SPIKING NEURAL NETWORKS: A BIOLOGICALLY INFORMED APPROACH TO CLASSIFICATION

Ethan Mitchell

Supervisors: Professor Erik Meijering and Professor Adelle Coster

School of Mathematics and Statistics
UNSW Sydney

July 2022

Submitted in partial fulfillment of the requirements of the degree of
Bachelor of Advanced Mathematics with Honours

Plagiarism statement

I declare that this thesis is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere.

I acknowledge that the assessor of this thesis may, for the purpose of assessing it:

- Reproduce it and provide a copy to another member of the University; and/or,
- Communicate a copy of it to a plagiarism checking service (which may then retain a copy of it on its database for the purpose of future plagiarism checking).

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct, and am aware of any potential plagiarism penalties which may apply.

By signing this declaration I am agreeing to the statements and conditions above.

Signed: E. Mitchell Date: 29/07/22

Acknowledgements

First and foremost, I must thank my supervisors Adelle Coster and Erik Meijering, who have given me an incredible amount of support over the last year. Every weekly catchup with you was an immeasurably valuable learning experience for me – despite the many long days and nights of study, I will always look back on this experience as an overwhelmingly positive one, thanks to you both. I would also like to thank my family, my partner Chrisna and my friends for supporting me over the last year and for all the encouragement to keep working hard.

Ethan Mitchell, 27 July 2022.

Abstract

This research investigated biologically-inspired, artificial neural networks comprised of spiking neurons, to benchmark their performance against traditional artificial networks on classification tasks. It also explored new mechanisms for structural plasticity in artificial spiking networks that were inspired by the biological process of neurogenesis. A brief background of spiking networks is presented and compared to traditional artificial networks. The way these networks process information and the learning schemes applied to them is also explored. Two experiments are used to compare the performance of traditional artificial networks comprised of perceptrons and that of spiking neural networks comprised of leaky integrate-and-fire neurons on classification tasks. A further experiment evaluates the effect on classification performance of modifications to the spiking network architecture via an artificial neurogenesis mechanism. The implications for our understanding of biological signal processing and recent claims made by neuroscientists that spiking networks could be the future of artificial intelligence are discussed.

Contents

Chapter 1

Introduction

The topic of this thesis is spiking neural networks, which are a category of artificial neural networks that closely resemble the processing behaviour of the brain [1]. In a traditional, artificial neural network, the underlying neuron model is heavily simplified, and the units typically transmit information once per process [2] (or at most, at a set of discrete time points, synchronously with the other neurons in their layer). In a spiking neural network, the neuron model is more biologically plausible, where differential equations are often used to model the membrane potential as a function of the input current from external stimuli or other neurons in the network [3]. In these networks, which approximate continuous time with small, discrete steps (between which neuron properties are integrated) [4], neurons can transmit signals at any time.

This thesis is focused on applying spiking neural networks to benchmarking tasks within the field of artificial intelligence. Researchers in the field of computational neuroscience also use spiking neural networks, comprised of very biologically accurate neuron models, such as the Hodgkin-Huxley neuron [5]. Their research is focused on approximating and modelling the behaviour of the brain, not for the purpose of computation, but instead to learn about how the brain processes information, and to make inferences about its internal network structures and the way information is represented.

There has been a disconnect between the work of artificial intelligence researchers, who primarily use traditional, artificial neural networks, and computational neuroscientists, who work almost exclusively with spiking neural networks. This great divide in neuron models didn't always exist. In fact, the first developments in the field of artificial intelligence were directly inspired by neuroscience findings, but over time, the fields have diverged, as artificial intelligence developments were engineered to improve capability and efficiency [6], whilst neuroscience developments optimised the biological realism of models of the brain.

Recently however, notable researchers in both fields have called for renewed collaboration between artificial intelligence and neuroscience, as they believe that the spiking networks used by computational neuroscientists, if introduced into mainstream artificial intelligence applications, could offer some computational benefits [7, 8]. They also claim that recent neuroscience discoveries (since the 1980s), could be used to inspire the development of totally new artificial models and algorithms [9]. However, recent work on spiking networks has demonstrated that current models tend to underperform when compared to state of the art artificial neural networks on a given task, both with respect to the usual performance metric of accuracy, and also when considering training time [10, 11].

This raises the important question of "why?". If the best neuroscience understanding of the human brain suggests that it functions as a spiking neural network, and humans can achieve excellent performance on almost all of the tasks that are used for benchmarking these networks, then why do the far simpler, traditional artificial networks consistently outperform the artificial spiking ones? Assuming that the neuron models are close approximations of how real brain cells work with respect to information processing on a macro scale, and that these artificial spiking networks are given sufficient exposure to training data to attempt to learn these benchmarking tasks, is the biological brain leveraging some other mechanisms for learning that are not captured in the artificial spiking neural networks? One such example is neurogenesis, the biological mechanism responsible for the creation of neurons in the nervous system [12].

The first aim of this thesis was to quantitatively compare the performance of state of the art artificial spiking neural networks and their traditional counterparts. To achieve this, a new, robust experiment was implemented for both types of neural networks, to quantify their performance on classification tasks, including the MNIST digit recognition task, which serves as a popular benchmarking problem in the computer vision field [13].

Once the performance of the two network types was quantified, the second aim of this thesis was to study the mechanism of neurogenesis in detail, in order to produce a mathematical model that is a suitable addition to artificial spiking neural networks. The performance of the artificial spiking network with neurogenesis was then determined on a small, benchmark classification task.

To assess either spiking neural networks (benchmarked against traditional neural networks) or the neurogenesis mechanism (benchmarked against a spiking network without this mechanism), the efficacy of a network on a classification task is measured by:

- the accuracy of outputs given an increasing set of training data, to evaluate the pace at which the network learns the given task, and the maximum performance attained;
- the stability of connection weights in the network during learning, to evaluate whether or not the network's representation of the problem reaches a steady state;
- the utilisation of connections in the network after training, where a smaller utilisation implies a sparser and hence more efficient representation of the task;
- the average number of neuron activations/spikes required to produce an output for a given input; and
- the average processing time to produce an output for a given input.

The first few chapters of this work are dedicated to building up contextual knowledge relevant to neural networks. First, the biological context is explored, then a series of mathematical models of neurons are reviewed, from biologically detailed ones to heavily simplified, binary models which are currently used in traditional neural networks. Artificial neural networks are then introduced: traditional networks of binary neurons and spiking networks of more realistic neurons, as well

as the learning rules that are applied to them, followed by a survey of the state of the art in both types of artificial neural networks.

The experimental setup is then covered, including a detailed description of the spiking network model used and a new mathematical model for neurogenesis, followed by an outline of the parameter optimisation procedure for all network types, and a breakdown of the metrics used to measure classification efficacy. The implementation of this experiment for both network types is then explained.

Following this experimental setup, three experiments are reviewed:

- a comparison of traditional and spiking networks on a small, benchmarking classification task with a very low-dimensional input and output, which permits extensive network visualisations;
- a comparison of the two network types on the significantly larger MNIST digit recognition task, to evaluate whether the qualities of the spiking network shown on the small task persist when learning this larger problem; and finally
- an evaluation of the neurogenesis mechanism on the smaller classification task, by comparing the performance of the spiking network with and without neurogenesis.

Finally, the thesis is concluded with a detailed discussion of all results and an exploration of several ideas for future studies in this area.

Chapter 2

Neurobiology

To study artificial neural networks, one must first gain a basic understanding of the brain and wider nervous system – the biological neural network that all artificial network models are inspired by. This chapter provides a brief background of this topic, first introducing neurons (the nodes of a biological neural network) and synapses (the connections between nodes), before looking at the human visual system, an example of how such a biological network processes information from input to output. Synaptic plasticity, the primary mechanism by which a biological neural network "learns", by adapting its circuitry in response to the network dynamics, is then explored, followed by neurogenesis, which is the biological mechanism by which new neurons are introduced to the nervous system.

2.1 Neurons

Neurons are specialised cells in the nervous system of the body that are electrically excitable, which allows them to transmit information to other cells via connections called synapses [14]. A typical neuron is illustrated in figure 2.1, highlighting its main components. The cell is surrounded by a largely impermeable membrane comprised of a bilayer of lipid molecules [15] that functions as an electrical insulator for the neuron, separating the cytoplasm inside the cell from other substances outside the cell, figure 2.2. The membrane acts electrically as a capacitor, meaning it has

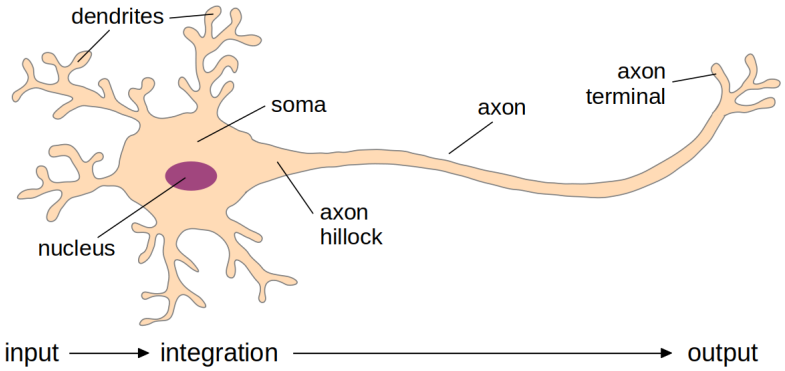

Figure 2.1: A typical neuron with dendrites, a soma and an axon with many branches, representing the input, integration and output of signals respectively. Neurons are termed electrically excitable cells. The concentration of electrically charged chemical ions inside the cell is determined by a series of coupled chemical and electrical interactions, starting at the dendrites, which then affect the soma region of the neuron, which in turn affect the axon of the cell.

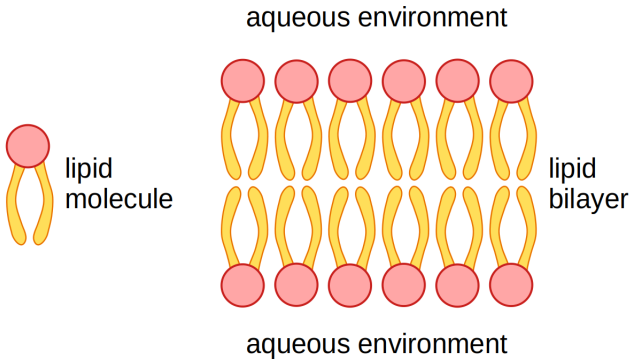

Figure 2.2: A lipid is an organic molecule with a hydrophilic head and multiple hydrophobic tails. When in an aqueous environment, these molecules combine to form a dense bilayer with the heads facing outwards, which is impermeable to ions and many other substances, making them ideal for their role as cell membrane.

the ability to separate charge between the inside and outside of the cell. Protein structures embedded in the membrane form channels that allow chemical ions such as sodium and potassium to cross the membrane and move between the inside and outside of the cell [15]. These come in two main varieties:

- passive channels, which are not specific to one type of ion, and are always open, allowing ions to flow continuously through them, where the direction of ion flow is down the ion-specific concentration gradient (reducing the difference in concentration of that ion between the inside and outside of the cell); and
- active channels, which transport a particular ion in a particular direction, and require work to be opened and closed, allowing or blocking the transit of ions. These come in three broad types:
  - voltage gated channels – where the probability of the channel being open to transport ions is determined by the electrical potential difference, or voltage, across the membrane. One type of voltage gated channel is the "ball and chain" type, where a tethered fragment of the channel protein is positioned either blocking the channel or allowing ions to pass. This mechanism is typical of sodium and potassium channels, figure 2.3;
  - chemically gated channels – where the probability of the channel being open is determined by the concentration gradient of its target ion across the cell membrane; and
  - ion pumps that actively transport ions in the opposite direction of their voltage and/or concentration gradients to maintain homeostasis (the stable/resting state of chemical ion distributions in the neuron).

The chemical imbalance of charged ions, which changes due to their movement through these channels and pumps, causes a position and time dependent electrical potential difference across the membrane [16]. This facilitates an "ionic charge system" for the neuron (where the physical movement of atoms carries charge, as opposed to the flow of electrons), in which the membrane acts as a capacitor. The two most significant ions in and around neurons are sodium ions (Na^+) and potassium ions (K^+) which both have a positive charge [15]. At the neuron's stable/resting

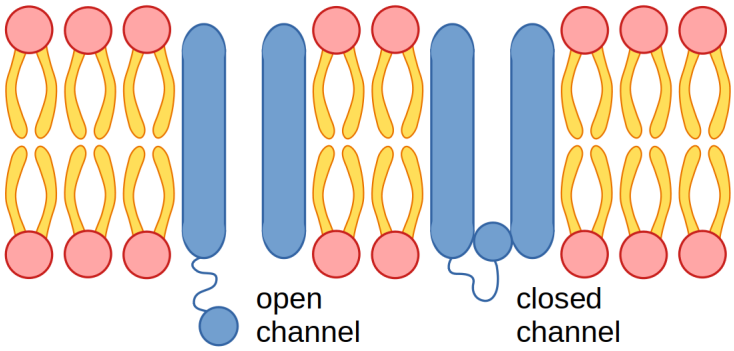

Figure 2.3: A "ball and chain" type ion channel (comprised of proteins) in its open and closed states. Voltage gated sodium and potassium channels in human neurons employ this mechanism, but there are other types for different ion channels.

state, the concentration of sodium ions is higher outside the cell than inside, whilst the concentration of potassium ions is higher inside the cell than it is outside.

When a voltage gated sodium channel opens (in response to an increase in the membrane potential of the cell), sodium ions flow down their concentration gradient into the cell, which further increases the membrane potential (because these sodium ions are carrying positive charge into the cell). This forms a positive feedback loop: as the membrane potential increases, neighbouring voltage gated sodium channels will open and transport even more sodium ions into the neuron.

If the membrane potential in the soma of the cell crosses the cell's excitation threshold, this movement of sodium ions and opening of sodium channels creates a runaway effect where a large number of sodium channels open in a short length of time, initiating a sharp increase in the membrane potential difference, which is called an action potential [16], figure 2.4. Concurrently, a "back flow" or leak through other channels and ion pumps acts to bring the potential back towards the neuron's resting state.

The spike in membrane potential activates other nearby voltage gated ion channels (primarily potassium channels). At its resting state, the concentration of potassium ions is higher inside the cell than outside, then as the action potential reaches its peak, potassium channels open and potassium ions flow out of the cell, down their concentration gradient, which reduces the membrane potential. Near this peak, sodium channels also start to close, which stops more sodium ions from flowing into the cell and results in a rapid reduction in the membrane potential.

The potassium channels generally stay open long enough that the potential difference dips below the resting level, but as more of the potassium channels close, the passive channels and ion pumps restore the neuron to its resting state.

The sodium and potassium ion channels also take a small amount of time to reset after an action potential (where they have been opened and then closed again), during which they are unresponsive (they cannot be opened again, even if their normal voltage or concentration criteria for opening is met). This causes a brief "refractory period" in the neuron, during which no new action potentials can occur.

The approximate temporal shape of an action potential is shown in figure 2.4, identifying the key events that give the spike its usual shape. The first phase of an

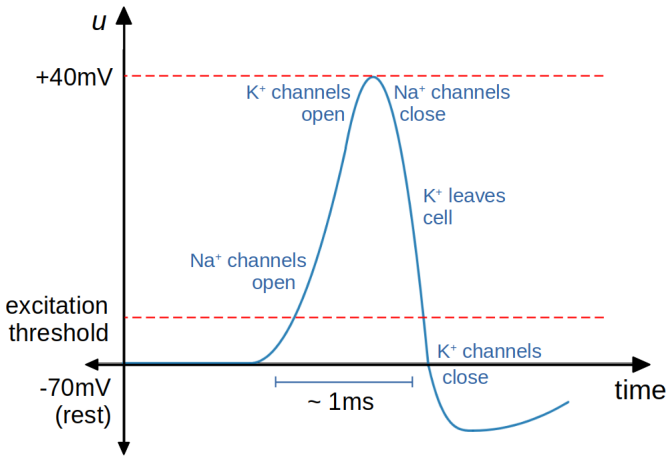

Figure 2.4: Typical time course of the membrane potential difference, u, during an action potential, showing the approximate resting potential of -70mV and the peak potential of +40mV. The scale bar indicates 1ms.

action potential is called the depolarisation of the neuron, where the membrane potential difference rapidly increases, and the second phase is called hyperpolarisation, where the potential comes back down [16]. This is because of the "physiologists's convention": an inward flow of positive ions into the cell is considered to be a negative current.

Action potentials are not global phenomena in the neuron: instead, they occur locally on small patches of membrane. An action potential on one patch of membrane affects neighbouring patches (as ions move around and nearby ion channels open and close), so the action potential also propagates along the membrane of the neuron.

2.2 Synapses

The trigger to initiate the opening of the sodium channels in the soma of the neuron is the integrated effect of changes in the inputs at the dendrites of the cell. The dendrites, which are branched protrusions off the soma of the neuron [14], receive synaptic inputs (stimulation from other neurons) and forward that input by propagating electric stimulation towards the soma, through ion channel activation. As the electrical potential changes in the branched structure of the dendrites, it is integrated, or combined with other inputs as it heads towards the final integrator – the soma.

Synaptic input can have an excitatory effect on the neuron (increasing the likelihood of an action potential by raising the membrane potential) or an inhibitory effect (decreasing the likelihood of an action potential by reducing the membrane potential) [17]. The dendrite's branches mean that a neuron can receive synaptic input from many other neurons (which can arrive at different times), and these varying inputs are combined in the dendrites and soma. Depending on the recent synaptic inputs received at any given time, these inputs may integrate to trigger an action potential in the soma.

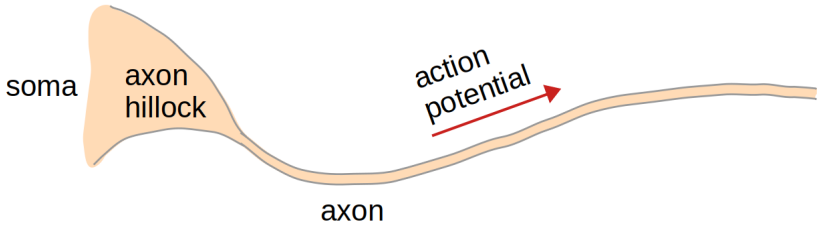

Figure 2.5: The axon hillock of the neuron leading into its axon, which is generally much longer than the soma of the neuron, which allows for connections to distant neurons elsewhere in the nervous system.

There is a special region of the soma called the axon hillock, which has the highest density of voltage activated sodium channels in the soma, making it very excitable, and hence action potentials occurring locally on patches of soma membrane are likely to initiate the propagation of an action potential in this region.

Connected to the axon hillock is the axon (which most but not all neurons have), which is a long, thin protrusion from the soma, up to tens of thousands of times longer than the diameter of the soma, figure 2.5. After an action potential occurs in the axon hillock, it propagates very quickly down the axon. Axons often branch, and each branch carries the action potential independently towards a swollen end called an axon terminal. When an action potential reaches an axon terminal, voltage activated calcium ion channels in the membrane rapidly open, allowing calcium ions to flow into the terminal. This causes synaptic vesicles inside the cell (very small, spherical membrane containers) to rise to the surface of the terminal and fuse with the lipid bilayer structure of the cell membrane. These vesicles house neurotransmitters [18], signalling chemicals used by neurons, which come in more than 100 known varieties, which are released when the vesicle fuses with the membrane. The neurotransmitters then travel across the synaptic cleft, which is a small gap (about 20nm) between the pre-synaptic neuron secreting the neurotransmitters and the dendrite of the post-synaptic neuron receiving them. This connection structure, comprising the axon terminal of the pre-synaptic neuron, the synaptic cleft, and a dendrite of the post-synaptic neuron is called a synapse [19]. A typical synapse is depicted in figure 2.6. The neurotransmitter molecules, once released, diffuse across the synaptic cleft and bind to receptors on the membrane of the dendrite of the post-synaptic neuron [20]. This binding of the neurotransmitters affects ion channels on the post-synaptic dendrite by triggering them to open or close, which can have a local excitatory or inhibitory effect on the post-synaptic neuron, depending on the type of neurotransmitter chemical, the dendrite's structure and other physical properties. Once the input has been received by the post-synaptic dendrite, the integration of inputs at the soma and the triggering of an action potential to fire through the axon (input to integration to output) in the post-synaptic neuron continues the cycle of activation.

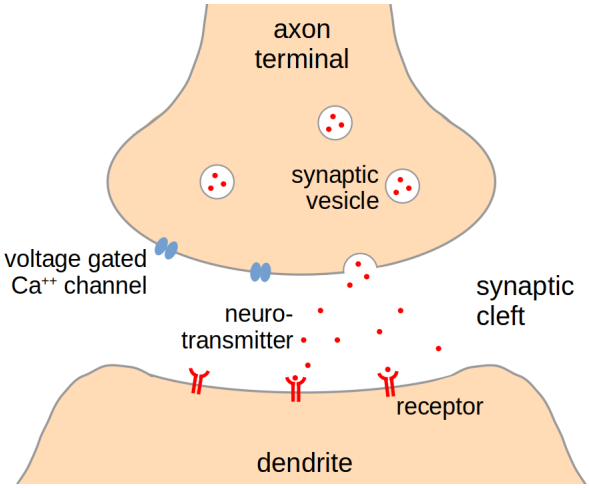

Figure 2.6: A synapse with the axon terminal of the pre-synaptic neuron shown at the top, the synaptic cleft in the middle, and the dendrite of the post-synaptic neuron shown at the bottom. Neurotransmitters are released from the axon terminal and diffuse across the synaptic cleft towards the post-synaptic dendrite (so here, the signal direction is top to bottom).

Just like the dendrite's branched structure means that a neuron can have many input connections, the axon's branched end structure means that a neuron can also have many output connections. Through these synapses, signals can travel as action potentials through many neurons, and one action potential can trigger a large number of downstream spikes (or none at all, for inhibitory connections), as a result of the multiplicity of outgoing connections. Signals can originate in neurons in the central nervous system and propagate to motor neurons to cause the movement of muscles, or travel to different areas of the brain to facilitate complex thoughts or memories. Figure 2.7 shows an example of a network of neurons cultured from stem cells.

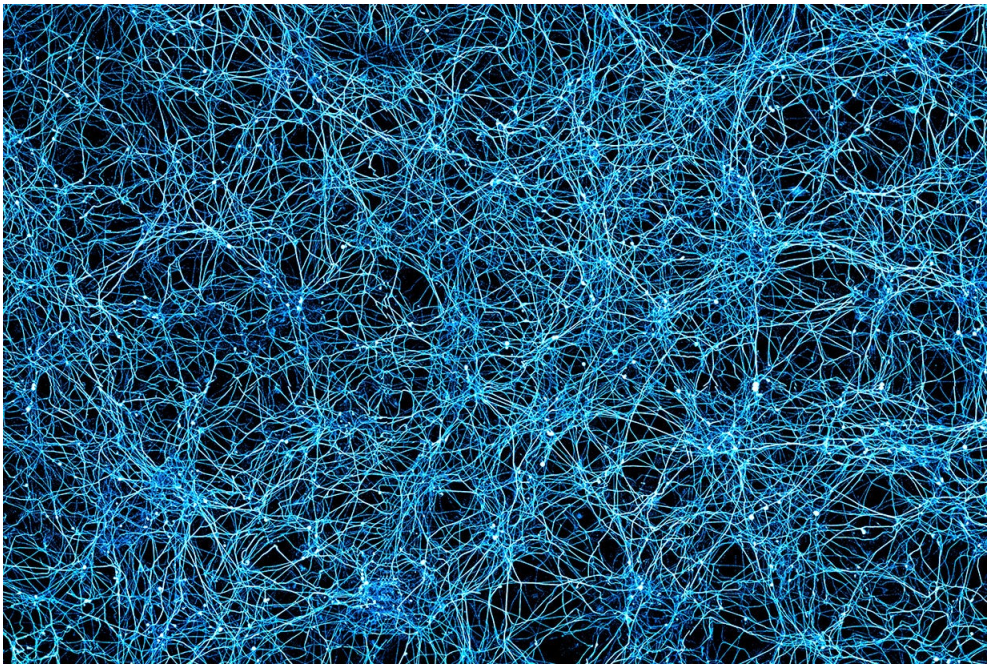

Figure 2.7: A human neural circuit formed in a petri dish by stem cells derived from skin cells. Note the vast complexity of connections that are present. Reproduced from [21].

2.3 Information processing

The human brain is a very large neural network comprised of approximately 86 billion neurons, and a similar number of other supporting cells [22]. In this network, the role of the neuron is to trigger action potentials to contribute towards information processing. Circuits comprised of neurons implement a variety operations including extracting information from sensory input, storing information as memories and translating it into actions [23]. The general consensus amongst neuroscientists today is that the brain uses a "spike timing code" where the precise spike times of neurons are used to encode information, as opposed to their firing rates (a firing rate code) [24], although there is still quite some debate on this topic.

The early stages of the human visual system, including the retina, optic nerve and primary visual cortex, serve as a well-understood example of information processing in the brain [25]. In the retina, incoming light is converted into an electrical signal by an outer layer of photoreceptor cells, which come in two main varieties: rods (which respond to dim light) and cones (which respond to brighter light and specific colours). These photoreceptor cells forward the electrical signal to other interneurons in the retina through synaptic connections. Eventually, the signal reaches retinal ganglion cells, which are the output neurons of the retina, and whose spike patterns correspond to the intensity of light detected by the retina.

The axons of the retinal ganglion cells form the optic nerve and extend to the lateral geniculate nucleus in the thalamus, where neurons here receive the retina's output signals and relay them to the primary visual cortex, figure 2.8.

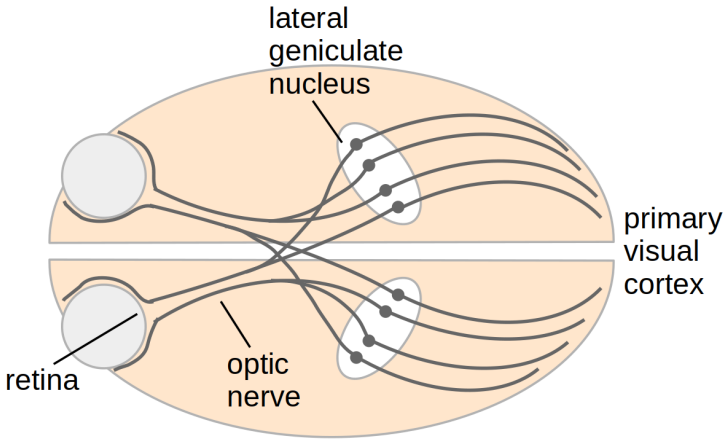

Figure 2.8: A heavily simplified, top-down view of the human brain where the eyes are on the left, the right hemisphere is on top and the left hemisphere is on the bottom. Axons protruding from the retinal ganglion cells in the retina form the optic nerve, which extend a long distance through the brain to the lateral geniculate nucleus of either hemisphere, where their signals are relayed by neurons whose axons extend into the primary visual cortex at the rear of the brain.

Neurons in the primary visual cortex only respond to signals in specific regions of the visual field known as their "receptive fields". These neurons exhibit enhanced firing rates in response to illumination in the receptive field that is either above or below the background level over the whole visual field. This circuitry allows for spatial information of the visual input to be encoded in the firing patterns of the primary visual cortex, which is key for downstream processing and generating reactions in response to this input.

2.4 Synaptic plasticity

A neural network "learns" by tuning the strengths of its synaptic connections to change the processing behaviour of the system. In biological neural networks, the strength of a synapse is a function of several properties [26] including:

- the amount of neurotransmitter released by the pre-synaptic axon terminal;
- the density and efficiency of neurotransmitter receptors on the post-synaptic dendrite;
- the number of ion channels on the dendrite; and
- the surface area and local branching structure of the dendrite.

The tuning of synapse strengths is called "synaptic plasticity", which includes short-term synaptic enhancement/depression [27] (which usually acts on a millisecond or second timescale) and long-term potentiation/depression [28] (which acts on a minute, hour or longer timescale).

The first suggestion of a mechanism for synaptic plasticity was raised by Donald Hebb in his book "The Organisation of Behaviour" in 1949, where he introduced a theory that is now commonly known as Hebb's postulate [29]:

When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

This theory was developed to explain the observed phenomenon of Hebbian learning, where two connected cells spiking at similar times increases the strength of the synapse between them. Spike timing dependent plasticity is a specialisation of this theory [30], where simple proximity of spike times is not sufficient to increase synaptic strength, but instead:

- if the pre-synaptic cell spikes just before the post-synaptic cell, this implies causality so the synapse is strengthened (biologically, this means the size of the current pulse received by the post-synaptic cell is increased); and
- if the pre-synaptic cell spikes just after the post-synaptic cell, the synapse is weakened (because this event suggests that the pre-synaptic neuron is not the cause of the post-synaptic cell spiking).

Over time, spike timing dependent plasticity will reduce the strength of synapses between non causally correlated neurons to zero, leaving only the most "important" causal connections in place.

More than 70 years after Hebb introduced the theory, the biological mechanisms that implement synaptic plasticity are now well understood. For example, early-phase long-term potentiation [28] is brought about via phosphorylation in the post-synaptic dendrite – a chemical reaction where a phosphate group (a phosphorus atom bonded to four oxygen atoms) attaches to another molecule to change its behaviour [28]. When a pre-synaptic and post-synaptic neuron fire in short sequence:

- existing neurotransmitter receptors are phosphorylated to increase their efficiency at converting neurotransmitter into local excitation; and
- phosphate groups use phosphorylation to modulate the process by which additional neurotransmitter receptors (stored in a pool near the membrane) are inserted into the membrane.

It is also theorised that increased phosphorylation in the post-synaptic dendrite can trigger the creation of a messenger chemical that travels back across the synaptic cleft to the pre-synaptic axon terminal, where it modulates the availability of synaptic vesicles or the likelihood of their release, in order to strengthen the connection on the pre-synaptic side [31].

2.5 Neurogenesis

Neurogenesis is the biological mechanism by which neurons are created from neural stem cells, migrate to different regions of the nervous system, and are eventually integrated into existing neural circuitry [12]. There are two types of neurogenesis: developmental and adult [32], and developmental neurogenesis occurs at a much faster rate than adult neurogenesis, to facilitate the construction of the body's nervous system while the embryo is growing. This process begins in "neurogenic niches", which are specialised regions of the brain containing stem cells (such as radial glial cells), which differentiate into neural progenitor cells, which then divide

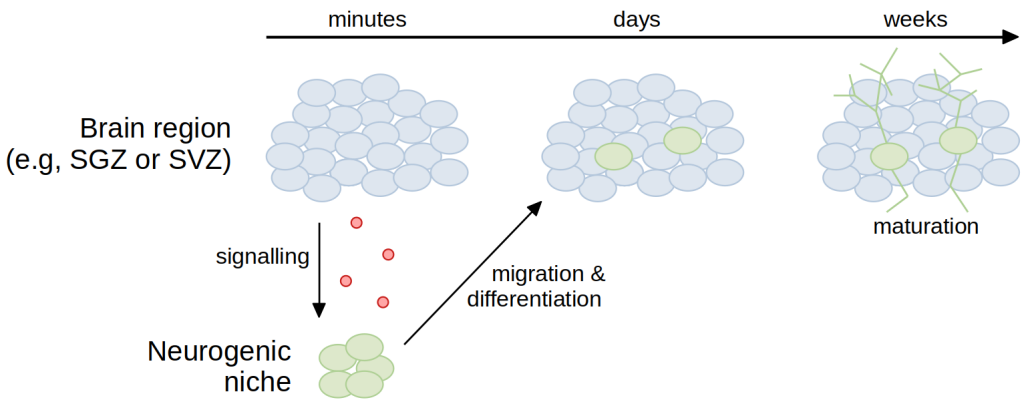

Figure 2.9: Neurogenesis begins in neurogenic niches, which are regulated by neurotransmitter chemicals released by mature neurons in the brain. Stem cells in the niches differentiate and migrate to their target region, where they slowly mature as they form synapses with existing neurons to begin contributing to processing.

to create functional neurons. Neurons do not divide (most remain functional for the lifespan of the animal), so neurogenesis is the only mechanism by which new neurons can be introduced to the nervous system [33]. Signalling between a target brain region and a neurogenic niche takes place on a minute timescale, then the migration and differentiation of neural stem cells to that region takes several days, where these new neurons often take a few weeks to mature into adult neurons, as illustrated in figure 2.9.

Compared to mature cells, neurons arising from neurogenesis demonstrate enhanced excitability and synaptic plasticity during the critical period after maturation while they are integrating into neural circuitry. It is hypothesised that this enhanced state may give new neurons an advantage over existing neurons for forming and stabilising new synapses, and may also allow these new neurons to make unique contributions to processing during this critical period. After these new neurons have fully matured, they exhibit the same levels of excitability and plasticity as far older neurons, so this enhanced state is only temporary. Not only is the rate of neurogenesis slower in the adult brain than in the developing embryo, but the pace of neuronal maturation is also slower (the enhanced state of new neurons is longer in adult neurogenesis). More details on adult neurogenesis can be found in section A of the appendix.

Neurogenesis is a type of structural plasticity in the brain, where even existing, mature neurons can be considered to be plastic as they regularly forge new functional synapses with new neurons born from neurogenesis. Neurogenesis can also be likened to "axonal regeneration", another structural plasticity phenomena where existing neurons sprout new axons and synapses to repair (often partial) circuit function after an injury to the nervous system [34], joining circuitry they were not previously part of. The addition of such a neuron to an existing, damaged circuit to repair functionality is very similar to the addition of a neuron via neurogenesis, as both result in the integration of a neuron that was not previously connected to the circuit, for the purpose of increasing capacity. Axonal regeneration is known to occur throughout the human nervous system, but is particularly

effective in the peripheral nervous system, which includes all neurons and axons outside of the brain and spinal cord [34].

In an artificial neural network, neurons are defined as mathematical models which approximate the complex, biological detail of real neurons with either a system of differential equations (where the main state variable is a membrane potential), or simply a function that receives numerical inputs and returns a numerical output. Synapses are often represented by weighted connections between artificial neurons, where the output of the pre-synaptic neuron is combined with the connection weight and delivered as immediate input to the post-synaptic neuron. These neurons and connections form an "architecture" for the network, where some neurons are labelled as "input" nodes, which receive external, numerical input, whilst some others are labelled as "output" nodes, whose spikes or activations are recorded to produce the network's output. The remaining neurons are often referred to as "hidden" nodes in the network. Finally, a learning rule is used to tune the connection weights either in response to the network's dynamics or to minimise a given loss function. Some of the common mathematical models encoding different abstractions of the neurobiology explored in this chapter are introduced in the following chapter.

Chapter 3

Mathematical review

To appreciate the current state of the art of both traditional, artificial neural networks and their spiking network counterparts in the artificial intelligence field, one must understand the history of neuron models relevant to these networks, the manner in which artificial neurons are organised to create them and the learning schemes used to optimise them for a given task.

Mathematical models of neural behaviour today are still largely based on the seminal work on Hodgkin and Huxley in 1952 [35], who modelled action potentials in the giant squid axon. Their model is very biologically detailed and its parameters can be fit to experimental data from real neurons, but its mathematical complexity makes it an unsuitable candidate as a processing unit in artificial neural networks.

On the opposite end of the spectrum to the Hodgkin-Huxley model lies the perceptron – the heavily simplified processing unit, predating biologically realistic descriptions, that is used in traditional artificial networks. This model has seen three significant stages of development:

- the introduction of the McCulloch-Pitts model in 1943 [36];
- the improvement that was the Rosenblatt perceptron in 1958 [37]; and
- the refinement to the modern perceptron by Minsky and Papert in 1969 [38].

The leaky integrate-and-fire neuron, which was first suggested by Lapicque back in 1907 [39], is a very popular neuron model for artificial spiking networks, as it features substantially more biological accuracy than the perceptron without the complexity of the Hodgkin-Huxley neuron.

