like its predecessor, the cell sizes (appendix 3 table B) of each sub cell served to increase the complexity of the system, and as cell size increases, so does the amount of partial solving systems.

The next tables, appendix 3 table C and D, examine the effects of external and internal sparsity on the cells ability to solve all logic gates (left), and linear logic gates (right). Overall, it seems that as internal sparsity increases, the amount of successes decrease, with an internal sparsity of 0.8 solving overwhelmingly more (144 successes, next highest is 100) than any other sparsity value. On the other hand, as external sparsity increases, so do the amount of cells that can solve the linearly separable logic gates, this trend continues till sparsity 1 (no connections), where the highest amount of success stands (200, as opposed to the next highest success rate of 143). With sparsity now split into internal and external, the results indicate that the more separate the two sub cells are, the better the overall system is able to solve the linearly separable logic

gates (high external sparsity). Additionally, for the chosen testing variables of our system, a sparsity of 0.8 has the most success, likely due to the fact that as internal sparsity increases, the chances of input reaching output decreases due to the reduced chances of connections being made within the cell. With two sub cells each required to have a functional input to output connection, the successes and partials of a lower sparsity is further amplified.

Although XOR and XNOR were seldom solved, overall, the remaining results were fairly consistent; each of the variables (cell size, connection strength) behaved comparably to the previous single cell iterations (Appendix 2). While the nonlinear logic gates were not simultaneously solved in any cell configurations using perceptron learning, the use of control nodes could prove to be different. In the next section, the architecture of the control nodes is introduced and simulated.

5.3 Control nodes

To mimic the effect of the control nodes, as described in Chen et al. (2020), three effects of the control nodes (illustrated in figure 9) are assumed to influence the relationship between input and output:

- Controls 1 and 2 (C1/C2): Reverse the connection sign (+ to – and vice versa)
- Controls 3 and 4 (C3/C4): Increase or decrease activation of all sub cell nodes
- Control 5 (C5): Increase or decrease activation of output node

C3 and C4 achieve non linearity through the vast and random connections within the network; due to the random connections, each node has a unique effect when additional activation is given to it (via C3 and C4), as the amount and strength of connections made between it and other nodes will vary from different nodes in the system. Additionally, the effects of the controls are applied prior to the activation function that squashes the activation of each reservoir node between -1 and 1, in an attempt to induce further nonlinearity. Together, C3 and C4 mimic the functions and capabilities of the perceptron learning algorithm by manipulating the input to output relationship through decreases and increases in activation, and as such should be able to solve known solvable perceptron problems.

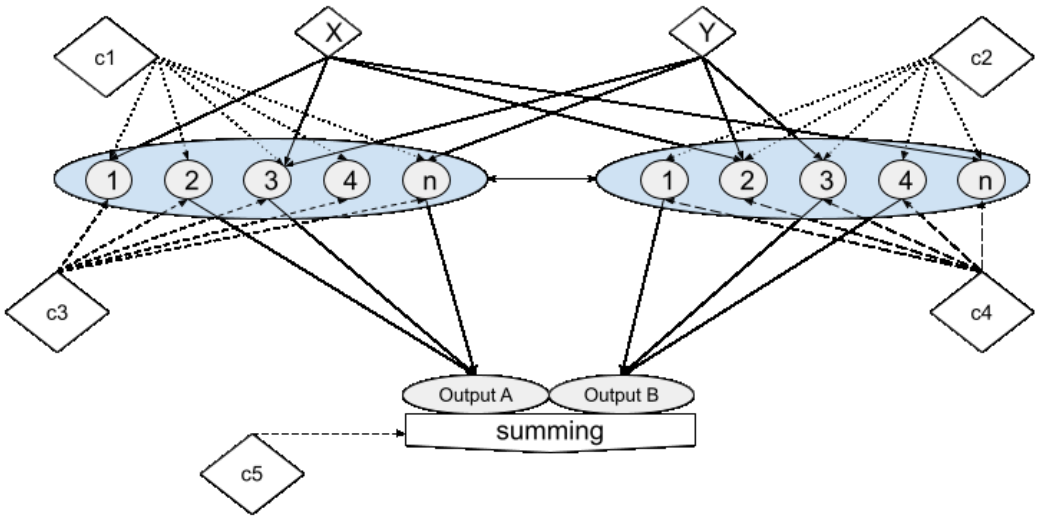


Figure 9. Schematic for control nodes implemented onto the boron dopant sub cell algorithm. C1 and C2 reverses the input connection signs of either one of the sub cells. C3 and C4 add or subtract activation onto all cells within sub cell A and sub cell B respectively. C5 adds or subtracts activation from the output (summing) node.

Unlike the previous simulation, where perceptron based learning rule is used, the current system of controls does not adjust any weights in the network, but instead influences the output by directly adding, subtracting or reversing activations in the reservoir. Functionally speaking, it is very similar to the perceptron learning rule, in the sense that it has the capabilities to reverse input activation (reverse sign connections in the case of controls, or negative/positive bias in the case of perceptron) and the ability to increase or decrease the output activation. Their discrepancy lies in the way the learning algorithm affects the boron-reservoir cell, and the way additional activation is given or taken from the cell. While perceptron learning increases or decreases the output connection weights and bias to the output, the activation changes realized through the control nodes are spread through the reservoir first before it reaches the output. An additional advantage of the controls is that C1/C2 (the sign connection reversers) are able to instantly reverse activation, whereas the perceptron learning rule will have to slowly and linearly decrease and adjust until the output connection weights are below 0; this makes solving problems like NOR and NAND much faster. Moreover, the effect of control nodes would be implementable in neuromorphic hardware as illustrated with the actual boron system (assuming that the effects of control are similar). In contrast, implementing a learning rule (and connection weights in general) in neuromorphic hardware is very difficult.

The results for the sub cell with control simulation is displayed in tables 3a and 3b. Since it appeared that the randomizer seed didn't make that much of a difference, each sub cell now has a dedicated seed ([1, 2, 3, 4] and [5, 6, 7, 8]) in order to reduce the amount of time it takes to run all configurations while varying seed. Table 3b compares the results obtained with the sub cell approach using control in figure 9 (column 'Control' in table 3b) with the results obtained with the sub cell approach using perceptron learning given in table 2b (copied in column 'Sun cell' in table 3b).

Table 3a.
Overall successes of all cell configurations with sub cell and control nodes.

	Success	Partial	Failure
OVERALL	0	404	172

Table 3b.
Comparison of successes of each logic gate in all cell configurations in both control node (3rd column) and sub cell (2nd column) simulations.

Logic gate	Control (%)	Sub cell(%)
OR	403(70.0)	6548(71.1)
NOR	403(70.0)	6446(69.9)
AND	242(42.0)	1364(14.8)
NAND	245(42.5)	1060(11.5)
XOR	23(4)	11(0.1)
XNOR	16(2.8)	13(0.1)

The results appear to be far better in comparison to the previous perceptron based simulation (Table 2). Improvements in success rate can be seen across all logic gates. However, overall, there were still no configurations that could simultaneously solve all logic gates. Regardless, it appears that adjusting control nodes that affect the reservoir results in an increased rate of success for different cell configurations compared to perceptron learning.

In an attempt to further improve the success rates of solving the nonlinear logic gates, each sub cell is further split into two cells, creating four boron patches (four boron sub cells). By further splitting the sub cells, the input signals within each sub cell can be split and controlled to artificially induce higher dimensionality within the reservoir, but also to guarantee at least two distinct input signals within all randomly generated reservoirs. Since an external sparsity of 1 gave the best results in the previous iteration, the boron sub cells (B1+B2 and B3+B4) will have no connection in between them, instead, external sparsity now dictates the connection between boron sub cells (B1 and B2, B3 and B4 in figure 10). Now, the control nodes can be used more specifically to solve for XOR and XNOR Figure 10 and 11 shows the new architectural design for the boron cell and the role of control in each case to solve for XOR and XNOR respectively.

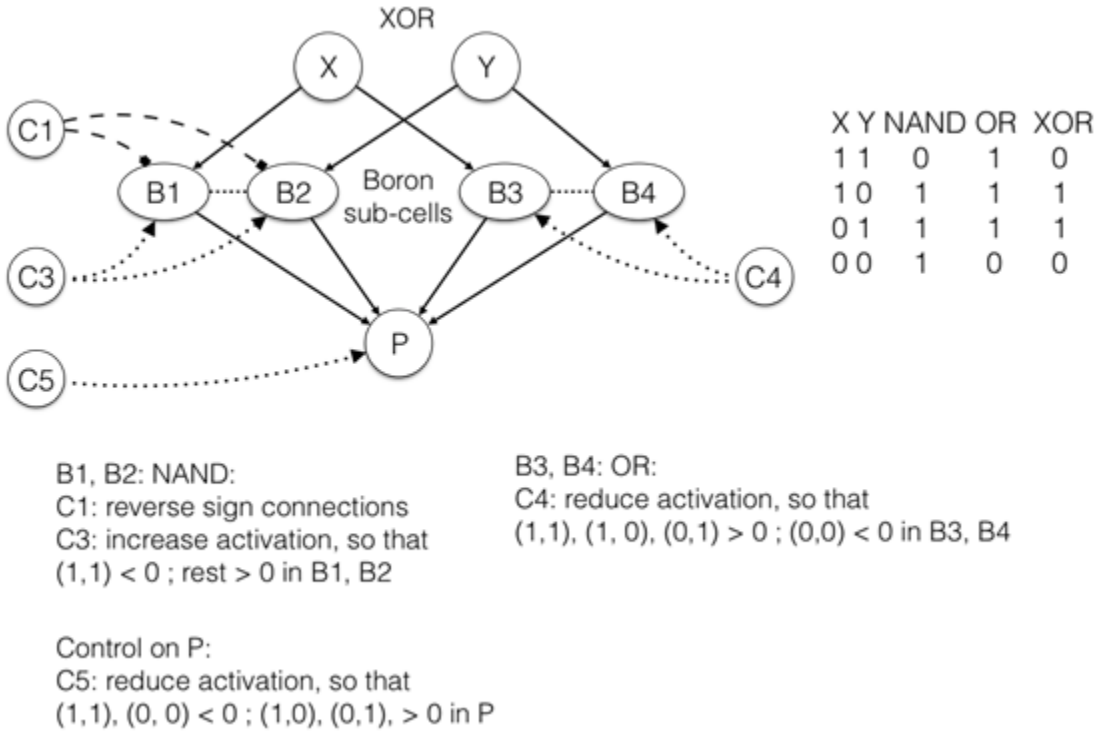


Figure 10. Boron Cell schematic for solving XOR using control nodes and four sub cells. C1 and C3 will help the sub cell pair B1 and B2 solve for NAND, while sub cell pair B3 and B4 solves for OR. Their resulting summer activation at output node P is reduced until XOR is solved. In the case of the figure, the summed output [1, 2, 2, 1] for inputs [(1,1), (1,0), (0,1), (0,0)] are reduced by 1 so that the output becomes [0, 1, 1, 0], solving for XOR.