Although the perceptron hasn't undergone significant change since the late 1960s, and the leaky integrate-and-fire neuron is more than 100 years old, the study of artificial neural networks is a field of research that is home to much innovation. This innovation often lies in the architecture of neural networks: the structure in which these neurons are organised and connected; as well as the learning schemes used to train them on a given classification problem [6].

A survey of spiking neural networks applied to the MNIST classification task [13] (a digit recognition dataset that is a popular benchmark task in the field of computer vision) found that most successful approaches explored in the last decade have all relied on mechanisms that are not biologically plausible [40]. One prominent example of a state of the art approach is the leaky integrate-and-fire network developed by Peter Diehl and Matthew Cook in 2015 [11]. This network uses a variant of the leaky integrate-and-fire neuron model that involved labelling neurons as either "excitatory" or "inhibitory" (based on their outgoing connections) and using conductance variables at each neuron to collect incoming stimuli.

The mathematical models for some biologically realistic and biologically inspired artificial neurons are developed below. Networks of these units and the common algorithms by which their connections are adjusted to learn – or optimise their output – for a given computational task are then reviewed.

3.1 Neuron models

3.1.1 Hodgkin-Huxley neuron

The Hodgkin-Huxley neuron model was developed in 1952 for the purpose of modelling action potentials in the giant squid axon [35]. Despite not knowing of the existence of ion channels embedded in the neuron membrane (as discussed in section 2.1), Hodgkin and Huxley hypothesised that the membrane potential fluctuated with its permeability to different ions – in particular, sodium, potassium and a leak current (mostly comprised of chloride ions). If the membrane is particularly permeable to one ion over the others, then its potential will tend towards the Nernst potential of that ion (also called the equilibrium potential, which is the voltage at which there is no net flow of that ion between the inside and outside of the membrane). Hodgkin and Huxley represented their neuron model by a circuit diagram much like the one depicted in figure 3.1. The variability of the sodium and potassium resistances allows this circuit to model their hypothesis about the membrane's varying permeability to these ions.

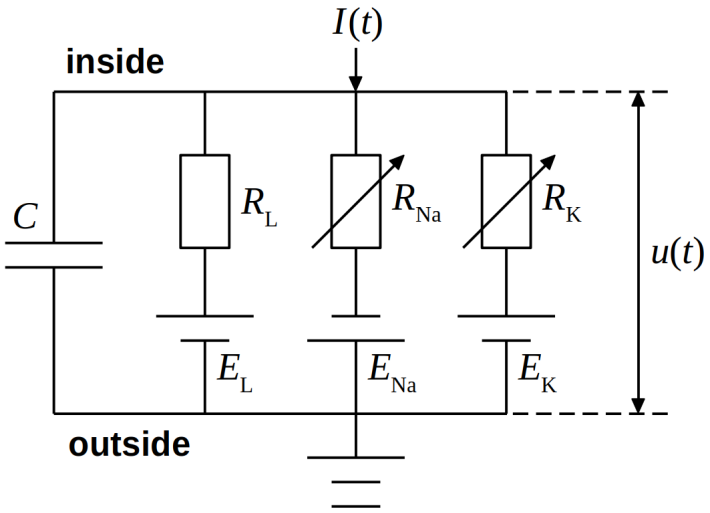

Figure 3.1: The circuit diagram for the Hodgkin Huxley neuron adapted from [5]. The membrane acts as a capacitor with capacity C as it separates the inside of the cell (top of the diagram) from the extracellular liquid (bottom of the diagram), resulting in a membrane potential difference $u(t)$. Given an input current $I(t)$ into the cell, charge may be added to the capacitor or cause the flow of ions through the semi-permeable membrane. The permeability of the membrane to each ion type is represented by a resistor in parallel with the membrane capacitor. The generic leak has a fixed resistance R_L, sodium ionic flow has a variable resistance R_{Na} and potassium ionic flow has a variable resistance R_{K}. The Nernst potential for each ion type is represented by a battery in series with the corresponding resistor, with voltages E_{L}, E_{Na} and E_{K} for the leak, sodium and potassium ions respectively.

By the conservation of electric charge, an external input current $I(t)$ to the neuron can be considered as the sum of the capacitor charging current $I_C(t)$ and each of the ionic flow currents $I_{\mathrm{L}}(t)$, $I_{\mathrm{Na}}(t)$ and $I_{\mathrm{K}}(t)$ through the corresponding resistors in figure 3.1, so

$$I(t) \;=\; I_C(t) + I_{\mathrm{L}}(t) + I_{\mathrm{Na}}(t) + I_{\mathrm{K}}(t). \tag{3.1.1}$$

Current is the rate of change of charge over time, and the voltage-charge relationship of a capacitor is defined as $C = q/u$, where q is the charge held on the capacitor and u is the electrical potential difference across it. Therefore the capacitor charging current can be expressed as

$$I_C(t) \;=\; \frac{dq}{dt} \;=\; C\frac{du}{dt}. \tag{3.1.2}$$

The ionic leak has a fixed conductance $g_{\mathrm{L}} = 1/R_L$. The voltage across the neuron's membrane is $u(t)$ and the leak resistor is in series with a battery of voltage E_{L}, and hence the voltage at the leak resistor is $u - E_{\mathrm{L}}$. By Ohm's law (voltage is the product of current and resistance), the leak current can be expressed as

$$I_{\mathrm{L}}(t) \;=\; g_{\mathrm{L}}(u(t) - E_{\mathrm{L}}). \tag{3.1.3}$$

The sodium and potassium currents can be similarly defined, except their conductances are not fixed, and instead depend on the membrane potential and time. To model these, Hodgkin and Huxley defined three gating variables for their neuron: h, m and n, which are dimensionless and take values in the range $[0, 1]$. These represent the processes in the neuron that affect the membrane's permeability to sodium and potassium ions. The sodium conductance is defined as $g_{\mathrm{Na}}m^3h$ and the potassium conductance as $g_{\mathrm{K}}n^4$, where g_{Na} and g_{K} are the maximum sodium and potassium conductances respectively. Thus

$$I_{\mathrm{Na}}(t) \;=\; g_{\mathrm{Na}}m^3h(u(t) - E_{\mathrm{Na}}) \tag{3.1.4}$$

$$\text{and } I_{\mathrm{K}}(t) \;=\; g_{\mathrm{K}}n^4(u(t) - E_{\mathrm{K}}). \tag{3.1.5}$$

Combining equations 3.1.1 - 3.1.5,

$$I(t) \;=\; C\frac{du}{dt} + g_{\mathrm{L}}(u(t) - E_{\mathrm{L}}) + g_{\mathrm{Na}}m^3h(u(t) - E_{\mathrm{Na}}) + g_{\mathrm{K}}n^4(u(t) - E_{\mathrm{K}}).$$

This equation can be rearranged to formulate a set of differential equations for the membrane potential and the gating variables,

$$\begin{aligned} C\frac{du}{dt} &= I(t) - g_{\mathrm{L}}(u - E_{\mathrm{L}}) - g_{\mathrm{Na}}m^3h(u - E_{\mathrm{Na}}) - g_{\mathrm{K}}n^4(u - E_{\mathrm{K}}), \\ \frac{dh}{dt} &= (1-h)\alpha_h(u) - h\beta_h(u), \\ \frac{dm}{dt} &= (1-m)\alpha_m(u) - m\beta_m(u), \\ \frac{dn}{dt} &= (1-n)\alpha_n(u) - n\beta_n(u). \end{aligned}$$

The Nernst potentials are approximately $E_{\mathrm{L}} = -65\mathrm{mV}$, $E_{\mathrm{Na}} = 55\mathrm{mV}$ and $E_{\mathrm{K}} = -77\mathrm{mV}$, whilst typical values for the maximum conductances in mammalian neurons are $g_{\mathrm{L}} = 0.3$ mS/cm^2, $g_{\mathrm{Na}} = 40$ mS/cm^2 and $g_{\mathrm{K}} = 35$ mS/cm^2 (where S stands for siemens, the SI unit for conductance). The gating functions α_h, α_m and α_n vary and are determined empirically for a given type of neuron. For pyramidal neurons in the cortex [5, 41], these coefficients are

$$\begin{aligned} \alpha_h(u) &= 0.25\exp\left(-\frac{u+90}{12}\right), \\ \beta_h(u) &= 0.25\frac{\exp\left(\frac{u+62}{6}\right)}{\exp\left(\frac{u+90}{12}\right)}, \\ \alpha_m(u) &= 0.182\frac{u+35}{1-\exp\left(-\frac{u+35}{9}\right)}, \\ \beta_m(u) &= -0.124\frac{u+35}{1-\exp\left(\frac{u+35}{9}\right)}, \\ \alpha_n(u) &= 0.02\frac{u-25}{1-\exp\left(-\frac{u-25}{9}\right)}, \\ \beta_n(u) &= -0.002\frac{u-25}{1-\exp\left(\frac{u-25}{9}\right)}. \end{aligned}$$

The Hodgkin-Huxley model was so effective at producing biologically accurate membrane potential deflections in response to input currents, that Hodgkin and Huxley received the 1963 Nobel Prize in Physiology or Medicine for this research. Today, this model continues to see use as one of the most popular biologically detailed neuron models in the neuroscience field. However, its level of biological accuracy makes this neuron model computationally inefficient for artificial neural networks (for the purpose of information processing and learning).

3.1.2 Leaky integrate-and-fire neuron

The behaviour of biological neurons can be heavily simplified down to two major components. The first of these is an integration process, where the neuron accumulates input received via its dendrites. The second is a "spiking mechanism", which is activated when the neuron's membrane potential reaches some spiking threshold, which triggers an action potential in the cell. The "spike time" associated with this action potential is usually defined as the time when the membrane potential

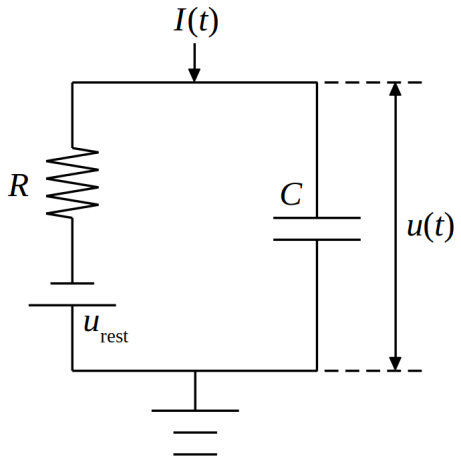

Figure 3.2: The circuit diagram of a leaky integrate-and-fire neuron which has a driving current $I(t)$, a capacitor C in parallel with a resistor R and a battery of potential u_{rest}. The membrane potential $u(t)$ is defined as the voltage across the capacitor.

reaches the spiking threshold, rather than any other point during the roughly 1ms duration of the action potential.

As discussed in section 2.1, action potentials in a particular neuron always have approximately the same shape (though different neuron types may exhibit different action potential shapes) [42]. Hence, the shape of these action potentials is not used to transmit information, but instead, information is only communicated by the presence or absence of a spike. This suggests a neuron model comprised of an equation for the membrane potential that controls its summation of input, and a spiking threshold that triggers action potentials as discrete events at precise points in time (specifically, when the potential difference reaches the threshold from below).

This is precisely the leaky integrate-and-fire model, which was introduced by Lapicque in 1907 [39], and remains a popular choice for simulation and modelling of spiking neurons today. This model defines a linear differential equation for the membrane potential $u(t)$ at time t, and a spiking threshold u_{thresh}, where if $u(t)$ reaches u_{thresh}, the neuron fires an action potential. The following derivation of the leaky integrate-and-fire model is based on the description by Gerstner et al. [3].

The electrical circuit representing a leaky integrate-and-fire neuron, figure 3.2, contains a capacitor C representing the cell membrane in parallel with a resistor R representing the ion channels, driven by an input current $I(t)$. The resistor is in series with a potential u_{rest} to ensure that u_{rest} is the minimum value of $u(t)$, the voltage across the capacitor. The constant resting potential u_{rest} represents the non-uniform distribution of charge inside and outside the cell when it is not firing or receiving electrical input.

The neuron receives an input current $I(t)$ at time t, which is the sum of the current through the resistor, $I_R(t)$, and the capacitor charging current $I_C(t)$, because the membrane capacitance is in parallel with the leak resistor. Between times t_1

and t_2, the electrical charge is

$$q \;=\; \int_{t_1}^{t_2} I(t)\, dt.$$

The membrane is an imperfect insulator for the cell, so any charge it separates will slowly leak away through ion channels and pores with some resistance R (this charge leak is where the term "leaky" in the model name comes from). The potential difference across the leak resistor is given by Ohm's Law,

$$u_R(t) \;=\; I_R(t)R$$

where $u_R(t)$ is the voltage across the resistor, $I_R(t)$ is the current passing through it and R is its resistance. The membrane potential in the leaky integrate-and-fire circuit is the sum of the rest potential and the potential across the leak resistor,

$$u(t) \;=\; u_{\text{rest}} + u_R(t) \;=\; u_{\text{rest}} + I_R(t)R.$$

Rearranging,

$$\begin{aligned} u(t) - u_{\text{rest}} \;&=\; I_R(t)R \\ I_R(t) \;&=\; \frac{u(t) - u_{\text{rest}}}{R}. \end{aligned}$$

The voltage-charge relationship of a capacitor is given by

$$q(t) \;=\; Cu(t)$$

where $q(t)$ is the charge held on the capacitor and $u(t)$ is the electrical potential difference across it, and hence

$$q(t) \;=\; C\left(u(t) - u_{\text{rest}}\right).$$

Differentiating this yields the capacitor charging current

$$I_C(t) \;=\; \frac{dq}{dt} \;=\; C\frac{du}{dt}.$$

Thus, the input current can be expressed as

$$I(t) \;=\; I_R(t) + I_C(t) \;=\; \frac{u(t) - u_{\text{rest}}}{R} \;+\; C\frac{du}{dt}.$$

Rearranging this equation yields

$$RC\frac{du}{dt} \;=\; -(u(t) - u_{\text{rest}}) + RI(t), \tag{3.1.6}$$

which is a first order, linear differential equation for the membrane potential $u(t)$ of the neuron when it is below its threshold potential.

A "spike" occurs when $u(t)$ reaches the threshold u_{thresh} from below. When this happens, an instantaneous action potential is triggered (which, in a network, would transmit an input current to any outgoing synapses) and then the voltage $u(t)$ is instantaneously set to some $u_{\text{reset}} \leq u_{\text{rest}}$. For a realistic refractory period for the neuron, $u_{\text{reset}} < u_{\text{rest}}$, delaying the onset of a subsequent action potential.

The leaky integrate-and-fire model is a "switching regime", with separate sub-threshold and threshold dynamics for the membrane potential $u(t)$,

$$\begin{aligned} RC\frac{du}{dt} &= -(u(t) - u_{\text{rest}}) + RI(t) \text{ when } u < u_{\text{thresh}}, \\ u(t) &= u_{\text{reset}} \text{ when } u = u_{\text{thresh}}. \end{aligned}$$

Formally, the reset condition is defined as

$$\lim_{\Delta t \to 0^+} u(t^{(f)} + \Delta t) = u_{\text{reset}},$$

where the $t^{(f)}$, $f = 1, 2, \ldots$, are the firing times

$$t^{(f)} \in \{t : u(t) = u_{\text{thresh}}\}.$$

Consider a leaky integrate-and-fire neuron with an initial voltage $u(0) = u_{\text{rest}} + \Delta u$ for some $\Delta u > 0$, and no input current so $I(t) = 0$ for all $t \geq 0$. This simplifies the governing differential equation, 3.1.6, to

$$RC\frac{du}{dt} = -(u(t) - u_{\text{rest}}).$$

This has the solution

$$u(t) = u_{\text{rest}} + \Delta u e^{-t/RC},$$

and hence $u(t) \to u_{\text{rest}}$ as $t \to \infty$, which is the expected behaviour in the absence of any input. The time constant of the membrane potential u, which is the time taken for the potential to drop to $1/e$ of its initial value (relative to u_{rest}), is $\tau = RC$.

Now consider the case of a constant input current $I(t) = I_0$ with $u(0) = u_{\text{rest}}$, so the neuron is initially at rest. The differential equation 3.1.6 then becomes

$$\tau\frac{du}{dt} = -(u(t) - u_{\text{rest}}) + RI_0,$$

which has the solution

$$u(t) = u_{\text{rest}} + RI_0(1 - e^{-t/\tau})$$

for $u(t) < u_{\text{thresh}}$. Therefore, $u(t) \to u_{\text{rest}} + RI_0 < u_{\text{thresh}}$ as $t \to \infty$, unless the input current I_0 is large enough to raise the membrane potential to its spiking threshold (which would invoke the threshold dynamics and reset the potential). For a short pulse of input current of duration t_0, the membrane potential reaches a sub-threshold value of

$$u(t_0) = u_{\text{rest}} + RI_0(1 - e^{-t_0/\tau}).$$

For pulse durations $t_0 \ll \tau$, a first order Taylor series ($e^{-x} \approx 1 - x$) can be used to approximate this as

$$u(t_0) \;\approx\; u_{\text{rest}} + RI_0\frac{t_0}{\tau}.$$

Therefore, the voltage increase

$$u(t_0) - u(0) \;=\; u(t_0) - u_{\text{rest}} \;\approx\; RI_0\frac{t_0}{\tau}$$

depends linearly on both the size I_0 and duration t_0 of sufficiently short current pulses. Hence, it is the charge delivered during the pulse,

$$q = \int_0^{t_0} I(t)\,dt \;=\; I_0 t_0$$

that controls the change in voltage. Thus, an equivalent voltage increase to the neuron can be achieved by changing the duration of the pulse, so long as the height is adjusted to preserve the product $I_0 t_0$. The limiting case of this is an instantaneous input pulse, where delivering charge q causes a discrete voltage increase

$$\Delta u \;=\; \frac{Rq}{\tau} \;=\; \frac{q}{C}. \tag{3.1.7}$$

Now consider the case of a neuron that is initially "fully charged" by a constant input current I_0, meaning $u(0) = u_{\text{rest}} + RI_0 < u_{\text{thresh}}$. The input has been switched off, so $I(t) = 0$ for all $t \geq 0$. The differential equation 3.1.6 simplifies to

$$\tau\frac{du}{dt} \;=\; -(u(t) - u_{\text{rest}}),$$

which has the solution

$$u(t) \;=\; u_{\text{rest}} + RI_0 e^{-t/\tau},$$

which again approaches the steady state at u_{rest} as $t \to \infty$.

Examples of the behaviour of leaky integrate-and-fire neurons under different current stimuli are shown in figure 3.3, using the neurobiologically realistic parameter values:

- spiking threshold $u_{\text{thresh}} = -50\text{mV}$;
- resting membrane potential $u_{\text{rest}} = -65\text{mV}$;
- post-spike reset membrane potential $u_{\text{reset}} = -70\text{mV}$;
- time constant $\tau = 10\text{ms}$; and
- membrane resistance $R = 1$ Ohm.

First, consider a neuron at rest which receives a single-pulse input defined by the step function

$$I(t) \;=\; \begin{cases} I_0 & \text{for } t \in [0, t_0], \\ 0 & \text{otherwise.} \end{cases}$$

Figure 3.3 shows the membrane potential $u(t)$ over a simulated period of 100ms for three different input pulses with the same duration $t_0 = 50\text{ms}$, but different amounts of current: 10mA, 20mA and 30mA, delivering 500μC, 1000μC and 1500μC

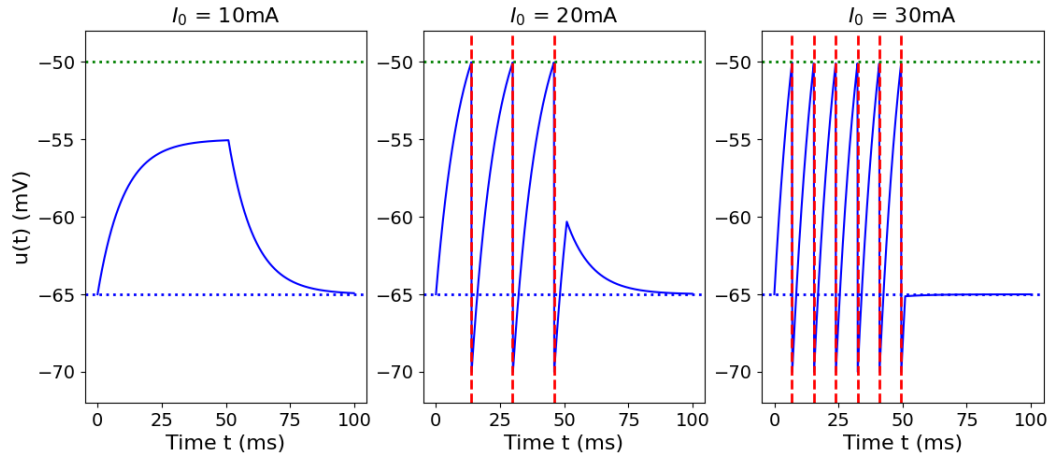

Figure 3.3: The membrane potential of three leaky integrate-and-fire neuron simulations with different input pulses. The spikes are shown as red, dashed, vertical lines, the spike threshold is shown as a horizontal, dashed green line and the resting level is shown as a horizontal, dashed blue line. Each simulation starts with the neuron at rest. All three input pulses last for 50ms, with current $I_0 = 10$mA, $I_0 = 20$mA and $I_0 = 30$mA from left to right.

of charge respectively. The input current of 10mA is not sufficient to push the membrane potential above the spiking threshold, which is the expected behaviour as the product $RI_0 = 10$mV, whilst the difference between the spiking threshold and resting level is 15mV. For $I_0 = 20$mA and 30mA, spiking behaviour of the leaky integrate-and-fire neuron is observed. It can be seen that the firing rate of the neuron depends on the size of the fixed-duration input pulse received, with larger input pulses (delivering more charge per unit time to the membrane) resulting in a faster firing rate. Note that the potential $u(t)$ tends towards u_{rest} at the conclusion of the current pulse.

The dynamics of the membrane potential is not biologically accurate, however the repeated spiking in response to a sufficiently large input current is an accurate reflection of the spiking behaviour of biological neurons stimulated in this manner [25]. There is a significant benefit in the simplicity of this model (defined by a single first order ODE and a basic reset condition).

Now consider a network of three leaky integrate-and-fire neurons, figure 3.4, where the two neurons on the left receive an input pulse, and are connected via synapses (from left to right) to the neuron on the right.

Whilst the leaky integrate-and-fire neuron has an analytic solution for constant input currents, its behaviour becomes more complex for varying currents. Rather than pursuing an analytic solution for a network of coupled neurons, a numerical implementation of the model was developed in Python using the Brian package [43, 4], which is discussed in detail in section B of the appendix. The implementation assumes that the synaptic current input to the neuron is of sufficiently short

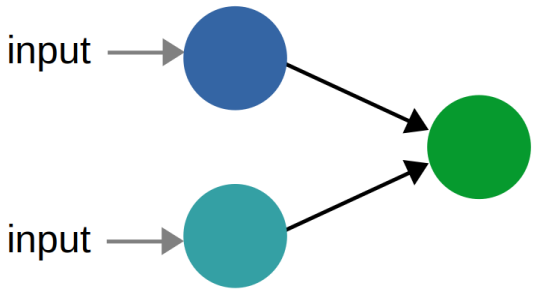

Figure 3.4: An example architecture of a leaky integrate-and-fire network, showing the two pre-synaptic neurons on the left, which receive an external input pulse, and the post-synaptic neuron on the right, which receives synaptic input whenever a connected pre-synaptic neuron spikes. The neurons are colour-coded to match their membrane potential plots in figure 3.5.

duration (see equation 3.1.7) that the input can be considered as an instantaneous change in the potential of Δu.

Two examples of the output of this network are shown in figure 3.5. Pre-synaptic neuron 1 (dark blue) receives a single input pulse of 20mA from $t = 0$ms to $t = 50$ms, whilst pre-synaptic neuron 2 (light blue) receives a delayed input pulse of 20mA from $t = 5$ms to $t = 50$ms, and post-synaptic neuron 3 (green) receives only synaptic input. The first simulation in figure 3.5 increments the post-synaptic potential by $\Delta u = 6$mV (delivered instantaneously upon a pre-synaptic spike), whilst the second simulation in figure 3.5 has $\Delta u = 12$mV. At the lower synaptic connectivity, the connection strength is not sufficient to cause the post-synaptic neuron to spike, but does increase its membrane potential to within 5mV of the spiking threshold. At the higher connectivity, pairs of two pre-synaptic spikes result in the post-synaptic neuron spiking at approximately 20 and 35ms into the simulation period. The network dynamics in figure 3.5 demonstrate that the connection strength or weight of synapses in a network of leaky integrate-and-fire neurons plays a major role in determining the behaviour of post-synaptic neurons in response to pre-synaptic spikes. Increasing the strength of the two synapses in the network resulted in a greater firing rate of the post-synaptic neuron, which is the desired result for an artificial spiking network (despite the simplicity of the synapse model that was used), which may encode information in the rate of spike production.

This illustrates that the leaky integrate-and-fire neuron model is a suitable candidate for simulation in an artificial spiking network. It is able to approximate the integration/summation behaviour of the membrane potential of biological neurons and the mechanism of firing an action potential, whilst making use of the simplifying assumption that action potential shapes do not carry information, to reduce these to discrete, instantaneous events. This neuron model consists of just one differential equation and a spiking threshold, and is therefore very efficient for computer simulation, even in large networks.

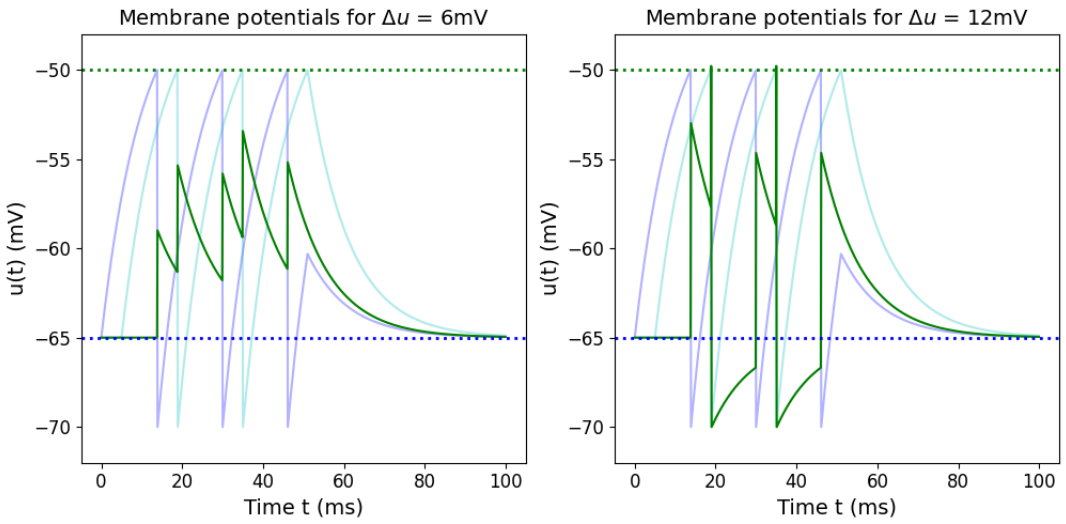

Figure 3.5: The membrane potentials of the leaky integrate-and-fire network in figure 3.4 for different connection strengths Δu. The spike threshold is shown as a horizontal, dashed green line and the resting level is shown as a horizontal, dashed blue line. Both simulations start with all neurons at rest. The membrane potentials of the two pre-synaptic neurons (in dark blue and light blue) are partially faded in this plot to draw attention to the membrane potential of the post-synaptic neuron in green.

3.1.3 Perceptrons

Predating biologically realistic systems, the earliest neuron model designed for computational use was the McCulloch-Pitts model, proposed in 1943 [36], which is now commonly referred to as the "classical perceptron". This was a heavily simplified model of the neuron that had binary inputs and no weighted connections. Indeed, there was no simplified notion of synaptic plasticity at this time, as Hebb's postulate was not published until 1949, six years after this model was introduced. The classical perceptron had a binary output controlled by a fixed threshold. The lack of understanding of modern neuroscience was not the only reason this model is so simple – another major reason was that computing power was very limited and difficult to access in the 1940s [44], so computing with a more complex model was not physically possible.

In 1958, fifteen years after the McCulloch-Pitts model was introduced, the Rosenblatt perceptron was invented at the Cornell Aeronautical Laboratory [37] in the US. This model was designed to resolve the key issues of its predecessor, by introducing input weights and making the threshold a parameter of the model that could be learned. An increase in computing power over this 15 year period allowed this more complex formulation to be computationally feasible, but this model is still a heavy simplification of a biological neuron (the computing power at the time was still very limited compared to the resources that are widely available today). This Rosenblatt model is now commonly referred to as the "perceptron", as

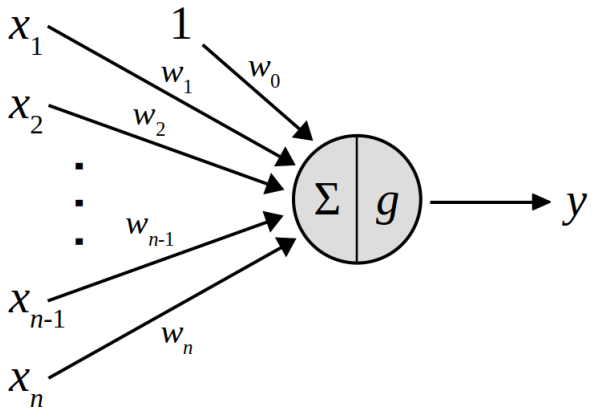

Figure 3.6: A perceptron with n real inputs $x_1, \ldots, x_n$ with weights $w_1, \ldots, w_n$ and an additional constant bias input with weight w_0. The single real output, y, is given by the activation function, g, of the weighted sum of the inputs.

further developments mostly involved enhancements to this model, rather than a fundamentally new model altogether.

In 1969, eleven years after Rosenblatt developed his perceptron model, Minsky and Papert [38] proposed a significant improvement to the perceptron in the form of an activation function. Despite this progress, their book detailed a very pessimistic view of the future potential of neural networks, and instead encouraged a focus on symbolic expert systems. This is often attributed as eventually leading to the "AI winter" of the 1980s [45], which was a period of significantly reduced interest in artificial intelligence research caused by issues with the performance of expert systems in industry.

Formally, a perceptron, figure 3.6, is a computational unit with $n \in \mathbb{N}$ real inputs $x_1, \ldots, x_n \in \mathbb{R}$. In a network, the inputs represent either external inputs or input from synapses of other neurons. These n incoming connections are each associated with a weight, the strength of the connection, $w_1, \ldots, w_n \in \mathbb{R}$.

The dynamics of a perceptron is discrete. All its inputs are considered to be received simultaneously at discrete times associated with firing events in the system. The output $y \in \mathbb{R}$ is determined by an activation function $g : \mathbb{R} \to \mathbb{R}$ where $y = g(z)$ and

$$z = \sum_{i=1}^{n} w_i x_i$$

is the weighted sum of the inputs.

A classical, McCulloch-Pitts perceptron has binary inputs $x_1, \ldots, x_n \in \{0, 1\}$ with no tunable weights, so $w_1 = \ldots = w_n = 1$. This neuron has a hardcoded threshold $\theta \in \mathbb{R}$ (or equivalently, $\theta \in \mathbb{N}$), which it uses to produce a binary output $y \in \{0, 1\}$ by summing its inputs and comparing this result to the threshold value. The output, y, is given by $y = g(z) = H(z - \theta)$, where H is the Heaviside function

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0. \end{cases}$$

The classical perceptron can be used to implement the logical AND function of two binary inputs x_1 and x_2 (which returns 1 if and only if $x_1 = x_2 = 1$), by setting $n = 2$ and $\theta = 2$. It can also implement the logical OR function (which returns 1 if at least one of $x_1 = 1$ or $x_2 = 1$ holds), by setting $n = 2$ and $\theta = 1$. However, this neuron is not able to implement a non-linear function of its inputs, such as the XOR function (there is no value of θ for $n = 2$ that will have $y = 1$ if $x_1 = 1$ and $x_2 = 0$ or $x_1 = 0$ and $x_2 = 1$, and $y = 0$ otherwise). This model does not handle non-binary inputs, has no mechanism to evolve its threshold θ and does not assign different weights to inputs. These are key issues with the classical perceptron that were resolved by the next evolution of this model, Rosenblatt's perceptron.

Rosenblatt's perceptron improved on the classical perceptron by taking real inputs $x_1, \ldots, x_n \in \mathbb{R}$ with tunable weights $w_1, \ldots, w_n \in \mathbb{R}$. The comparison of the weighted input to the threshold value θ is encoded by the inclusion of an additional bias input, $x_0 = 1$ with weight $w_0 = -\theta$, so the weighted sum of inputs becomes

$$Z \;=\; \sum_{i=0}^{n} w_i x_i.$$

However, this perceptron is still restricted to producing a binary output $y \in \{0, 1\}$, where

$$y \;=\; H(Z) \;=\; H\left(\sum_{i=0}^{n} w_i x_i\right).$$

A single instance of this perceptron functions as a binary classifier of real input data if an output of 1 corresponds to a positive classification of the input, and an output of 0 means a negative classification. By modifying the weights $\boldsymbol{w}$ (including the bias weight $w_0 = -\theta$) according to a learning scheme, this single perceptron can be tuned to perform any linearly separable binary classification task, given a suitable learning (weight modification) rate and enough training time.

An example of a simple learning scheme with learning rate $\nu \in (0, 1)$ is to iterate through training samples $\boldsymbol{x}^j$ with corresponding target outputs $d^j \in \{0, 1\}$, $j = 1, \ldots, N$. At each iteration,

- the perceptron's classification y^j is computed; then
- each weight is updated such that $w_i^{j+1} = w_i^j + \nu(d^j - y^j)x_i^j$ for weights $i = 0, \ldots, n$.

The weight updating terminates once all training samples have been correctly classified without a single error / weight adjustment. The Rosenblatt model was a significant improvement on its predecessor, the McCulloch-Pitts perceptron, but it still suffered from not being able to implement a non-linear function of its inputs (such as the XOR function), which prompted the development of further enhancements.

Minsky and Papert's perceptron improved on Rosenblatt's binary thresholded model by introducing a generalised activation function g that permits a real output, so $y = g(Z) \in \mathbb{R}$. This serves to add non-linearity into a network of Minsky and Papert perceptrons, allowing these networks to learn a wider class of tasks (or equivalently, approximate a wider class of functions of the input). Examples of popular activation functions for the perceptron include

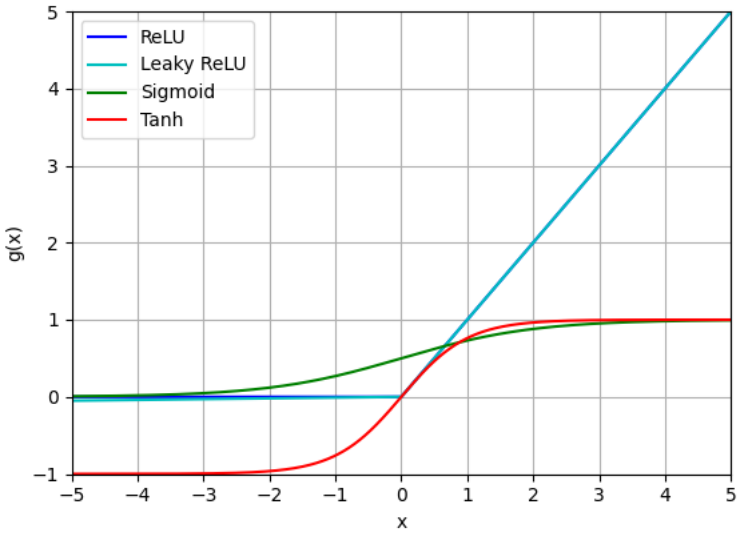

Figure 3.7: A comparison of popular activation functions: the ReLU (dark blue), leaky ReLU (light blue), sigmoid (green) and tanh (red) for inputs $x \in [-5, 5]$. For $x \geq 0$, ReLU and leaky ReLU produce the same output.