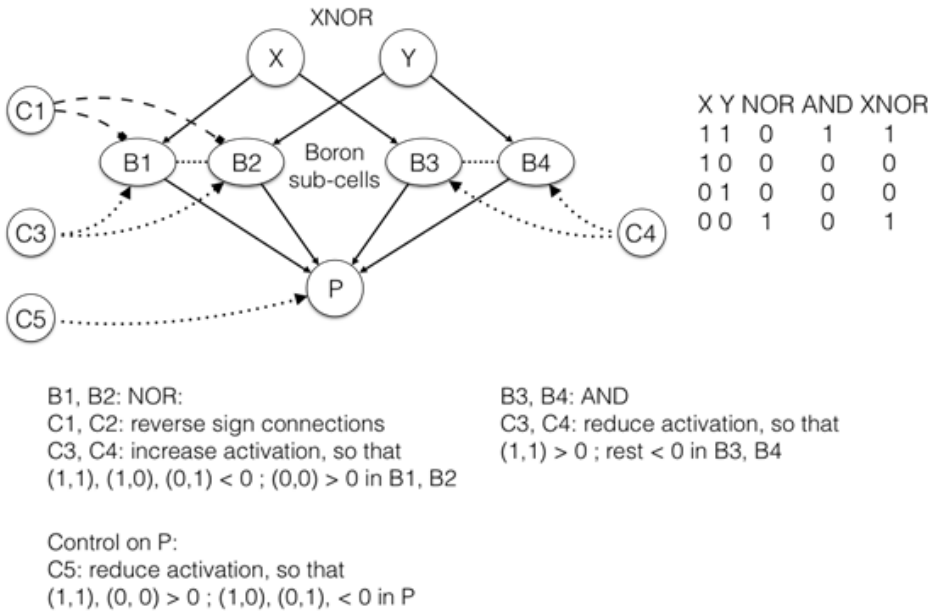


Figure 11. Boron Cell schematic for solving XNOR using control nodes and four sub cells. C1 and C3 will help the sub cell pair B1 and B2 solve for NOR, while C4 helps sub cell pair B3 and B4 solves for AND. Their resulting summer activation at output node P is reduced until XNOR is solved. In the case of the figure, the summed output [1, 0, 0, 1] for inputs [(1,1), (1,0), (0,1), (0,0)] does not need to be reduced to solve XNOR.

Table 4a.
Overall successes of all cell configurations with 4 sub cell and control nodes.

	Success	Partial	Failure
OVERALL	0	398	177

Table 4b.
Comparison of successes of each logic gate in all cell configurations in both the previous control node (3rd column) and the new split control (2nd column) iterations.

Logic gate	4 cell control (%)	Control (%)
OR	399(69.2)	403(70.0)
NOR	397(68.9)	403(70.0)
AND	178(31.0)	242(42.0)
NAND	179(31.1)	245(42.5)
XOR	2(0.3)	23(4)
XNOR	24(4.2)	16(2.8)

Overall, the new four sub cell iteration performed slightly worse than the previous simulation. It appears to have a significantly harder time solving for AND and NAND, while OR and NOR remain fairly unaffected. Tables 4a and 4b display the results and a comparison with the previous iteration (the column 'Control' in table 4b, copied from table 3b).

Unexpectedly, the rate of success in solving nonlinear logic gates is poor, with only 2 XOR gate and 24 XNOR gates being solved out of a total 576 cases respectively. It appears that the architecture is still too linear to consistently solve XOR or XNOR. There could be several potential reasons for this. Most notably, when solving XOR, since the OR gate sub cell does not need to be adjusted to be solved (it only needs to distinguish activation from no activation), whereas the NAND sub cell requires several adjustments to be solved. This means that the OR sub cell output will be significantly larger than the NAND output, in particular, with inputs (1,1). Tables 5a and 5b shows the effects of the controls on the XOR truth table.

Table 5a.
XOR truth table after C1 (*-1)

X	Y	NAND	OR	XOR
1	**1**	-2	2	0
1	**0**	-1	1	0
0	**1**	-1	1	0
0	**0**	0	0	0

Table 5b.
XOR truth table after C3 (+1.5) and C4 (+0)

X	Y	NAND	OR	XOR
1	**1**	-0.5	2	1.5
1	**0**	0.5	1	1.5
0	**1**	0.5	1	1.5
0	**0**	1.5	0	1.5

As C1 reverses the input activation for the NAND gate (table 5a), enabling input (1,1) to produce a negative output (table 5a, -2 in the NAND column), inputs (1,0) and (0,1) become negatives (table 5a, -1s in the NAND column) as well. To compensate, C3 (table 5b) adds activation to the reservoir to increase the outputs of inputs (1,0), (0,1) and (0,0) to a positive while keeping input (1,1) a negative. As a result, both the NAND and OR gates are now solved, however, XOR remains unsolved. The problem lies in the lack of adjustments on the OR gate at inputs (1,1) and (0,0); alongside the inflated NAND output for input (0,0). In summary, the controls C3, C4 and C5 are not able to balance the outputs in a manner which keeps inputs (1,1) and (0,0) lower than that of (1,0) and (0,1) to solve for XOR.

Further inspection of the data confirmed our schematic ideas in the cases for some cells, as the outputs of many XOR cases are either [0, 0, 0, 0] or [0, 0 ,0, 1]. In the case of [0, 0, 0, 1], the amount of adjustment that NAND requires to make input (0,0) positive is uncompensated for by the OR cell which requires no adjustments. To combat the imbalance in adjustments, the idea of a saturation cell is introduced. The saturation cells will normalize the output of sub cells, and furthermore, could potentially act as a method of normalizing outputs in the real boron cell as well in order to solve the large differences in outputs of different logic gates as presented in

Chen et al. (2020) and illustrated in table 1c. The schematics and architecture for the saturation cell implementation is described in the following section.

5.4 Saturation cells

The results of the control node tests indicate that the present network architecture still is not able to solve the nonlinear logic gates in a sufficient manner. This is likely because the network is still too linear, since an input of (1, 1) and (0, 0) will result in a much larger output than (1, 0) and (0, 1) as shown previously. To induce further nonlinearity, saturation cells, which standardizes the output of either sub cell (to the standards of the saturation cell configurations), are introduced to increase the output of inputs (1, 0) and (0, 1). Figure 12 illustrates the use of saturation cells.

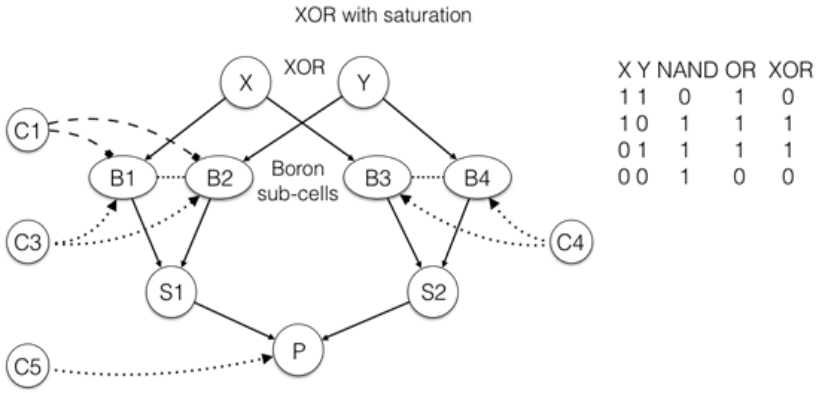


Figure 12. Schematic for four boron sub cell with saturation cells. Saturation cells S1 and S2 "normalize" the outputs of each sub cell to 1 or -1, thus, removing the problem of outputs that are highly positive or negative due to the randomness of the networks.

The sub cells B1 and B2 feed into saturation cell S1 and the sub cells B3 and B4 feed into saturation cell S2. The direct effects of the saturation cells are essentially to normalize the outputs according to the amount of sub cell nodes and the activation function. In the case of the chosen activation function for the algorithm (tanh(x)), it would maximize all outputs (B1 + B2, B3 + B4) so that all values above zero become one (1), and all values below zero become negative one -1). This eliminates the huge differences between sub cell outputs, so that the control node C5 can reduce inputs (1, 1) and (0,0) to be lower than (1, 0) and (0, 1). The schematic and effect of the saturation cell is illustrated further in figure 12.

The results of the saturation cell simulations are displayed in tables 6a and 6b. Narrowing cell sizes down to 20 and 50 (10 had too little successes, 100 took a significantly longer time to run) but keeping the remaining parameters the same, the new architecture significantly improved the rate at which the nonlinear logic gates are solved. Specifically, there are now 31 cell configurations which are able to solve all logical gates, including both XOR and XNOR, compared to the zero cells in the previous control iteration of the algorithm. Individually, there are now 32 XOR, and 42 XNOR gates that are solved.

Table 6a.
Overall successes of all cell configurations with 4 sub cell and control nodes.

	Success	Partial	Failure
OVERALL	31	197	60

Table 6b.
Comparison of- successes of each logic gate in all cell configurations in both the previous control node (3rd column) and the new split control (2nd column) iterations.