- the rectified linear unit (ReLU) function $g(x) = \max\{0, x\}$;
- the leaky ReLU function $g(x) = \max\{0.01x, x\}$;
- the sigmoid $g(x) = (1 + e^{-x})^{-1}$; and
- the tanh function $g(x) = \tanh(x)$.

Figure 3.7 compares these functions for inputs in $[-5, 5]$. All four of these activation functions introduce nonlinearity to the perceptron, but only the sigmoid and tanh functions return output bounded in a finite range. For the sigmoid, $g(x) \in (0, 1)$, and for tanh, $g(x) \in (-1, 1)$.

The Minsky and Papert model is the standard model of a perceptron used in artificial neural networks today, and has not been significantly changed since its introduction more than 50 years ago.

3.2 Artificial neural networks

3.2.1 Traditional neural networks

A traditional neural network is comprised of perceptrons, each of which is termed a node of the network. The architecture, or set of directed connections between the nodes, is usually set such that fixed groups of nodes, or layers, have rich connections between them, but no connections exist between nodes within the same layer [2]. If a node j has incoming connections from nodes $i = 1, \ldots, n$ in the network, then the output of node j is formulated as

$$y_j = g\left(\sum_{i=1}^{n} w_{ij} y_i\right),$$

where w_{ij} is the strength of the directed connection from node i to node j, y_i is the output of node i and g is an activation function. The weight w_{ij} is a parameter

of the network that will be adjusted during training according to some learning paradigm, to allow the network to learn a given task.

In general, artificial networks are organised into ordered layers without internal connections, starting with

- the input layer, which contains special nodes that are directly assigned a value from the real vector of external inputs given to the network (and hence the number of such nodes in this layer of the network must match the dimensionality of the input); the nodes in this layer are then connected to nodes in the next
- "hidden layer(s)" of perceptrons, each of which have incoming connections from the previous layer and outgoing connections to the next layer; and finally
- the output layer, where these neurons have no outgoing connections, but instead, their real, numerical outputs are collected into an "output vector" for the network (where the dimensionality of this vector equals the number of neurons in this layer).

This simple type of network architecture is called a deep, fully-connected, feed-forward network. Figure 3.8 shows an example of such a network with an input layer, one hidden layer and an output layer. Here, the term "deep" refers to the fact that the network has at least one hidden layer, containing nodes that are not directly connected to the input or output of the network. The term "fully-connected" means that between every pair of neighbouring layers in the network, every node in the first layer has an outgoing connection to every node in the second layer. The example network in figure 3.8 has 15 connections between the first two layers and six connections between the last two. The term "feed-forward" refers to the fact that signals only travel "forward" through the network (from left to right in figure 3.8).

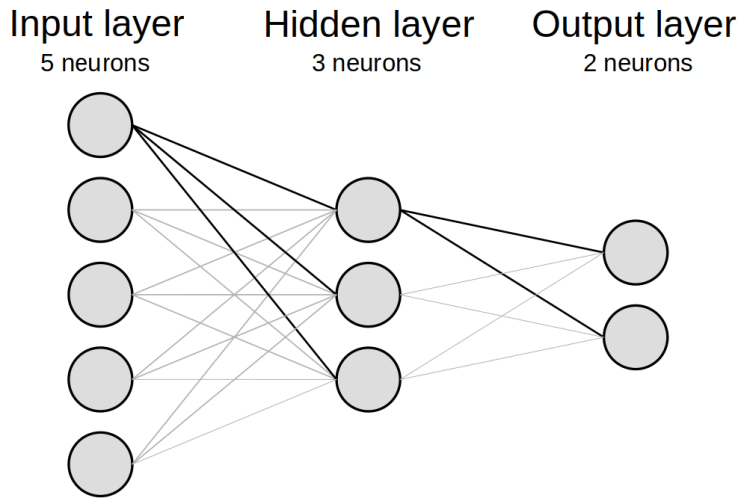

Figure 3.8: An example of a small, fully-connected, feed-forward network with five input nodes, one layer of three hidden perceptrons and two output perceptrons. The fully-connected architecture results in 15 connections between the input and hidden layers, and six connections between the hidden and output layers.

Given a vector of inputs, and tasked with producing an output, an artificial network starts by assigning each numerical value in the input vector to its corresponding node in the input layer. Next, the nodes in the input layer deliver their numerical value via all of their outgoing connections. The perceptrons in the second layer of the network receive this input via all of their incoming connections simultaneously. The outputs of the second layer are then determined – each perceptron computes the weighted sum of its inputs, passes this value to its activation function to produce an output, and delivers this output through each of its outgoing connections. This process continues, layer-by-layer, where the output of every perceptron in a network layer is calculated exactly once, at the same moment as all other nodes in its layer, until the output layer of the network is reached. When this layer receives inputs from the last hidden layer of the network, the output value of each perceptron is combined to form the output vector for the network.

There are many variations of this standard, fully-connected, feed-forward network architecture [46], such as

- residual networks, which are feed-forward networks with added skip connections, where some nodes have outgoing connections to nodes in non-neighbouring (later) layers of the network;
- recurrent networks, where the connection scheme forms a directed graph, and the signal is directed back to a previous network layer from a downstream one, a finite number of times;
- convolutional neural networks, where connections between nodes in some pairs of adjacent layers share weights to implement a convolution of the input, rather than a regular connection scheme where all weights are independent (i.e, there is a linking of the weights for nodes in close proximity). Such networks also often utilise pooling layers, where outputs are combined to drastically reduce the dimensionality of the next layer, usually by taking the maximum or average of local clusters of nodes); and
- autoencoders, which are usually fully connected networks, but where the middle layer has the smallest size and the input and output layers have the same size. The objective is for the output to recreate the input as accurately as possible, therefore producing a compact representation of the input at the middle layer.

3.2.2 Spiking neural networks

A spiking neural network is an artificial network where the nodes are spiking neuron models such as a leaky integrate-and-fire neuron, and hence these networks more closely resemble biological neural networks than their perceptron-based counterparts [47]. These networks are also usually organised into some architecture, but the time-flexibility of spiking neurons means that layers will not necessarily process their outputs all at once, and hence recurrent connections are more commonly used in spiking networks than those comprised of perceptrons.

Just like traditional artificial networks, spiking networks also contain a layer of input neurons which function differently to the other neurons in the network. Instead of directly receiving an activation value from the vector of inputs, spiking input neurons receive either a firing rate or list of precise spike times, depending on the type of input encoding used (either a rate coding or temporal encoding scheme)

[48]. To process this input stimulus, the network is simulated, usually by iterating through very small, discrete time steps, and numerically integrating the network variables between each step. During this simulation period, the network is being continuously stimulated via its input neurons spiking according to their firing rate or set spike times.

A simple example of a rate coding scheme, given a vector of inputs where each entry is assigned a unique input neuron in the network, is to convert each numeric input into a firing rate for its neuron by rescaling the input into a set range of frequencies, such as 0-10Hz. Given a firing rate r, the input neuron could then spike every r^{-1} units of simulated time. Alternatively, a simple example of a temporal coding scheme is to rescale the vector of inputs into values in the range $[0, 1]$, which are interpreted as offsets from which the corresponding input neurons are set to fire at constant-length intervals, such as every second (so a rescaled value of 0.3 produces spike times of 0.3, 1.3 and 2.3 seconds, and so on).

Connections in a spiking neural network are activated whenever an action potential is generated in the pre-synaptic neuron. These synapses send a signal proportional to the weight of the connection. The precise manner in which this signal modifies the post-synaptic membrane potential depends on the type of spiking neuron model used, but the simplest scheme is to instantaneously increase the membrane potential by an amount proportional to the synaptic weight. Neurons in a spiking network emit an action potential whenever their membrane potential difference crosses their spiking threshold, at real-valued times. This is very different to perceptron neural networks, where in the feed-forward case, firing events for a node occur at the same time as all other nodes in the same layer.

The asynchronous firing of nodes in spiking networks means that processing the output of the network is a more complicated procedure than in traditional networks, because any neurons labelled as "output neurons" in the spiking network can fire any number of times while the network is being simulated. A standard procedure is to record the spike times of all output neurons for the entire duration of the simulation, and then post-process the results (for example, each neuron submits a "vote" for a prescribed network output at each spike, and the final result is decided via a majority vote).

3.3 Learning in neural networks

3.3.1 Classification tasks

In machine learning, a classification task is a type of supervised learning problem where the input data is some vector $\boldsymbol{x}^i \in \mathbb{R}^n$ for samples $i = 1, \ldots, N$ and the inputs are said to have $n \in \mathbb{N}$ "features", which is the size of the input vectors. Each sample comes with a label identifying the class to which it belongs, usually represented as an integer $y^i \in \{0, \ldots, L\}$, where $L \in \mathbb{N}$ is the number of classes present in the dataset. The learning algorithm for this classification problem needs to adjust its internal parameters (for a neural network, these are the connection weights and possibly biases) such that the system can receive any sample in the dataset as input, process it, and produce a correct output classification (which may involve interpreting or post-processing its output signal) [2]. For example, each

sample may correspond to an animal, the input features to different measurements of the creature, and the output label to its species.

Humans show great proficiency in classifying things they observe, which suggests that the biological brain is adept at learning classification tasks using its implementation of a spiking neural network. An example of this is the human visual system, discussed in section 2.3, where the firing rates of neurons in the primary visual cortex are believed to encode spatial information from visual input, which can be used to classify objects in the field of view. Other brain regions known to be involved in the categorical representation of information include the prefrontal cortex, parietal cortex, motor cortex and the hippocampus [49]. The human brain's ability to learn categorical representations of information is likely key to our higher cognition, which separates us from other species. This is evidenced by the fact that some of these brain regions (in particular, the prefrontal cortex) implementing classification are much larger in the human brain than in the brains of other closely-related species.

To apply a traditional, perceptron-based neural network to learning a classification task, the input layer of this network should contain one node for each input feature, which receives the numerical value of that feature for any given sample. The output layer of the network should have one neuron per output class in the dataset. The standard procedure is to normalise the output vector from the network, and interpret each entry as representing the likelihood of the input sample belonging to the corresponding class. The predicted class of the network is therefore the class with the largest entry in this output vector.

A similar approach can be used in artificial spiking networks undertaking classification tasks. The input layer of neurons should also have a 1-1 correspondence with the input features, where each of these neurons is given a firing rate or sequence of spike times via preprocessing of the input vector. The output layer of the network may also have one neuron per class in the dataset, where the network's prediction after processing an input sample could be the class that corresponds to the output neuron which fired the most times during the simulation period.

When an artificial network is learning a classification task, the dataset is usually split into training data and testing data (with the proportion being a hyperparameter of the learning process, commonly 80% training data and 20% testing data). The training data (both inputs $\boldsymbol{x}^i$ and corresponding correct outputs y^i) is used to "train" the network using a learning scheme to tune its connection weights, and the testing data is then used to evaluate its performance on unseen samples.

There are two broad regimes that can be used for learning a classification task: supervised or unsupervised learning. A supervised learning scheme uses the correct outputs y^i to compute a "loss function" from the network's output for each sample, which quantifies the error in its predictions and is used to modify the network weights to attempt to reduce this error. An unsupervised learning scheme instead modifies the network weights without reference to the correct output, by simply processing the training inputs $\boldsymbol{x}^i$ and tuning its weights in response to its internal dynamics.

The standard learning paradigm for traditional, perceptron-based neural networks is backpropagation with stochastic gradient descent. In this scheme, the gradient of a given loss function is computed with respect to its connection weights and biases [50], and then the stochastic gradient descent algorithm is used to iterate

over batches of the training data and update the weight values in the opposite direction to the gradient, that is the direction of steepest descent of the loss function. The backpropagation algorithm is reviewed in detail below.

The standard learning paradigm for artificial spiking networks is an unsupervised learning rule of spike timing dependent plasticity while processing training samples, which tunes synapse weights in the network using a function of the spike times of pre and post synaptic neurons [48], as described in more detail below.

After an artificial network has been trained for a classification task, it is evaluated on the test data. This involves fixing the network parameters, then inputting each test sample to the network and computing the network's output, which is its predicted class for the sample. A standard metric such as accuracy, which is the number of correct predictions divided by the total number of test samples, can then be computed using comparisons with the true labels of the test dataset, in order to compute a "score" measuring how well the network has learned the classification task.

3.3.2 Backpropagation

Backpropagation is an efficient algorithm for computing the gradient of a loss function with respect to the weights and biases of a traditional neural network. This gradient can then be used in an optimisation procedure such as gradient descent, to train the network to learn a given task.

This algorithm was formally introduced in the context of training neural networks in a 1986 paper by Rumelhart, Hinton and Williams [51], where it was demonstrated to provide significant performance benefits compared to earlier approaches that were used at the time. This algorithm is still used today to compute gradients for supervised learning in neural networks.

To demonstrate the algorithm, consider a classification task in which each input sample $\boldsymbol{x} \in \mathbb{R}^{n_1}$ is associated with an output class $m = 1, \ldots, n_L$. The output class m can be represented as a one-hot vector $\boldsymbol{y} \in \{0, 1\}^{n_L}$ where $y_m = 1$ and $y_i = 0$ for all $i \neq m$, $i = 1, \ldots, n_L$. The dataset contains M pairs of inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$.

A fully-connected, feed-forward network with $L \in \mathbb{N}$ layers is to be used to classify this data. Each layer $l \in \{1, \ldots, L\}$ consists of $n_l \in \mathbb{N}$ nodes – specifically, Rosenblatt perceptrons with Minsky-Papert activation functions for layers $l = 2, \ldots, L$, and external inputs for $l = 1$. All nodes in layer l connect to each node in the next layer $l + 1$ (unless $l = L$, where this is the output layer with no forward connections). This network architecture is shown in figure 3.9.

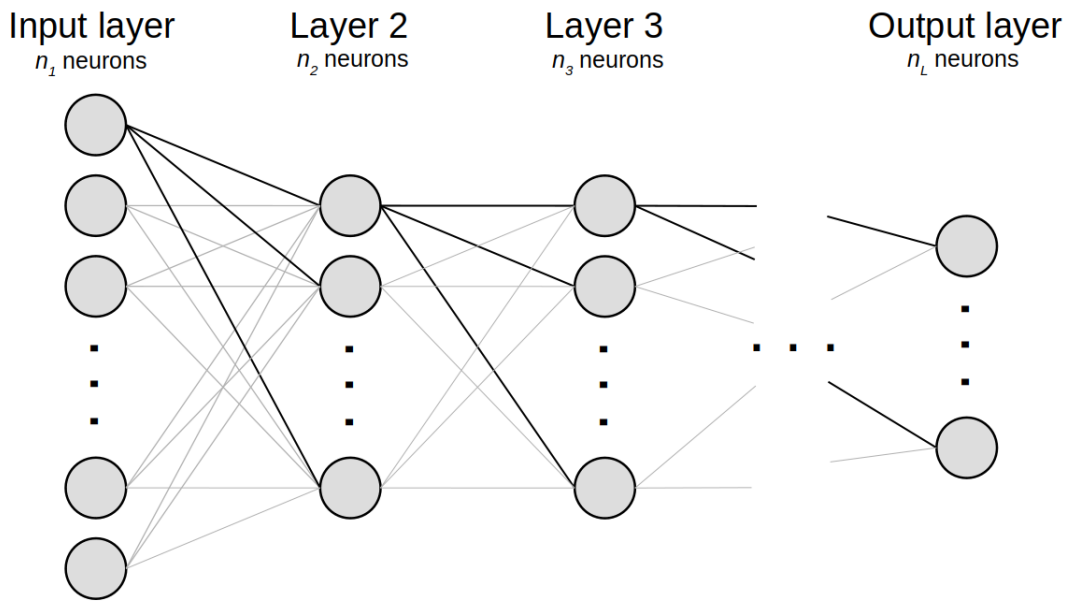

Figure 3.9: A fully-connected, feed-forward network with L layers $l = 1, \ldots, L$, each comprising $n_l \in \mathbb{N}$ nodes. This architecture is appropriate for a classification task where the inputs $\boldsymbol{x}$ have n_1 features, each associated with one of n_L classes.

Assume for simplicity that every neuron in the network uses the same activation function g, so the output/activation a_j^l of neuron j in layer l is given by [50]

$$a_j^l \;=\; g(z_j^l)$$

where

$$z_j^l \;=\; \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1} + b_j^l \tag{3.3.1}$$

and

- $w_{jk}^l \in \mathbb{R}$ is the weight of the connection from the k^{th} neuron in layer $l-1$ to the j^{th} neuron in layer l for $l \in \{2, \ldots, L\}$, $k \in \{1, \ldots, n_{l-1}\}$ and $j \in \{1, \ldots, n_l\}$; and
- b_j^l is the bias of the j^{th} neuron in layer l for $l \in \{2, \ldots, L\}$ and $j \in \{1, \ldots, n_l\}$.

The activation for layer l is therefore

$$\boldsymbol{a}^l \;=\; \boldsymbol{g}(\boldsymbol{z}^l) \;\in\; \mathbb{R}^{n_l}$$

where the function $\boldsymbol{g} : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ is the element-wise activation where each component $g_j(\boldsymbol{z}^l) = g(z_j^l)$ for $j = 1, \ldots, n_l$. The weighted input to the activation function at layer l is the vector

$$\boldsymbol{z}^l \;=\; W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \in \mathbb{R}^{n_l},$$

where

- $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ is the matrix of input weights w_{jk}^l to the l^{th} layer for $j = 1, \ldots, n_l$, $k = 1, \ldots, n_{l-1}$; and

- $\boldsymbol{b}^l \in \mathbb{R}^{n_l}$ is the vector of biases $b^l_j, j = 1, \ldots, n_l$ for the l^{th} layer.

In order to assess the performance of the network in classifying the data, a loss or cost function $C : \mathbb{R}^{n_L} \to \mathbb{R}_{\geq 0}$ is defined, which measures the inaccuracy of the network processing the input data $\boldsymbol{x} \in \mathbb{R}^{n_1}$ given its output $\boldsymbol{a}^L$ and the associated correct output $\boldsymbol{y} \in \mathbb{R}^{n_L}$. The scalar "loss" output of the cost function is zero if $\boldsymbol{a}^L = \boldsymbol{y}$ (the network returned the correct output) and increases when $\boldsymbol{a}^L$ is more dissimilar to $\boldsymbol{y}$ (the output gets more "incorrect").

One example of a popular loss function for a classification task with n_L classes is cross-entropy error

$$C(\boldsymbol{a}^L) \;=\; -\sum_{j=1}^{n_L} y_j \log_2(a^L_j).$$

In the case of classifying the output class, m, this simplifies to

$$C(\boldsymbol{a}^L) \;=\; -\log_2(a^L_m) \tag{3.3.2}$$

as $y_i = 0$ for all $i \neq m$ and $y_m = 1$. Note that because this loss is being evaluated given a fixed sample $(\boldsymbol{x}, \boldsymbol{y})$, the y_j can be considered as parameters of this function, rather than additional inputs.

Given a fixed input $\boldsymbol{x}$ with target output $\boldsymbol{y}$, the goal of backpropagation is to find the gradient of the loss function with respect to the parameters of the network – specifically the weights w^l_{jk} and the biases b^l_j – which can then be used to minimise the loss function. This requires computing the partial derivatives

$$\frac{\partial C}{\partial w^l_{jk}} \;\text{ and }\; \frac{\partial C}{\partial b^l_j}.$$

To achieve this, define another intermediate quantity δ^l_j, which is termed the error for the j^{th} neuron of the l^{th} layer of the network, $l \in \{1, \ldots, L\}$ and $j \in \{1, \ldots, n_l\}$, and is defined as

$$\delta^l_j \;=\; \frac{\partial C}{\partial z^l_j}.$$

This is the change in the final cost caused by change to the weighted sum input z^l_j to this neuron. First, consider the output layer error

$$\begin{aligned}
\delta^L_j \;&=\; \frac{\partial C}{\partial z^L_j} \\
&=\; \sum_{k=1}^{n_L} \frac{\partial C}{\partial a^L_k}\frac{\partial a^L_k}{\partial z^L_j} \quad \text{by the chain rule} \\
&=\; \frac{\partial C}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j} \quad \text{as } \frac{\partial a^L_k}{\partial z^L_j} = 0 \text{ when } j \neq k \\
&=\; \frac{\partial C}{\partial a^L_j}\frac{dg}{dz^L_j}.
\end{aligned}$$

The first factor is a simple partial derivative of the cost function that is easy to compute. For example, for cross-entropy loss as given by equation 3.3.2,

$$\frac{\partial C}{\partial a_j^L} \;=\; -\frac{y_j}{\ln(2) a_j^L}.$$

The second factor is the derivative of the activation function g at z_j^L, which, for the usual activation functions outlined in section 3.1.3, is easy to compute. In vector form, this equation looks like

$$\boldsymbol{\delta}^L \;=\; \nabla_{\boldsymbol{a}^L} C \odot \boldsymbol{g'}(\boldsymbol{z}^L),$$

where $\boldsymbol{\delta}^L$ is the vector of errors in layer L and $\odot$ is the Hadamard element-wise product of vectors.

The backpropagation algorithm gets its name from the process by which the error $\boldsymbol{\delta}^l$ in layer l is determined from the error $\boldsymbol{\delta}^{l+1}$ in layer $l+1$, in the opposite direction to the determination of the network's output given its input $\boldsymbol{x}$. To backpropagate through the network to calculate errors, consider the error at layer l and neuron j,

$$\begin{aligned}
\delta_j^l \;&=\; \frac{\partial C}{\partial z_j^l} \\
&=\; \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \quad \text{by the chain rule} \\
&=\; \sum_{k=1}^{n_{l+1}} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}.
\end{aligned} \tag{3.3.3}$$

From equation 3.3.1,

$$\begin{aligned}
z_k^{l+1} \;&=\; \sum_{j=1}^{n_l} w_{kj}^{l+1} a_j^l + b_k^{l+1} \quad \text{for } k = 1, \ldots, n_{l+1} \\
&=\; \sum_{j=1}^{n_l} w_{kj}^{l+1} g(z_j^l) + b_k^{l+1}, \\
\text{so} \quad \frac{\partial z_k^{l+1}}{\partial z_j^l} \;&=\; w_{kj}^{l+1} \frac{dg}{dz_j^l}.
\end{aligned}$$

Substituting this expression into equation 3.3.3 gives

$$\delta_j^l \;=\; \sum_{k=1}^{n_{l+1}} w_{kj}^{l+1} \delta_k^{l+1} \frac{dg}{dz_j^l}.$$

In vector form, the backpropagation error for layer l is

$$\boldsymbol{\delta}^l \;=\; \left(\left(W^{l+1}\right)^T \boldsymbol{\delta}^{l+1} \right) \odot \boldsymbol{g'}(\boldsymbol{z}^l).$$

Thus, the algorithm takes the error $\boldsymbol{\delta}^{l+1}$ at layer $l+1$ and multiplies it by the transpose weight matrix $\left(W^{l+1}\right)^T$ to propagate the error back to the output of layer l. It then multiplies element-wise by $\boldsymbol{g'}(\boldsymbol{z}^l)$ to further propagate the error back through the activation function to the weighted input $\boldsymbol{z}^l$ of layer l.

Now consider the rate of change of the loss function C with respect to the bias b^l_j of neuron j in layer l:

$$\begin{aligned}\frac{\partial C}{\partial b^l_j} &= \frac{\partial C}{\partial z^l_j}\frac{\partial z^l_j}{\partial b^l_j} \quad \text{by the chain rule} \\ &= \delta^l_j \frac{\partial}{\partial b^l_j}\left(\sum_{k=1}^{n_{l-1}} w^l_{jk} a^{l-1}_k + b^l_j\right) \\ &= \delta^l_j.\end{aligned}$$

Similarly, an equation for the rate of change of C with respect to the weight w^l_{jk} is

$$\begin{aligned}\frac{\partial C}{\partial w^l_{jk}} &= \frac{\partial C}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}} \quad \text{by the chain rule} \\ &= \delta^l_j \frac{\partial}{\partial w^l_{jk}}\left(\sum_{k=1}^{n_{l-1}} w^l_{jk} a^{l-1}_k + b^l_j\right) \\ &= \delta^l_j a^{l-1}_k.\end{aligned}$$

Thus, the backpropagation algorithm can be defined as follows. Given a fixed input $\boldsymbol{x}$ with target output $\boldsymbol{y}$, compute the gradient of the loss function by the following steps:

1. input: feed the input $\boldsymbol{x}$ to the input layer of the network to set $\boldsymbol{a}^1 = \boldsymbol{x}$;
2. feedforward: for each layer $l = 2, 3, \ldots, L$, evaluate the weighted input $\boldsymbol{z}^l = W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$ and then the activation $\boldsymbol{a}^l = \boldsymbol{g}(\boldsymbol{z}^l)$;
3. output layer errors: compute $\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L} C \odot \boldsymbol{g'}(\boldsymbol{z}^L)$ using the target output $\boldsymbol{y}$;
4. backpropagate: for $l = L-1, \ldots, 2$, evaluate the errors in layer l as $\boldsymbol{\delta}^l = \left(\left(W^{l+1}\right)^T \boldsymbol{\delta}^{l+1}\right) \odot \boldsymbol{g'}(\boldsymbol{z}^l)$; then
5. gradient: evaluate each of the partial derivatives $\dfrac{\partial C}{\partial b^l_j} = \delta^l_j$ and $\dfrac{\partial C}{\partial w^l_{jk}} = \delta^l_j a^{l-1}_k$ to find the gradient of the loss function with respect to the tunable parameters of the network.

Following backpropagation, an optimisation scheme such as stochastic gradient descent can utilise the computed gradients to reduce the loss by adjusting the weights w^l_{jk} and biases b^l_j of the network, as outlined in the following section.

3.3.3 Stochastic gradient descent

Stochastic gradient descent is an iterative algorithm for optimising an objective function: here, to minimise the loss function with respect to the weight and bias parameters of a neural network [50]. The term stochastic refers to how this method makes a stochastic approximation of the actual gradient with respect to the training dataset as a whole, using the gradient calculated from a random subset of the

training data. Gradient descent refers to how the method iteratively shifts the parameter values in the opposite direction of the gradient, which is the direction of steepest descent in the loss function. Backpropagation is used to calculate the loss gradient for each random batch of training samples.

To train a network using stochastic gradient descent, the weights and biases of the network are typically initialised as small random values. Then, given a random batch of $m \in \mathbb{N}$ training samples $(\boldsymbol{x}^i, \boldsymbol{y}^i)$ for $i = 1, \ldots, m$ and a learning rate $\nu \in \mathbb{R}_{>0}$, the algorithm is:

1. for each training sample $(\boldsymbol{x}^i, \boldsymbol{y}^i)$ in the batch, run the feedforward and backpropagate steps of the backpropagation algorithm to evaluate the partial derivatives $\left(\frac{\partial C}{\partial b_j^l}\right)^i$ and $\left(\frac{\partial C}{\partial w_{jk}^l}\right)^i$ for layers $l \in \{2, \ldots, L\}$ and nodes $k \in \{1, \ldots, n_{l-1}\}$ and $j \in \{1, \ldots, n_l\}$;
2. compute the average gradients over the batch

$$\overline{\left(\frac{\partial C}{\partial b_j^l}\right)} = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{\partial C}{\partial b_j^l}\right)^i \quad \text{and} \quad \overline{\left(\frac{\partial C}{\partial w_{jk}^l}\right)} = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{\partial C}{\partial w_{jk}^l}\right)^i;$$

3. update the network parameters in the direction of steepest descent, proportional to the learning rate ν:

$$b_j^l \leftarrow b_j^l - \nu \overline{\left(\frac{\partial C}{\partial b_j^l}\right)}$$

and

$$w_{jk}^l \leftarrow w_{jk}^l - \nu \overline{\left(\frac{\partial C}{\partial w_{jk}^l}\right)}.$$

This process is repeated for different, randomly-sampled batches of m samples from the training set until a termination condition is reached, such as

- the magnitude of the gradient vector ∇C is below some threshold; and/or
- a preset number of epochs of training have been completed, where one epoch consists of batches whose disjoint union is the set of all training samples (e.g., for 100 training samples and a batch size of $m = 20$, one epoch would consist of 5 batches, where each epoch has its 5 batches randomly drawn from the training set without replacement); and/or
- between epochs of training, the network is evaluated on a test dataset to compute the loss C, and training is stopped when C no longer decreases between these tests.

3.3.4 Spike timing dependent plasticity

While traditional, perceptron-based neural networks typically learn using the supervised scheme of stochastic gradient descent (with backpropagation used to compute the gradient of the loss function), artificial spiking networks are typically trained by the unsupervised learning rule of spike timing dependent plasticity [48]. Here, "plasticity" refers to tuning the weights of synapses between neurons, using the

spike times of the pre and post synaptic neurons to determine the change in the weight. Spike timing dependent plasticity modifies the network connectivity in the following manner.

Consider a pair of neurons connected by a single synapse, which may form part of a larger network. Let w be the weight of this connection between the pre-synaptic neuron i and post-synaptic neuron j. The spike timing dependent plasticity rule [52] uses the recorded spike times t_i^m for $m = 1, 2, \ldots, M$ for neuron i and t_j^n for $n = 1, 2, \ldots, N$ for neuron j, during a given period, to compute the weight adjustment Δw. In general,

$$\Delta w \;=\; \sum_{m=1}^{M} \sum_{n=1}^{N} f(t_j^n - t_i^m) \tag{3.3.4}$$

where f is any function that satisfies

- $f(0) = 0$;
- $f(t) > 0$ when $t > 0$ and f is a decreasing function in this range, such that $f(t) \to 0$ as $t \to \infty$; and
- $f(t) < 0$ when $t < 0$ and f is also a decreasing function in this range with $f(t) \to 0$ from below as $t \to -\infty$.

The motivation for this updating scheme is that, for a pair of spike times t_i^m and t_j^n that are in quick succession with the pre-synaptic neuron firing first, the synapse weight w will increase, as this timing suggests that neuron i was important for triggering a spike in neuron j. On the other hand, if t_i^m and t_j^n are in quick succession but the post-synaptic neuron j fires first, the synapse weight w will decrease, because this timing suggests that the firing events were uncorrelated. Spikes that occur at equal times, so $t_i^m = t_j^n$, are considered to be non-causal, and so no adjustment to the synapse weight is made. If the difference between t_i^m and t_j^n is large, these spikes are not connected and so will not contribute a significant change to the weight w. Therefore, a close approximation of the rule 3.3.4 is to compute the weight adjustment Δw upon each spike of the post-synaptic neuron, using its spike time t_j, and only the most recent spike time t_i of the pre-synaptic neuron, so

$$\Delta w \;=\; f(t_j - t_i). \tag{3.3.5}$$

A simple example of a discontinuous weight adjustment function f is

$$f(t) \;=\; \begin{cases} Ae^{-t/\tau_A} & \text{for } t > 0 \\ Be^{t/\tau_B} & \text{for } t < 0 \\ 0 & \text{for } t = 0, \end{cases} \tag{3.3.6}$$

where $A > 0$ and $B < 0$. These coefficients may depend on the current synaptic weight w, whilst the time constants τ_A and τ_B are positive values of the same order of magnitude as the neuron membrane potential (in a biologically realistic neuron, this would be on the order of 10ms). This function is illustrated in figure 3.10.

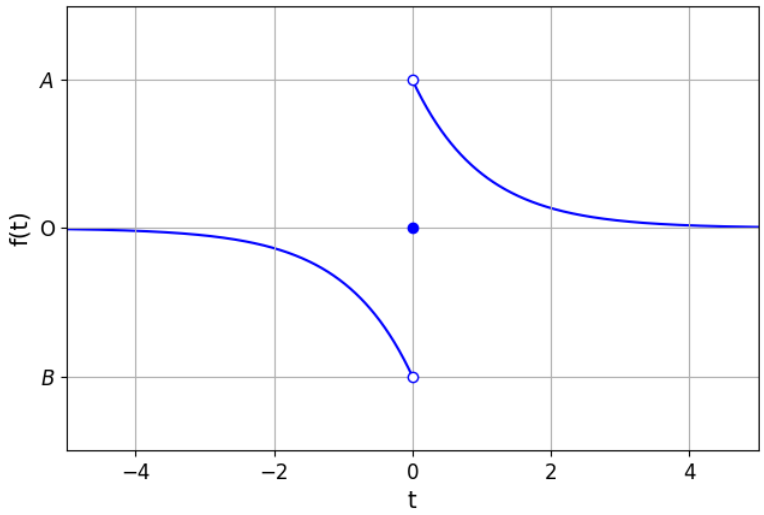

Figure 3.10: A plot of the piecewise weight adjustment function 3.3.6 on $t \in [-4, 4]$, where the $t < 0$ branch corresponds to reducing the synapse weight and the $t > 0$ branch corresponds to increasing it.

To train a spiking neural network with a spike timing dependent plasticity learning rule, given $m \in \mathbb{N}$ training samples $(\boldsymbol{x}^i, y^i)$ for $i = 1, \ldots, m$, the following scheme is employed:

1. convert each input sample $\boldsymbol{x}^i$ into a vector of spike times for each input neuron in the network (using either a rate or temporal coding scheme);
2. for each training sample, simulate the network for some duration of time, where stimulation to the network is delivered via the input neurons;
3. at each instance of a non-input spike in the network, apply the spike timing dependent plasticity rule to all incoming synapses to the neuron that has fired, according to
$$w \;\to\; w + f(t_j - t_i),$$
where j is the post-synaptic neuron that spiked and i is the pre-synaptic neuron;
4. also during the simulation period for each training sample, record the spike times $t_i^1, \ldots, t_i^s$ for every output neuron in the network;
5. once the training samples have been exhausted, process the spike times using the correct classification y^i for each sample to assign a label/class to each output neuron (the method for which depends on the task).

A simple supervised labelling scheme is to assign each output neuron in the network to the class that caused it to spike most frequently during training. Given these output labels, when the network processes a test sample, the classification can be computed as the class whose associated output neurons spiked the highest number of times during the simulation period. An example of classification using a spiking neural network is explored in section 3.4.2.

3.4 Classification using spiking neural networks

A survey of spiking neural networks applied to classification tasks finds that most successful approaches explored in the last decade have all relied on mechanisms that are not biologically plausible [40].

A popular system is to train a traditional neural network on the task using backpropagation and stochastic gradient descent, and then convert it into a spiking network [53]. In this method, perceptrons are directly converted into spiking neurons (often of the leaky integrate-and-fire variety), and the weighted connections between them are copied over to synapses between the spiking neurons. The weights usually require rescaling because of the different parameters of the spiking network, such as the time constant of membrane potential. The conversion approximately maps the activations (final outputs) of the original perceptrons into the firing rates of the spiking neurons, and as such, this conversion method is a type of rate-based learning. An extension of this method involves defining constraints on the perceptron network during training that force the learned weights to be directly applicable to the downstream spiking network (such that no rescaling is required) [54]. These approaches are clearly not biologically plausible, as the brain uses (likely unsupervised) learning mechanisms that operate directly on its spiking neurons, and no conversion is required.

Another popular approach is to use a modified spike timing dependent plasticity learning rule that implements supervised learning for a spiking network containing special "teacher neurons" that provide feedback through connections to the other learning neurons in the network. The most notable algorithm that follows this approach is the ReSuMe "remote supervised method" [48]. When the network produces an output spike sequence for a training sample, this output is compared with the desired spike sequence, and the teacher neurons send a teaching signal generated from this comparison to fine tune the synaptic weights of the network (to reduce the output error). This approach is not biologically plausible in general, as the human brain has demonstrated the ability to learn in unsupervised environments.

A further approach is to define a modified backpropagation algorithm that can operate on the spike times of neurons, to allow a regular supervised learning scheme such as stochastic gradient descent to be directly applied to a spiking network. A prominent example of this is the SpikeProp algorithm [55], first developed in the early 2000s, which applies backpropagation to a proxy for the first spike times of the output neurons in a network, that is real-valued and almost-everywhere differentiable. Another approach involved applying stochastic gradient descent to the membrane potentials of spiking neurons, by filtering out any discontinuities arising from direct synaptic input [56]. Again, this approach is not biologically plausible because it implements a supervised learning scheme.

The focus on biologically plausible spiking networks follows from the objective of this research: to investigate the possible advantages of the biological brain's processing method over traditional perceptron networks. This investigation could help answer the question of why the biological brain implements such a complex spiking network instead of a network comprised of perceptron-like nodes.

3.4.1 MNIST digit recognition task

The MNIST (modified National Institute of Standards and Technology) dataset is a very popular classification dataset in the field of computer vision [57]. It is commonly used to benchmark the performance of new machine learning models and network architectures before moving onto larger or more complex real-world learning problems. Hence, this task serves as a suitable entry-level exercise for this project, to compare the performance and suitability of traditional and spiking neural networks.

This dataset consists of 28×28 grayscale images of handwritten digits [57], with 60,000 training images and 10,000 test images that were derived from the original NIST database of binary (black and white) images of handwritten digits and characters. Each digit is labelled with its true class between 0 and 9. The original images were size-normalised to fit into a 20×20 square, anti-aliased to smooth the result and introduce gray levels between black and white, and centered by translating each digit so that its center of mass is in the middle of the image. Some examples of the images in the dataset are shown in figure 3.11.

The dataset was created to improve upon the NIST database that was produced by the US government agency of the same name, which contained over 800,000 128×128 binary images of handwritten digits and characters that were segmented from handwriting sample forms given to a variety of US citizens [58]. Both the training and test sets contain an equal split of images derived from the handwriting samples of adults and teenagers, to ensure a variety of writing styles are featured. A feature of this dataset is that the set of writers who produced the training digits is disjoint from the set of writers who produced the test digits, meaning a successful classifier is learning to predict patterns in the digits rather than patterns in the handwriting of specific individuals.

In the MNIST digit recognition task, a classifier is trained on the digit images with their corresponding class labels from the training set (often with data augmentation using skewing, slanting, shifting or blurring as a preprocessing step to improve the robustness of the training), and is then evaluated by its classification accuracy on the test dataset, to quantify how well it has learned the task.

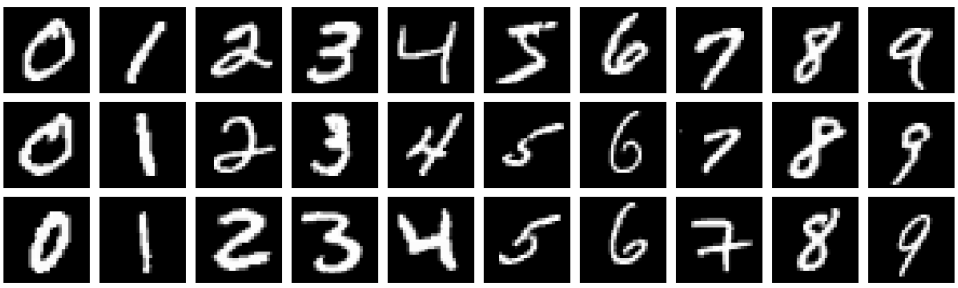

Figure 3.11: A sample of 28×28 images from the MNIST dataset.

3.4.2 Diehl & Cook's leaky integrate-and-fire network

A biologically plausible spiking neural network was implemented for the MNIST classification task by Peter Diehl and Matthew Cook in 2015 [11]. This classifier used a variant of the leaky integrate-and-fire neuron with conductance-based synapses, and was trained by the unsupervised spike timing dependent plasticity learning rule. Every neuron was deemed to be either excitatory or inhibitory, which defined the type of outgoing synapses of the neuron (an excitatory neuron's outgoing connections have an excitatory effect on the post-synaptic neuron's membrane potential). The neural unit is taken to be a leaky integrate-and-fire model, described in section 3.1.2, where the membrane potential u behaves according to the linear, first order differential equation

$$\tau_u \frac{du}{dt} = (E_{\text{rest}} - u) + g_e(E_{\text{exc}} - u) + g_i(E_{\text{inh}} - u).$$

In this model, E_{rest} is the resting membrane potential and τ_u is the time constant for membrane voltage (which the authors varied for excitatory and inhibitory neurons in their network). Synaptic inputs are modelled as input currents through conductances g_e and g_i which connect to incoming excitatory and inhibitory synapses respectively (where a synapse is labelled according to the type of pre-synaptic neuron). The equilibrium potentials of excitatory and inhibitory input are E_{exc} and E_{inh} respectively.

This model, like the standard leaky integrate-and-fire neuron, has a spiking threshold E_{thresh}, where a neuron spikes if its membrane potential reaches this level from below. After it spikes, a neuron's voltage is instantaneously set to E_{reset}, the reset potential, and it begins a refractory period of duration r seconds, within which it cannot spike again.

The synaptic conductances are incremented by a positive synapse weight w at the onset of a pre-synaptic spike from the corresponding type of neuron. In the absence of any input spikes, they decay exponentially according to

$$\tau_{g_e} \frac{dg_e}{dt} = -g_e \text{ and } \tau_{g_i} \frac{dg_i}{dt} = -g_i,$$

where τ_{g_e} is the time constant of excitatory conductance decay and τ_{g_i} is the time constant of inhibitory conductance decay.

The network architecture, figure 3.12, consisted of three layers, and depended on a parameter N, which dictated the size of the square non-input layers. The authors experimented with different sizes N from 100 to 6,400. The model had an input layer of $28 \times 28 = 784$ neurons, each of which corresponded to a unique pixel in the MNIST input images, and were fully-connected to an excitatory layer of N neurons via excitatory synapses. The neurons in the excitatory layer encoded the classification output of the system using class labels assigned upon the completion of training. The excitatory layer was 1:1 connected to an inhibitory layer with N neurons, which had $N - 1$ outgoing synapses to all but their corresponding neurons in the excitatory layer.

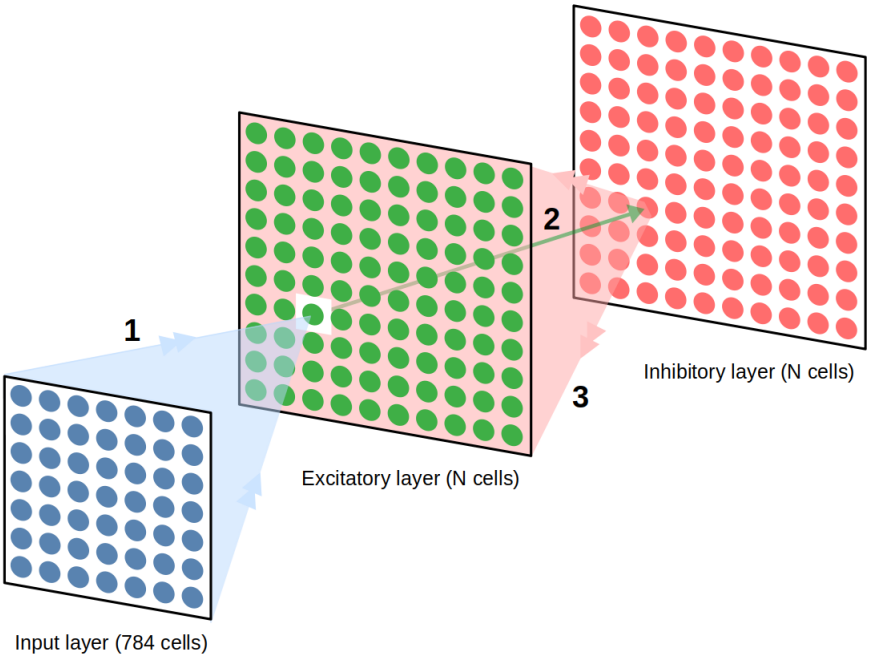

Figure 3.12: Diehl and Cook's network architecture (recreated from [11]). This network consists of an input layer of 784 neurons that is fully connected to an excitatory layer of N neurons, which is 1:1 connected to an inhibitory layer, also containing N neurons. There are feedback connections from each neuron in the inhibitory layer to all but the corresponding neuron in the excitatory layer. The spike times of the excitatory neurons are used to produce the network's output.

A power-law spike timing dependent plasticity rule was used to learn the synapse weights during training, where changes to the weight w of a synapse are calculated using a pre-synaptic trace variable x_{pre} for that synapse, which is a leaky, short-memory count of the number of spikes that have passed through the synapse. Whenever a pre-synaptic spike arrives at the synapse, x_{pre} is increased by 1, otherwise, it decays exponentially according to

$$\tau_{x_{\text{pre}}} \frac{dx_{\text{pre}}}{dt} = -x_{\text{pre}},$$

where $\tau_{x_{\text{pre}}}$ is the time constant of the trace. Whenever the post-synaptic neuron spikes, the synapse adjusts its weight w according to

$$w \leftarrow w + \Delta w,$$

where

$$\Delta w = \nu(x_{\text{pre}} - x_{\text{tar}})(w_{\max} - w)^{\mu}.$$

In this rule,

- $\nu > 0$ is the learning rate;
- $x_{\text{tar}} \geq 1$ is the target value of the pre-synaptic trace that would classify the synapse as "important" at the time of a post-synaptic spike, which serves to

further silence synapses that rarely trigger a post-synaptic spike (if $x_{\text{pre}} < x_{\text{tar}}$ then $\Delta w < 0$ for $w < w_{\max}$);
- $w_{\max} > 0$ is the maximum weight; and
- $\mu \in [0, 1]$ is the power that determines the dependence of the update Δw on the previous weight value.

Using this scheme, weight updates are only calculated when the post-synaptic neuron spikes. In the study, it was reported that the network tended to converge to a state where neurons had very low firing rates on average, meaning this system is very efficient for computation, even in large networks.

An additional mechanism that was implemented in the model was intrinsic plasticity, in which spike-induced changes are made to the electrical properties of a neuron (as opposed to synaptic plasticity, where spiking causes the properties of a synapse to change). Specifically, an adaptive spiking threshold $E_{\text{thresh}} + \theta$ was employed for each neuron, where θ was increased by a constant voltage θ_+ whenever the neuron spiked, and otherwise decayed exponentially with time constant τ_θ. Hence, when a neuron spiked, its spiking membrane threshold increased, which made it slightly harder for the neuron to fire again for a short time (until the decay brings its threshold back down to a reasonable level). This limited the firing rates of neurons in the network and also served to encourage all neurons to have approximately the same firing rates (to achieve homogeneity of firing rates within the network).

Each neuron in the input layer corresponds to a unique pixel in the images in the MNIST dataset. To convert the image input into spikes for these neurons, the intensity value $I \in [0, 255] \cap \mathbb{Z}$ of each pixel is divided by 4 to return a firing rate $r = \frac{I}{4}$ Hz $\in [0, 63.75]$ for its corresponding neuron. Then, the input neurons were set to fire as a homogeneous Poisson process over a time interval Δt to stimulate the network, where in this period, the chance of an input neuron spiking n times is

$$\mathbb{P}(n \text{ spikes during } \Delta t) \;=\; e^{-r\Delta t}\frac{(r\Delta t)^n}{n!}.$$

The study set the simulation period to $\Delta t = 350$ms (for processing one input sample). If less than five spikes were recorded from the excitatory neurons in the second layer of the network, the maximum firing rate was increased by 32 Hz, so $r \in [0, 95.75]$ Hz. The input intensities were then rescaled to compute new firing rates in this larger range, and the network was simulated again with this stronger stimulation. This process was repeated until five spikes were recorded in the excitatory layer. During training, the network was simulated for 150ms with no stimulation between different input images, to ensure that all neuron membrane potentials u and other properties (such as g_e, g_i and x_{pre}) decayed to their resting levels before the next input was given.

Once all training samples had been processed through the network, the learning rate ν was set to zero (to fix the values of the synapse weights w) and each neuron's adaptive spiking threshold θ was also fixed. Each neuron in the excitatory layer was labelled by the class whose inputs caused it to have the largest average spiking rate during training. Then, when testing or otherwise using the network to classify

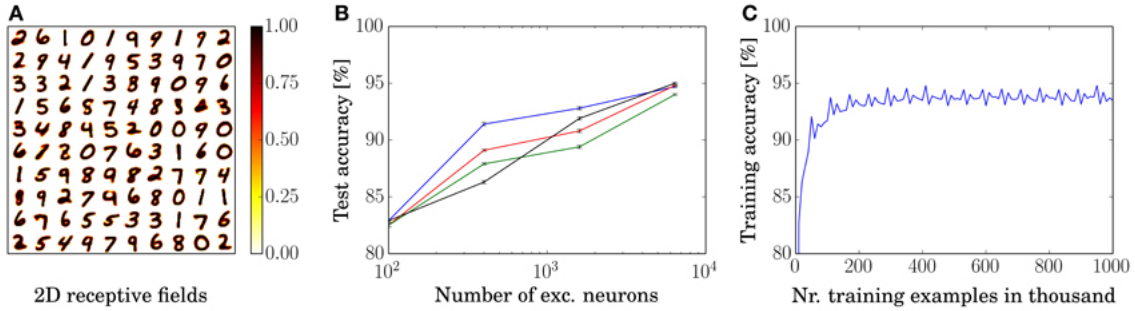

Figure 3.13: Experimental results of spiking network classifications. The left plot visualises the weights of the input to excitatory synapses for a network with $N = 10 \times 10$ excitatory neurons, where the colour and digits show the receptive field of each of these neurons. The middle plot shows the final test accuracy for four different learning rules as a function of the network size N, where the power law rule explained above is shown in red. The right plot shows the training accuracy for a network with $N = 1{,}600$ as a function of the number of training samples, which demonstrates convergence after approximately 200,000 samples. Reproduced from [11].

MNIST images, the same input stimulation process was used, and the output prediction of the network was the class whose assigned neurons had the largest average firing rate.

The network performance was evaluated for several sizes of the excitatory and inhibitory layers: $N = 100$, 400, 1,600 and 6,400 neurons. The accuracy increased with network size, with final test accuracies (the percentage of correct classifications) of 82.9%, 87.0%, 91.9%, and 95.0% respectively, with the larger networks requiring a few hundred thousand training samples to converge to a stable, well-performing state. Some of the results are shown in figure 3.13.

The impressive performance achieved by this biologically plausible spiking neural network, trained using an unsupervised learning scheme, makes a strong case for the claim that spiking networks are a robust basis for artificial intelligence systems. The spiking network model of Diehl and Cook forms the basis of the investigations in this project.

Chapter 4

Study design, methodology and implementation

The use of a spiking neural network for classification outlined in section 3.4.2 forms the basis for the current study. The first aim was to characterise the performance of this type of spiking neural network classifier and compare it to that of a traditional, perceptron-based neural network. The next aim of the investigation was to explore the effects of modifications to the network architecture in the form of neurogenesis on the efficacy of the spiking network. The model of the spiking neural network was that outlined in the previous section 3.4.2, and that of a comparator traditional network, are outlined below, followed by a new mathematical model for an artificial neurogenesis mechanism which builds on the spiking network. The parameter optimisation procedures for the spiking network, neurogenesis mechanism, and traditional network are then explored, before detailing the evaluation metrics that will be used to measure classification efficacy. The implementation of the traditional and spiking networks are discussed, followed by a brief explanation of the implementation of classification tasks in general.

4.1 Methodology

4.1.1 Spiking network

The spiking network implemented in the study was directly based on Diehl and Cook's leaky integrate-and-fire network that was presented in section 3.4.2. A neuron in this network with a single incoming synapse with weight w, which delivers a spike at time t_{input}, can be summarised by the system of three equations

$$\begin{aligned}
\tau_u \frac{du}{dt} &= (E_{\text{rest}} - u) + g_e(E_{\text{exc}} - u) + g_i(E_{\text{inh}} - u), \\
\frac{dg_e}{dt} &= -\frac{g_e}{\tau_{g_e}} + C(w, 1)\delta(t - t_{\text{input}}), \\
\frac{dg_i}{dt} &= -\frac{g_i}{\tau_{g_i}} + C(w, 0)\delta(t - t_{\text{input}}),
\end{aligned}$$

where

- $u(t)$ is the membrane potential difference;
- τ_u is the time constant of membrane potential difference;
- E_{rest} is the resting membrane potential;
- $g_e(t)$ and $g_i(t)$ are the conductances of the neuron that connect to incoming excitatory and inhibitory synaptic inputs respectively (where a synapse is labelled according to the type of pre-synaptic neuron);

- τ_{g_e} and τ_{g_i} are the time constants of excitatory and inhibitory conductances respectively; and
- E_{exc} and E_{inh} are the equilibrium potentials of excitatory and inhibitory input respectively.

In these equations, $C(w, b)$ (where b is a binary input in $\{0, 1\}$) is a function that depends on the type of synapse:

$$C(w,b) \;=\; \begin{cases} bw & \text{if the pre-synaptic neuron is excitatory} \\ (1-b)w & \text{if the pre-synaptic neuron is inhibitory,} \end{cases}$$

and $\delta(t)$ is the Dirac delta function with

$$\int_{-\infty}^{\infty} \delta(t)\, dt \;=\; 1,$$
$$\text{and} \quad \delta(t) \;=\; 0 \text{ for } t \neq 0,$$

which delivers a point impulse to the appropriate conductance at the time of the pre-synaptic spike t_{input}. A discrete condition must also be defined to allow the neuron to spike:

$$\text{if } u(t) \geq E_{\text{thresh}} + \theta(t), \text{ set } u(t) = E_{\text{reset}} \text{ and record } t_{\text{spike}} = t.$$

In this rule, E_{thresh} is the neuron's spiking threshold, E_{reset} is the reset potential after a spike, and $\theta \geq 0$ implements an adaptive spiking threshold for the neuron, which increases by some amount θ_+ upon a spike and otherwise decays exponentially, so

$$\frac{d\theta}{dt} \;=\; -\frac{\theta}{\tau_\theta} + \theta_+ \delta(t - t_{\text{spike}}).$$

At the time t_{spike}, all outgoing synapses from the neuron deliver the impulse to their post-synaptic neurons, while this neuron will enter a refractory period of duration r seconds, within which it cannot spike again. A synapse in this network maintains a weight w and a pre-synaptic trace x_{pre}, which behaves according to

$$\frac{dx_{\text{pre}}}{dt} \;=\; -\frac{x_{\text{pre}}}{\tau_x} + \delta(t - t_{\text{pre}}),$$

where t_{pre} is the time of a pre-synaptic spike, at which point x_{pre} is increased by 1. At the time of a post-synaptic spike t_{post}, the weight w is discretely updated according to the rule

$$\Delta w \;=\; \nu(x_{\text{pre}} - x_{\text{tar}})(w_{\max} - w)^\mu,$$

where

- $\nu > 0$ is the learning rate;
- x_{tar} is the target value of the pre-synaptic trace, above which $\Delta w > 0$;
- $w_{\max} > 0$ is the maximum weight; and

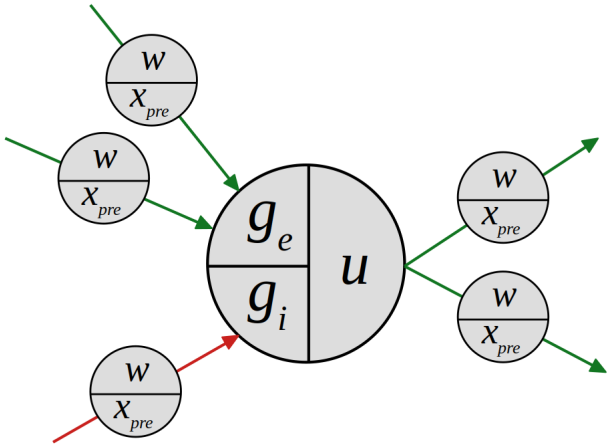

Figure 4.1: A compartmental model of a leaky integrate-and-fire neuron in a network with a few synapses connected to it, where synapses are coloured according to their type (green for excitatory and red for inhibitory). Each neuron is governed by differential equations for its u, g_e and g_i variables, and each synapse maintains a weight w and pre-synaptic trace x_{pre}. This neuron is an excitatory neuron because its outgoing synapses are shown in green.

- $\mu \in [0, 1]$ is the power that determines the dependence of the update Δw on the previous weight value.

A network of these modified leaky integrate-and-fire neurons with synapses connecting them is a large system of coupled ODEs, where each neuron is governed by a set of differential equations that are coupled by synapses, each represented by a separate additive term, figure 4.1. This system will exhibit complex behaviour as the synapse weights adapt via the learning rule.

4.1.2 Comparator traditional network

To compare Diehl and Cook's leaky integrate-and-fire network to a traditional, perceptron neural network, the traditional network should have a similar size and structure. Hence, any differences in performance that arise during experimentation can be attributed to the different neuron models (leaky integrate-and-fire neuron vs perceptron) and/or learning schemes (unsupervised spike timing dependent plasticity vs supervised stochastic gradient descent). The three layer structure of Diehl and Cook's scalable network suggests a deep traditional network with two hidden layers, each containing some variable $N \in \mathbb{N}$ perceptrons, using a feed-forward, fully-connected architecture (as this is the most common architecture for perceptron-based networks). This network has an input layer of 784 neurons (where, just like in the spiking network, each neuron corresponds to one pixel in the input image) and an output layer of ten neurons (each corresponding to one digit class), and is depicted in figure 4.2.

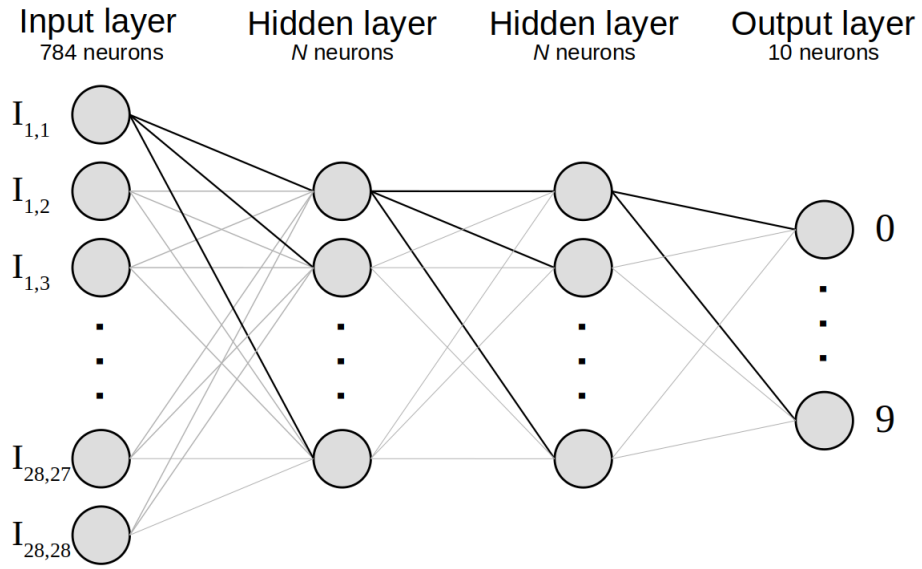

Figure 4.2: A feed-forward, fully-connected perceptron neural network with an input layer of 784 neurons (each corresponding to a pixel in the MNIST digit images), two hidden layers of a variable N number of neurons, and an output layer of 10 neurons, each corresponding to one digit class.

This network differs from the spiking network in its additional output layer of 10 neurons – the spiking network used its second layer of excitatory neurons to produce an output. It also differs in its feed-forward, all-to-all connection scheme – the spiking network only featured all-to-all connections from its input layer to its excitatory layer, and used more restricted connections between the excitatory and inhibitory layers in both directions. This traditional network has 10 extra nodes, but when $N = 100$, this is a network size increase of approximately 1%. The spiking network has $784N + N + N(N-1) = N^2 + 784N$ synapses, which is 88,400 when $N = 100$. On the other hand, the traditional network has $784N + N^2 + 10N = N^2 + 794N$ connections, which is 89,400 when
$N = 100$, also representing an approximately 1% increase in the number of connections. Therefore, the two networks have roughly the same size, which makes for a fair comparison.

To train this perceptron network, the standard approach for a classification task will be used: stochastic gradient descent with a cross-entropy loss, using batch sizes of 100 training samples at a time. Different values for the learning rate $\nu \in [0, 1]$ for stochastic gradient descent will be considered.

In the original MNIST paper [59], Yann LeCun experimented with training a wide variety of neural networks on this task, including three layer networks such as this one, which showed good performance (an accuracy score of more than 95%) with just 400 neurons between the two hidden layers (where the two layers were not necessarily of the same size). Hence, the network architecture presented here can be used as a suitable representative of traditional neural networks for this classification task.

4.1.3 Neurogenesis

Given this spiking network model, the objective was to define a biologically plausible neurogenesis mechanism to implement structural plasticity for the network in response to its dynamics, as discussed in section 2.5. This work was inspired by the network structure optimisation algorithms for traditional neural networks that are researched in the "neural architecture search" field [60]. The challenge was to define a neural architecture search algorithm that was

- compatible with a spiking neural network;
- compatible with an unsupervised learning scheme (spike timing dependent plasticity); and
- biologically inspired by neurogenesis in the human brain.

This was a unique challenge that had not previously been researched: work has been done on unsupervised neural architecture search methods, but these have all been built on perceptron-based, traditional neural networks, and none of them are biologically inspired.

The neurogenesis mechanism presented in this thesis adds four new variables to each neuron in the network. Firstly, define $n \in [0, 1]$, which is an abstraction of the local neuromodulator density near the neuron, and represents its level of recent activity. Given a spike time t_{spike} for the neuron, n behaves according to

$$\tau_n \frac{dn}{dt} = -n + \min\{n_s,\ 1 - n\}\delta(t - t_{\text{spike}}),$$

meaning it increases by a fixed amount n_s whenever the neuron spikes, and otherwise decays to zero. Each neuron also inherits the properties x, y and z, which hold its position in a 3D network space. Cell death is introduced into the network to prune inactive nodes: if a neuron's neuromodulator level n dips below a threshold n_d, this neuron and all of its adjacent synapses are deleted.

A neurogenesis event is the insertion of a newborn neuron into the network, reminiscent of a biological neuron migrating through blood vessels in the brain and finally reaching its target region, as discussed in section 2.5. To generate neurogenesis event times in the artificial network, define a neurogenesis rate r_n and variance parameter σ_r. When a neurogenesis event occurs, the time to the next event is sampled from the normal distribution

$$\Delta t \sim \mathcal{N}(\frac{1}{r_n}, \sigma_r).$$

The network parameter $t_{\max}$ controls the neurogenesis cut-off time, meaning if $t + \Delta t > t_{\max}$, no more neurogenesis events will be triggered. This feature models the biological brain's preference to apply neurogenesis primarily during the early stages of embryonic development, by only allowing the creation of neurons during the first $t_{\max}$ seconds of simulated time.

At a neurogenesis event, a single new neuron is created, with its (x, y, z) position determined by the neuromodulator densities n throughout the network, such that

the neuron is added to either the most or least active region. To achieve this, first define the 1-norm

$$||\boldsymbol{x}||_1 \;=\; \sum_{i=1}^{N} |x_i| \quad \text{for } \boldsymbol{x} \in \mathbb{R}^N,$$

and define a vector noise function

$$\text{noise}(\boldsymbol{x}) = \boldsymbol{z}^T\boldsymbol{x} \text{ where } \boldsymbol{z} \in \mathbb{R}^N \text{ and each } z_i \sim \text{Lognormal}(0, \sigma_n^2)$$

with standard deviation set by a network parameter σ_n.

To implement a neurogenesis mechanism where new neurons gravitate to the most active (in recent memory) areas of the network, define the normalised, noisy vector of neuromodulator levels

$$\boldsymbol{n} \;=\; \frac{\hat{\boldsymbol{n}}}{||\hat{\boldsymbol{n}}||_1} \text{ where } \hat{\boldsymbol{n}} \;=\; \text{noise}\left(\begin{pmatrix} n_1 & n_2 & \dots & n_N \end{pmatrix}^T\right) \in \mathbb{R}^N,$$

where N is the number of neurons in the network and n_i is the neuromodulator density of the i^{th} neuron. Then, define the position vector

$$\boldsymbol{x} \;=\; \begin{pmatrix} x_1 & x_2 & \dots & x_N \end{pmatrix}^T$$

whose entries are the x-coordinates of the N neurons in the network, and similarly define position vectors $\boldsymbol{y}$ and $\boldsymbol{z}$. Finally, use the dot product to determine the position of the new neuron as

$$x_{N+1} = \boldsymbol{n}^T\boldsymbol{x}, \;\; y_{N+1} = \boldsymbol{n}^T\boldsymbol{y} \;\text{ and }\; z_{N+1} = \boldsymbol{n}^T\boldsymbol{z}.$$

The new neuron forms synapses by iterating through its local neighbourhood in proximity order (closest first), where the probability of forging a new synapse with neuron j depends on both the distance D_j to neuron j and its neuromodulator level n_j, according to

$$\mathbb{P}(\text{synapse}) \;=\; n_j^{s_n} \exp(-d_n D_j),$$

where s_n and d_n are dependency parameters of the network. The direction of these new synapses either abides by the general direction of information flow in the network (by comparing x_{N+1} and x_j), or can be generated randomly (with equal probability of forwards or backwards). This iteration continues until either the network's neurons have been exhausted or the new neuron has created $s_{\max}$ synapses, which is another parameter of the network.

Newborn neurons experience a temporary, enhanced period of duration T_{enh} seconds, which gives them a competitive advantage over mature neurons in the network. This enhanced period is characterised by

- enhanced excitability, where the neuron's conductance variables abide by the modified rules

$$\begin{aligned}\frac{dg_e}{dt} &= -\frac{g_e}{\tau_{g_e}} + C(g_{\text{enh}}w, 1)\delta(t - t_{\text{input}}),\\ \frac{dg_i}{dt} &= -\frac{g_i}{\tau_{g_i}} + C(g_{\text{enh}}w, 0)\delta(t - t_{\text{input}}),\end{aligned}$$

  meaning any impulses received via synapses of weight w increase the appropriate conductance variable by $g_{\text{enh}}w$ where the multiplicative factor $g_{\text{enh}} > 1$ is a parameter of the network; and
- enhanced plasticity for outgoing synapses with modified learning rate $\nu_{\text{enh}}\nu$ where the factor $\nu_{\text{enh}} > 1$ is a parameter of the network.

As discussed in section 2.5, this enhanced period helps with the formation and stabilising of the new neuron's synapses, by allowing the new neuron to actively contribute to information processing as soon as it is created.

This neurogenesis mechanism, where new neurons gravitate to the most active (in recent memory) areas of the network, can be thought of as trying to build redundancy into the network by creating multiple parallel pathways for important signals to travel through. This is because the new neurons are being added where lots of signal traffic is already flowing through the existing neurons, and the new neuron will form synapses with its local neighbours. There is evidence in the visual cortex that the human brain does employ such parallel pathways. The "two-stream" hypothesis proposed by Ungerleider and Mishkin in 1982 [61] claims that two parallel pathways connect the V1 region in the visual cortex to the temporal cortex: the dorsal stream and the ventral stream. Therefore, this neurogenesis mechanism and its strategy for the positioning of new neurons can be considered biologically plausible.

On the other hand, consider a neurogenesis mechanism where new neurons gravitate to the least active (in recent memory) areas of the network, via the modified positioning rule

$$\boldsymbol{n} = \frac{\hat{\boldsymbol{n}}}{||\hat{\boldsymbol{n}}||_1} \text{ where } \hat{\boldsymbol{n}} = \text{noise}\left(\begin{pmatrix}1-n_1 & 1-n_2 & \dots & 1-n_N\end{pmatrix}^T\right) \in \mathbb{R}^N,$$

where the coordinates x_{N+1}, y_{N+1} and z_{N+1} are computed from $\boldsymbol{n}$ as defined previously. This mechanism can be thought of as trying to avoid redundancy when also paired with neuron decay (where neurons are removed if their n variable decreases below a set threshold), because it adds neurons into the network to create entirely new pathways where there previously was a lack of activity. There is strong evidence to suggest that during embryonic development, there is a growth phase (after neurons have mostly finished migrating to their specified regions) where innumerable synapses are formed between neurons, followed by a phase of competition among these connections where many redundant pathways are pruned to drastically reduce the number of connections and stabilise the ones that remain [62]. Therefore, this strategy for the positioning of new neurons is also biologically plausible.

Given that both of these neurogenesis mechanisms (bolstering the most active regions or strengthening the quietest regions) can be argued to be biologically plausible, despite the fact that they represent opposing extrema in the space of neurogenesis mechanisms (with respect to the positioning of new neurons), these mechanisms need to be fairly benchmarked in order to determine which type of rule is a suitable candidate for further study.

4.1.4 Parameter optimisation for spiking network

The spiking network outlined in section 4.1.1 has a large set of parameters which are listed in table 4.1, the values for which need to be optimised to reach the maximum attainable performance on any given classification task.

Parameter	Range	Units	Definition
Δt	$\mathbb{R}_{>0}$	s	Network timestep duration.
τ_u	$\mathbb{R}_{>0}$	s	Time constant of membrane potential u.
E_{rest}	$\mathbb{R}$	mV	Resting membrane potential.
E_{reset}	$\mathbb{R}$	mV	Post-spike reset membrane potential.
E_{thresh}	$\mathbb{R}$	mV	Spiking threshold membrane potential.
E_{exc}	$\mathbb{R}$	mV	Equilibrium potential of excitatory input.
E_{inh}	$\mathbb{R}$	mV	Equilibrium potential of inhibitory input.
τ_{g_e}	$\mathbb{R}_{>0}$	s	Time constant of excitatory conductance g_e.
τ_{g_i}	$\mathbb{R}_{>0}$	s	Time constant of inhibitory conductance g_i.
τ_θ	$\mathbb{R}_{>0}$	s	Time constant of adaptive spiking threshold θ.
r	$\mathbb{R}_{\geq 0}$	s	Duration of neuron refractory period.
θ_0	$\mathbb{R}_{\geq 0}$	mV	Initial value of adaptive spiking threshold θ.
θ_+	$\mathbb{R}_{>0}$	mV	Discrete increase to adaptive spiking threshold θ.
τ_x	$\mathbb{R}_{>0}$	s	Time constant of pre-synaptic trace x_{pre}.
ν	$\mathbb{R}_{>0}$	1	Learning rate of synapse weights.
$w_{\max}$	$\mathbb{R}_{>0}$	1	Maximum synapse weight w.
μ	$\mathbb{R}_{\geq 0}$	1	Dependence of weight change on previous weight.
x_{tar}	$\mathbb{R}_{>0}$	1	Target value of pre-synaptic trace x_{pre}.

Table 4.1: The parameters of the spiking network without neurogenesis, listed with their value ranges, units and definitions.

Optimising all of these parameters simultaneously would require searching within a continuous, 18 dimensional parameter space, which is computationally infeasible. Choosing just two values for each listed parameter would result in $2^{18} = 262,144$ unique parameter sets, which would each need to be evaluated by training and testing a spiking network with these parameter values. Simulating a small network (less than 20 neurons) for a single input has a runtime on the order of seconds (using the implementation detailed in section 4.2.2). Assuming a one second runtime per input sample, training a small network on 500 samples from a dataset and evaluating its performance on 100 further samples would require 10 minutes of runtime. Performing this evaluation procedure on each of the 262,144 parameter sets would therefore take almost 5 years to run in series, or require 1,820 processes running in

parallel to complete the procedure in 24 hours. Therefore, it is necessary to identify a subset of these parameters to optimise, and set the remaining parameters to biologically plausible values.