Logic gate	Saturation cell (%)	4 cell control (%)
OR	225(78.9)	399(69.2)
NOR	203(71.0)	397(68.9)
AND	100(35.1)	178(31.0)
NAND	103(36.0)	179(31.1)
XOR	32(11.2)	2(0.3)
XNOR	42(14.7)	24(4.2)

While the saturation cells have significantly improved the rate of which XOR and XNOR can be solved, the solving rate is still low compared to the linear logical gates (10.7% of all tested cell configurations solved both XOR and XNOR). Nevertheless, this illustrates one of the Boron cells fundamental problems. Chen et al.'s Boron cells are not guaranteed to solve all logic gates, as the randomness involved in synthesizing a cell could entail that not every synthesized cell has the connection patterns that enables it to solve all logic gates. Moreover, even when the nonlinear logical gates are solved they would produce significantly lower output voltages (as illustrated in figure 2c). Hence, by changing the overall architecture of the boron cell configuration, such as by introducing sub cells and saturation cells as illustrated in figure 11, the effects of the randomness can be somewhat controlled and reduced to increase the chances of

creating a functional architecture with Boron cells. For example, the use of saturation cells could solve the problem of highly reduced output from XOR as seen in figure 2c.

With the control-based boron cell simulation now capable of solving all Boolean logic gates in a semi-reliable manner, the architecture can be further tested on a digit recognition test.

6 Digit recognition

Now that all of the Boolean logic gates have been solved, the final test is to see whether an architecture or network of (simulated) boron (sub) cells as reservoir networks can do a simple digit recognition tasks, similar to what is described in Chen et al. (2020). To mimic the filters used in Chen et al. (2020), two Boron cells (reservoirs), each consisting of two sub cells (reservoirs) for two input nodes (totaling four input nodes) are initialized, as illustrated in figure 13. This forms the basis for a two by two input space, from which the overall system (network), that is each filter, can receive input.

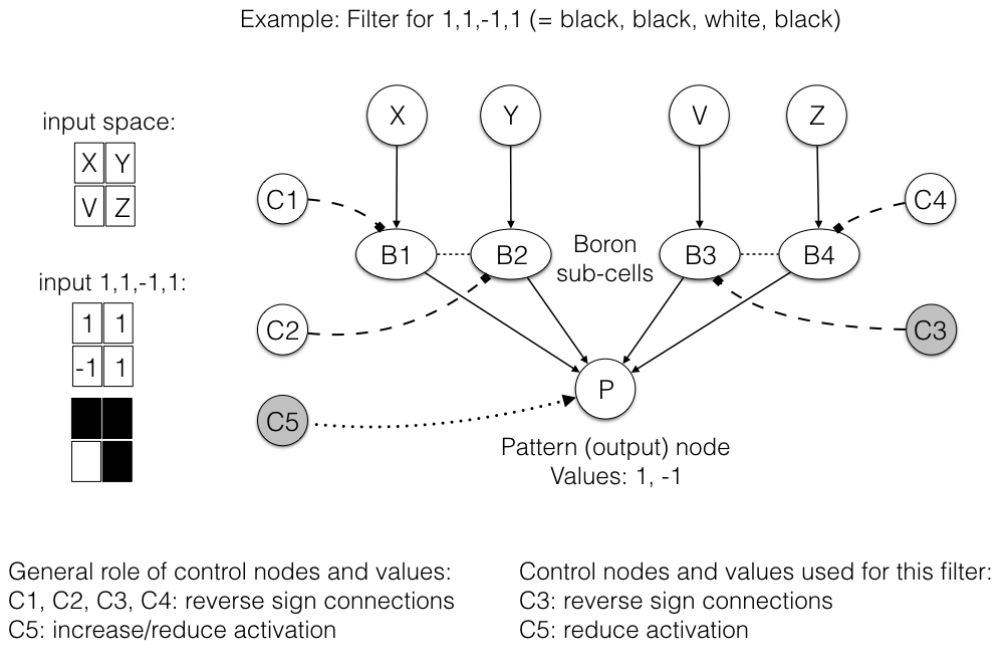


Figure 13. Boron cell schematic for number recognition using control nodes. There are now four individual inputs, each of them inputting to a boron sub cell. Since only AND gates are needed for solving very basic visual pattern recognition problems, only two types of control are needed, as illustrated for this filter. The input space X, Y and Z are the input nodes of the boron-reservoir, taking an input of 1, for black blocks and -1 for white blocks.

Parts of the controls as well as the saturation are removed in order to save algorithm run time, as the only logic gates required to be solved in this simple visual pattern recognition are AND gates. Using the two by two input space, 16 filters can be created to use for visual discrimination of 16 different patterns, as illustrated in figure 14.

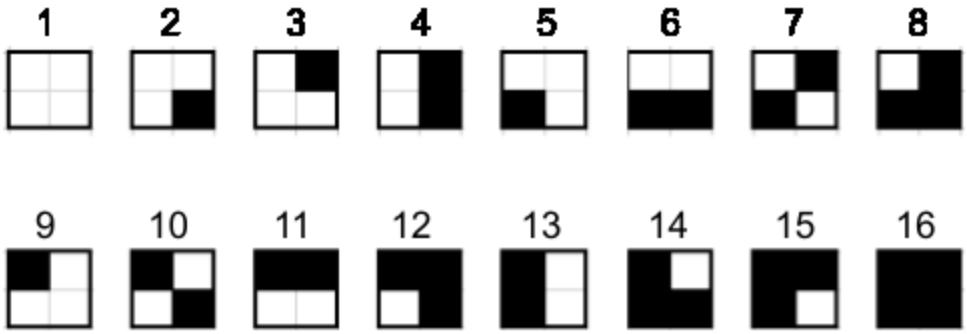


Figure 14. 2x2 filters simulated in excel. With 4 input blocks, there are a total of 16 different patterns that can be created, each of them representing a filter for visual pattern recognition. Combined, they can create digit forms on a 6x6 canvas in the manner illustrated in figure 14.

Figure 13 illustrates how pattern 12 in figure 14 is classified by the simulated boron system. Each filter is an array of four blocks, representing the two by two input space, holding either a negative (-1) or a positive (1) value (white and black blocks respectively). In figure 13, the inputs X, Y and Z activate their corresponding boron sub-cells with positive activation (1). Input V activates its boron cell with negative activation (-1). This is reversed by the input from the control cell C3 that operates on this boron sub cell. In this way, all activations from the boron sub cells to the output cell P are positive. This will not be the case for another input pattern to this filter. For example, if X = -1 in a pattern it will activate the boron sub cell of X with negative activation (-1), which is not reversed by input from control cell C1 on this sub cell, because C1 control is not used for the X sub cell in this filter. So, effectively, a filter is characterized by the set of control inputs (C1, C2, C3 or C4) that operate on it. Each unique filter will have a different set of control inputs. By setting control C5, the required filter output can be selected by reducing all output values until only one is positive.

So, effectively the input to a filter is multiplied by the filter control values (1 for no C1, C2, C3 or C4 control input, -1 for C1, C2, C3 or C4 control input), and produces a positive activation if the control value matches the input value (1 * 1 = 1 in the case of input 1 and filter 1; -1 * -1 = 1 in the case of input -1 and filter -1). If the input does not match the filter (1 * -1 = -

1; -1 * 1 = -1) then the resulting output will be negative activation. By summing the outputs of all four blocks (or sub cells), each filter produces a unique output based on the same input. The matching filter should produce an output that is much larger than the other filters outputs, thus indicating to the algorithm which pattern is the input.

To simulate the use of the filters illustrated in figures 13 and 14, a test input canvas was created in excel, where six by six blocks of cells are filled in with black and white to create patterns in the overall canvas (figure 14 right) corresponding to the numbers 1 to 9. In Chen et al. (2020), black is mapped onto the input voltage 0.5, whereas white is mapped onto -0.5 input voltage. In the current algorithm, black is mapped to the activation value 1, whereas white is mapped onto to the activation value -1. The six by six number canvas is divided into two by two blocks, (figure 15, middle). Each of these two by two blocks serves as the input space of the boron filters outlined in figures 12 and 13. The filters are then simulated (one by one) and for each two by two block, with each 2x2 block holding 16 different filter outputs per input. The network holds hard-coded information about the filters used in each number pattern (figure 14 right) and, given the input, will sum the activation of all filters used in a number pattern for each number. The resulting nine number activations are then reduced (by C5 in figure 13) until only one activation sum remains positive. A number pattern in the six by six canvas is classified when only one number activation sum is positive. Figure 15 illustrates the two by two patterns that need to be classified correctly to classify the pattern for the number 1 in the six by six canvas.

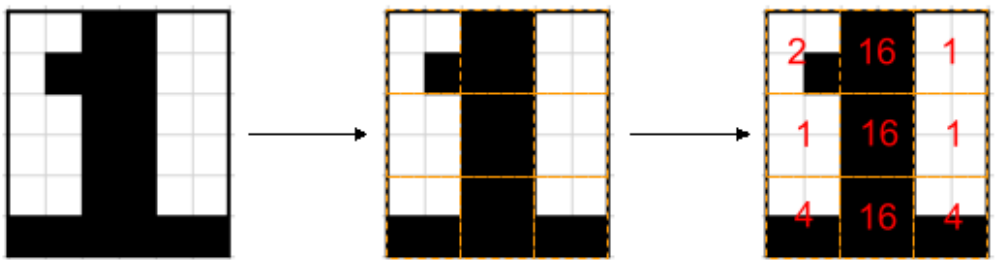


Figure 15. Illustration of the number recognition system in a six by six canvas using 2x2 filters for each two by two block in the canvas (left). Each 2x2 block (middle) in the 6x6 reading holds the activation values for all 16 filters. The algorithm has hard coded information about the filters (left) used to create all nine number forms, and sums the filter outputs of all filter outputs for each number, creating nine number activation sums. All number activations are then reduced, and a number pattern is classified when only one of the nine number activations are positive.

Because the negative input instead of zero input is mapped to white (empty) blocks, as illustrated in figure 2, AND gates are much easier to solve with boron-reservoir cells, as the distinction between (1,1) and (1,-1) will be much larger than that between (1,1) and (1,0). The results of our number classification algorithm, presented in table 19, show a high success rate with this approach. Successes in the present algorithm are cell configurations that were capable of recognizing all number patterns (0-9). For the algorithm to identify a number pattern, only one of the nine number pattern activation values can be positive. Partials in this case means cell configurations that were capable of solving some, but not all of the input number patterns, whereas failures signify that none of the number patterns were solvable for the cell configuration. The digit recognition tests were done using cell sizes 10, 20, 50 and 100; internal sparsity values 0.8, 0.85, 0.88, 0.92, 0.95, 0.98; external sparsity values 0.2, 0.5, 0.85, 0.9, 0.95, 1; and connection weight values 0.2, 0.5, 0.8 and random (0-1).