A rescaling of the leaky integrate-and-fire neuron equations in the spiking network can be used to reveal the relationships between different parameters, and hence suggest which parameters should be fixed, and which should be optimised. Ignoring any instantaneous terms, the leaky integrate and fire neuron equations used in the spiking network are

$$\begin{aligned}
\tau_u \frac{du}{dt} &= (E_{\text{rest}} - u) + g_e(E_{\text{exc}} - u) + g_i(E_{\text{inh}} - u), \\
\tau_{g_e} \frac{dg_e}{dt} &= -g_e, \\
\tau_{g_i} \frac{dg_i}{dt} &= -g_i.
\end{aligned}$$

These equations feature six of the network's parameters: τ_u, E_{rest}, E_{exc}, E_{inh}, τ_{g_e} and τ_{g_i}. Seeking a dimensionless system, define the rescaled variables

$$x = \frac{u}{u_0},\ s = \frac{t}{t_0},\ y = \frac{g_e}{g_{e0}} \text{ and } z = \frac{g_i}{g_{i0}},$$

then applying the chain rule yields

$$\begin{aligned}
\frac{du}{dt} &= \frac{du}{dx}\frac{dx}{ds}\frac{ds}{dt} = u_0 \frac{dx}{ds}\frac{1}{t_0} = \frac{u_0}{t_0}\frac{dx}{ds}, \\
\frac{dg_e}{dt} &= \frac{dg_e}{dy}\frac{dy}{ds}\frac{ds}{dt} = g_{e0}\frac{dy}{ds}\frac{1}{t_0} = \frac{g_{e0}}{t_0}\frac{dy}{ds}, \text{ and} \\
\frac{dg_i}{dt} &= \frac{dg_i}{dz}\frac{dz}{ds}\frac{ds}{dt} = g_{i0}\frac{dz}{ds}\frac{1}{t_0} = \frac{g_{i0}}{t_0}\frac{dz}{ds}.
\end{aligned}$$

Starting with the equation for $\dfrac{du}{dt}$, substituting the rescaled variables and the derivative expansions finds

$$\begin{aligned}
\tau_u \frac{u_0}{t_0}\frac{dx}{ds} &= (E_{\text{rest}} - u_0 x) + g_{e0} y (E_{\text{exc}} - u_0 x) + g_{i0} z (E_{\text{inh}} - u_0 x) \\
\frac{dx}{ds} &= \frac{t_0 E_{\text{rest}}}{\tau_u u_0} - \frac{t_0}{\tau_u} x + \frac{g_{e0} t_0 E_{\text{exc}}}{\tau_u u_0} y - \frac{g_{e0} t_0}{\tau_u} xy + \frac{g_{i0} t_0 E_{\text{inh}}}{\tau_u u_0} z - \frac{g_{i0} t_0}{\tau_u} xz.
\end{aligned}$$

Aiming for the simplest coefficients, set $t_0 = \tau_u$, which yields

$$\frac{dx}{ds} = \frac{E_{\text{rest}}}{u_0} - x + \frac{g_{e0} E_{\text{exc}}}{u_0} y - g_{e0} xy + \frac{g_{i0} E_{\text{inh}}}{u_0} z - g_{i0} xz.$$

Now looking at the equation for $\frac{dg_e}{dt}$, substituting finds that

$$\begin{aligned}\frac{g_{e0}\tau_{g_e}}{t_0}\frac{dy}{ds} &= -g_{e0}y \\ \frac{dy}{ds} &= -\frac{t_0}{\tau_{g_e}}y = -\frac{\tau_u}{\tau_{g_e}}y,\end{aligned}$$

and similarly

$$\frac{dz}{ds} = -\frac{\tau_u}{\tau_{g_i}}z.$$

Hence, still aiming for simplicity, set the scaling constants

$$u_0 = E_{\text{rest}},\ g_{e0} = \frac{E_{\text{rest}}}{E_{\text{exc}}} \text{ and } g_{i0} = \frac{E_{\text{rest}}}{E_{\text{inh}}}.$$

Finally, the rescaled system of three coupled ODEs is

$$\begin{aligned}\frac{dx}{ds} &= 1 - x + y - \frac{E_{\text{rest}}}{E_{\text{exc}}}xy + z - \frac{E_{\text{rest}}}{E_{\text{inh}}}xz, \\ \frac{dy}{ds} &= -\frac{\tau_u}{\tau_{g_e}}y \text{ and} \\ \frac{dz}{ds} &= -\frac{\tau_u}{\tau_{g_i}}z.\end{aligned}$$

This rescaling implies that amongst the six parameters considered (τ_u, E_{rest}, E_{exc}, E_{inh}, τ_{g_e} and τ_{g_i}), there are four important ratios:

$$\frac{E_{\text{rest}}}{E_{\text{exc}}},\ \frac{E_{\text{rest}}}{E_{\text{inh}}},\ \frac{\tau_u}{\tau_{g_e}} \text{ and } \frac{\tau_u}{\tau_{g_i}},$$

and hence this subset of the parameter space is actually just four dimensional.

To evaluate the performance of different parameter sets for the proposed model, a parameter sweep can be applied within a lattice subset of its parameter space, where at each point, a network with those parameter values is initialised, trained and then evaluated to return a performance measure such as an accuracy score. The results of this process can be visualised on a grid containing a heatmap for every pair of parameters, where each point in one of these heatmaps displays the best, average or worst performance score achieved by a parameter set with the corresponding pair of values. See sections 5.1 and 6.1 for examples of such heatmap grids.

Any parameter sweep in this research project needs to be able to run on the Katana computational cluster managed by UNSW, which requires that jobs have a runtime of no more than 12 hours, and suggests that 48 CPU cores is a reasonable upper bound for regular jobs. By running a small parameter sweep on the spiking network for benchmarking purposes, it was determined that a parameter sweep iteration cannot include more than 8,000 combinations of parameter values if it is to finish successfully within these constraints. To form accuracy heatmaps (within

a wider grid) of reasonable resolution, each parameter should be considered over 5-6 different values. Combining these constraints, the number of parameters that can be considered per parameter sweep iteration is the maximal n such that $6^n \leq$ 8,000, which is $n = 5$.

This research was scoped to considering only biologically plausible spiking networks, and hence the set of parameters in the network that require optimisation was reduced by fixing some parameters to biologically plausible values, where such justifications can be made. Parameters that correspond to known properties of the biological neuron, and hence can be fixed to a plausible value, were:

- the resting membrane potential E_{rest}, which typically lies between -60 and -70mV [16], so it was fixed at -65mV;
- the spiking threshold E_{thresh}, which typically ranges from -55 to -50mV [63], so it was fixed at -50mV;
- the reset potential after an action potential, E_{reset}, which should be less than E_{rest} because of afterhyperpolarisation (the overshoot of membrane potential below the resting level after an action potential), and hence a value of -75mV was appropriate; and
- the target membrane potential of excitation E_{exc}, which was set to the sodium ion equilibrium voltage of +55mV [16], which is approximately the peak value of an action potential [16] (and is often referred to as E_{Na} in the literature).

Considering the remaining free parameters, the five most significant (those with the largest influence on network behaviour) are

- the time constant of neuron membrane potential τ_u, where a biologically plausible value is 10ms [63], so a parameter sweep should search around this value;
- the time constants of neuron post-synaptic conductances $\tau_{g_e} = \tau_{g_i}$;
- the time constant of the adaptive firing threshold τ_θ for neurons;
- the time constant of the pre-synaptic trace τ_x for synapses; and
- the learning rate ν in spike timing dependent plasticity for synapse weights w.

When optimising these parameters of the spiking network on a given classification task, multiple iterations of parameter sweeps may be required to identify a region within the parameter space where performance is stable and near-optimal. A stable region of the parameter space should result in heatmaps with large sections of near-constant accuracy, where one can be confident that choosing a parameter set from within this region will guarantee the indicated level of performance. Otherwise, if there are "cliffs" or "holes" in the heatmap surface near the chosen region (where accuracy significantly drops off), one can assume that a slight change in network conditions (such as the specific training samples given to the network) could result in much poorer performance than expected.

4.1.5 Parameter optimisation for neurogenesis

The neurogenesis mechanism outlined in section 4.1.3 has many parameters that need to be optimised to maximise performance on any classification task, and these are listed in table 4.2.

Parameter	Range	Units	Definition
τ_n	$\mathbb{R}_{>0}$	s	Time constant of neuromodulator level n.
n_s	$\mathbb{R}_{>0}$	1	Increase to n upon a spike.
n_d	$\mathbb{R}_{\geq 0}$	1	Neuromodulator cell death threshold.
r_n	$\mathbb{R}_{>0}$	s^{-1}	Neurogenesis rate.
$t_{\max}$	$\mathbb{R}_{\geq 0}$	s	Neurogenesis cutoff time.
σ_r	$\mathbb{R}_{\geq 0}$	s	Standard deviation of neurogenesis timing noise.
g_{enh}	$\mathbb{R}_{\geq 1}$	1	Newborn neurons enhanced excitability factor.
ν_{enh}	$\mathbb{R}_{\geq 1}$	1	Newborn neurons enhanced learning rate factor.
T_{enh}	$\mathbb{R}_{\geq 0}$	s	Newborn neurons enhanced period duration.
s_n	$\mathbb{R}_{\geq 0}$	1	New synapse neuromodulator dependency parameter.
s_d	$\mathbb{R}_{\geq 0}$	1	New synapse distance dependency parameter.
$s_{\max}$	$\mathbb{R}_{>0}$	1	Maximum number of synapses for a newborn neuron.
σ_n	$\mathbb{R}_{\geq 0}$	m	Standard deviation of newborn neuron position noise.
Method	A or I	s	Method for positioning newborn neurons.

Table 4.2: The parameters of the neurogenesis mechanism, listed with their value ranges, units and definitions. Note that for the method parameter, A denotes the "active" method and I denotes the "inactive" method of neuron positioning.

The most significant parameters in this list are

- the time constant of the neuromodulator level τ_n (including this in the sweep means that the increase amount n_s can be fixed to a reasonable value such as 0.1, as varying τ_n is equivalent to varying n_s and the threshold n_d);
- the neuromodulator cell death threshold n_d, where setting this to a value less than 0 is equivalent to "disabling" cell death in the mechanism, so two values that were be swept over are -1 and 0.05;
- the neurogenesis rate r_n;
- the neurogenesis cutoff time $t_{\max}$;
- the new synapse distance dependency parameter s_d;
- the maximum number of synapses $s_{\max}$ for a newborn neuron; and
- the newborn neuron positioning method, which can be set to "active" or "inactive".

It is difficult to determine biologically plausible values for these parameters to sweep over, as neurogenesis has not been the subject of much mathematical modelling. As such, a broad range of values were considered, to increase the likelihood of an optimal region within this large parameter space being found. For example, the time constant τ_n of the neuromodulator level n was considered over the values: $10^{-1}, 10^{-2}, 10^{-3}$ and 10^{-4}.

To avoid the stochastic parts of the neurogenesis mechanism influencing the evaluation of its performance, σ_r and σ_n were set to zero. Considering nonzero values of these parameters would have required each simulation experiment containing neurogenesis to be run several times to determine the average performance score, which would have significantly increased the computing resources and time required, beyond those available for this project. For the simulation experiments in this investigation, the two newborn neuron enhanced factors g_{enh} and ν_{enh} were set to

1.25, i.e., the excitability of newborn neurons and the plasticity of their adjacent synapses received a boost of 25%.

For any classification task, a neurogenesis parameter sweep requires the parameters of the underlying spiking network model to be fixed. As such, the spiking network parameter sweep process as outlined in the previous section 4.1.4 was run first with neurogenesis disabled. An optimal parameter set for the spiking network was identified from the results and fixed prior to the neurogenesis parameter sweep.

4.1.6 Parameter optimisation for traditional network

A parameter sweep was also performed for the traditional network, where the two most significant parameters were

- the learning rate ν of stochastic gradient descent, where the values $10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}$ and 10^{-1} were considered; and
- the momentum ρ, also from the stochastic gradient descent algorithm, where the values $0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.975, 0.99, 0.995$ and 0.999 were considered.

This parameter sweep ran completely independently of the spiking and neurogenesis parameter sweeps on any given classification task. Because only two parameters were considered in this process, the results can be visualised as a single heatmap for ν against ρ, to summarise the entire parameter space in a single plot, see section 5.3.

4.1.7 Measuring classification efficacy

To compare the computation and performance of traditional and spiking artificial neural networks on classification tasks, the following procedure was used:

1. a suitable architecture for each network type was defined for the task, of similar structure and capacity, to ensure a fair comparison;
2. a parameter sweep was performed independently for each network to identify a stable region of its parameter space that achieved near-optimal performance; where the evaluation used a very limited set of training and testing data to make the parameter sweeps computationally feasible;
3. a varied set of several parameter values within the optimal regions were defined for each network; and
4. a comparison experiment was run using the parameter sets for both networks, using a larger set of training and testing data to robustly measure and compare the computation and performance of the networks.

The comparison experiment measured:

- the accuracy of the networks on the test set, given an increasing set of training samples, to evaluate the pace at which the networks were able to learn the given task, and the maximum performance they attained;
- the connection weights in the networks over an epoch of the training data, to evaluate whether or not the networks' representations of the problem reached a steady state, and if so, to gauge their stability;
- the utilisation of connections in the networks (the proportion of connections with a nonzero weight after training), where a smaller utilisation implies a sparser and hence more efficient representation of the task;

- the average number of spikes required to produce an output for a given input; and
- the average processing time to produce an output for a given input.

An ideal network would achieve a higher maximum accuracy than its counterpart, requiring less training samples to reach this level of performance. It would learn a sparse representation of the task, with its weights reaching a stable steady state quickly, which suggests that the network is robust to perturbations in its parameters and training data/procedure. It would also process inputs efficiently, requiring less spikes than its counterpart and less processing time to simulate the network.

The results of this comparison experiment on any classification task may have implications for our understanding of biological signal processing – in particular, why the biological brain has evolved to implement a spiking neural network and not one comprised of simpler, binary units such as perceptrons. They may also be useful for validating recent claims made by neuroscientists that spiking networks could be the future of artificial intelligence research, by identifying possible advantages that spiking networks offer over traditional networks for classification tasks.

These comparison metrics can also be used to measure the influence on computation of the neurogenesis mechanism when applied to the artificial spiking network. The procedure in this case was to

1. take a set of optimal parameter values for the spiking network from the comparison experiment;
2. perform a parameter sweep on a subset of the neurogenesis parameters to identify a stable region of this parameter space that achieves near-optimal performance;
3. a varied set of neurogenesis parameter values within this optimal region were chosen; and
4. a comparison experiment was run to compare the spiking network with and without neurogenesis, using both sets of optimal parameter values (i.e., for each spiking network parameter set, the metrics were evaluated on the network with neurogenesis turned off, and then on the network with neurogenesis turned on, for each neurogenesis parameter set).

A positive result for the neurogenesis mechanism would be an increase in the maximum accuracy achieved by the spiking network, by adapting it towards a more efficient network architecture that yields a more sparse representation of the task. This improved architecture would require less spikes and thus less time to process inputs.

4.2 Implementation

4.2.1 Traditional network

The traditional network was implemented in Python using PyTorch, an open source machine learning library [64]. In PyTorch, neural networks can be defined as a class, which specifies the architecture via the composition of objects. The code snippet below defines a three layer network (one input layer, one hidden layer and one output layer) with ten input nodes, six (by default) hidden nodes and two output nodes.

```
from torch import nn

class ThreeLayerNetwork(nn.Module):
    def __init__(self, hidden_units=6):
        super(ThreeLayerNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.network = nn.Sequential(
            nn.Linear(10, hidden_units),
            nn.ReLU(),
            nn.Linear(hidden_units, 2)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.network(x)
        return logits
```

In this network definition, `nn.Linear(10, hidden_units)` stipulates that the input and hidden layers of the network are fully-connected, `nn.ReLU()` assigns the rectified linear unit activation function to the nodes in the hidden layer, and finally `nn.Linear(hidden_units, 2)` stipulates that the hidden and output layers are also fully-connected. To train a neural network, one must define a loss function (called `criterion` in the code snippet below) and a learning regime (called `optimiser` in the code snippet below) to minimise it given a set of training data. For this research project, the cross-entropy loss function

$$-\sum_{i=1}^{n} y_i \ln(p_i)$$

for true labels $\boldsymbol{y} \in \{0,1\}^n$ and network predictions $\boldsymbol{p} \in \mathbb{R}^n$ was used, which is a popular choice for training neural networks on classification tasks [65]. The network was trained using stochastic gradient descent, section 3.3.3, and requires a learning rate $\nu \in \mathbb{R}_{>0}$ and momentum $\rho \in \mathbb{R}_{\geq 0}$ to be defined as follows.

```
from torch import nn, optim
criterion = nn.CrossEntropyLoss()
optimiser = optim.SGD(network.parameters(), lr=nu, momentum=rho)
```

Given a neural network which has been initialised as `network = ThreeLayerNetwork(6)`, the training procedure can be implemented in just a few lines of code, thanks to PyTorch's automatic differentiation routines. First, the training data `X` is ran forward through the network to return a tensor of predictions `preds`, which are used along with the true labels `y` to compute the loss using the `criterion` function. This loss is then propagated backward through the network to compute its gradient for each network parameter, and finally the parameters are incremented in the direction of steepest descent, proportional to the learning rate ν.

```
optimiser.zero_grad()
preds = network(X)
```

```
loss = criterion(preds, y.long())
loss.backward()
optimiser.step()
```

The training data is split into batches of a fixed size, and this training procedure is performed iteratively for each batch (where a batch size of 1 means the network's parameters are optimised for each individual training sample). After training is complete, the network can be used to make predictions on new test data by simply feeding a 2D tensor of inputs `X` to the network, to return its predictions. This output will be a 2D tensor `preds` with a row per sample, where each row is interpreted as a vector $\boldsymbol{p}$ whose entries are the predicted likelihood of the corresponding class being the true label. This output can be turned into a final tensor of class predictions using `_,classes = torch.max(preds.data, 1)`.

4.2.2 Spiking network

To include a structural plasticity mechanism such as neurogenesis in an artificial spiking network, the implementation of this spiking network needs to support modifying the network architecture during runtime. The most popular library for implementing spiking networks in Python is Brian [43, 4], but this library does not support this advanced feature. To meet this requirement, an artificial spiking network was implemented in Python from scratch, relying only on lower-level libraries such as NumPy [66] for its vector data types and SciPy [67] for its integration routines.

A guiding principle behind the development of this artificial spiking network was to implement a simple object-oriented interface to initialise, train and test a model with minimal code required outside of the source files. This was inspired by the PyTorch library [64], which is introduced in the previous section, and which made defining a traditional neural network and using it very easy. The spiking network implementation is comprised of three main classes:

- a `Neurogenesis` class, which is initialised by providing an optional dictionary of parameters (whose default values are defined in the class' constructor, such as τ_n and r_n), and implements a `neurogenesis()` function which is responsible for creating new neurons;
- a `NeuronModel` class, which is initialised by providing an optional dictionary of parameters and a `Neurogenesis` object, and stores the neurons and synapses of the network, implements integration and handles spikes; and
- a `Network` class, which is initialised by providing an optional dictionary of parameters, a `NeuronModel` object and `Neurogenesis` object, and implements the main simulation loop and manages simulated time.

This modular setup allowed for the definition of multiple different neuron and neurogenesis models, which were able to be used automatically in any combination thanks to a standard set of methods each class had to implement.

In the `NeuronModel` class, neurons are stored in a dictionary mapping from each neuron's unique integer ID to a dictionary of its properties. These properties are

- its x, y and z coordinates in a 3D network space;
- its type, either `"exc"` or `"inh"`;
- its membrane potential u;

- its post-synaptic conductances g_e and g_i;
- its adaptive threshold θ;
- the time of its last spike t_{spike};
- its neuromodulator density n;
- a list of incoming synapse IDs and outgoing synapse IDs;
- a boolean property which determines whether or not the neuron is in an enhanced state, along with a time t_{enh} that this enhanced period should end; and
- a class label which is initially `None` and assigned (to neurons identified as belonging to the output layer) after training.

Synapses are also stored in a dictionary mapping from each synapse's unique integer ID to a dictionary of its properties, which are

- its weight w;
- its pre-synaptic trace x_{pre};
- its pre-synaptic neuron ID and post-synaptic neuron ID.

The `Network` class implements a `train_with_replay()` function which iterates through a given NumPy array of training samples, first generating spike times for the network's input neurons by sampling from a Poisson process, and then calling a `run()` function which simulates the network for a set duration provided as an argument (which defaults to 350ms). This `run()` function implements the main simulation loop by iterating through timesteps of Δt (a parameter of the network), where at each timestep, it

1. integrates all neuron and synapse properties;
2. handles spikes in input and hidden/output neurons; and
3. calls routines from its composed `Neurogenesis` object to trigger and handle neurogenesis events.

Integration of neuron and synapse properties is performed by an `integrate_step()` method in the `NeuronModel` class. This function uses SciPy's `odeint()` function [68], which in turn uses the `LSODA()` routine from a FORTRAN library called ODEPACK [69], which uses advanced numerical integration techniques that operate on the Jacobian matrix of a system of ODEs. The precise technique used depends on whether or not `LSODA()` determines that the system of ODEs is stiff [70] (meaning some methods will be numerically unstable without a very small step size). In the non-stiff case, Adams methods [71] (which encompass the forward and backward Euler methods and the trapezoidal rule) are used, and in the stiff case, a backward differentiation formula method is used. In the `integrate_step()` function, each neuron is integrated independently as a system of three ODEs (for u, g_e and g_i), as shown in the source code snippet below, which was taken from the `NeuronModel` class.

```
@staticmethod
def neuron_gradients(
    variables, t, tau_u, u_rest, u_exc, u_inh, tau_ge, tau_gi
):
    dudt = (
        u_rest-variables[0] + variables[1]*(u_exc-variables[0])
        + variables[2]*(u_inh-variables[0])
```

```
    )/tau_u
    dgedt = -variables[1]/tau_ge
    dgidt = -variables[2]/tau_gi
    return [dudt,dgedt,dgidt]

def integrate_step(self, t, dt, freeze=False):
    self.spikes = []
    for n in self.neurons:
        neuron = self.neurons[n]
        if neuron['type'] == 'exc':
            params = self.neuron_params_exc()
        else:
            params = self.neuron_params_inh()

        [neuron['u'],neuron['g_e'],neuron['g_i']] = odeint(
            self.neuron_gradients,
            [neuron['u'],neuron['g_e'],neuron['g_i']],
            [t, t+dt],
            args=params
        )[-1]
```

For input neurons, spikes are triggered by comparing the current simulation time to their pre-defined list of spike times; whilst for all other neurons, spikes are triggered by comparing their membrane potential u to the spiking threshold E_{thresh}. A spike is handled by

1. resetting the neuron's membrane potential u to E_{reset};
2. incrementing the neuron's adaptive threshold θ by θ_+;
3. calling the `Neurogenesis` model's `handle_spike()` function which increments the neuron's neuromodulator density n by n_s;
4. incrementing the pre-synaptic trace x_{pre} of all outgoing synapses by 1;
5. incrementing the relevant post-synaptic conductance g_e or g_i of all outgoing connected neurons by the corresponding synapse weight w; and
6. applying spike timing dependent plasticity to adjust the weight w of all incoming synapses.

Finally, at each timestep, an `is_time()` method of the `Neurogenesis` object is called to check if a neurogenesis event should be triggered. If this function returns `True`, the `neurogenesis()` method is called, which

1. determines the position of the new neuron using the neuromodulator levels of existing neurons in the network, as described in section 4.1.3;
2. calls the `NeuronModel` method `create_neuron()` to add the new neuron to the network and initialise its variables; and
3. iteratively defines synapses for the new neuron as described in section 4.1.3.

The code snippet below demonstrates the simple interface that this modular implementation exposes for training and testing a model (this example is for a binary classification task), which is reminiscent of PyTorch's simple user-facing functions.

```
input_secs = 0.35
cooldown_secs = 0.15
```

```
classes = [0,1]
network.train_with_replay(X_train, input_secs, cooldown_secs)
network.label_with_replay(X_train, input_secs, y_train, classes)
y_pred = network.test_with_replay(X_test, input_secs)
```

To improve the performance of this spiking network implementation, several avenues of optimisation were investigated. The first such enhancement was to reduce the frequency at which some neuron and synapse properties were integrated, by only integrating them when an updated value was required for computation. Notable examples of these properties are

- the pre-synaptic trace x_{pre} for each synapse, which is only updated before it needs to be incremented or before the synapse weight is tuned by the spike timing dependent plasticity learning rule, using the exact formula

$$x_{\text{pre}} \leftarrow x_{\text{pre}} e^{-\frac{t-t_0}{\tau_x}}$$

  where t_0 is the last time the trace was updated (a property managed by each synapse); and
- the neuromodulator density n for each neuron, which is only updated when the neuron spikes and hence n needs to be incremented, or when a neurogenesis event occurs and the neuromodulator density is needed to position the new neuron, which is done using a formula similar to the one for x_{pre} above.

The next such enhancement came after noticing that a lot of integration during the network's runtime was spent updating neuron or synapse properties that were essentially zero. This wasted computation was avoided by defining a threshold `float_tol = 1e3*np.finfo(float).eps` equal to 1000 times the machine epsilon of Python's floating point number system. When a variable in the network (such as a neuromodulator density n or synapse weight w) reaches (its absolute value) below this threshold, it is automatically sent to zero to avoid having to integrate it until it is incremented above this level.

Another optimisation that was investigated was pre-integration of the membrane potential u and conductances g_e and g_i. Before the network was simulated, a three dimensional grid was constructed over these variables, whose resolution was proportional to the simulation timestep Δt, where each point corresponded to a unique set of values that these variables could take. Numerical integration using SciPy's `odeint()` function was performed to compute the values of these properties after one timestep, and this data was stored in a table, which mapped the set of initial values to the integrated values after one timestep. This table was then loaded upon the initialisation of the network, which instead of numerically integrating these coupled properties per neuron per timestep during simulation, would simply query the table for the closest match and update the values accordingly. This drastically sped up the network's simulation by at least one order of magnitude, but came with a significant memory cost: a grid with N values per variable would result in a table with N^3 records. Setting 1,000 evenly-spaced values per variable, and assuming each row of the table stored three 64-bit floating point numbers and hence required 24 bytes of storage, the table would use approximately 24GB of memory (which would need to be loaded into memory from disk each time the network was

run). This memory requirement was several orders of magnitude more than the network previously required, and hence this optimisation was not used in the final implementation.

Another component of the spiking network implementation is its `visualise_input()` function, which produces a video of the network while processing a given input sample. This video shows the neurons and synapses of the network in two dimensions, in a graph layout that is constructed using Python's NetworkX library [72]. Each timestep of the network corresponds to one frame of the video, and a framerate argument can be passed to the `visualise_input()` function to control the duration of the video. Python's multiprocessing library [73] is used to generate the frames as separate images in parallel with NetworkX's `draw()` function, and the OpenCV library's `VideoWriter()` function [74] is used to compile them into a video file, where the simulation time is annotated onto each frame. This functionality proved very useful for studying the network dynamics on the MAGIC task, chapter 5, and for identifying bugs in the network during development.

Sample code for this spiking network implementation is discussed in section C of the appendix.

4.2.3 Classification tasks

A classification task in Python is typically provided as a 2D array of training data `X` and a 1D array of associated class labels `y`, which are usually split into separate training and testing data (where the training data may also be further split into training and validation datasets). The data is also run through a set of preprocessing steps, where

1. each feature (column of `X`) is normalised so it lies in $[0, 1]$;
2. for the spiking network, the input is converted into a firing rate for the Poisson input neurons in the network by multiplying `X` by the maximum firing rate; and
3. the classes are balanced by undersampling from over-represented classes (such that each class is represented an equal number of times in `y`).

Given a new classification task, the first step is to perform parameter sweeps for both the spiking and traditional networks. These sweeps are defined by specifying a map from each parameter to a list of values to trial. An example is shown in the code snippet below, which was the initial sweep definition for the MAGIC task.

```
PARAM_GRID = {
    "tau_u": [0.005, 0.01, 0.02, 0.03],
    "tau_g": [0.0005, 0.001, 0.005, 0.01, 0.1],
    "tau_theta": [0.0005, 0.001, 0.005, 0.01, 0.1],
    "tau_x": [0.0005, 0.001, 0.005, 0.01, 0.1],
    "nu": [0.0001, 0.001, 0.01],
}
```

This is used to generate a list of parameter sets which contain every possible combination of listed values, and for each of these, a network is initialised with these values, then trained and tested on the classification task. An accuracy score is produced using the metrics module in the Scikit library for Python [75], and each result is written to a log file along with its parameter values.

Parameter set evaluations were run in parallel on Katana (UNSW's computational cluster) using Python's multiprocessing library [73], making use of semaphores to limit the number of parallel processes and ensure that only one process was writing to the results file at a time. Once the parameter sweep had finished, a separate script was run which read the results file and produced heatmap visualisations, which were used to define the next parameter sweep iteration, or to identify an optimal region of the parameter space. The comparison experiments were also run on Katana, where each network evaluation (a separate evaluation was performed for each training dataset size) was run as a parallel process, the results written to a results file, and a separate script used to produce plots of the results.

To ensure that the parameter sweep rules out networks that learn an empty or trivial representation of the problem, some post-processing was applied to the output vector of predictions:

- any empty predictions, where the network returns `None` instead of an integer class label, were converted into a wrong label to ensure this sample contributed to the accuracy score as a misclassification; and
- if the network predicted the same class for every sample (which in a class-balanced binary classification task would yield a 50% accuracy score), its accuracy score was set to zero.

Chapter 5

MAGIC classification task

This chapter details the first experiment of the study, which uses a small, benchmarking classification task to compare the performance of an artificial spiking network with a traditional network of the same architecture. The reason for starting with such a small task is that it allows for networks with a small number of nodes to be used, which permits more detailed visualisations of the connection weights within these networks during learning. A parameter optimisation was first performed for both network types before the results of the comparison experiment (the metrics for which were introduced in section 4.1.7) and the insights they offer into the benefits of processing with a spiking network are explored.

5.1 MAGIC dataset

The Penn Machine Learning Benchmarks package, produced by the Computational Genetics Lab at the University of Pennsylvania, is a collection of benchmark datasets sourced from a wide range of applications, which are suitable for evaluating and comparing machine learning algorithms [76]. The datasets in this package cover both classification and regression tasks, with a variety of input types and sizes, including a breast cancer classification task, a wine quality dataset and the MAGIC particle classification problem, outlined below.

There are 162 classification datasets in the package, with the number of input features ranging from two to 1,000 and the number of samples ranging from 32 to 1,000,000. To choose a simple dataset suitable for benchmarking the performance of the spiking neural network and later the neurogenesis mechanism, five constraints were defined:

- to allow the neurogenesis mechanism to develop a variety of different network structures (by varying the parameters of the mechanism), there should be at least five input features;
- to keep the required network size small (so simulation can be fast and a visualisation of the network architecture can be interpreted), there should be at most 15 input features;
- to give the neurogenesis mechanism enough training time to recognise patterns of behaviour, there should be at least 2000 samples;
- to allow for the entire dataset to be used for training, validating and testing, without requiring an extensive runtime, there should be at most 20,000 samples; and
- to keep network evaluation simple, there should only be a binary output.

The best dataset in the benchmarks package that lies within these constraints is the MAGIC (Major Atmospheric Gamma Imaging Cherenkov Telescope project)

dataset. Each sample in the data corresponds to one set of Monte-Carlo simulated high-energy gamma particle shower measurements (by an atmospheric Cherenkov telescope), and the objective is to classify the particle type as gamma (a real signal) or hadron (background noise) [77].

This dataset has ten input features, all of which are real numbers (which vary in magnitude, so the input data should first be normalised), and 19,020 samples, with a slight class imbalance of $\frac{2}{3}$ hadron (corresponding to an output label of 0) and $\frac{1}{3}$ gamma (corresponding to an output label of 1).

5.2 Network architecture

To ensure the comparison between the spiking and traditional networks was fair, the same architecture was used for both network types, figure 5.1. This way, the only differences between the spiking and traditional networks were the neuron model and learning scheme, and hence any differences in performance could be attributed to these two qualities.

This architecture is small enough that visualisations of the networks after learning can be produced, to study the sparsity of the learned representations. This architecture contains 72 total connections, the weights for which can be analysed over a training session to study the rates at which the networks learn the task and converge to a steady state of their connection weights. A larger architecture

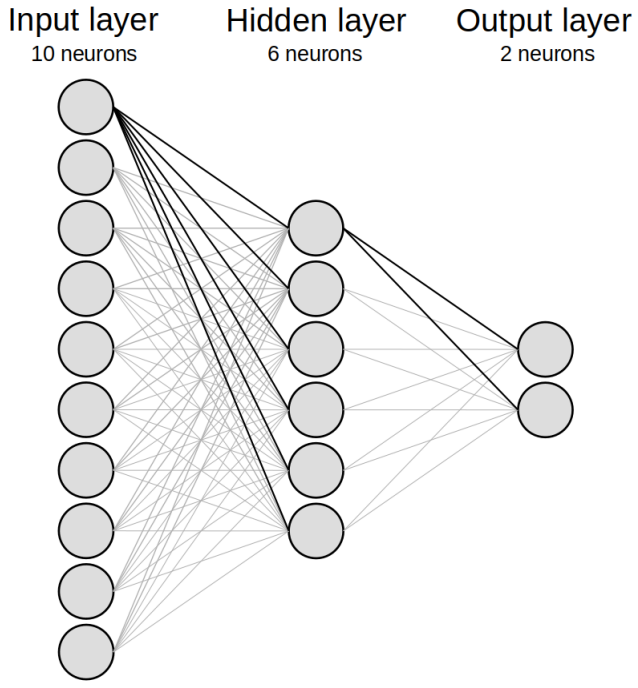

Figure 5.1: A visualisation of the feed-forward, fully-connected architecture used for both the spiking and traditional networks on the MAGIC classification task, with an input layer of ten neurons, a hidden layer of six neurons and an output layer of two neurons. The outgoing connections from the first node in the input and hidden layers are highlighted to demonstrate the full connectivity of this network.

would not permit easy visualisations of this evolution, and hence would make the comparison of these qualities of the networks significantly more difficult.

5.3 Parameter optimisation

To optimise the parameters of the spiking network for the MAGIC task, two iterations of parameter sweeps were required. The sweeps considered the five most significant network parameters: the membrane potential time constant τ_u, post-synaptic conductance time constants τ_{g_e} and τ_{g_i}, adaptive firing threshold time constant τ_θ, pre-synaptic trace time constant τ_x and learning rate ν. Each parameter set in the network was evaluated using 500 training samples and 200 testing samples, to ensure the reliability of the discovered optima. Figure 5.2 illustrates the second, final parameter sweep iteration. This parameter sweep suggests that the time constant of the pre-synaptic trace $\tau_x = 0.02\text{s}^{-1}$ is contraindicated, whilst

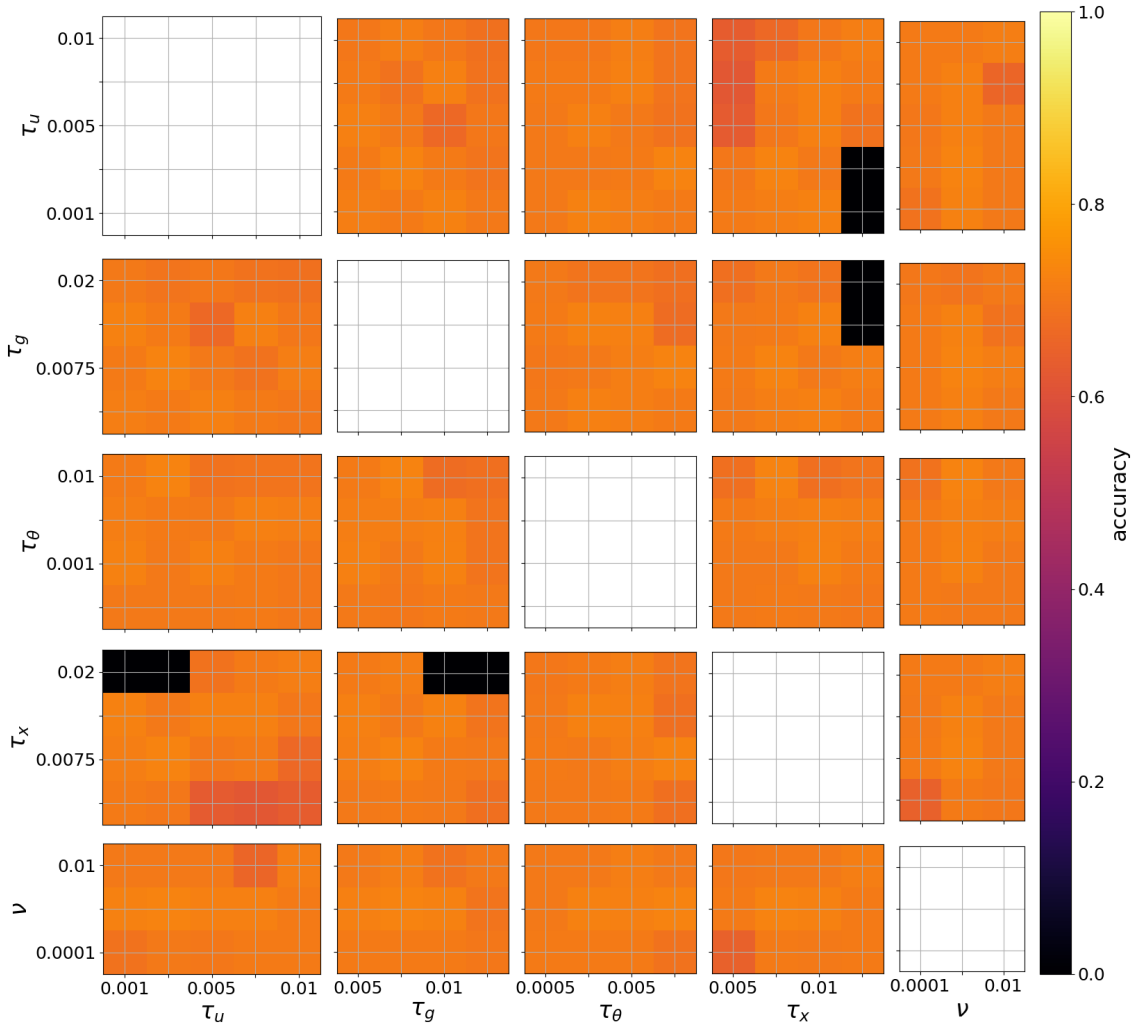

Figure 5.2: Heatmaps of the accuracy of the classifications for each pair of parameters in the spiking network. The maximum test accuracy score at each point is indicated by the colour. Note that the label τ_g is used to refer to the post-synaptic conductance time constants τ_{g_e} and τ_{g_i}, which were set to equal values in this sweep.

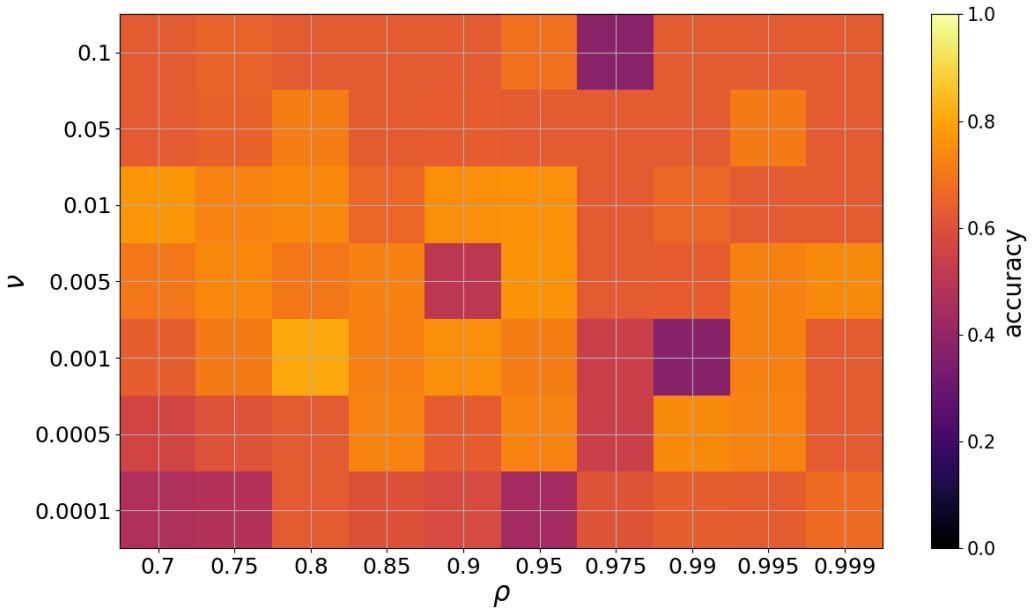

Figure 5.3: A heatmap of classification accuracy with learning rate ν on the vertical axis and momentum ρ on the horizontal axis, showing the test accuracy score indicated by the colour.

all other parameter values produce fairly similar best-case results. To understand why this value of the pre-synaptic trace hindered the network's ability to learn: it is useful to know that this value for τ_x was the largest value considered, which means that networks with this parameter value had the slowest decay of the pre-synaptic trace x_{pre}. Considering the spike timing dependent plasticity learning rule

$$\Delta w \;=\; \nu(x_{\text{pre}} - x_{\text{tar}})(w_{\max} - w)^{\mu},$$

synapse weights in these networks would strengthen more frequently (and by a larger amount) than those with faster pre-synaptic trace decay. The high value of τ_x may have resulted in these networks tending towards a very dense representation of the problem through their connection weights, as opposed to a sparse one, which spiking neural networks are known to prefer. Apart from this value for τ_x, however, the parameter sweep results suggest that the spiking network is, in general, not very sensitive to the values of its parameters. To validate this, several high-performing parameter sets from across the parameter space were included in the comparison against the traditional network.

To optimise the parameters of the traditional network for this task, only one parameter sweep iteration was required. The sweep considered the two most significant network parameters: the stochastic gradient descent learning rate ν and momentum ρ. To mirror the spiking parameter sweep, each parameter set in the traditional network was evaluated using the same 500 training samples and 200 testing samples that were used for the spiking network. Figure 5.3 illustrates the results of this parameter sweep. These results suggest that the traditional network is not very sensitive to these two parameters (but does demonstrate slightly more variance in its test scores than the spiking network, shown in figure 5.2), and hence

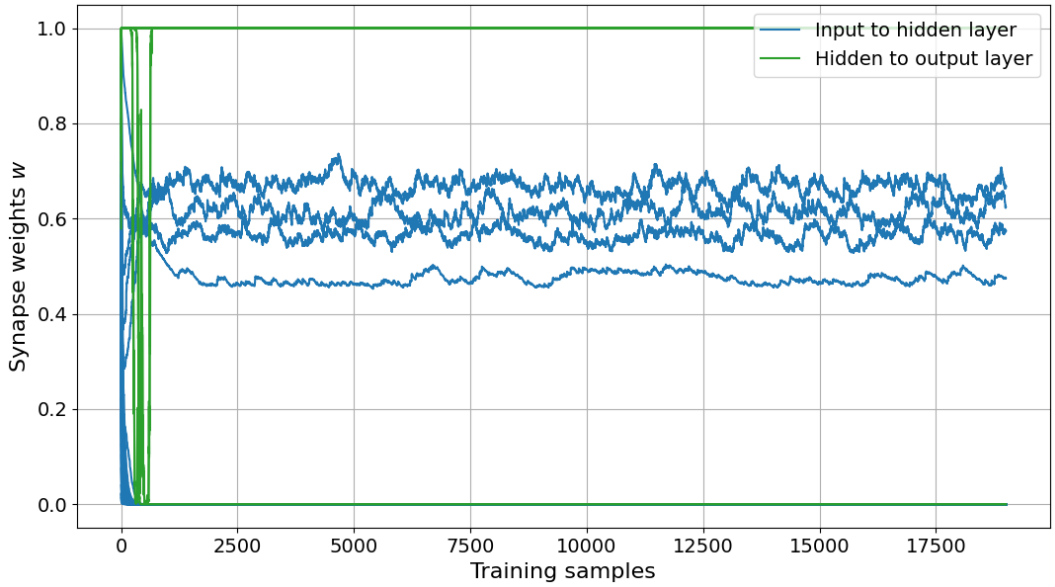

Figure 5.4: The connection weights of the spiking network with an optimal parameter set, over one epoch of the MAGIC dataset, where the connections between the input and hidden layers are shown in blue, and the connections between the hidden and output layers are shown in green. Note that many of the weights in both layers quickly became zero, and the non-zero weights between the hidden and output layers rapidly approached 1.

several high-performing parameter sets were chosen from across this two dimensional parameter space to include in the comparison against the spiking network.

5.4 Comparing traditional and spiking neural networks

One top-performing parameter set from the spiking network ($\tau_u = 0.001$, $\tau_{g_e} = \tau_{g_i} = 0.01$, $\tau_\theta = 0.001$, $\tau_x = 0.01$, and $\nu = 0.001$) and one top-performing parameter set from the traditional network ($\nu = 0.001$ and $\rho = 0.8$) were chosen to run over one epoch of the MAGIC training dataset (approximately 19,000 samples). The connection weights in these networks were then monitored over the training period, with the evolution of the spiking network weights as a function of the number of training samples depicted in figures 5.4 and 5.5, and the evolution of the traditional network weights depicted in figure 5.6.

The weights in this network take random initial values in the range [0,1], which is also the range they are able to adapt within. Notice a rapid adjustment of the hidden-output weights within the first 500-1,000 samples (which is shown in more detail in figure 5.5). After approximately 1,500 samples the weights appear to have reached a relatively stable steady state. It can be seen that the network has learned a very sparse representation of the problem, with only four out of the 60 input-hidden weights having a non-zero value in the steady state.

Notice in figure 5.5 that all hidden-output layer weights initially tend towards 1 (before some of them revert to zero later, as seen in figure 5.4), whereas most input-hidden layer weights tend towards zero at a slower rate. Figure 5.5 suggests that the classification efficacy of the network might be improved if the hidden-output

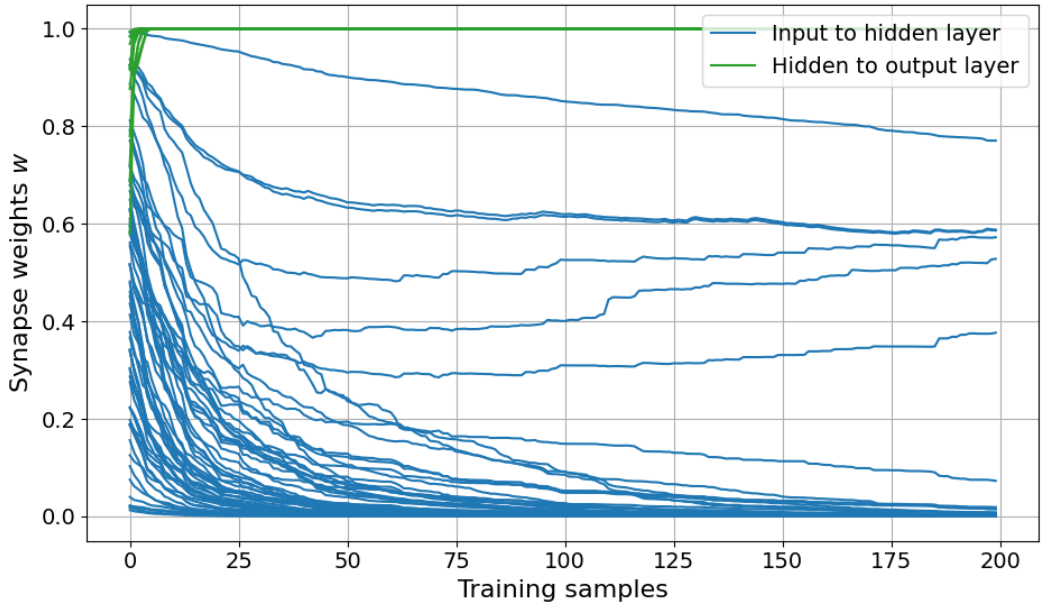

Figure 5.5: A detailed view of the early evolution of spiking network weights shown in figure 5.4. The connection weights of the spiking network with an optimal parameter set, over the first 200 training samples of the MAGIC dataset, where the connections between the input and hidden layers are shown in blue, and the connections between the hidden and output layers are shown in green. Note the rapid adjustment of synaptic weights in this early period of training.

layer weights were adapted at a different, much slower learning rate than those in the input-hidden layer.

The weights in the comparator traditional network take initial values in a small range centered on zero, but are able to take on any real value as the network learns. The evolution of the network weights as a function of the number of training samples is shown in figure 5.6. Notice that many of the network weights appear to still be slowly evolving near the end of the epoch (19,000 samples), unlike the spiking network, figure 5.4, which rapidly reached a steady state (with small variance in the non-zero and non-unity weights). The traditional network weights that were approximately in steady state at the end of the epoch showed more variance than those in the spiking network.

Both networks were able to learn the task with reasonably similar accuracies, figures 5.2 and 5.3, however, the spiking network reached its steady state approximately ten times faster than the traditional network. This steady state was a very sparse representation of the problem (utilising only four out of the 60 weights between the input and hidden layers – where a weight of one can be considered a straight through connection), whereas the traditional network utilised most of its weights to represent the information learned. The spiking network also demonstrated far less variance once its weights reached the steady state, which suggests that this network is more robust to perturbations such as varying the order of training samples and varying the initial weight values. Figure 5.7 shows an example of

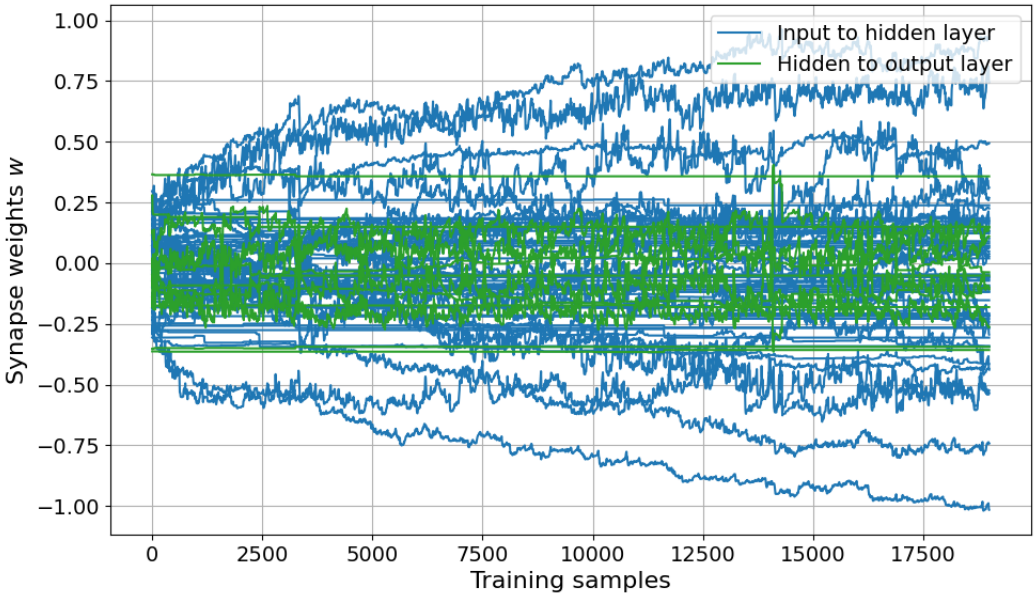

Figure 5.6: The connection weights of the traditional network with an optimal parameter set, over one epoch of the MAGIC dataset, where the connections between the input and hidden layers are shown in blue, and the connections between the hidden and output layers are shown in green.

a spiking network after training on the MAGIC dataset, which indicates the final weights and shows the sparseness of its learned representation.

Interestingly, only four of the ten input features (each associated with one of the input neurons) were used in the final network state to encode information about the training data. The others had all zero weights for their outgoing synapses, whilst five out of the six hidden neurons were utilised, with one having no outgoing connections.

The eight highest performing parameter sets from both the spiking and traditional networks were then taken and trained over a variety of dataset sizes: 100, 250, 500, 1,000, 2,500 and 5,000 samples, and evaluated on a fixed testing dataset of 500 samples. The objective of this experiment was to compare how quickly both network types tended to learn the MAGIC task, the levels of accuracy they were able to reach, and how sensitive these networks were to their parameter values. A further objective was to compare the connection utilisation or sparseness of the learned representations (fraction of non-zero weights). Figure 5.8 shows the test accuracy achieved by these networks against the number of training samples, whilst figure 5.9 shows the synapse utilisation of these networks against the training samples.

Notice in figure 5.8 that the spiking network showed consistently better accuracy than the traditional network, and had a smaller range of test scores for training set sizes of 100, 500 and 1,000. However, it had some significant underperforming outliers for 250, 2,500 and 5,000 training samples.

Notice in figure 5.9 that the spiking network demonstrated significantly lower synapse utilisation than the traditional network, keeping less than 20% of its original connections after training on 5,000 samples, compared to the traditional network's

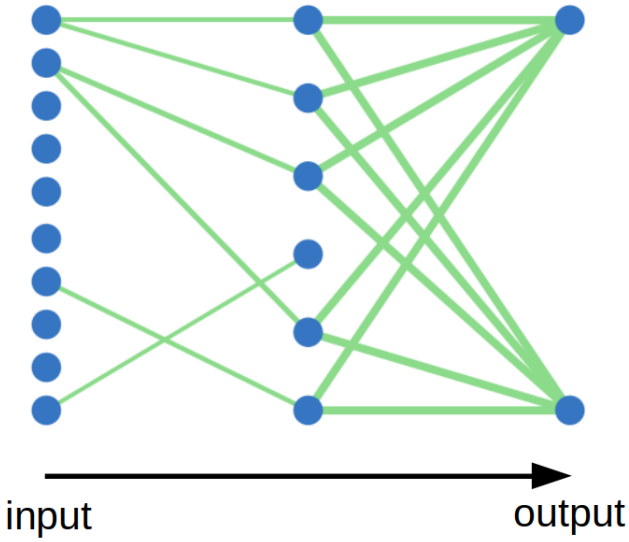

Figure 5.7: The sparse architecture of a spiking network after training on the first 1,500 samples of the MAGIC dataset, where the neurons are shown as blue circles and the thickness of the edges connecting them are proportional to the weight of the corresponding synapse (and hence, synapses with sufficiently small weights do not show in this representation of the data).

65%. Also notice that the average utilisation of the spiking network clearly decays as the number of training samples increases, whilst the traditional network demonstrates a very shallow reduction in utilisation. This very clearly demonstrates that the spiking network, with its unsupervised learning scheme, has learned a far more sparse representation of the problem than the traditional network.

On average, the spiking network required 0.4 seconds of CPU time and 1,415 spikes to process each input, which is several orders of magnitude more than the traditional network's 6×10^{-5} seconds of CPU time and just 18 spikes. This vast difference in processing time is likely a result of the custom implementation of the spiking network, which is clearly not able to compete with the heavily optimised PyTorch library (a result of it being a widely used, open source package). The very large number of spikes recorded in the spiking network may be a product of the sparse representation of the problem that it learned – the remaining synapses dominated during learning to the point where they were so efficient that they almost always result in post-synaptic spikes. This suggests that more research is needed to develop enhancements to the spiking network to encourage it to learn a representation of the problem that requires a minimal number of spikes (clearly the adaptive spiking threshold θ is not sufficient in this case).

This comparison experiment demonstrates that the spiking network learned the MAGIC task faster (in terms of number of training samples, not processing time) and more accurately, forming a significantly sparser representation of the problem than the traditional network. The spiking network outperformed the traditional network with respect to these performance metrics, despite using an unsupervised learning rule to tune its connection weights (where the only supervised component of its training was the labelling of its output neurons). Meanwhile, the traditional

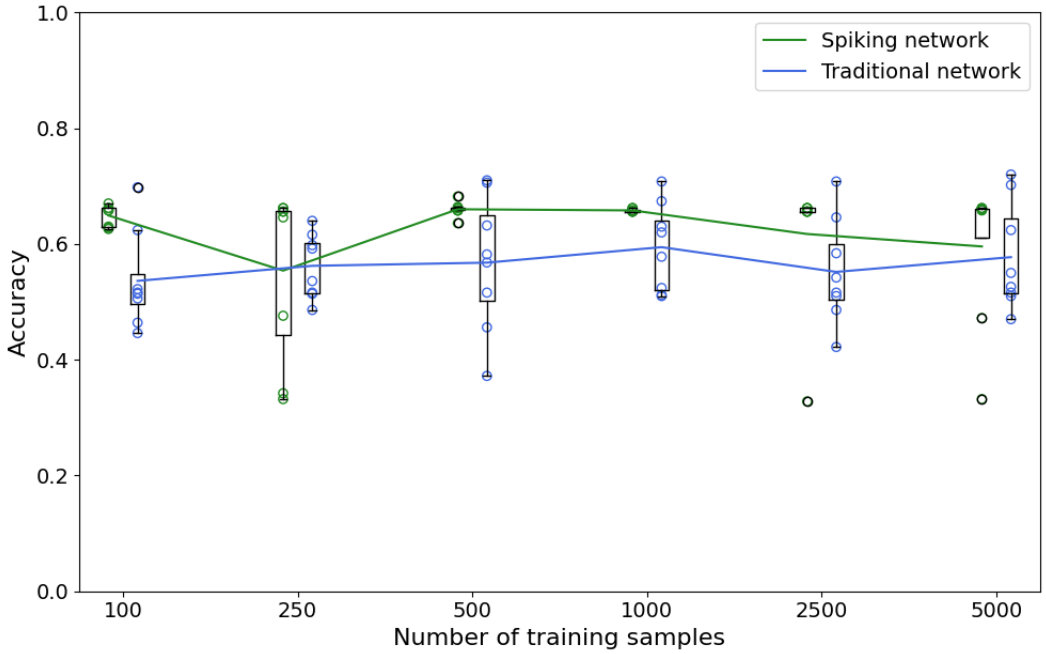

Figure 5.8: Test accuracy as a function of the number of training samples for the spiking (green) and traditional (blue) networks on the MAGIC task given different numbers of training samples. At each training set size, the spiking network accuracy scores and quartile box are shown in the left column, whilst the traditional network scores and quartile box are shown in the right column – note these correspond to the same number of training samples. The results for the individual parameter sets are shown as open circles for each network type. The lines indicate the average accuracy scores for each network.

network used a supervised learning scheme, adjusting its weights in the direction of steepest error descent during the training process, and yet required more training data to learn the task. Ignoring the drastically longer processing time of the spiking network, these results suggest that spiking neural networks are a more suitable choice for small classification problems such as the MAGIC task, especially when limited training data is available. However, in order to make them more feasible for widespread use, further work needs to be done in optimising the implementation.

An example spiking network visualisation on the MAGIC task from this project can be found online at: `https://vimeo.com/695135246`. This visualisation was produced by the custom spiking network implementation, as outlined at the end of section 4.2.2. This video shows the network processing one input from the MAGIC task for 350ms of simulated time. The nodes in blue correspond to neurons (which flash when the neuron emits an action potential), and the edges between them to synapses (where the shade of green corresponds to the weight w of the synapse, with a darker shade meaning a larger weight w).

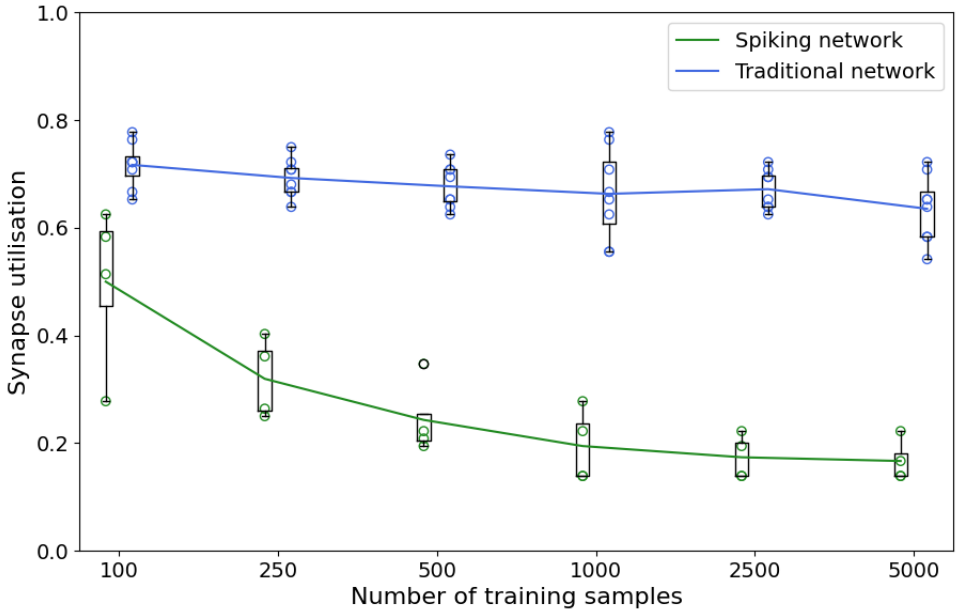

Figure 5.9: The synapse utilisation of the spiking (green) and traditional (blue) networks on the MAGIC task given different numbers of training samples. At each training set size, the spiking network utilisation scores and quartile box are shown in the left column, whilst the traditional network scores and quartile box are shown in the right column – note these correspond to the same number of training samples. The results for the individual parameter sets are shown as open circles for each network type. The lines indicate the average utilisation scores for each network.

5.5 Summary

The comparison experiment for the MAGIC dataset suggests that artificial spiking networks are better suited to small (both in terms of the number of features and the number of training samples) classification tasks than traditional, perceptron networks of the same size. This is evidenced by the spiking network's higher average accuracy score over the majority of training dataset sizes and its significantly faster convergence to a steady state of connection weights.

The spiking network's steady state demonstrated far less variance in weight values than that of the traditional network, which suggests that it is more robust to perturbations such as varying the order of its training data. The fast pace with which this more stable steady state was reached suggests that a discrete deflection to the spiking network's weight values would be corrected faster and with more accuracy than if such a deflection was made to the traditional network.

It is also clear that the spiking network learned a significantly more sparse representation of the problem, as reflected by the proportion of connections with non-zero weights after learning. The synapse utilisation scores in figure 5.9 indicate that the spiking network converged towards utilising less than 20% of its original connections, whilst the traditional network used about 65% of its connections after training on 5,000 samples. Assuming the implementation was appropriately optimised and deployed to appropriate hardware the spiking and traditional networks

would require similar amounts of memory and processing power per node and connection, then the spiking network for this task would be approximately three times as efficient for simulation than the traditional network.

These results may suggest one reason why the biological nervous system has evolved to implement a spiking neural network, as opposed to one comprised of binary thresholding units such as perceptrons. In order to survive, humans must be capable of learning tasks with very limited training data (such as learning to recognise animals as prey or predators). Therefore, the rapid pace at which the artificial spiking network reached its steady state (the point at which it "learned" the task to a reasonable level of accuracy) suggests that spiking networks are a good representation for information processing in the human brain. Further positives are that its sparse representation requires less energy to process information than a more dense network, and its lower utilisation of neurons means a brain with a fixed number of neurons can hold a greater number of different circuits for learning a greater number of tasks.

Chapter 6

MNIST digit recognition task

This chapter covers the second experiment of the study, which compares the performance of artificial spiking and traditional networks of approximately the same size on the MNIST digit recognition task, introduced in section 3.4.1. This task is significantly larger than the MAGIC classification task in chapter 5, both in terms of the size of its input and the number of training samples. The MNIST task is of particular relevance because digit recognition is a problem that the human visual system is clearly capable of learning, and hence a biologically plausible, artificial spiking network is expected to demonstrate a reasonable level of performance on this dataset.

A parameter optimisation was performed for both network types before the results of the comparison experiment were explored. There were significantly more connections in the networks used for the MNIST task compared to that for the MAGIC task, so it was not possible to fully explore the evolution of the weights in these systems within the scope of this project. Rather, a comparison of the accuracy and final synapse utilisation for high performing networks was explored. This chapter ends with a discussion of the results and the insights they offer into why the brain's visual system, which was first introduced in section 2.3, has evolved to implement a spiking neural network.

6.1 Network architecture

For this task, the network architecture proposed by Diehl and Cook [11], which was discussed in detail in section 3.4.2 and shown in figure 3.12, was used for the spiking network. The size parameter N was set to 100, making for excitatory and inhibitory layers of 10×10 neurons each, which was the smallest size Diehl and Cook considered in their investigation, but enabled results for this study within the time constraints.

To ensure the comparison between the spiking and traditional networks was fair, a fully-connected, feed-forward architecture for the traditional network was defined which had a similar size and structure, as discussed in section 4.1.2 and shown in figure 4.2. The number of input neurons is set by the size of the input data and the number of output neurons by the number of classes associated with the classification task. The size parameter N for the traditional network sets the number of neurons in each of the two hidden layers, and was also set to 100 for this network, meaning this traditional network had 994 neurons in total, just 10 more than the comparator spiking network.

Due to the significant increase in the number of neurons in the spiking network, from 18 in the MAGIC experiment to 984 in this experiment, the custom implementation of the spiking network detailed in section 4.2.2 was too slow to permit any extensive experimentation. To alleviate this issue, an equivalent implementation of the spiking network was developed using the Brian simulator [43, 4], which was based on an implementation uploaded by Peter Diehl to Github in 2015 [78] (with significant refactoring for this experiment, to improve clarity and performance). Unfortunately, the use of this type of implementation meant that further investigations of the effects of neurogenesis were not possible for this system. A detailed overview of the Brian simulator can be found in appendix section B, and sample code of this implementation is discussed in appendix section D.

6.2 Parameter optimisation

To optimise the parameters of the spiking network for the MNIST task, a small parameter sweep was defined, informed by the results of the smaller MAGIC classification task. An initial investigation trialled several optimal parameter sets from the MAGIC task (such as $\tau_u = 0.001$, $\tau_{g_e} = \tau_{g_i} = 0.01$, $\tau_\theta = 0.001$, $\tau_x = 0.01$, and $\nu = 0.001$) on this new, larger problem. Unfortunately the networks using these parameter sets all failed to learn the task and achieved zero accuracy.

Since it was infeasible in this project to thoroughly search a wider parameter space to identify a region where non-zero accuracy could be achieved, the second parameter sweep for this task was anchored around the optimal parameter values reported by Peter Diehl and Matthew Cook in their 2015 paper [11]. These values were

- an excitatory membrane potential time constant $\tau_{u_{\text{exc}}} = 100\text{ms}$;
- an inhibitory membrane potential time constant $\tau_{u_{\text{inh}}} = 10\text{ms}$ (this is different to the MAGIC task, where the membrane potential time constants for excitatory and inhibitory neurons were forced to be equal);
- an excitatory conductance time constant $g_e = 1\text{ms}$;
- an inhibitory conductance time constant $g_i = 2\text{ms}$ (also forced to equal the excitatory conductance time constant in the MAGIC parameter sweep); and
- an adaptive threshold time constant $\tau_\theta = 10^7\text{ms}$.

This parameter set is very different from any that were considered for the MAGIC task. Here, the time constant for the membrane potential of excitatory neurons is different to that for inhibitory neurons (by one order of magnitude), and likewise for excitatory and inhibitory conductances (by a factor of two). The time constant of the adaptive membrane potential is also set to a significantly larger value than what was considered previously, meaning in this network, the adaptive threshold would decay far slower than in the network used for the MAGIC task, which would promote a very sparse spiking behaviour.

The accuracy of the spiking network at parameters around the local neighbourhood of this point in the parameter space is shown in figure 6.1, where 72 total parameter sets were considered. Each network was trained on 2,000 samples from the MNIST dataset, and then evaluated on a further 200 samples to produce the accuracy score. Each of the 72 network evaluations comprising this parameter sweep took approximately 2 hours of CPU time to process.

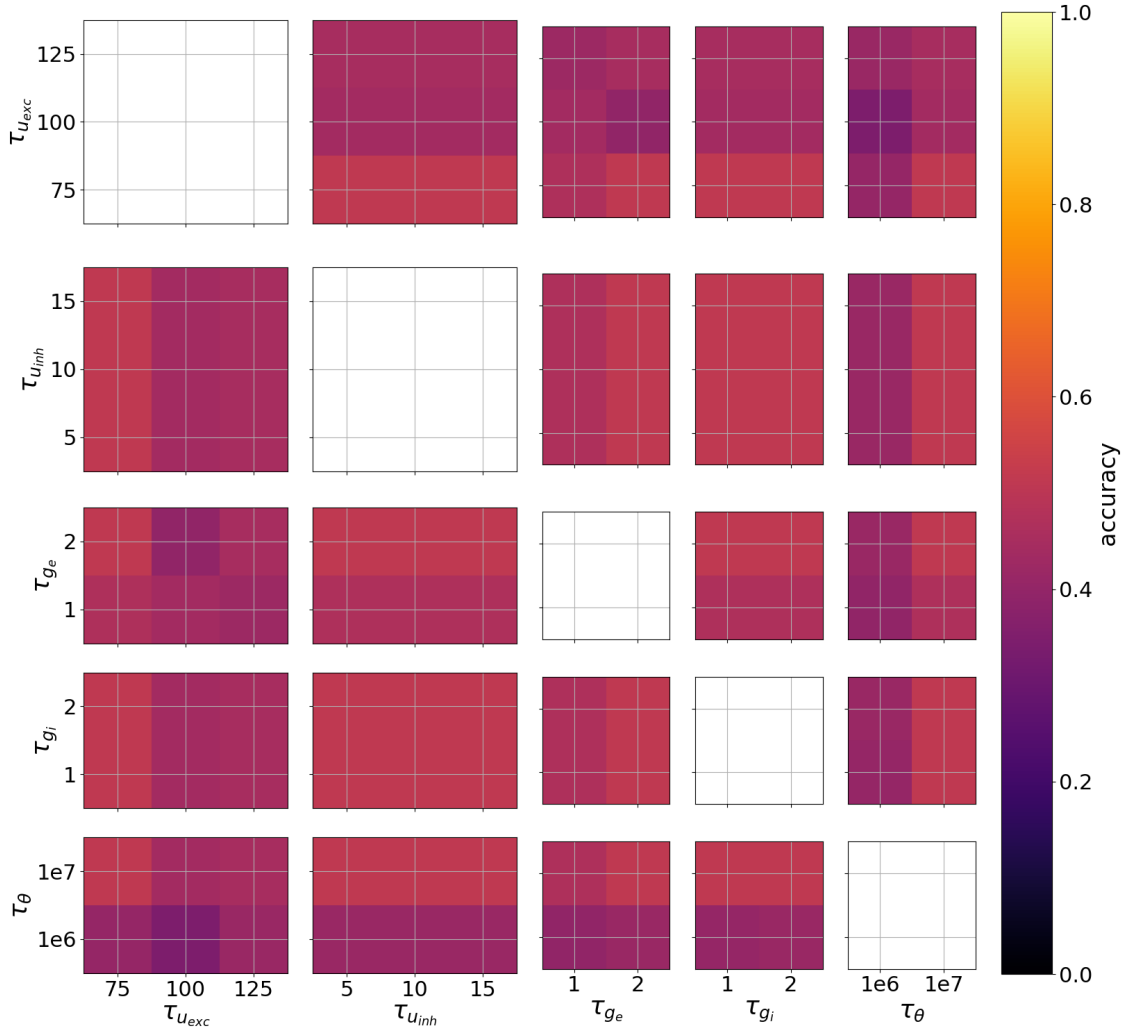

Figure 6.1: Heatmaps of the accuracy of the spiking network's classifications in the MNIST task for each pair of network parameters, showing the maximum test accuracy score at each point for the spiking network, indicated by the colour.