Table 7a.

Overall successes of all cell configurations with sub cell architecture

	Success	**Partial**	**Failure**
Overall	283	99	194

Table 7b.

Example output of a partial success boron-reservoir cell (cell size: 50; internal sparsity: 0.88; external sparsity: 0.5; connection weight: 0.8). The expected highest output is in bold-face.

Expected Number	**Recognized Number**	**Learning Runs**	**Output per number filter: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**
0	None	99	[**59.4**, -187.0, -81.4, -11.0, -55.8, -63.8, -28.6, -132.6, 8.2, -12.6]
1	1	53	[-141.0, **105.4**, -0.20, -70.6, -131.4, -69.0, -88.2, -19.4, -89.8, -69.0]
2	2	88	[-70.4, -35.2, **70.4**, 0.0, -115.2, -68.8, -52.8, -38.4, -35.2, -33.6]
3	None	99	[-11.0, -116.6, -11.0, **59.4**, -91.0, -44.6, -28.6, -62.2, 24.2, -28.6]
4	4	77	[-33.8, -155.4, -104.2, -69.0, **81.4**, -70.6, -86.6, -85.0, -85.0, -0.2]
5	5	88	[-52.8, -104.0, -68.8, -33.6, -81.6, **70.4**, 0.0, -52.8, -17.6, -35.2]
6	None	99	[-28.6, -134.2, -63.8, -28.6, -108.6, -11.0, **59.4**, -99.0, 6.6, -100.6]
7	7	50	[-83.6, -16.4, -0.4, -13.2, -58.0, -14.8, -50.0, **108.4**, -48.4, -50.0]
8	None	99	[8.2, -135.8, -46.2, 24.2, -107.0, -28.6, 6.6, -97.4, **59.4**, -63.8]
9	9	87	[-0.6, -103.0, -32.6, -16.6, -10.2, -34.2, -88.6, -87.0, -51.8, **71.4**]

There were varying degree of successes within the partial successes (2-9 digits unclassified). However, after looking into the outputs of all partial successes, it appears that all partially successful cells could have fully solved with more learning runs or a higher/lower learning rate. Table 7b displays the outputs of all number filters given an input number pattern

for a specific set of cell configurations. In this case, a system with cell size 50, internal sparsity 0.88, external sparsity 0.5 and a connection weight of 0.8 produces a partially successfully system that is incapable of solving for number forms 0, 3, 6 and 8. These number patterns weren't solved because there are more than one positive values in the array of nine number pattern activation sums (table 7b, column 4). Nevertheless, looking at the outputs, the expected number appears to always have the highest activation. This means that with more learning runs or rate, the algorithm is still capable of recognizing the correct number pattern from the number filter outputs.

Table 7c.

Example output of a failure boron-reservoir cell system (cell size: 100; internal sparsity: 0.8; external sparsity: 1; connection weight: 0.8). The expected highest output is in bold-face

Expected Number	Recognized Number	Learning Runs	Output per number filter: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
0	None	99	[**556.2**, -423.0, 13.8, 353.0, 97.0, 130.6, 276.2, -168.6, 405.8, 281.0]
1	None	99	[-423.0, **556.2**, 119.4, -219.8, -392.6, -147.8, -200.6, -69.4, -272.6, -227.8]
2	None	99	[-26.2, 159.4, **516.2**, 177.0, -114.2, -42.2, -37.4, 151.4, 31.4, 111.4]
3	None	99	[353.0, -219.8, 217.0, **556.2**, -48.6, 183.4, 188.2, 34.6, 410.6, 196.2]
4	None	99	[97.0, -392.6, -154.2, -48.6, **556.2**, -120.6, -125.4, -141.4, -53.4, 170.6]
5	None	99	[130.6, -147.8, -2.2, 183.4, -120.6, **556.2**, 257.0, -37.4, 188.2, 127.4]
6	None	99	[276.2, -200.6, 2.6, 188.2, -125.4, 257.0, **556.2**, -183.0, 333.8, 1.0]
7	None	99	[-168.6, -69.4, 191.4, 34.6, -141.4, -37.4, -183.0, **556.2**, -111.0, -15.0]
8	None	99	[405.8, -272.6, 71.4, 410.6, -53.4, 188.2, 333.8, -111.0, **556.2**, 130.6]
9	None	99	[281.0, -227.8, 151.4, 196.2, 170.6, 127.4, 1.0, -15.0, 130.6, **556.2**]

Likewise, all failures (that have a connection from input to output) appear to be solve-able, and only lack more learning runs, or rate to reduce all number outputs till only one is above 0. This is evident in tables 7c, where the expected number filter always produces the highest output, but not (yet) as the only output above 0. This means that while the learning settings were inadequate for solving cells with certain parameters, all of the cells configurations appear to be capable of recognizing simple digit forms. This demonstrates the resistance to the inherent randomness of a reservoir that this architecture realizes for simple visual perceptual recognition problems.

Using the same architecture as the digit recognition system, the Serre et al. (2007) S1 visual system architecture can be simulated to create Gabor filters. In the next section, the digit recognition model is applied to a 3x3 Gabor filter recognition model, and further elaborated to create more complex form recognition capabilities using the concept of S2 cells from the Serre model.

7 Testing simple cell architecture using Gabor filter recognition

The Serre et al. (2017) model of pattern classification in the visual cortex was presented in section 2.1. In the model, the simple to complex cell hierarchy (based on Hubel and Wiesel, 1965) begins with retinotopic S1 simple cells in the primary visual cortex that directly respond to different orientations and scales of contoured bars (lines) in the retina's receptive field (see figure 16). In the Serre model, only four orientations are used for the S1 cells: horizontal, vertical, and two diagonals. The S1 simple cells are then pooled together in the complex layer C1, which results in the invariance towards translational position and scale (see section 2.1).

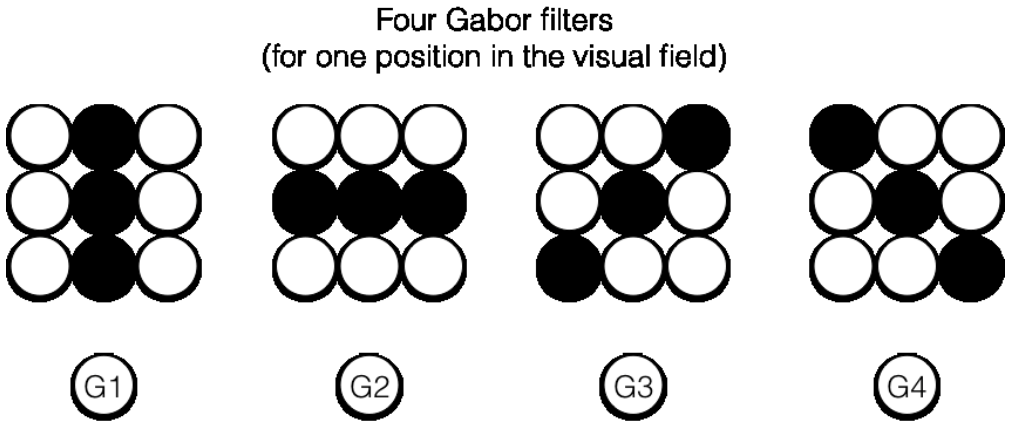


Figure 16 Four different Gabor filters, each with a differently oriented line segmentations. Each filter consists of a 3x3 block, where white represents empty (-1) input and black represents filled input (1). Like in Serre, the four Gabor filters are line segments arranged vertically (G1), horizontally (G2), and diagonally (G3 and G4).

Figure 16 illustrates the Gabor filters (G1, G2, G3 and G4) that will be simulated here using a boron-reservoir cell system. Each filter corresponds to a line orientation (given by the row of black input nodes) in a three by three canvas of input nodes that serves as input to the Gabor filters. The black versus white nodes correspond with the center-surround nature of Gabor filters in the Serre model and the visual cortex (see section 2.1). Here, the center is black and the surround is white. The reverse (white lines in a black surround) is also found in the visual cortex and in the Serre model, but is not simulated here.

Figure 16 displays the boron-reservoir system used to simulate the S1 cells. Like in the digit recognition model in section 5, there is one boron cell-reservoir for each input node. The total input to the system is given by the input nodes in the 3x3 input space. Each boron cell receives input from one input node and from one control node. For example, boron cell B-t receives input from input node T and from control node C1-t. The role of the control nodes is to reverse the activation in the boron-reservoir cell. Without the effect of control, a boron cell receives positive activation from the black input nodes in figure 16 (grey nodes in figure 17) and negative input activation from the white input nodes.

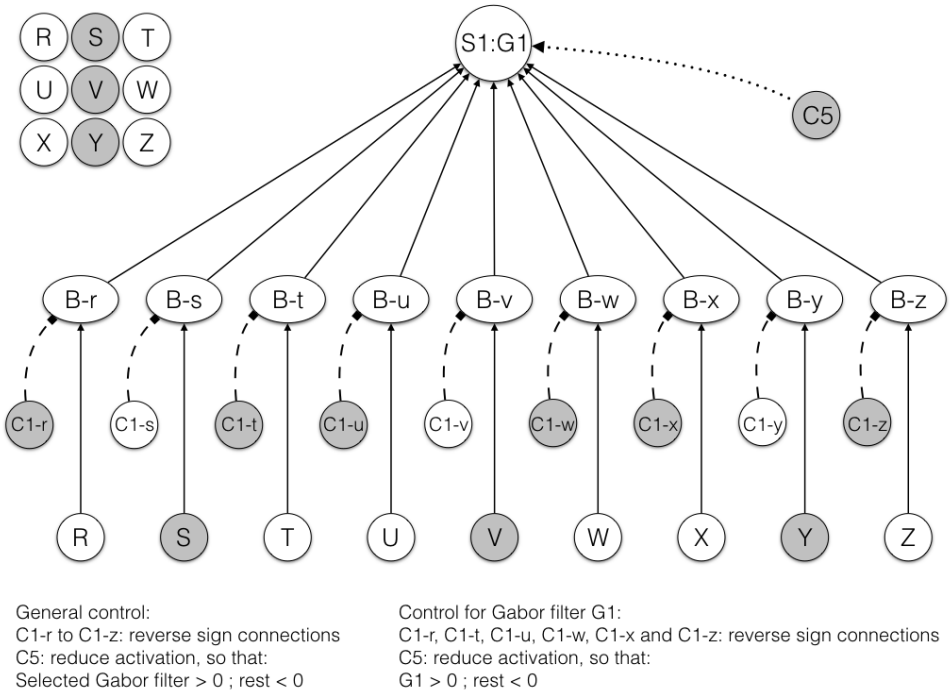


Figure 17. Input (top left) and S1 Gabor filter (here: G1) of the Serre et al. (2007) Gabor filter recognition system realized in the simulated boron-reservoir system. 9 input nodes (R to Z) make up the 3x3 input space. C1-r to C1-z are control nodes that reverse the sign of the activation in the boron cell produced by the input node. The activation of these control nodes is based on the desired filter properties, the activation values of the control nodes are: activation (value -1) if the desired input of their corresponding input node is white (input activation -1), no activation (value 1) if the desired input is black (illustrated with grey here, input activation 1). The values of C1-r to C1-z are pre-programmed into the system as prior knowledge of the Gabor filter control values. Inputs are then fed into B-r to B-z boron-reservoir cells. The activation of these cells is summed into the output node (S1) that represents a Gabor filter. The summed output can be reduced by control node C5 until only one filter output is above 0, which is then chosen as the recognized filter. Here, the desired filter output is Gabor filter G1, represented by the output node S1:G1. The active input nodes and the active control nodes are grey.

As with the classification of the numerical patterns in section 5, an inactive control node has activation value 1, which is multiplied with the activation in the corresponding boron cell. This has no effect on the activation of that boron cell. When a control node is active (activation value -1) it reverses the sign of the activation in the boron cell it is connected to. Therefore, by selecting the activation of the nodes C1-r to C1-z, a filter is created that selects an input pattern. The activation of all boron-reservoir cells is fed into the output cells. There are four output cells in the system. Each one is an S1 cell that corresponds to one of the Gabor filters illustrated in figure 16. Figure 17 illustrates the S1 cell that corresponds to Gabor filer G1 in figure 16. By activating the control cell C5, the activation of each S1 cell can be reduced until one of them remains as the only activation above 0. The boron-reservoir system implements a Gabor filter successfully, if the selected S1 cell corresponds to the input pattern, as S1 in figure 17 corresponds to the input pattern G1 in figure 16.

For the purpose of this study, the assumption is made that if a Gabor filter can be simulated on one position of the visual field, it will also operate in the same way at other positions (without explicit modeling). Simulating multiple receptive fields would be far too time consuming to test at this stage of development. Furthermore, the focus of the Gabor filter implementations will be between S1 to S2 (skipping C1), where the simple S1 outputs are fed into S2 simple cells at the next layer, which combine S1 Gabor patterns into more complex patterns (C1 cells in the Serre model simply pool the responses of S1 cells with the same line orientation but with difference receptive fields).

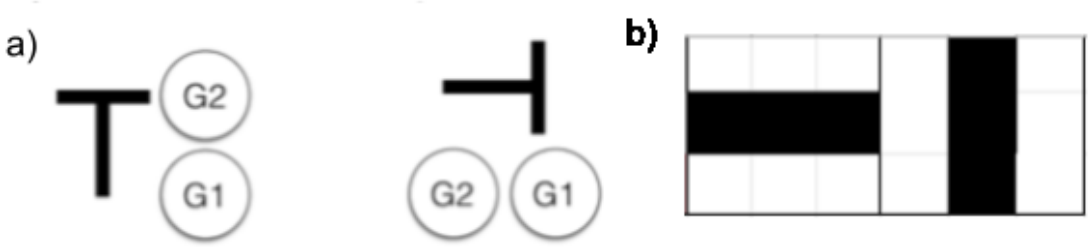


Figure 18. a) Illustration of complex patterns that can be classified by S2 cells using Gabor filter outputs from the S1 cells. The horizontal (upright) and the vertical (rotated) pattern 'T' can be created by combining G1 and G2 at adjacent positions in the visual field. b) Illustration of the implemented version (in excel) of the Gabor filters G2 and G1 in their respective positions for the vertical 'T' in (a). Filled black cells indicate an input of 1 while white cells are -1.

Figure 18a and b shows the complex patterns that S2 cells should classify. By using two adjacent visual field positions, Gabor filters recognized in those positions can be combined further to activate an S2 cell, which would classify more complex patterns like a horizontal or vertical T. The connection from S1 to S2 cells in the visual hierarchy raises more interesting questions for the Boron cell approach, namely, whether outputs created in the S1 layer can be used in a further Boron cell layer, or whether the Boron cell architecture is even capable, or useful, for solving multilayered visual perceptual problems. The architecture for S1 to S2 cell in figure 18b is illustrated in figure 19.

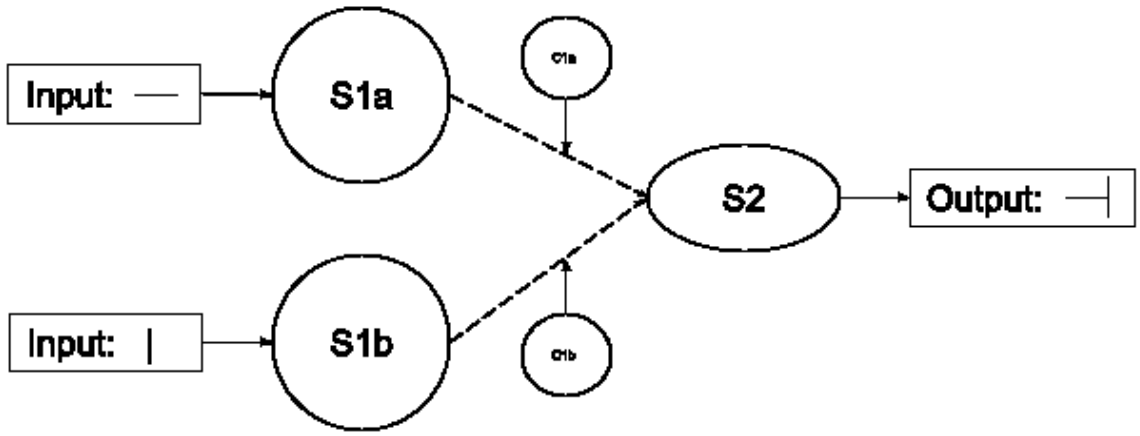


Figure 19. Illustration of the S1 to S2 architecture in figure 17b from Serre et al. (2007) realized in a simulated boron architecture. S2 receives input from S1 cells, and is activated if the adjacent S1a and S1b outputs are the desired (controlled by C1a and C1b) horizontal (G2), and vertical (G1) line filter outputs. If the desired inputs are received at S2, it will recognize the new complex pattern

S1a and S1b represent two adjacent positions in the visual field, each preprogrammed to recognize a Gabor filter. S1a recognizes a horizontal line (Gabor filter G2 in figure 15), while S1b recognizes a vertical line (Gabor filter G1 in figure 15). The input from the cells S1a and S1b is combined with the input from the control nodes C1a (for S1a) and C1b (for C1b), which operate in the same way as the control nodes in figure 17. As in figure 17, they are (for now) hard coded to represent a desired pattern. If the outputs for both S1a and S1b are active and in line with the desired pattern set by the control nodes, they will in turn activate S2, which classifies the vertical T shape complex pattern.

Using these ideas and structure, a network is initialized based on functioning values from prior experiments: input values (1, -1), a boron-reservoir cell size of 50, internal sparsity of 0.80, external sparsity of 1 (no connections between boron-reservoir cells) and a connection strength

of 0.2. The input canvas used is likewise represented excel, based on the 3x3 input space illustrated in figure 16. The overall input canvas now consists of two adjacent 3x3 blocks that are used to test the S2 cell outputs (figure 19).

Table 8a, b, c, d shows a sample of S1 and S2 filteroutputs from varying inputs in either position of the canvas.

Table 8a.
Output of all S1 and S2 filters with inputs: G2 G1.

	Input	
Output	**G2**	**G1**
S1:G1	-0,713	12,243
S1:G2	12,243	-0,713
S1:G3	-3,485	-1,272
S1:G4	-3,663	-1,449
S2:(G2G1)		3,6753

Table 8b.
Output of all S1 and S2 filters with inputs: G3 G2.

	Input	
Output	**G3**	**G2**
S1:G1	-0,472	-0,713
S1:G2	-2,6854	12,243
S1:G3	13,043	-3,485
S1:G4	-3,422	-3,663
S2:(G2G1)		-3,6585

Table 8c.
Output of all S1 and S2 filters with inputs: G4 G1.

	Input	
Output	**G4**	**G1**
S1:G1	-0,649	12,243
S1:G2	-2,863	-0,713
S1:G3	-3,422	-1,272
S1:G4	13.043	-1,449
S2:(G2G1)		0,209

Table 8d.
Output of all S1 and S2 filters with inputs: G2 G4.

	Input	
Output	**G2**	**G4**
S1:G1	-0,713	-0,649
S1:G2	12,243	-2,863
S1:G3	-3,485	-3,422
S1:G4	-3,663	13,043
S2:(G2G1)		-0,1922

Looking at the results, the S1 cells S1:G1, S1:G2, S1:G3 and S1:G4 were able to correctly classify their desired inputs. From the outputs of the S2 cell, selective for the combination G2G1, S2:(G2G1) illustrated in figures 18b and 19, it is clear that the correct input (table 8a) gives the highest activation in comparison to mismatched inputs. For the inputs that are only half mismatched (table 8c, table 8d) the outputs of S2:(G2G1) are still significantly lower. There are some minor differences between the mismatch output activation of S2:(G2G1) in the tables 8c and 8d that can be attributed to the randomness in the S1 cells. In particular, while the

inputs from both G4 G1 (table 8c) and G2 G4 (table 8d) differ from G2 G1 by one position (G4), the first one produces a positive (0,209) activation in S2:(G2G1) while the other a negative (-0,1922) activation in S2:(G2G1). These differences can be attributed to the randomness within the 9 boron-reservoir cells that activate the S1 cells. For example, this causes a difference in output for mismatching G2 (-2,863) and mismatching G1 (-0,649) when the input is G4 (table 8c). However, such small differences make little difference in further layers. Most importantly, the mismatched positive output of S2:(G2G1) in table 8c is very small compared to the correct positive output of S2:(G2G1) in table 8a. Higher layers of a classification system would be able to distinguish between these correct and mismatched S2:(G2G1) outputs.