The results suggest that, within this region of the parameter space, the spiking network's accuracy is not particularly sensitive to its parameter values. The optimal values for the spiking network on this very small subset of training data were $\tau_{u_{\text{exc}}} = 75\text{ms}$, $\tau_{g_e} = 2\text{ms}$ and $\tau_\theta = 10^7\text{ms}$, whilst $\tau_{u_{\text{inh}}}$ and τ_{g_i} were free to take any value considered, since they do not appear to have a substantial effect on performance. This optimal parameter set differs from the one reported by Diehl & Cook, having a smaller time constant for the membrane potential of excitatory neurons and having τ_{g_e} take the value of 2ms instead of 1ms. This result is encouraging because a membrane potential time constant of 75ms is more biologically realistic than 100ms, and hence a network with this parameter value can still be considered biologically plausible. The accuracy scores achieved in this parameter sweep are in the neighbourhood of 50%, which is not a particularly high score, but it is reasonable when considering that the network only trained on 2,000 of the 60,000 available samples in the dataset.

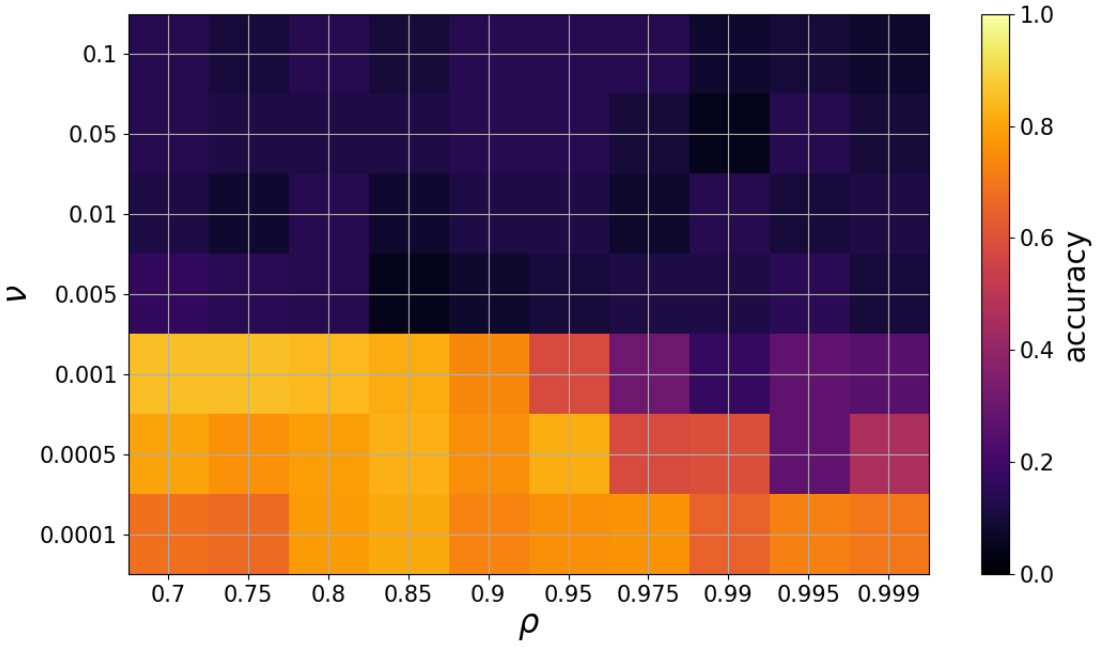

Figure 6.2: Heatmap of classification accuracy for the MNIST task using a traditional network, as a function of the learning rate ν and momentum ρ, where the score is shown by the colour scale.

To optimise the parameters of the traditional network for this task, only one parameter sweep iteration was required, exploring the sensitivity of the accuracy to changes in the learning rate ν and momentum ρ of stochastic gradient descent. To mirror the spiking parameter sweep, each parameter set in this network was evaluated using 2,000 training samples and 200 testing samples. Figure 6.2 illustrates the results of this analysis. These results suggest that the traditional network is quite sensitive to its two parameters, with its accuracy score varying between 10% and 90%, with a significant drop-off in performance observed when moving from a learning rate of 0.001 up to a learning rate of 0.005. The sweep suggests that smaller learning rates and smaller momentums are better for this task, with an optimal parameter set lying anywhere in the range $\nu \in [0.0001, 0.001]$, $\rho \in [0.7, 0.9]$. The highest accuracy scores achieved in this sweep were significantly better than the scores achieved by the spiking network, which suggests that the traditional network may generally outperform the spiking network for this task.

6.3 Comparing traditional and spiking neural networks

Mirroring the comparison experiment for the MAGIC task, four top-performing parameter sets from both the spiking and traditional networks were taken and trained over a variety of dataset sizes: 500, 1,000, 2,500, 5,000 and 10,000 samples, and evaluated on a fixed testing dataset of 500 samples. The objective of this experiment was to compare the ability of the two network types to learn the MNIST task, and the sparseness of their learned representations. Figure 6.3 shows the test accuracy achieved by these networks as a function of the number of training samples, whilst figure 6.4 shows the synapse utilisation of these networks.

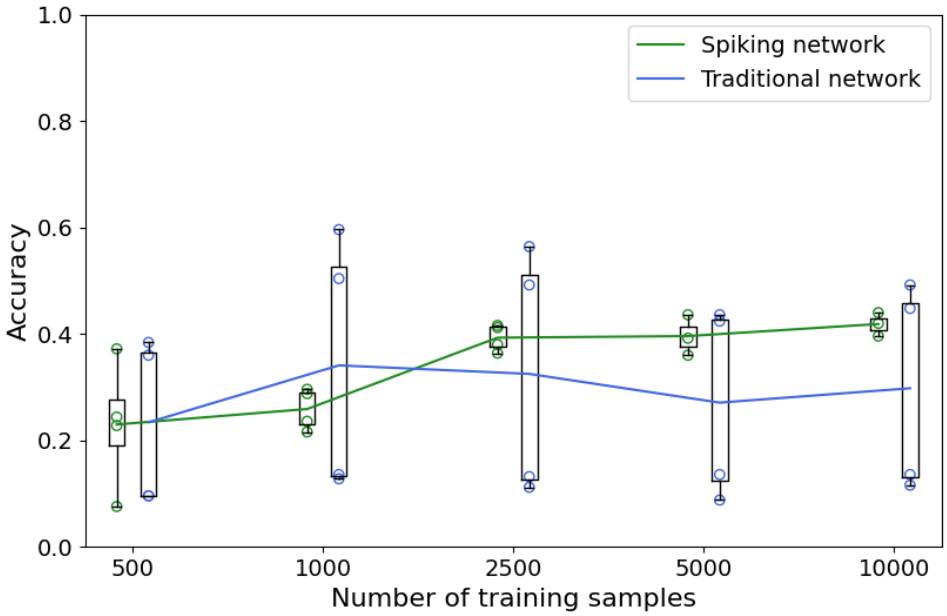

Figure 6.3: Test accuracy as a function of the number of training samples for the spiking (green) and traditional (blue) networks on the MNIST task given different numbers of training samples. At each training set size, the spiking network accuracy scores and quartile box are shown in the left column, whilst the traditional network scores and quartile box are shown in the right column – note these correspond to the same number of training samples. The results for the individual parameter sets are shown as open circles for each network type. The lines indicate the average accuracy scores for each network.

Notice in figure 6.3 that the spiking network's accuracy was on par with the traditional network here, despite both networks reaching lower scores than were suggested by their parameter sweeps (which hints that the specific data used for the sweep may have been easy to learn and possibly overfit). Just like the MAGIC task, the spiking network demonstrated far less variance in its accuracy scores across the board, compared to the variance of the traditional network accuracies. The traditional network accuracies showed a bimodal distribution at each training sample size, despite the results being from the top four performing instances of the traditional network (and not due to the dramatic change in accuracy observed in figure 6.2 with changes in the learning rate). The spiking network also demonstrated a slow improvement in performance as the number of training samples increased, unlike the traditional network.

Notice in figure 6.4 that both networks demonstrated very high synapse utilisations on this task, unlike those for the MAGIC task, where the spiking network converged to using only 20% of its connections on the largest dataset size. Here, the utilisation scores of the two network types were on par, but the spiking network showed a downward trend in its utilisation, with very little variance in the scores that were achieved. Meanwhile, the traditional network demonstrated a constant level of synapse utilisation across the board. Interestingly, there was a subset of spiking network instances that showed smaller utilisations for smaller sample sizes.

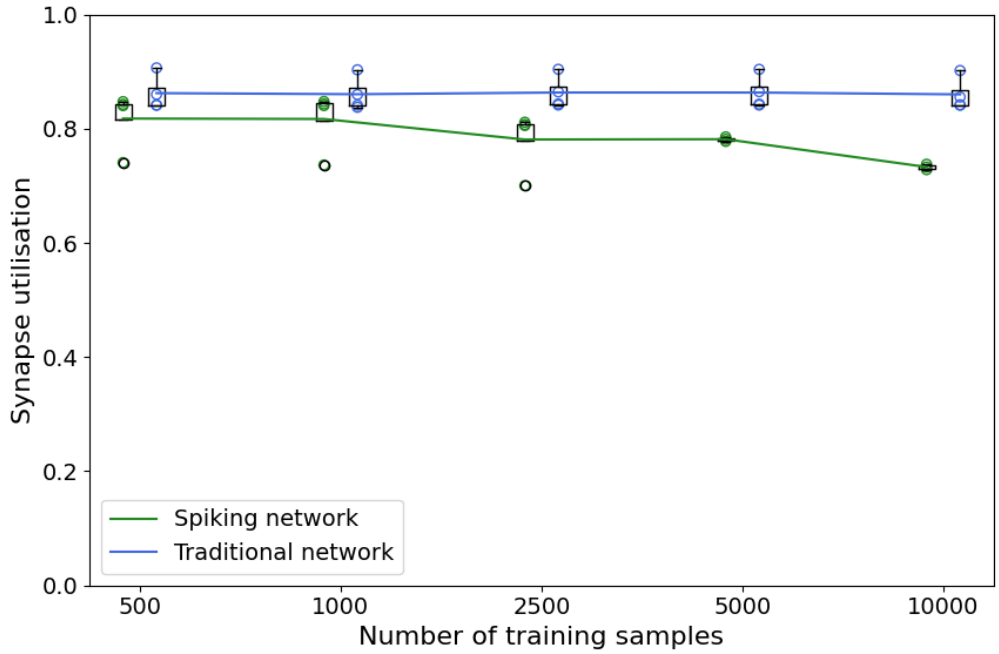

Figure 6.4: The synapse utilisation of the spiking (green) and traditional (blue) networks on the MNIST task given different numbers of training samples. At each training set size, the spiking network utilisation scores and quartile box are shown in the left column, whilst the traditional network scores and quartile box are shown in the right column – note these correspond to the same number of training samples. The results for the individual parameter sets are shown as open circles for each network type. The lines indicate the average utilisation scores for each network.

The accuracy of these networks was similar to the other spiking networks at these sample sizes, so this may indicate that an appropriate optimisation algorithm could be developed to further minimise the utilisation whilst maintaining accuracy.

On average, the spiking network required 4.75 seconds of CPU time and only 33 non-input spikes to process each sample, whilst the traditional network required just 1×10^{-5} seconds of CPU time, but a much larger 210 non-input spikes. Just like with the MAGIC task, there is a five orders of magnitude difference between the runtime of the traditional and spiking networks, which is likely a result of the far greater complexity of the spiking network, and the lack of optimisation in its implementation when compared to the widely used PyTorch library for the traditional network. However, unlike the MAGIC task, the spiking network required far less spikes amongst its excitatory and inhibitory layers, which suggests that the spiking network's architecture for this experiment has succeeded in enforcing sparse spiking behaviour in the network, unlike the feedforward architecture that was used for the MAGIC task and the traditional networks here.

6.4 Summary

The comparison experiment for the MNIST digit recognition task demonstrates that the spiking network is able to perform at least on-par with the traditional network. The MNIST task is a significantly larger problem (in terms of the input size) than the MAGIC task which was previously studied. Despite this, the spiking network's accuracy showed little variance and a clear upward trend as the amount of training data provided to it increased, whilst its synapse utilisation showed a shallow downward trend. However, there is no clear evidence to suggest that the spiking network is capable of learning this task faster (with less training data) than the traditional network – the accuracy comparison, figure 6.3, suggests that the spiking network initially performed worse than the traditional network, and it took until 2,500 samples for its average performance to overtake its counterpart. Once again, ignoring the significantly longer processing time of the spiking network, these results suggest that spiking neural networks are a suitable candidate for small classification tasks in the field of computer vision, such as this MNIST digit recognition task.

These results may provide some insight into why the biological nervous system has implemented a spiking neural network for vision, as outlined in section 2.3. One clear advantage of the spiking network in the results is that it showed significantly sparser firing behaviour than the traditional network, requiring only 33 spikes from its excitatory and inhibitory layers of 200 neurons in total, over the entire simulation period of 350ms for each sample. This suggests that neurons in the excitatory layer learned to respond very selectively to specific types of input, thereby making for a simple output calculation via the majority voting scheme (the more spikes from these output neurons, the more "noise" when computing the final classification). This could be an advantage in the biological brain, where neurons in the primary visual cortex that spike in response to very specific types of visual input from the retina can form meaningful connections to downstream neurons in the visual cortex or other brain regions for higher-order processing or thought. Neurons that instead spike in response to a wide variety of visual input might struggle to maintain downstream connectivity as their firing rates or spike times would not encode relevant information.

Chapter 7

Neurogenesis for the MAGIC classification task

This chapter explores the final experiment in the study, which uses the small MAGIC classification task to compare the classification performance of an artificial spiking network with and without the neurogenesis mechanism defined in section 4.1.3. This task was chosen because it can be learned by small networks, as demonstrated in chapter 5, and permits the visualisation of evolving architectures during learning, as a result of neurogenesis. The simple, feed-forward, fully-connected network architecture investigated in section 5.2 and used for the original MAGIC experiment was also used for this experiment. Thus, the spiking network with neurogenesis was initialised with the same architecture as the spiking network without it, to ensure a fair performance comparison.

7.1 Parameter optimisation

The parameter optimisation for the spiking network with neurogenesis was made in two stages. Firstly, the parameters of the spiking network were explored, and then the parameters associated with the neurogenesis mechanism were adjusted to determine the resulting classification accuracy. The first stage of this optimisation was reported in chapter 5, and the parameters for the spiking network used in this experiment are fixed at the optimal values of $\tau_u = 0.001$, $\tau_{g_e} = \tau_{g_i} = 0.01$, $\tau_\theta = 0.001$, $\tau_x = 0.01$, and $\nu = 0.001$.

The exploration of the effects of neurogenesis considered the seven most significant neurogenesis parameters: the neuromodulator density time constant τ_n, cell death neuromodulator threshold n_d, neurogenesis cell production rate r_n, neurogenesis cut-off time $t_{\max}$, dependency of new synapses on distance parameter s_d, maximum number of synapses for newborn neurons $s_{\max}$ and the cell positioning method discussed in section 4.1.3. The accuracy of the classifications for each neurogenesis parameter set was determined using 500 training samples and 200 testing samples to ensure the reliability of the discovered optima. Figure 7.1 illustrates the classification accuracy of the networks as a function of the parameters. The accuracy results immediately suggest that the neurogenesis mechanism introduces a lot of variance into the performance of the spiking network. The best scores achieved with neurogenesis are on par with, or slightly below, those achieved by the spiking network without neurogenesis. The optimal subspace of the neurogenesis parameter space is comprised of small distinct regions neighbouring those with poor accuracy – indicating that the results could be quite sensitive to neurogenesis parameter values. This is in contrast to the performance as a function of the network parameters themselves, which was largely insensitive to changes in values, figure 5.2.

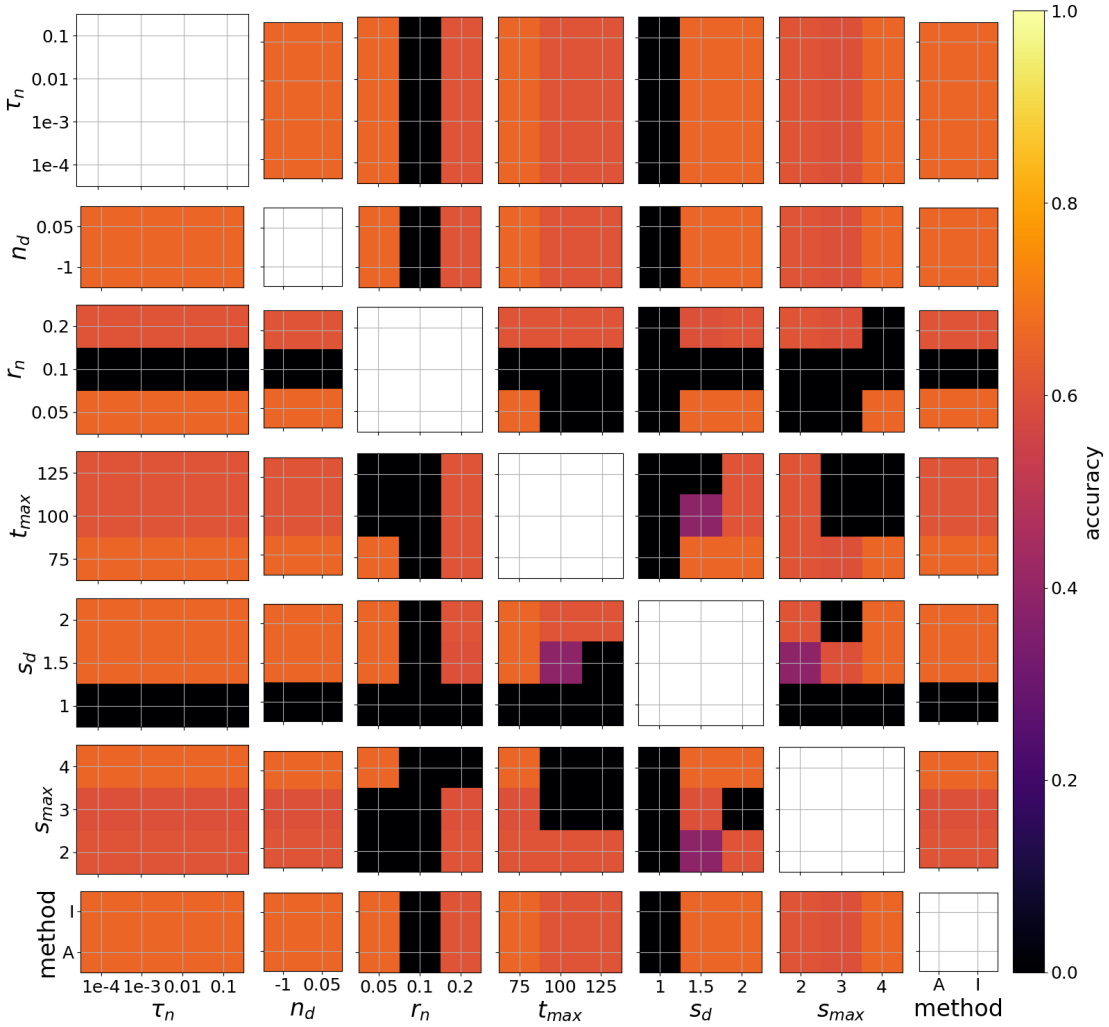

Figure 7.1: Heatmaps of the classification accuracy of the spiking network with neurogenesis, as a function of the neurogenesis parameters. The network parameters were fixed at the optimal values found in chapter 5 in all cases. The accuracy is shown in the colour scale and the method parameter takes one of two values, "A" for the active method and "I" for the inactive method of newborn neuron placement.

From these limited parameter sweep results it can be seen that a neurogenesis cell production rate r_n of 0.1 events/sec produces much poorer accuracy than a value of 0.05 events/sec, or a neurogenesis event every 20 seconds (approximately every 60 training samples). These results also suggest that the distance dependency parameter s_d used when positioning newborn neurons should take a value larger than 1 to enable classifications. Larger values of this parameter mean newborn neurons are less likely to form synapses with far-away mature neurons. This indicates that classifiers with reasonable accuracy are produced when meaningful connectivity can be formed in the spiking network of predominantly local connections. The last suggestion that can be inferred from these results is that, for reasonable accuracy, the maximum number of synapses $s_{\max}$ that a newborn neuron can form should be at least 4, as smaller values for this parameter consistently resulted in poorer performance. The architecture of the spiking network upon its initialisation has

all hidden neurons having 10 incoming synapses and 2 outgoing ones, and hence a newborn neuron with 2 or 3 synapses in total would likely struggle to spike at a rate comparable to the mature neurons in the network, due to its restricted set of possible inputs.

The optimal region within this neurogenesis parameter space is $\tau_n \in [0.0001, 0.1]$, $n_d = 0.05$ or -1 (where -1 means no cell death), $r_n = 0.05$, $t_{\max} = 75$, $s_d \in [1.5, 2]$, $s_{\max} = 4$ with either the active or inactive method. The spiking network parameters were fixed, and several high-performing networks with neurogenesis parameters from these sets were chosen for the comparison to the spiking network without neurogenesis.

7.2 Measuring benefit of neurogenesis to computation

The first investigation into the effectiveness of the neurogenesis mechanism was to visualise the architectures that were being produced, to evaluate the positioning of newborn neurons. For this, a spiking network with the parameter set $\tau_u = 0.001$, $\tau_{g_e} = \tau_{g_i} = 0.01$, $\tau_\theta = 0.001$, $\tau_x = 0.01$ and $\nu = 0.001$ was used, and then combined with several different optimal neurogenesis parameter sets, including $\tau_n = 0.0001$, $n_d = -1$, $r_n = 0.05$, $t_{\max} = 75$, $s_d = 1.5$, $s_{\max} = 4$ and the active method. Some interesting architectures that resulted are shown in figure 7.2.

The architectures shown in figure 7.2 all depict a newborn neuron with interesting connectivity. Figure 7.2a shows a newborn neuron being positioned in between the hidden and output layers, with an incoming synapse from the third hidden neuron and an outgoing synapse to the first output neuron. This newborn neuron therefore provides a parallel pathway between its connected hidden and output neurons. Figure 7.2b shows a newborn neuron being positioned within the hidden layer (which suggests that the neuromodulator levels of the output neurons were low in this case), with an incoming synapse from the third hidden neuron and an outgoing one to the fifth hidden neuron. This added pathway could serve to synchronise the spiking behaviour of the two connected hidden neurons. Figure 7.2c shows a similar connectivity to figure 7.2b, where the neuron's position has been slightly moved to the right by the neuromodulator levels of the ouput neurons. Figure 7.2d, which shows the architecture after 5,000 training samples, shows that the neurogenesis mechanism created three new neurons during training, but these were not able to integrate fast enough into the local circuitry, and thus any meaningful connectivity created for them was removed by spike timing dependent plasticity. The only remaining synapse likely resulted from the post-synaptic, newborn neuron spiking in response to large amounts of activity from the pre-synaptic neuron (at the bottom of the middle, hidden layer), allowing this connection to persist. However, this synapse ultimately serves no purpose in producing the network's output as the neuron has no outgoing synapses. These results suggests that the neurogenesis mechanism tends to produce newborn neurons with sensible positions and interesting connectivity patterns, but that these neurons struggle to integrate into the local circuitry and contribute to information processing, despite their temporary enhanced period.

Four top-performing spiking network parameter sets and four top-performing neurogenesis parameter sets were combined in all possible combinations (including

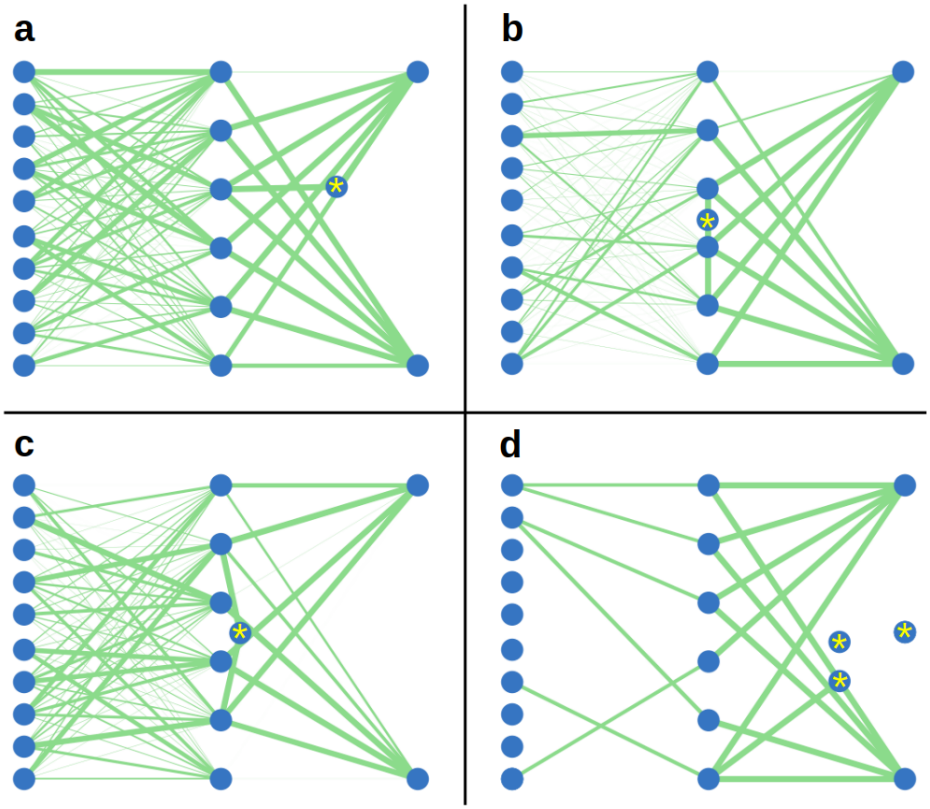

Figure 7.2: Four examples of the architectures formed by spiking networks with neurogenesis for the MAGIC task. The thickness of a synapse is proportional to its weight. The neurons are shown as blue dots. Newborn neurons, added by the neurogenesis mechanism, are indicated with an overlaid star. Architectures a, b and c were at the time of the first neurogenesis event in these networks, to study the quality of connections created for the newborn neuron, and architecture d was after training on 5,000 samples.

both networks with and without neurogenesis). These 20 networks were then trained over a variety of dataset sizes: 100, 250, 500, 1,000, 2,500 and 5,000 samples, and evaluated on a fixed testing dataset of 500 samples. The objective of this experiment was to measure the influence of the neurogenesis mechanism on the pace at which the spiking network learned the MAGIC task and the maximum accuracy achieved. Another objective was to determine the sensitivity of output to the neurogenesis parameter values within the optimal region identified in section 7.1. A further objective was to quantify the impact of neurogenesis on synapse utilisation and the sparseness of the network's learned representation of the task. Figure 7.3 shows the test accuracy achieved by these networks as a function of the number of training samples, whilst figure 7.4 shows the synapse utilisation of these networks.

Notice in figure 7.3 that in approximately 50% of the networks with neurogenesis, the accuracy score was zero, where the added neurons and synapses created by neurogenesis disrupted the network and drove it towards a trivial state of no connectivity. Considering only the non-zero accuracy scores, the neurogenesis mechanism still increased the variance in the spiking network's performance over that without

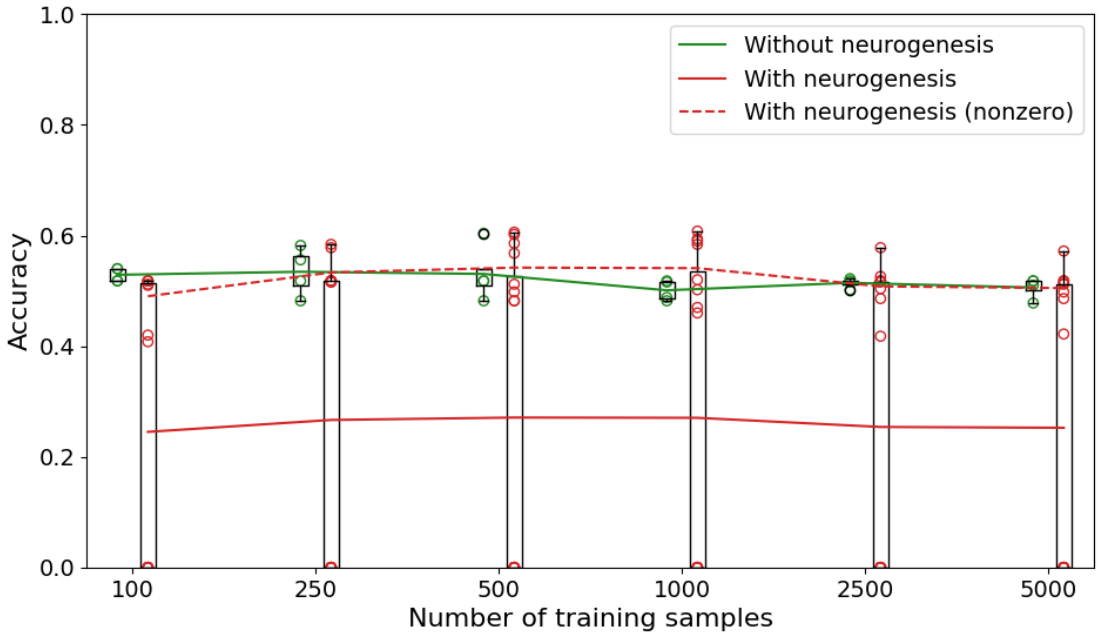

Figure 7.3: Test accuracy as a function of the number of training samples for the spiking network without neurogenesis (green) and with neurogenesis (red) for the MAGIC task as a function of the number of training samples. At each training set size, the without-neurogenesis accuracy scores and quartile box are shown in the left column, whilst the with-neurogenesis scores and quartile box are shown in the right column – note these correspond to the same number of training samples. The results for the individual parameter sets are shown as open circles for each network type. The lines indicate the average accuracy scores for each network, and the red dotted line shows the average non-zero accuracy scores for networks with neurogenesis.

neurogenesis. However, for systems with more than 1,000 training samples, the network with neurogenesis was able to reach a slightly higher peak accuracy score (for at least one parameter set) than without. This suggests that the neurogenesis mechanism has the potential to improve the spiking network, but is very sensitive to its parameter values, and clearly a much smaller optimal region in its parameter space would need to be identified to reduce the variance in the performance of these networks.

Figure 7.4 shows the synapse utilisation of the networks with and without neurogenesis for the MAGIC task as a function of the number of training samples. Notice that approximately half of the networks with neurogenesis converged to a trivial state of no connectivity (which appears as a zero utilisation score). However, considering the non-trivial networks, synapse utilisation in networks with neurogenesis was consistently higher than those without neurogenesis at all training sample sizes, which suggests that at least some of the new synapses created for newborn neurons persisted during training and may have contributed to computation. However, these extra synapses increase the complexity of the learned representation of

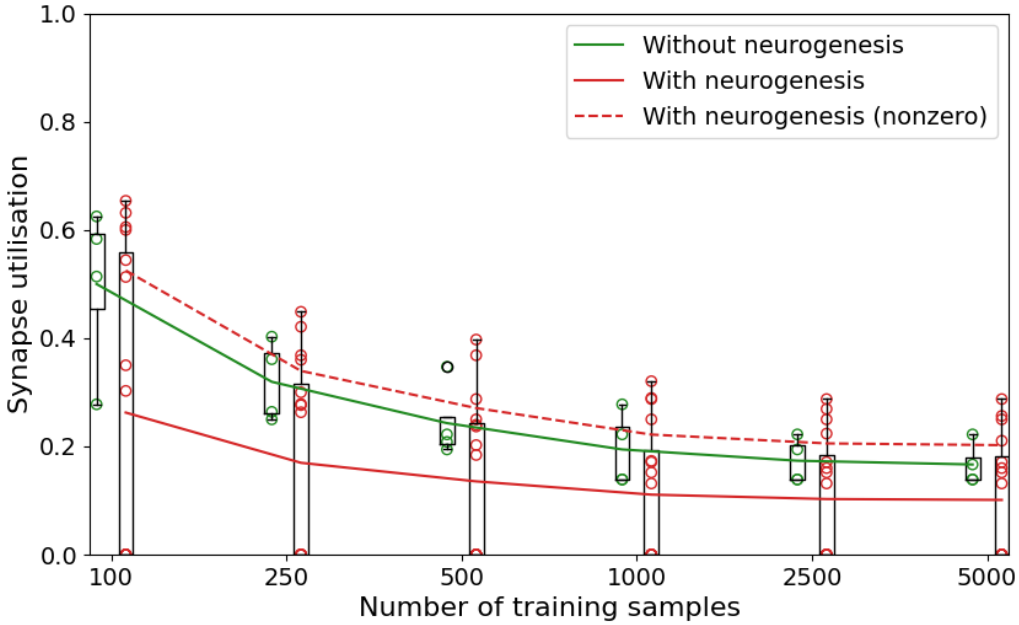

Figure 7.4: The synapse utilisation of the spiking network without neurogenesis (green) and with neurogenesis (blue) for the MAGIC task given different numbers of training samples. At each training set size, the without-neurogenesis utilisation scores and quartile box are shown in the left column, whilst the with-neurogenesis scores and quartile box are shown in the right column – note these correspond to the same number of training samples. The results for the individual parameter sets are shown as open circles for each network type. The lines indicate the average utilisation scores for each network, and the red dotted line shows the average non-zero utilisation scores for networks with neurogenesis.

the problem and therefore increase the computational resources (memory and CPU time) needed to process each input.

Recall from section 5.4 that on average, the spiking network required 0.4 seconds of CPU time and 1415 spikes to process each input. With neurogenesis, this was increased to 0.66 seconds of CPU time and 1574 spikes. The processing time increased by more than 50%, whilst the number of spikes increased by approximately 10%. The larger increase in processing time is likely a result of the neurogenesis mechanism adding extra computational steps: integrating the neuromodulator level n of neurons and running neurogenesis events (which involves computing with a vector neurodmoulator levels and iterating through all neurons in the network for creating new synapses). The increase in the number of spikes correlates with the results in figure 7.4, which showed that neurogenesis consistently increased synapse utilisation – more synapses in the network will tend to result in more spikes given the same input.

This comparison experiment demonstrates that the neurogenesis mechanism has some potential to improve the spiking network, but comes with multiple disadvantages. The mechanism is very sensitive to its parameter values, and has a bimodal effect on the network, where in approximately 50% of the cases studied, the mechanism caused an otherwise-optimal spiking network to converge to an empty state of

no connectivity. Even considering only the non-trivial cases, the variance in the test accuracy scores increased significantly when neurogenesis was added, the sparseness of the learned representation of the problem was reduced, and the runtime of the network increased by more than 50%.

An example spiking network visualisation with neurogenesis from this study for the MAGIC task can be found online at: `https://vimeo.com/695141414`. This visualisation was produced by the custom spiking network implementation, as outlined at the end of section 4.2.2. This video shows that the neurogenesis mechanism in this case has added three new neurons to the network, which contribute to information processing, as they emit some action potentials during the 350ms of simulated time.

7.3 Summary

This experiment represents an initial investigation into the efficacy of the biological mechanism of neurogenesis to artificial spiking networks undertaking classification tasks. The results suggest that more work needs to be done to improve the mathematical model that was designed to replicate this mechanism, which currently only results in a minor performance gain for a small set of parameter values.

The investigation showed that the mechanism is very sensitive to its parameter values, as very similar parameter sets can have drastically different impacts on the spiking network during training. In approximately 50% of cases, the mechanism has a small impact on the network's performance, increasing the variance in its accuracy scores, and only sometimes improving them; and in the other 50% of cases, the added connectivity disrupted the previously stable network state and drove the network towards a trivial state of no connectivity. Further investigations need to be made to better understand why the neurogenesis mechanism is having this impact on computation, in order to improve its mathematical model to avoid these scenarios.