8 General Discussion

Overall, the final implementations were capable of solving all the problems that were initially set as benchmarks. These benchmarks included solving all Boolean logic gates, performing basic digit recognition task in Chen et al. (2020), and finally, simulating (partially) the S1 to S2 layer of the Serre model in a boron-reservoir network. First, the results of the simulations will be discussed.

The standard perceptron solves for all linear logic gates, as no randomness exists within the perceptron nodes. The introduction of a reservoir into a perceptron system showed that the random complexity inherent to the reservoir enables even a single layer perceptron to solve nonlinear logic gates under certain reservoir settings. This could illustrate why these logic gates could be solved with the actual boron system of Chen et al. (2020). However, the random complexity came at a cost, as it highly reduces the networks capability to solve the linear logic gates. Again, this might occur in the actual boron system of Chen et al. (2020) as well, which could indicate why a long process of evolutionally computing is needed to find the correct control settings. Much of this can be attributed to reservoirs that were generated without connections from input to output. In these cases, the output simply received no activation regardless of the input. However, there were also a number of cell settings that over expanded, positively or negatively, activations to the point where information gradient was lost, for example, when inputs (1,1) and (1,0)/(0,1) result in outputs that are indistinguishable. This occurs when connection weights are high (0.5+) and high amount of nodes (50+) and/or

connections (sparsity < 0.5). Due to the inability to differentiate outputs given by inputs (1,1) and (1,0)/(0,1), AND, NAND, XOR and XNOR gates are unable to be solved. On the other hand, OR and NOR are still capable of being solved, since the only discrimination made is between activation, and no activation.

The results from the implementation of control nodes showed that the use of control nodes, over perceptron learning at the output weights, significantly improved the solving rate of the linear logic gates AND and NAND (table 3b). This is partially due to the splitting of the cell (to accommodate for control structures), as doing so highly increases the chances that at least two distinct input signals and dimensionality within the reservoir. These findings confirm that the assumed effects of the control nodes are just as effective as perceptron learning for tuning the reservoir outputs. However, at this point the architecture still had difficulties solving either of the nonlinear logic gates.

In the final saturation iteration of the program, the percentage of cells capable of solving all logic gates are 10.7%, with a solving rate of the nonlinear XOR and XNOR logic gates at 11% and 14% respectively. In comparison to previous, unsaturated versions, the saturation network drastically improved the rate at which the nonlinear gates are solved, and slightly improved the rate at which linear logic gates can be solved. These results show the potential of a saturation effect in the boron cell approach to reduce effects of the inherent randomness in boron cells, increase nonlinearity of the solution space and ultimately granting more randomly generated cells the ability to solve all logic gates. Moreover, unlike the highly varied activation levels of logic gate outputs from the original boron cell from Chen et al. (2020), as illustrated in figure 2c, the effects of the saturation also ‘standardizes’ all outputs to a uniform level of activation. This would make it easier for (potential) higher levels in the system to interpret the results obtained by the boron cells. That is, without the saturation effects, an up-scaled multilayer interconnected network of cells would have a difficult time interpreting such varied outputs. Conclusively, while the original boron cell itself may not have the functions of the saturation cell, such an effect can aid in the reduction of randomness effects that lead to cells that are incapable of solving nonlinear logic gates.

Despite architectural changes, the reliability of a simulated boron-reservoir cells still remains fairly low for logic gates outside of OR and NOR. But, interestingly, the results obtained

here for the linear and non-linear logic gates (partly) reflect the differences in output current observed for these gates with the actual Chen et al. boron dopant cell. For example, as illustrated in figure 2c, the output current for NOR is twice that of AND and ten times that of XOR. This could result from the inherent difficulty of obtaining these results due to the random reservoir nature of boron cells, as observed with the simulations presented here.

For the purpose of a simple visual pattern recognition, as given by digit recognition and the S1 and S2 units in the Serre model, the architecture was further adapted to be more suited for solving AND gates. The digit recognition architecture with four distinct input signals was capable of recognizing all numbers (0 – 9) using almost any cell setting (many required a different learning run and rate setting, but the cell architecture produces a range of outputs that are solve-able regardless of the cell configuration). By artificially splitting the input signals, the complexity and randomness of the cell is potentially reduced, but in turn, is more capable of reliably solving simple problems due to the guaranteed number of dimensions in each randomly generated cell. Adopting the same concepts, the Gabor filter S1 – S2 architecture was also successfully simulated. In both cases, the boron cell simply solves multiple AND gates, and as such, the success in the digit recognition algorithm is easily replicated in the Gabor filter algorithm. Moreover, adopting this architecture for visual perception highly reduced the randomness inherent to boron cells. While the randomness is still present, it is small enough that almost any generated cell is capable of solving visual recognition problems. Conclusively, the Gabor filter simulation shows that the simulated boron system is capable of supporting a S1 to S2 structure as described by Serre et al, (2007).

There are a number of limitations regarding the conceptual implementations of the simulated boron-reservoir network; namely, the amount of nodes within the simulated reservoir compared to the actual reservoir, the control node effects, and the lack of quantum properties in the simulation that would better mimic electron hopping effects. A potential explanation to the low solving rate of nonlinear logic gates is that the simulation does not capture the complexity that the actual boron cell system provides, considering that the actual boron cell consists of millions, perhaps hundreds of millions of boron dopant atoms, whereas our simulation consists of a maximum 100 simulated boron nodes. In this sense, it could be that not all 100 node boron cell simulations contained multi-layer like systems that are complex enough to create the

nonlinear projections needed for solving the nonlinear logic gates. With millions, if not hundreds of millions of boron atom nodes, the chances of a network capable of creating nonlinear projections is much higher. Moreover, with a significantly larger amount of boron atoms (nodes), the activation would vary a lot less, as the network is less dependent on each node, leading to smaller output differences between nodes that are interconnected. For example, in a smaller network of 100 nodes, there is a higher chance of bottleneck nodes that are required to be connected in order for activation to reach the output, or flow through a large patch of nodes. With hundreds of millions of nodes, the network becomes more resistant to bottleneck nodes, as the chances of another node providing similar connections would be higher (unless connectivity within the cell is low). To that end, an actual boron network, with hundreds of millions of nodes, designed for the simple to complex cell hierarchy could work without the need for multiples of exactly the same cells or a different filtering scheme. Simulation of networks like the boron cell could be done with much higher computing power, particularly, making use of parallel CMOS computing systems like ASICS and GPUs to increase the efficiency of calculating parallel matrix/vector additions and multiplications. As such, higher amounts of nodes could be simulated for a more accurate representation of the complexity of the boron atom network.

Another limitation to the study concerns the speculated effects of the control nodes (ultimately, the structure within the reservoir that accommodates the controls as well), and how they affect boron atom patches (or 'sub cells'). The effects of the control nodes are speculated to be fairly simple, adding or subtracting activations from nodes, and creating inhibiting behaviour by reversing input connection weights from positive to negative. There is a potential discrepancy between the effects of the control nodes within the actual boron cell, and the simulation, that is difficult to determine without additional information from Chen et al. (2020). One effect that partially mischaracterizes the boron cell is the way in which the controls affect boron atoms (nodes) within the cell. In the current simulation, to accommodate for the positional effects of the control and input electrodes, the cell was split into two equal sub cells. Considering the randomness of boron atom positions, two equally sized boron patches formed around the effective ranges of their respective control nodes is highly unlikely. Instead, the actual boron cell may consist of multiple, unequally sized patches of boron atoms. To increase the accuracy of the simulation, control nodes, namely C3 and C4 (adding and subtracting activation directly from, and into, cell nodes) could be simulated using additional connection matrices that randomly

connect cell nodes to the control electrodes. Considering the assumption that control electrodes are only capable of affecting cell nodes within their vicinity, locational proximity can be determined using the connection strengths of each node (the stronger, the closer), and the spreading of activation could be loosely modeled with spatial position in mind. This can be further aided by creating and using visualizations of the node network within the reservoir. Lastly, further interpretations of the effects, or architecture of the control node could also yield interesting results. Instead of directly influencing the activation of all nodes in the reservoir, the controls could be to influence the connection weights between boron nodes, or boron patches (sub cells). As the present thesis has demonstrated, exploring divergent architectural paradigms could further increase the reliability and usefulness of boron-reservoir cells in practical scenarios. Future studies can test the boron-reservoir cell under different RC properties (mentioned in section 4.3) and architecture. For example, varying reservoir sizes, increasing dimensionality by using more sub cells (boron patches) and reinterpreting control node effects.

Finally, as a neuromorphic computing device that takes advantage of the quantum properties of interconnected dopant boron atoms, the present simulation lacks a quantum understanding and representation of the actual cell. Specifically, concerning the hopping regime that transports charge between each boron atom. While these concepts may be beyond our current understanding of the system, library packages like Qiskit, Cirq or PyQu are capable of simulating quantum behaviours, and could be used to create more accurate simulations. Nevertheless, the fundamental ideas of the architecture are captured in our simulation, using control nodes as a method of adjusting input to output relationship within a reservoir rather than through synapse (updating connection weights). This paradigm of "learning", implicit to the hardware, is valuable for neural network problems of all complexities, as emulating synapse in physical hardware is incredibly difficult, and has been a long standing problem within the realm of neuromorphic hardware research. The benefit of this can be seen by looking at the digital simulation of the Serre et al. model depicted in figure 1. To simulate this model, representations are needed of numeral values such as activation values of nodes and connection weights. In turn, these need to be multiplied and used as inputs in activation functions that provide the (updated) activation values of each node in the system. All of these aspects are difficult to implement in neuromorphic hardware. But, as the Serre simulation illustrates, boron cells used as reservoirs could circumvent the problems of representing and calculating activation values and activation

functions, and control nodes could circumvent the problem of representing and controlling connection weights.

9 Conclusion

The results from the simulations suggest that the randomness inherent in constructing a boron-reservoir cell can be reduced and controlled through different architectural paradigms of the boron system, for example, by splitting the cell into sub cells and adding saturation effects to the output. To that end, the small energy foot print and the efficient computational power of boron dopant architecture could be used for systems that require massive amounts of simple parallel computations; like the lower level, retinotopic areas of the human visual cortex (V1). Consisting of millions of line segment filters, overlapping in all positions of our retinal field, the enormous amounts of parallel computation involved in V1 can be done using the inherent parallelism and energy efficiency in the boron cell. Furthermore, the nanoscale of the boron architecture allows the system to be scaled up without creating physically large systems of hardware.

Conclusively, while the majority of current machine vision research focus on increasing accuracy and performance without consideration of computational restraints, the boron dopant cell architecture provides a paradigm that has the potential to increase computational capacity while reducing energy costs and physical volume. Architectural changes may still be in order as the inherent randomness in creating a boron cell is highly detrimental to its practicality, and combatting the randomness through architectural changes has been proven to be effective in the present thesis. As conventional silicon based CMOS systems approach their inevitable limit, the potential advantages that neuromorphic hardware can offer for computationally intensive tasks like visual perception becomes more and more attractive. Future studies on the topic should continue to focus and develop the reliability and scalability of the boron dopant cell from an architectural paradigm.

References

Alonso, J.-M. (2002). Book Review: Neural Connections and Receptive Field Properties in the Primary Visual Cortex. *The Neuroscientist, 8*(5), 443-456. doi:https://doi.org/10.1177/107385802236967

Brown, R. E. (2020). Donald O. Hebb and the Organization of Behavior: 17 years in the writing. *Molecular Brain, 13*(1), 55. doi:10.1186/s13041-020-00567-8

Chen, T., van Gelder, J., van de Ven, B., Amitonov, S. V., de Wilde, B., Ruiz, H.-C. E., . . . van der Wiel, W. G. (2020). Classification with a disordered dopant-atom network in silicon. *Nature, 577*, 341-345. doi:https://doi.org/10.1038/s41586-019-1901-0

Da, L., Chen, X., Becchi, M., & Zong, Z. (2016). *Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs*. Paper presented at the 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), Atlanta, GA, USA.

Dsouza, J. (2020). What is a GPU and do you need one in Deep Learning? Retrieved from https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d

Fukushima, K. (1980). Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position *Biological Cybernetics, 36*, 192-202. doi:https://doi.org/10.1007/BF00344251

García-Martína, E., Faviola Rodrigues, C., Riley, G., & Grahna, H. (2019). Estimation of energy consumption in machine learning. *Journal of Parallel and Dsitributed Computing, 134*, 75-88. doi:https://doi.org/10.1016/j.jpdc.2019.07.007

Giannaris, E. L., & Rosene, L. D. (2012). A stereological study of the numbers of neurons and glia in the primary visual cortex across the lifespan of male and female rhesus monkeys. *The Journal of Comparative Neurology, 520*(15), 3492-3508. doi:https://doi.org/10.1002/cne.23101

Goodfellow, I., Yoshua, B., Courville, Aaron. (2016). Convolution NetworksDeep Learning (pp. 326-366): MIT Press. Retrieved from https://www.deeplearningbook.org/contents/convnets.html.

Gupta, S. (2019, 19.06.2020). Neuromorphic Hardware: Trying to Put Brain Into Chips. *Computational Neuroscience.* Retrieved from https://towardsdatascience.com/neuromorphic-hardware-trying-to-put-brain-into-chips-222132f7e4de

Han, Y., Roig, G., & Poggio, T. (2020). Scale and translation-invariance for novel objects in human vision. *Scientific Reports, 10*(1411). doi:https://doi.org/10.1038/s41598-019-57261-6

Hinaut, X., Dominey, P. F. (2013). Real-Time Parallel Processing of Grammatical Structure in the Fronto-Striatal System: A Recurrent Network Simulation Study Using Reservoir Computing. *PLoS ONE, 8*(2), 1-18. doi:10.1371/journal.pone.0052946

Huang, J., Rathod, V., & Sun, C. (2017). *Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors*. Paper presented at the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology, 160*(1), 106-154. doi:10.1113/jphysiol.1962.sp006837

Jaeger, H. (2008). *A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach.* International University Bremen. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.378.4095&rep=rep1&type=pdf

LeCun, Y. (2019, 17-21 Feb. 2019). *1.1 Deep Learning Hardware: Past, Present, and Future.* Paper presented at the 2019 IEEE International Solid- State Circuits Conference - (ISSCC).

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. *IEEE, 86*(11), 2278 – 2324. doi: 10.1109/5.726791

Leuba, G., & Kraftsik, R. (1994). Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age. *Anatomy and Embryology, 190*, 351-366. doi:10.1007/BF00187293

Linsley, D., Eberhardt, S., Sharma, T., Gupta, P., & Serre, T. (2018). *What are the visual features underlying human versus machine vision?* Paper presented at the IEEE.

Misra, J., & Saha, I. (2010). Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing, 74*(1), 239-255. doi:https://doi.org/10.1016/j.neucom.2010.03.021

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing, 2*(5), 183-197. doi:https://doi.org/10.1016/0925-2312(91)90023-5

Nurvitadhi, E., Sheffield, D., Sim, J., Mishra, A., Venkatesh, G., Marr, D. (2016, December 7-9). *Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC*

[Conference paper]. 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China. 10.1109/FPT.2016.7929192

Peters, T. (2020, October 2). *Floating point arithmetic: Issues and Limitations.* Python. https://docs.python.org/3/tutorial/floatingpoint.html

Preuveneers, D., Tsingenopoulos, I., & Joosen, W. (2020). Resource Usage and Performance Trade-offs for Machine Learning Models in Smart Environments. *Sensors (Basel, Switzerland), 20*(4), 1176. doi:10.3390/s20041176

Reddy, N. (2019). *A Survey on Specialised Hardware for Machine Learning*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408. doi:https://doi.org/10.1037/h0042519

Sato, K. (2018). What makes TPUs fine-tuned for deep learning? *AI & MACHINE LEARNING.* Retrieved from https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning

Schuller, I. K., Stevens, R., Pino, R., & Pechan, M. (2015). *Neuromorphic Computing – From Materials Research to Systems Architecture Roundtable*. Retrieved from United States: https://www.osti.gov/biblio/1283147

https://www.osti.gov/servlets/purl/1283147

Serre, T., Olivia, A., & Poggio, T. (2007). A feedforward architecture accounts for rapid categorization. *PNAS, 104*(15), 6424-6429. doi:10.1073/pnas.0700622104

Sharma, S. S., Simone. (2020). ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology, 4*(12), 310-316.

Shi, W., Alawieh, M. B., Li, X., & Yu, H. (2017). Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey. *Integration, 59*, 148-156. doi:https://doi.org/10.1016/j.vlsi.2017.07.007

Tanaka, G., Yamane, T., Heroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks, 115*, 100-123. doi: https://doi.org/10.1016/j.neunet.2019.03.005

Tian, H., Wang, T., Liu, Y., Qiao, X., & Li, Y. (2020). Computer vision technology in agricultural automation —A review. *Information Processing in Agriculture, 7*(1), 1-19. doi:https://doi.org/10.1016/j.inpa.2019.09.006

Vergara, O., Cruz-Sánchez, V., Ochoa, H., Nandayapa, M., & Flores Abad, A. (2014). Automatic Product Quality Inspection Using Computer Vision Systems. In J. L. G. M. Alcaraz, Aide Aracely Maldonado; Robles, Guillermo Cortes (Ed.), *Lean Manufacturing in the Developing World* (pp. 135-156). Chihuahua, Mexico: Springer.

Yang, J., Wang, C., Jiang, B., Song, H., & Meng, Q. (2021). Visual Perception Enabled Industry Intelligence: State of the Art, Challenges and Prospects. *IEEE Transactions on Industrial Informatics, 17*(3), 2204-2219. doi:10.1109/TII.2020.2998818

Yann, L., Leon, B., Bengio, Y., & Haffner, P. (1998). Gradient based learning applied to document recognition. *IEEE Transactions on Industrial Informatics*, 2278 - 2324. doi:10.1109/5.726791

Zhao, Y., Deng, B., & Wang, Z. (2002, 7-7 Nov. 2002). *Analysis and study of perceptron to solve XOR problem.* Paper presented at the The 2nd International Workshop on Autonomous Decentralized System, 2002.

Appendix

A.1 Appendix 1

Table A.