These results may provide some insight into why neurogenesis primarily occurs during the development of the embryo in the biological brain, rather than during adulthood. Perhaps spiking networks naturally do not respond well to changes in their structure (as was suggested by the results of this experiment), and therefore evolution converged towards most structural plasticity occurring before the brain needs to be fully functional (i.e., before birth). To improve the current artificial neurogenesis mechanism, further investigations into the known biology of adult neurogenesis in the human brain needs to be conducted, to further inform how newborn neurons should be positioned and integrated into the existing circuitry, in such a way that network function is undisturbed and performance steadily improves. An initial investigation of adult neurogenesis in the human brain can be found in appendix section A.

Chapter 8

Discussion and future work

8.1 Traditional vs spiking networks

The performance of spiking and traditional neural networks was explored for two different classification tasks, with the MAGIC task being much smaller in scale than the MNIST task. The smaller number of input features in the MAGIC data enabled the network architectures to have far fewer neurons and synapses, allowing this task to serve as a basis to compare the performance of spiking and traditional neural networks, as well as exploring the effects of neurogenesis in a spiking network undertaking classification.

The comparison between the spiking and traditional networks on the MAGIC task indicated that the spiking network consistently achieved higher average accuracy scores over all training dataset sizes (with less variance) and demonstrated much faster convergence to its steady state of connection weights than the traditional network. This steady state of weights appeared to be far more stable than the weights of the traditional network after one epoch of the training data, and was a far more sparse representation of the problem. One particular spiking network that was studied utilised only four of the 60 connections that were initialised between the input and hidden layers of the network. The wider comparison which evaluated several parameter sets showed a clear decay in the synapse utilisation of the spiking network, which, on average, used less than 20% of its original synapses after training on 5,000 samples, whilst the traditional network used approximately 65%. However, the spiking network took approximately four orders of magnitude more CPU time to process each sample, some of which can be attributed to the increased complexity of its neuron model, and the rest likely due to the lack of optimisation in its implementation compared to that for the traditional network. Whilst the spiking network was sparse in terms of its weights, it was not sparse in terms of its spiking behaviour – it required 1,415 spikes to process each input, whilst the traditional network required just 18.

The comparison experiment for the much larger MNIST digit recognition task saw the spiking network, with its specialised architecture, perform on-par with the feedforward traditional network, suggesting it to be a suitable alternative for this application. Some qualities of the spiking network on the MAGIC task remained – notably, its accuracy once again showed much less variance than that of the traditional network, suggesting that this network is not overly sensitive to its parameter values within an optimal region of its parameter space. Its synapse utilisation also remained consistently lower than the traditional network, and showed a clear downward trend, but was much higher here than on the MAGIC task, staying above

70% after training on 10,000 samples. The spiking network's performance demonstrated an upward trajectory when given more training data (towards a maximum of 10,000 samples), whilst the traditional network seemed to peak after training on just 1,000 samples. Further to this, the spiking network, on average, required only 33 non-input spikes to process each sample, which was much less than the traditional network's 210. This suggests that the excitation-inhibition architecture used for this task achieved its goal of promoting a very sparse firing behaviour in the network. However, the top performing outliers for the traditional network consistently outperformed the spiking network's best scores, and unlike on the smaller MAGIC task, there was no indication that the spiking network was capable of learning the MNIST task faster than the traditional network. The spiking network once again required several orders of magnitude more CPU time to process each sample.

The two comparison experiments suggest that the benefits of the spiking network over the traditional network for classification tasks are that

- it can perform at least on-par with a traditional network of the same size/ capacity, and tends to outperform it on smaller tasks;
- it is less sensitive to its parameter values within an optimal region of its parameter space;
- it consistently learns a sparser representation of the problem in terms of its synapse weights; and
- given the right architecture, can also demonstrate a sparser firing behaviour.

The most significant downside to the spiking network over its traditional counterpart is that it requires several orders of magnitude more CPU time to process each input, which constrains the size of parameter sweeps that can feasibly be performed for this network, and the number of samples that can be trained on during experimentation.

The results across these two experiments may provide some insight into why spiking neural networks for information processing have evolved in biological systems. Perhaps it is much more difficult to implement a perceptron-like binary unit in biology, or there are other biological constraints that drove evolution of biological neurons towards a spiking mechanism.

In this study, the spiking network learned the MAGIC task significantly faster than the traditional network, and on the MNIST task, it showcased its ability to continually learn as more training data was provided (whilst the traditional network showed no clear trend of improvement). These performance benefits, where the spiking network is able to learn faster and more consistently, may have provided an evolutionary advantage for this neural network implementation. A further advantage is that its sparse representation in terms of utilised neurons and connections requires less circuitry to implement, meaning a brain with a fixed number of neurons or capped density of connections can hold a greater number of circuits for learning a greater number of tasks. Given the right architecture, the sparsity of its firing behaviour also means the spiking network could require less energy to process information than its perceptron-based counterpart, assuming that spiking neurons and perceptrons in biology would require similar amounts of energy to fire an action potential. The spiking network's evidently lower sensitivity to its parameter values suggests that it could serve as a more stable implementation of a biological neural

network, where perturbations to the network's properties due to random mutations or developmental issues would be less likely to significantly impact the network's ability to learn.

8.2 Neurogenesis

The neurogenesis mechanism described in section 4.1.3 was evaluated on the MAGIC classification task, to assess its impact on computation in spiking networks. The mathematical model for neurogenesis was designed to mimic the broad process and qualities of the biological mechanism explored in section 2.5. The experimental results suggest that the model for neurogenesis requires significant improvements and further optimisation before it can be considered a reliable, positive addition to an artificial spiking network.

An analysis of the position and synapses of newborn neurons created by the neurogenesis mechanism, from the examples shown in figure 7.2, suggests that the model is responding appropriately to the network dynamics. In some cases, the newborn neurons and their added connections provided a parallel pathway for signals where connectivity already existed. This behaviour is biologically plausible, as the "two-stream" hypothesis proposed by Ungerleider and Mishkin in 1982 claims that two parallel pathways connect the V1 region in the visual cortex to the temporal cortex [61]. In other cases, the new synapses created an entirely new pathway between two mature neurons that were not previously connected, which is also biologically plausible, as during embryonic development, the brain undergoes a competition phase where redundant, parallel pathways are pruned, leaving mostly novel connections between neurons [62]. This suggests that the two methods for positioning new neurons (called "active" and "inactive"), and the method of connecting them to mature neurons in their local neighbourhood, are appropriate approximations of mechanisms in the biological brain.

Despite its biological plausibility, the experimental results demonstrate that the artificial neurogenesis mechanism is very sensitive to its parameter values, and has a bimodal effect on the spiking network during training (where very similar parameter sets may lie on either side of this effect). In approximately half of the cases that were studied, the mechanism had a very minor influence on the network's performance, increasing the variance in its accuracy scores and sometimes improving it (the maximum accuracy achieved with neurogenesis was 5-10% higher than without it). In the other 50% of cases, neurogenesis disrupted the otherwise stable spiking network and drove it towards a trivial state of no connectivity and hence a zero accuracy score. No clear explanation for this behaviour, and which parameter values were causing it, could be found at this stage. Further investigations need to be made to better understand why the neurogenesis mechanism is having this impact on computation, and whether this result is particular to the small network architectures explored, in order to improve its mathematical model to avoid these scenarios.

Despite the artificial neurogenesis mechanism not being a successful addition to the spiking network in its current state, the experimental results may provide some insight into why the biological neurogenesis mechanism is most active during embryonic development, and occurs at a much slower rate during adulthood. The

parameter sweep results suggest that a smaller value of the $t_{\max}$ parameter, which is the cut-off time for neurogenesis events during training, is better for network performance. Perhaps spiking networks in general do not respond well to structural changes during learning, which would explain the experimental results, and may also explain why our brains evolved such that most structural plasticity occurs before the brain needs to be fully functional (i.e., before birth). The artificial neurogenesis mechanism could benefit from a further investigation into the qualities of adult neurogenesis in the human SVZ and SGZ brain regions, to learn about the positioning, connectivity and maturation of newborn neurons in adults. Such an investigation could give insight into how biological neurogenesis is able to operate without significantly disturbing network function – and these lessons could then be used to enhance the mathematical model for neurogenesis in an artificial spiking network.

8.3 Suggestions for future work

Following on from this research, several suggestions for future work can be made. The first is to apply the spiking and traditional network comparison experiment to larger classification tasks, such as the CIFAR-10 dataset [79]. This is another popular benchmarking dataset in the computer vision field that consists of 60,000 32×32 colour images split into 10 classes such as "automobile" and "dog". A step-up from this is the CIFAR-100 dataset of the same size, containing 100 classes, each represented by 600 images in the dataset. Learning these problems would likely require an increase in the size of the network architecture (such as increasing the size of the non-input layers in the architecture used for the MNIST task from $N = 100$ to $N = 400$) and an increase in the amount of training samples provided to the network, possibly requiring multiple epochs of the training data to converge to a final network state. This is a challenge, considering that the spiking network for the MNIST task required almost 5 seconds of CPU time to process each sample. Assuming a four-fold increase in processing time to handle the CIFAR-10 dataset, training on all 60,000 samples would require approximately two weeks of CPU time. Hence, applying the spiking network to larger classification tasks would require significant optimisation of its implementation, in order to make this a computationally feasible exercise.

Learning tasks other than classification problems could also pose an interesting area of further research, to compare the qualities of the spiking network against traditional, state of the art approaches on tasks that involve processing sequences of information (as opposed to individual samples), or even transfer learning tasks. The field of natural language processing includes many popular sequential processing tasks, such as machine translation and sentiment analysis from streams of text, where some state of the art traditional approaches are the highly recurrent long short-term memory network [80] and autoencoder [81] models. Artificial spiking networks may be suited to these tasks because of the manner in which they process information over simulated time, which would allow for sequences of input to be fed to the network at regular intervals during a single simulation period, with the network processing continuously between each input. This naturally allows for important context to remain encoded within the network's spike times or firing

rates as it processes new information. Transfer learning problems involve a machine learning algorithm training on one dataset before cutting over and training on another, and then evaluating its performance on both tasks, to quantify how much knowledge of the first task was retained and how much knowledge of the second task was able to be absorbed. The biological brain appears to be adept at such tasks, as humans are able to supplement knowledge gained in one context with knowledge gained in another. The experimental results in this thesis demonstrate that spiking networks tend to learn very sparse representations of classification problems, which suggests that a spiking network could handle multiple circuits wired over one pool of neurons, and therefore may perform well on transfer learning tasks, given an appropriate architecture.

Another suggestion for future work is to investigate different spiking and traditional network architectures, such as convolutional neural networks, where connections between nodes in some pairs of adjacent layers share weights to implement a convolution of the input, rather than a regular connection scheme where all weights are independent. This architecture may improve the spiking network's ability to learn visual classification tasks such as the MNIST dataset explored in this thesis, because the convolution operation facilitates the efficient processing of spatially distributed input. Comparing such a spiking network to traditional, convolutional networks would also increase the relevance of this comparison experiment, as these networks are closer to the state of the art for classification in the computer vision field.

Expanding the investigation of the artificial neurogenesis mechanism presented in section 4.1.3 to larger classification tasks such as the MNIST digit recognition task and the CIFAR-10 dataset is another suggestion for future work. The focus of this investigation would be to determine whether the bimodal results observed for the spiking network with neurogenesis for the MAGIC task persist in larger architectures. Variations to the current neurogenesis mechanism could also be explored, such as a mechanism that splits the most active (in recent memory) synapse in the network by placing a newborn neuron in the middle of it, and connecting this neuron to the original pre and post synaptic neurons. Connecting this newborn neuron to other neurons in its local neighbourhood would allow this previously isolated connection to receive input from, and send output to, several local neurons in the network. This mechanism could alternatively split the synapse whose weight w recorded the largest magnitude of change since the last neurogenesis event, which would facilitate replacing synapses that have not yet reached a steady state.

A further possibility is to consider different structural plasticity mechanisms for the artificial spiking network – one suggestion is to model the formation of new synapses between existing neurons, rather than the creation of new neurons. This could be implemented by extending the philosophy of Hebb's postulate, essentially "cells that fire together wire together" [29] from a plasticity rule that tunes the weights of existing synapses to a mathematical model which forges new connections. This could connect neurons based on their physical proximity and the similarity of their firing patterns, possibly by evaluating the "distance" between neurons in some metric space that combines position and firing activity. Such a model, if not carefully implemented, could result in a significant increase in the memory or CPU footprint of the simulated network, since all $n(n-1)$ possible directed connections

between n neurons in a network must be considered. The modern neuroscience understanding of the process of synapse creation suggests that the target neuron (towards which an axon is growing) can assist with forming the connection [62]. It achieves this by releasing chemicals such as nerve growth factor for the axon to absorb via receptors, which can be used by the pre-synaptic neuron to power the growth of the axon. This suggests an implementation of this mechanism by modelling attraction/repulsion forces based on the aforementioned factors.

A final suggestion is to investigate alternative neuron models with a dual focus on biological accuracy and suitability for simulation. One example of such a neuron is the spike response model [82], which is a generalisation of the leaky integrate-and-fire neuron that includes a refractory period and a variable spiking threshold by default, and has previously been fit to experimental data of biological neurons [83]. Another example is the Galves–Löcherbach model [84] which is a stochastic neuron, where the likelihood of firing depends on a potential which is simply a weighted sum of previous spikes of other neurons in the network, which is reset to zero when the neuron spikes. A further idea is to experiment with compartmental neuron models [85], which are popular with computational neuroscientists, where dendrites are the focus of modelling (which can be split into multiple connecting segments or treated as continuous cables), as their branching structure determines how synaptic input is integrated towards the soma. An extended component of this study could be to develop a new neuron model that preserves the biological plausibility of these existing models, but trades some unneeded accuracy for simulation efficiency. An early idea for this endeavour is to substitute simple difference equations for the differential equations in the leaky integrate and fire neuron.

Chapter 9

Conclusion

Recent literature on artificial spiking networks has demonstrated that they tend to underperform when compared to traditional networks, both with respect to the usual performance metric of accuracy, and also when considering training time [10, 11]. The experimental results in this thesis suggest that the first of these claims is not the case – on both classification tasks considered, the spiking network achieved average accuracy scores that were at least on-par with the traditional network of the same size/capacity, and clearly outperformed it on the relatively small MAGIC task. This is a surprising result, given that the spiking network was trained with an unsupervised learning scheme, whilst the traditional network learned via stochastic gradient descent of its connection weights with respect to a cross-entropy loss function. Further to this, the spiking network demonstrated other advantages over its traditional counterpart – less sensitivity to its parameter values within an optimal region of its parameter space, and its tendency to learn a consistently sparser representation of the problem in terms of its connection weights. However, the second of the claims in the literature, that spiking networks require significantly more training time to learn a given task, was validated by the experimental results in this thesis. The spiking network required several orders of magnitude more CPU time to process each sample during training and testing than the traditional network, but perhaps this disadvantage could be lessened by pursuing a more heavily optimised spiking network implementation.

The investigation into adapting the spiking network architecture using neurogenesis implemented a model for the production and placement of newborn neurons, and the development of their connectivity. An analysis of the position and connectivity of newborn neurons created by this mechanism during learning validated that the model was producing varied and biologically plausible results. However, in terms of classification efficacy, the mechanism had a bimodal effect on the spiking network during training – in half of the cases studied, neurogenesis had a very minor impact on classification accuracy and sometimes improved it, but in the other half of cases, the insertion of new neurons drove the network towards a trivial state of no connectivity and hence a zero accuracy score. These results suggest that the neurogenesis model requires more improvements and optimisation before it can be considered a reliable, positive addition to an artificial spiking network.

References

[1] S. Ghosh-Dastidar and H. Adeli, "Spiking Neural Networks," *International Journal of Neural Systems*, vol. 19, no. 04, pp. 295–308, 2009.

[2] M. A. Nielsen, *Neural Networks and Deep Learning*, vol. 25, ch. 1. Using neural nets to recognize handwritten digits. Determination Press, San Francisco, CA, 2015.

[3] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics from single neurons to networks and models of cognition*, ch. 1.3 Integrate-And-Fire Models. Cambridge University Press, online ed., 2014.

[4] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *ELife*, vol. 8, p. e47314, 2019.

[5] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics from Single Neurons to Networks and Models of Cognition*, ch. 2.2 Hodgkin-Huxley Model. Cambridge University Press, online ed., 2014.

[6] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[7] A. H. Marblestone, G. Wayne, and K. P. Kording, "Toward an integration of deep learning and neuroscience," *Frontiers in Computational Neuroscience*, p. 94, 2016.

[8] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, no. 2, pp. 245–258, 2017.

[9] N. K. Medathati, H. Neumann, G. S. Masson, and P. Kornprobst, "Bio-inspired computer vision: Towards a synergistic approach of artificial and biological vision," *Computer Vision and Image Understanding*, vol. 150, pp. 1–30, 2016.

[10] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[11] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015.

[12] G.-l. Ming and H. Song, "Adult neurogenesis in the mammalian brain: significant answers and significant questions," *Neuron*, vol. 70, no. 4, pp. 687–702, 2011.

[13] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[14] C. Henley, *Foundations of Neuroscience*, ch. 1. The Neuron. Michigan State University, 2021.

[15] C. Henley, *Foundations of Neuroscience*, ch. 2. Ion Movement. Michigan State University, 2021.

[16] D. A. McCormick, *Fundamental Neuroscience*, ch. 6. Membrane potential and action potential, pp. 112–131. Academic press, 3 ed., 2008.
[17] C. Henley, *Foundations of Neuroscience*, ch. 5. Postsynaptic Potentials. Michigan State University, 2021.
[18] C. Henley, *Foundations of Neuroscience*, ch. 10. Neurotransmitter Release. Michigan State University, 2021.
[19] C. Henley, *Foundations of Neuroscience*, ch. 8. Synapse Structure. Michigan State University, 2021.
[20] A. Y. Deutch and R. H. Roth, *Fundamental Neuroscience*, ch. 7. Neurotransmitters, pp. 133–148. Academic press, 3 ed., 2008.
[21] M. Victor, "Building a Neural Circuit in a Dish." `https://ki-images.mit.edu/2019/victor-1`, 2019. Accessed: 08/05/2022.
[22] S. Herculano-Houzel, "The human brain in numbers: a linearly scaled-up primate brain," *Frontiers in Human Neuroscience*, p. 31, 2009.
[23] C. Koch and I. Segev, "The role of single neurons in information processing," *Nature Neuroscience*, vol. 3, no. 11, pp. 1171–1177, 2000.
[24] R. Brette, "Philosophy of the spike: rate-based vs. spike-based theories of the brain," *Frontiers in Systems Neuroscience*, p. 151, 2015.
[25] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT press, 2001.
[26] M. A. Lynch, "Long-term potentiation and memory," *Physiological Reviews*, vol. 84, no. 1, pp. 87–136, 2004.
[27] R. S. Zucker, W. G. Regehr, *et al.*, "Short-term synaptic plasticity," *Annual Review of Physiology*, vol. 64, no. 1, pp. 355–405, 2002.
[28] R. C. Malenka and M. F. Bear, "LTP and LTD: an embarrassment of riches," *Neuron*, vol. 44, no. 1, pp. 5–21, 2004.
[29] D. Hebb, *The Organization of Behavior*. New York: Wiley & Sons, 1949.
[30] M. Taylor *et al.*, "The problem of stimulus structure in the behavioural theory of perception," *South African Journal of Psychology*, vol. 3, pp. 23–45, 1973.
[31] N. J. Emptage, C. A. Reid, A. Fine, and T. V. Bliss, "Optical quantal analysis reveals a presynaptic component of LTP at hippocampal Schaffer-associational synapses," *Neuron*, vol. 38, no. 5, pp. 797–804, 2003.
[32] P. S. Eriksson, E. Perfilieva, T. Björk-Eriksson, A.-M. Alborn, C. Nordborg, D. A. Peterson, and F. H. Gage, "Neurogenesis in the adult human hippocampus," *Nature medicine*, vol. 4, no. 11, pp. 1313–1317, 1998.
[33] M. Bronner-Fraser and M. E. Hatten, *Fundamental Neuroscience*, ch. 16. Neurogenesis and Migration, pp. 351–372. Academic press, 3 ed., 2008.
[34] E. A. Huebner and S. M. Strittmatter, "Axon regeneration in the peripheral and central nervous systems," *Cell Biology of the Axon*, pp. 305–360, 2009.
[35] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952.
[36] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
[37] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory, 1957.

[38] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. MIT press, 2017.
[39] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, no. 5-6, pp. 303–304, 1999.
[40] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.
[41] Z. F. Mainen, J. Joerges, J. R. Huguenard, and T. J. Sejnowski, "A model of spike initiation in neocortical pyramidal neurons," *Neuron*, vol. 15, no. 6, pp. 1427–1439, 1995.
[42] C. Henley, *Foundations of Neuroscience*, ch. 6. Action Potentials. Michigan State University, 2021.
[43] D. F. Goodman and R. Brette, "The brian simulator," *Frontiers in Neuroscience*, vol. 3, p. 26, 2009.
[44] "Timeline of Computer History: 1943." `https://www.computerhistory.org/timeline/1943`, 2021. Accessed: 04/12/2021.
[45] "An AI alphabet: W is for winter." `https://www.ainewsletter.com/newsletters/aix_0501.htm`, 2005. Accessed: 04/12/2021.
[46] P. Baheti, "The Essential Guide to Neural Network Architectures." `https://www.v7labs.com/blog/neural-network-architectures-guide`, 2021. Accessed: 21/02/2022.
[47] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
[48] S. Schliebs and N. Kasabov, "Computational modeling with spiking neural networks," *Springer handbook of Bio-/Neuroinformatics*, pp. 625–646, 2014.
[49] C. A. Seger and E. K. Miller, "Category learning in the brain," *Annual Review of Neuroscience*, vol. 33, p. 203, 2010.
[50] M. A. Nielsen, *Neural Networks and Deep Learning*, vol. 25, ch. 2. How the backpropagation algorithm works. Determination Press, San Francisco, CA, 2015.
[51] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
[52] W. Gerstner, R. Kempter, J. L. Van Hemmen, and H. Wagner, "A neuronal learning rule for sub-millisecond temporal coding," *Nature*, vol. 383, no. 6595, pp. 76–78, 1996.
[53] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2015.
[54] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for Energy-Efficient Neuromorphic Computing," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.
[55] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.
[56] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016.

[57] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits." http://yann.lecun.com/exdb/mnist/, 1998. Accessed: 03/09/2021.
[58] P. J. Grother, "NIST special database 19 - handprinted forms and characters database," *National Institute of Standards and Technology*, 1995.
[59] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[60] X. He, K. Zhao, and X. Chu, "AutoML: A Survey of the State-of-the-Art," *Knowledge-Based Systems*, vol. 212, pp. 106–622, 2021.
[61] J. A. Bourne, "Unravelling the development of the visual cortex: implications for plasticity and repair," *Journal of Anatomy*, vol. 217, no. 4, pp. 449–468, 2010.
[62] S. Ackerman, *Discovering the Brain*, ch. 6. The Development and Shaping of the Brain, pp. 86–103. National Academies Press (US), 1992.
[63] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, ch. 5.4 Integrate-and-Fire Models, pp. 162–166. MIT press, 2001.
[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[65] D. M. Kline and V. L. Berardi, "Revisiting squared-error and cross-entropy functions for training neural network classifiers," *Neural Computing & Applications*, vol. 14, no. 4, pp. 310–318, 2005.
[66] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.
[67] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.
[68] SciPy developers, "scipy.integrate.odeint." https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html, 2022. Accessed: 05/11/2021.
[69] J. Burkardt, "ODEPACK ordinary differential equation solvers." https://people.sc.fsu.edu/~jburkardt/f77_src/odepack/odepack.html, 2008. Accessed: 05/11/2021.
[70] C. F. Curtiss and J. O. Hirschfelder, "Integration of stiff equations," *Proceedings of the National Academy of Sciences*, vol. 38, no. 3, pp. 235–243, 1952.
[71] M. Zeltkevic, "Adams methods." https://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node6.html, 1998. Accessed: 05/11/2021.
[72] NetworkX developers, "NetworkX Network Analysis in Python." https://networkx.org, 2022. Accessed: 01/03/2022.
[73] Python developers, "Multiprocessing – Process-based parallelism." https://docs.python.org/3/library/multiprocessing.html, 2022. Accessed: 23/01/2022.

[74] OpenCV developers, "cv::VideoWriter Class Reference." https://docs.opencv.org/3.4/dd/d9e/classcv_1_1VideoWriter.html, 2022. Accessed: 01/03/2022.

[75] Scikit developers, "3.3.2. Classification metrics." https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics, 2022. Accessed: 12/01/2022.

[76] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "PMLB: a large benchmark suite for machine learning evaluation and comparison," *BioData Mining*, vol. 10, pp. 1–13, Dec 2017.

[77] R. K. Bock, "MAGIC Gamma Telescope Data Set." https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope, 2004. Accessed: 13/01/2022.

[78] P. U. Diehl, "peter-u-diehl/stdp-mnist." https://github.com/peter-u-diehl/stdp-mnist, 2015. Accessed: 02/10/2021.

[79] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," Master's thesis, University of Toronto, 2009.

[80] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[81] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, "Autoencoder for words," *Neurocomputing*, vol. 139, pp. 84–96, 2014.

[82] W. Gerstner and J. L. van Hemmen, "Associative memory in a network of 'spiking' neurons," *Network: Computation in Neural Systems*, vol. 3, no. 2, pp. 139–164, 1992.

[83] R. Jolivet, A. Rauch, H.-R. Lüscher, and W. Gerstner, "Predicting spike timing of neocortical pyramidal neurons by simple threshold models," *Journal of Computational Neuroscience*, vol. 21, no. 1, pp. 35–49, 2006.

[84] A. Galves and E. Löcherbach, "Infinite systems of interacting chains with memory of variable length—a stochastic model for biological neural nets," *Journal of Statistical Physics*, vol. 151, no. 5, pp. 896–921, 2013.

[85] A. E. Lindsay, K. Lindsay, and J. Rosenberg, "Increased computational accuracy in multi-compartmental cable models by a novel approach for precise point process localization," *Journal of Computational Neuroscience*, vol. 19, no. 1, pp. 21–38, 2005.

[86] Brian authors, "Numerical integration." https://brian2.readthedocs.io/en/stable/user/numerical_integration.html, 2020. Accessed: 18/03/2022.

[87] Brian authors, "Source code for brian2.stateupdaters.exact." https://brian2.readthedocs.io/en/stable/_modules/brian2/stateupdaters/exact.html, 2020. Accessed: 18/03/2022.

[88] SymPy development team, "Integrals." https://docs.sympy.org/latest/modules/integrals/integrals.html, 2021. Accessed: 18/03/2022.

[89] J. C. Butcher, "A history of Runge-Kutta methods," *Applied Numerical Mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[90] Brian authors, "Older notes on code generation." https://brian2.readthedocs.io/en/stable/developer/oldcodegen.html, 2020. Accessed: 18/03/2022.

[91] Brian authors, "Computational methods and efficiency." https://brian2.readthedocs.io/en/stable/user/computation.html, 2020. Accessed: 18/03/2022.

Appendix

A Adult neurogenesis in the human brain

In the human brain, adult neurogenesis occurs in two specific "neurogenic" regions [12]:

- the subgranular zone (SGZ) of the dentate gyrus in the hippocampus, which is part of the temporal lobe of the brain (the SGZ primarily contributes to the formation of memories); and
- the subventricular zone (SVZ) generates neurons that migrate to the olfactory bulb (responsible for processing the sense of smell) and become interneurons (not connected to sensory input or functional output, but in the middle of a circuit).

An important difference between these neurogenic regions is the mode of migration. In both cases, the neural progenitor cells first differentiate into neuroblasts, which are cells that don't divide, but instead mature into a functional neuron after a migration to their target region. In the SGZ, neuroblasts typically migrate through blood vessels in the brain; whilst in the SVZ, they form a chain and migrate together to their target region (the olfactory bulb) through a tube of astrocytes (very numerous supporting cells in the nervous system).

Neuromodulators (chemicals transmitted by neurons to regulate populations of neurons and other brain functions) and other morphogens (signalling molecules) serve to regulate many facets of neurogenesis in the embryonic and adult brain, such as:

- in the adult SGZ, mature interneurons release GABA (gamma-aminobutyric acid, a neuromodulator) which regulates cell proliferation (the differentiation of stem cells and differentiation/division of neural progenitors to produce neurons), the pace of maturation and the development of dendrites and synapses in newborn neurons;
- astrocytes in the SGZ and SVZ release morphogens (some of which remain attached to their membrane) that regulate the proliferation and target neuron of stem cells and the migration, maturation and synapse development of newborn neurons; and
- microglia also release morphogens that regulate adult neurogenesis.

B Brian simulator

Brian is a popular open source Python library for simulating spiking neural networks [43, 4]. This simulator was designed to be easy to learn and very flexible, allowing neuron models to be defined by specifying their differential equations as strings in Python syntax. An example of this is the leaky-integrate-and-fire neuron, where the membrane potential is defined by the equation

`"du/dt = (-(u-urest) + R*I)/tau : volt"`

where `urest`, `R` and `tau` are constants (with respect to Brian SI units) defined as Python variables in the same script, and `I` is a simulated variable defined by the initialisation string `"I : amp"`, and later controlled by discrete update events in the simulation. The spiking behaviour of the neuron is defined by specifying a threshold condition `u > uthresh`, a reset condition of `u = ureset` and a refractory period duration in milliseconds.

When a group of neurons is initialised in Brian, an integration method can be specified as an argument (otherwise Brian will choose one automatically). The most notable of these methods are `"exact"`, `"euler"`, `"exponential_euler"`, `"rk4"` and `"rk2"` [86].

If the model's differential equations are linear, Brian can use exact integration, denoted by `"exact"`, to find expressions for the variables of the model with respect to time (and possibly other variables in the model), and then iteratively evaluate these at discrete time points in the simulation [87]. Brian achieves exact integration using the open source Sympy library in Python, which is a symbolic mathematical computation package. Sympy implements integration using a variety of strategies [88], with the simplest being an algebraic method of finding an antiderivative for the integrand, then applying the fundamental theorem of calculus (which works for elementary functions containing, for example, polynomial or trigonometric terms).

If the differential equations are not linear, Brian can only use a numerical method. For discussing these, let h be a constant time step size, consider the consecutive discrete time points t_n and $t_{n+1} = t_n + h$, and let $\frac{du}{dt} = f(u,t)$ unless otherwise stated (noting that in a coupled system with multiple state variables, the other state variables may also be parameters of f). Let $u_n = u(t_n)$ and $u_{n+1} = u(t_{n+1})$.

The first of these numerical methods is the forward Euler method, denoted by `"euler"`, where the update rule to find u_{n+1} given u_n is

$$u_{n+1} = u_n + hf(u_n, t_n).$$

If the ODE is second order, so $\frac{d^2u}{dt^2} = f(u, u', t)$, then the update rule is

$$\begin{aligned} u_{n+1} &= u_n + hu'(t_n), \\ u'(t_{n+1}) &= u'(t_n) + hf(u_n, u'(t_n), t_n). \end{aligned}$$

The second numerical method is exponential Euler integration, denoted by `"exponential_euler"`, which assumes that the differential equation is of the form

$$\frac{du}{dt} = -Au + N(u),$$

where A is a constant and $N(u)$ is a nonlinear term with respect to u. Then the update rule is

$$u_{n+1} = e^{-Ah}u_n + A^{-1}(1 - e^{-Ah})N(u_n),$$

where the linear term is integrated exactly but the nonlinear term is held constant over the timestep from t_n to t_{n+1}.

The classical Runge-Kutta method [89], denoted by `"rk4"`, works on any first-order initial value problems, where the update rule is

$$\begin{aligned} u_{n+1} &= u_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \text{ where} \\ k_1 &= f(u_n, t_n), \\ k_2 &= f(u_n + \frac{h}{2}k_1, t_n + \frac{h}{2}), \\ k_3 &= f(u_n + \frac{h}{2}k_2, t_n + \frac{h}{2}), \text{ and} \\ k_4 &= f(u_n + hk_3, t_n + h). \end{aligned}$$

In this method, k_1 is the slope at t_n, k_2 and k_3 are both estimates of the slope at the midpoint $\frac{1}{2}(t_n + t_{n+1})$ and k_4 is the slope at t_{n+1}, where greater weight is given to the midpoint slopes when averaging them to estimate u_{n+1}.

The second order Runge-Kutta method (also called the midpoint method), denoted by `"rk2"`, has the update rule

$$u_{n+1} = u_n + h \cdot f(u_n + \frac{h}{2}f(u_n, t_n), t_n + \frac{h}{2}),$$

which uses an approximate gradient at the midpoint. This method can also be adapted to integrate second order ODEs using the same approach that was used for Euler's method.

To convert equations defined as Python strings into executable code, Brian uses the following process [90]:

1. convert equations to abstract code – an integration method is chosen, then abstract code that uses only basic arithmetic operations (in Python syntax) to compute that integration method for the state variables for a single time step (and taking refractory behaviour into account) is generated from the provided equations;
2. convert abstract code to code snippets – given the abstract code and a target language (such as Python or C++), add the correct syntax and proper declarations to produce code that is "correct" for the target language, but not necessarily runnable on its own;
3. convert code snippets to code blocks – insert each code snippet into a predefined Jinja2 template for the target language to produce a block of runnable code; and
4. execute the code block – depending on the simulation method specified, either allocate memory, run the simulation loop and execute the code from Python, or save the code blocks with simulation control code in the target language to be run later.

There are two primary methods that Brian uses for running a simulation:

- runtime code generation – where the main simulation loop and memory allocation are handled by Python code (using Numpy) within the Brian package, and run live when the simulation `run(duration)` method is called, where

code blocks in the target language are generated and then executed remotely from Python control code; or
- standalone code generation – simulation control code is generated in the target language and bundled with generated code blocks (which are each saved as separate files) into a complete project, which can then be compiled and run independently (at a later time or even on a different machine).

The runtime method is the most popular, as it allows for the simulation to be executed immediately without requiring extra compiling (which suits quick experimentation with neuron models, which is what the Brian simulator was primarily built for), but is the slower method because of the overhead of running the main simulation loop in Python rather than the faster C++ [91]. The standalone method is more suited to large-scale networks where efficiency improvements offer significant time reduction, and for running simulations on isolated machines (such as a remote supercomputer).

C Spiking network implementation for the MAGIC task

A Python script containing source code for the custom spiking network implementation that was used for the MAGIC task in chapters 5 and 7 can be found online at: `https://github.com/ethanm-dev/honours/blob/main/appendix_custom.py`. This implementation was outlined in section 4.2.2. To run this script, first run the following command to install the necessary dependencies:

```
pip3 install numpy scipy tqdm pmlb scikit-learn
```

Then, download the `appendix_custom.py` script to your local filesystem. In the same directory as the Python script, simply run the command:

```
python3 appendix_custom.py
```

This script will train and test a spiking network with neurogenesis on a very small set of MAGIC data, using parameter values defined within.

D Spiking network implementation for the MNIST task

A Python script containing source code for the Brian spiking network implementation that was used for the MNIST task in chapter 6 can be found online at: `https://github.com/ethanm-dev/honours/blob/main/appendix_brian.py`. To run this script, first run the following command to install the necessary dependencies:

```
pip3 install numpy brian2 tqdm scikit-learn
```

Then, download the `appendix_brian.py` script and the data folder at `https://github.com/ethanm-dev/honours/tree/main/mnist_data` to your local filesystem. In the same directory as the Python script and `mnist_data` folder, simply run the command:

```
python3 appendix_brian.py
```

This script will train and test a spiking network on a very small set of MNIST data, using parameter values defined within.