Success rate of different learning run and learning rate combinations

Max learning Run	Learning Rate	Success	Partial	Failure	Success Rate %
100	0.1	38	232	210	7.92
100	0.01	16	254	210	3.33
100	0.001	0	270	210	0.00
1000	0.1	89	181	210	18.54
1000	0.01	78	192	210	17.71
1000	0.001	40	230	210	8.33

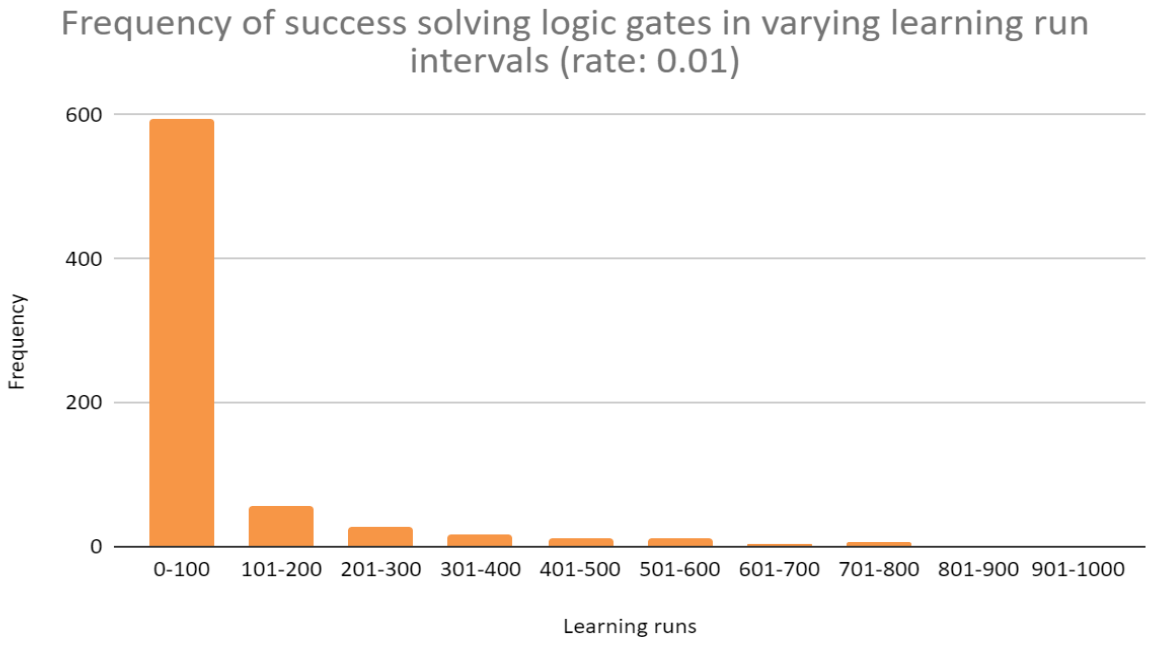


Figure A. Graph showing frequency of successful solving of logic gates in different learning run intervals (0-100, 101-200, 201-300..) using 1000 learning runs and 0.01 learning rate.

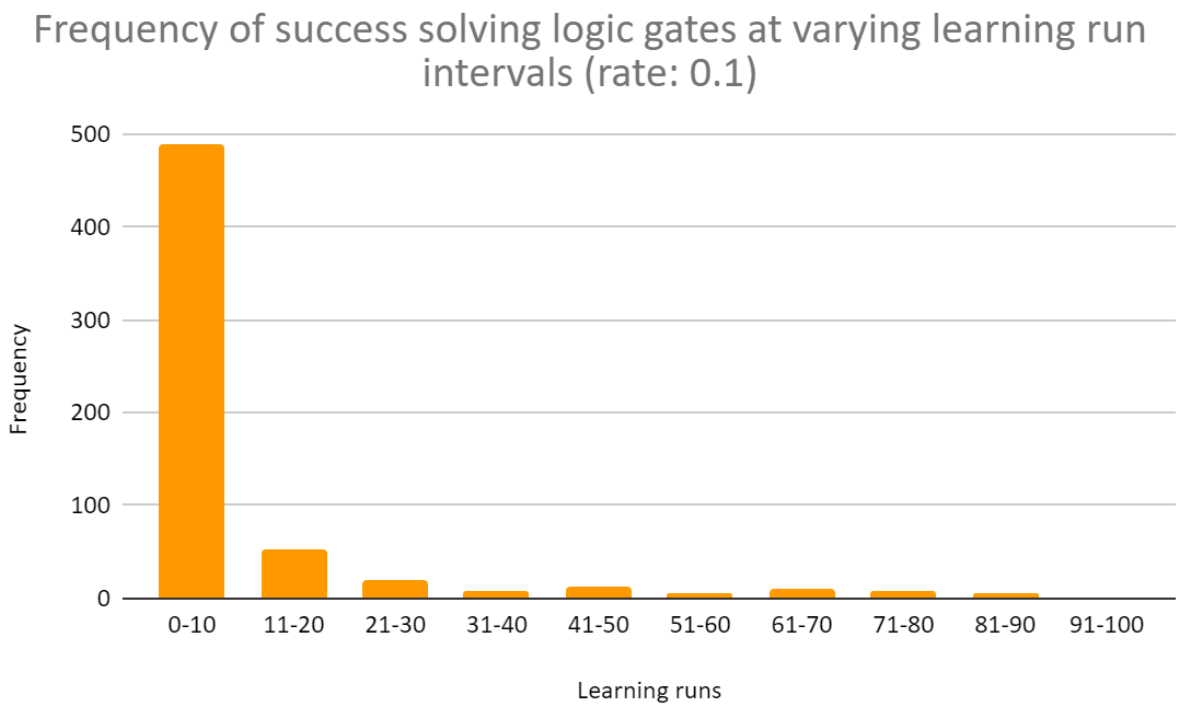


Figure B. Graph showing frequency of successful solving of logic gates in different learning run intervals (0-10, 11-20, 21-30..) using 100 learning runs and 0.1 learning rate.

A.2 Appendix 2

Table A.

Randomizer seeds

SEEDS	Success	Partial	Failure
[1, 2, 3]	5	53	38
[4, 5, 6]	5	41	50
[7, 8, 9]	5	49	42
[10, 11, 12]	3	54	39
[13, 14, 15]	2	54	40

Table B.

onnection strength

COSTR	Success	Partial	Failure
0.2	15	54	51
0.5	2	67	51
0.8	0	69	51
random	3	61	56

Table C.

Sparsity

SPARS	Success	Partial	Failure
0.8	5	67	8
0.85	6	50	24
0.88	3	48	29
0.92	6	40	34
0.95	0	31	49
0.98	0	15	65

Table D.

Cell Size

CSIZE	Success	Partial	Failure
10	1	11	108
20	3	50	67
50	11	80	29
100	5	110	5

Table E.

Cell size by Sparsity

CSIZE	SPARS	Success	Partial	Failure
10	**0.8**	8	4	8
10	**0.85**	0	0	20
10	**0.88**	0	0	20
10	**0.92**	0	0	20
10	**0.95**	0	0	20
10	**0.98**	0	0	20
20	**0.8**	10	10	0
20	**0.85**	8	8	4

CSIZE	SPARS	Success	Partial	Failure
50	**0.8**	4	16	0
50	**0.85**	5	15	0
50	**0.88**	5	15	0
50	**0.92**	6	14	0
50	**0.95**	6	5	9
50	**0.98**	0	0	20
100	**0.8**	0	20	0
100	**0.85**	1	19	0

20	**0.88**	5	6	9	**100**	**0.88**	2	18	0
20	**0.92**	3	3	14	**100**	**0.92**	5	15	0
20	**0.95**	0	0	20	**100**	**0.95**	5	15	0
20	**0.98**	0	0	20	**100**	**0.98**	5	9	6

Table F.

Cell size by connection strength

CSIZE	COSTR	Success	Partial	Failure	CSIZE	COSTR	Success	Partial	Failure
10	**0.2**	2	1	27	**50**	**0.2**	18	5	7
10	**0.5**	3	0	27	**50**	**0.5**	4	19	7
10	**0.8**	0	3	27	**50**	**0.8**	3	20	7
10	**random**	3	0	27	**50**	**random**	1	21	8
20	**0.2**	6	8	16	**100**	**0.2**	13	15	2
20	**0.5**	9	5	16	**100**	**0.5**	1	28	1
20	**0.8**	3	11	16	**100**	**0.8**	3	26	1
20	**random**	8	3	19	**100**	**random**	1	27	2

Table G.

Connection Strength by Sparsity

COSTR	SPARS	Success	Partial	Failure	COSTR	SPARS	Success	Partial	Failure
0.2	**0.8**	11	7	2	**0.8**	**0.8**	1	17	2
0.2	**0.85**	7	7	6	**0.8**	**0.85**	0	14	6
0.2	**0.88**	7	6	7	**0.8**	**0.88**	0	13	7
0.2	**0.92**	8	4	8	**0.8**	**0.92**	3	9	8
0.2	**0.95**	6	2	12	**0.8**	**0.95**	2	6	12
0.2	**0.98**	0	3	17	**0.8**	**0.98**	3	1	16
0.5	**0.8**	4	14	2	**random**	**0.8**	6	12	2
0.5	**0.85**	4	10	6	**random**	**0.85**	3	11	6
0.5	**0.88**	3	10	7	**random**	**0.88**	2	10	8
0.5	**0.92**	3	9	8	**random**	**0.92**	0	10	10
0.5	**0.95**	2	6	12	**random**	**0.95**	1	6	13
0.5	**0.98**	1	3	16	**random**	**0.98**	1	2	17

A.3 Appendix 3

Table A: Results per seed combination used to generate the subcells. The results were the same for subcell A and B, and every combination with other cells (e.g. [1,2,3,4] for cell A was tested with all seed combinations in cell B).

SEEDS	Success	Partial	Failure
[1. 2. 3. 4]	0	1622	682
[5. 6. 7. 8]	0	1648	656
[9. 10. 11. 12]	0	1660	644
[13. 14. 15. 16]	0	1618	686

Table B. Results filtered by cell size. Left: Success, partial, failure rate of all logic gates. Right: Success, partial, failure rate of linear logic gates.

CELLSIZE	Success	Partial	Failure
10	0	231	2073
20	0	1394	910
50	0	2001	303
100	0	2248	56

CELLSIZE	Success	Partial	Failure
10	72	159	2073
20	229	1165	910
50	150	1857	297
100	80	2168	56

Table C. Results filtered by external sparsity. Left: Success, partial, failure rate of all logic gates. Right: Success, partial, failure rate of linear logic gates.

EXT_SPARS	Success	Partial	Failure
0.2	0	1028	508
0.5	0	1028	508
0.85	0	998	538
0.9	0	998	538
0.95	0	995	541
1	0	827	709

EXT_SPARS	Success	Partial	Failure
0.2	8	1020	508
0.5	13	1015	508
0.85	75	923	538
0.9	92	906	538
0.95	143	853	540
1	200	632	704

Table D. Results filtered by internal sparsity. Left: Success, partial, failure rate of all logic gates. Right: Success, partial, failure rate of linear logic gates.

INT_SPARS	Success	Partial	Failure
0.8	0	1326	210
0.85	0	1185	351
0.88	0	1128	408
0.92	0	1058	478
0.95	0	710	826
0.98	0	467	1069

INT_SPARS	Success	Partial	Failure
0.8	144	1182	210
0.85	92	1093	351
0.88	93	1035	408
0.92	100	958	478
0.95	60	655	821
0.98	42	426	1068