School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Comparison of Neural Simulator Performance for Recurrent Networks

Callum Lillywhite-Roake

Submitted in accordance with the requirements for the degree of
MEng, BSc Computer Science

2021/22

COMP3931 Individual Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (27/04/22)
Experiment Code Repository and Raw Data	`https://github.com/C41Underscore/MIIND-NEST-Analysis`	Uploaded to GitHub (27/04/22)

Summary

Neural simulators fall in the field of computational neuroscience, an intersection of computer science and neuroscience. The aim of any neural simulator is to use biologically derived mathematical models to simulate the behaviour of populations of neurons, and this typically happens in one of two ways. We either simulate the individual neurons and the connections between them, something known as Monte Carlo simulation, or we can simulate the behaviour of an entire group of neurons, describing the behaviour at population-level, known as population methods. MIIND is such a simulator that uses population-level modelling, whilst NEST is a simulator that performs Monte Carlo simulations. In this report, we aim to compare the performance of these simulators when simulating recurrent networks, which are networks of neurons which feedback their output to themselves. Specifically, it has been noted that MIIND shows unusual results from simulation of recurrent networks, and we aim to see how MIIND's results differs from that of NEST's when simulating these so-called recurrent networks.

Acknowledgements

I would like to acknowledge my supervisor Marc de Kamps, who has been extremely helpful and took time out to ensure that I understood the core concepts outlined in this report, and answering the many questions that I sent them. They also helped me get back on track when it had become apparent that I needed for reinforcement in key areas of this subject.

I would like to acknowledge Hugh Osborne, Marc de Kamp's PhD student, who answered my many questions in great detail and providing me with great insight and intuition as to how MIIND works. They helped me with issues getting simulations running, all despite my very erratic emails and questions asking for help.

I would also like to acknowledge my assessor Abdulrahman Altahhan, who provided me with valuable feedback on my project progress, and raised several extremely useful points about where my current progress was lacking, something that I was able to build upon in this report.

Finally, this work was undertaken on ARC3, part of the High Performance Computing facilities at the University of Leeds, UK.

Contents

Chapter 1

The Biological Neuron

Neurons are considered to be the basic building blocks of our brain and the nervous system [1] which is responsible for signalling information throughout our body [2]. The human brain itself consists of an incredible amount of these units, roughly 85 billion [3] [4], and are the cells responsible for receiving external sensory input, relying messages to our muscles, and our behaviour [1]. Here we shall discuss some basic anatomy of neurons and the biology that underpins them. It should be noted that here we shall specifically be talking a type of neuron called a Multipolar neuron, which is the most common neuron in mammalian nervous systems [2].

1.1 Basics of Neurons

Neurons operate together by communicating with one another [5] and sending messages to build-up complex behaviour [2]. This communication consists of messages of small electrical pulses know as action potentials. Action Potentials are the signals that neurons receive, integrate, and send information with, and generally tend to be on the order of 100mV within the neuron [2]. These action potentials are then transmitted to other neurons along small gaps called synapses [2], and when an action potential reaches the synapse, it causes the release of chemical neurotransmitters, which allows for more sophisticated communication [1]. The process of a neuron producing an action potential is called firing.

1.1.1 Structure of Neurons

Neurons have a "tree-like" [1] structure, and consists of four defined regions [2]. Firstly, there is the Soma, also known as the cell body [1], and this contains the cell's nucleus, which itself contains the DNA of the cell [2]. It is the responsibility of the Soma to perform processing of the incoming signals from other neurons, we can treat it as the 'central processing unit' [6] of a neuron. Furthermore, it connects two key areas of the neuron: Dendrites and the Axon.

Dendrites are considered to be the main way that a neuron will receive signals from others [2], and contains the receiving sites for the chemical neurotransmitters released by the sending neuron in the synapse [1]. It is said that the summation of dendritic inputs is a key factor in what causes a neuron to fire [1].

The axon of the neuron is the medium of transmission used by a neuron to get an action potential from the dendrites across to other neurons [2]. It is a thin membrane, with a diameter ranging between 2μm to 20μm, and can be as long as 1m [2]. A neuron axon is able to branch, which allows a neuron to connect to multiple other neurons, where the end of an axon branch, known as a terminal [2], will propagate it's action potential across the synapse. The action potential is generated at the start of an axon, in an area called the initial segment [2], and is

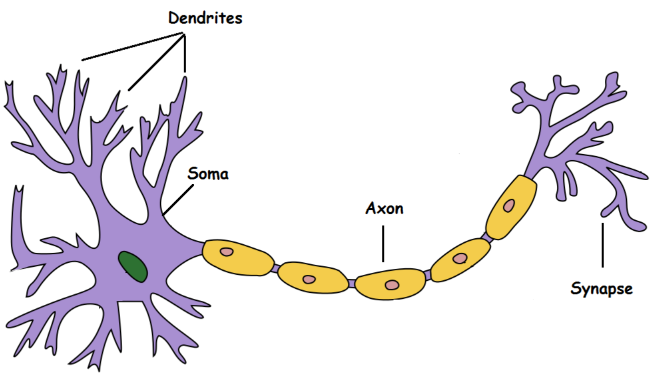

Figure 1.1: The basic anatomy of a neuron, obtained from [7]

constantly regenerated as it move along the membrane to ensure that it stays at a constant potential [2].

1.1.2 Spiking Neurons

Signals produced by neurons consist of short electrical pulses known as spikes, and these spikes happen as a direct result of an action potential being generated in a neuron, and in reality the terms spike and action potential refer to the same phenomena [6]. Each spike produced will have am amplitude of roughly 100mV and will last for 1-2ms [6], and the spike does not change once it has been generated and moves along the axon [6], in other words, it has no internal variation.

A spike train is defined as a series of spikes emitted by a single neuron that can occur at regular or irregular intervals over a given period of time [6], and importantly, these spikes are stereotyped [6], meaning that each spike is very similar to all other spikes produced. As a consequence of this, it's believed that spiking neurons don't encode information in individual spikes, but rather in the timing and number of spikes produced by a neuron [6]. Spikes in spike trains tend to be well separated [6], and this is because after a neuron fires, it enters a refractory state, where it is near-impossible to generate a second spike [6].

1.1.3 Membrane Potential

Membrane Potential refers to the electrical potential that a neuron membrane has [2] and is generated as a result of a difference between the inner and outer surfaces of the membrane [2]. The outer layer of a membrane has an excess amount of positive ions, whilst the inner part of the membrane, also known as the cytoplasmic side, has an excess amount of negative ions[2]. On the outer layer of the membrane, we see concentrations of sodium, calcium, and chloride [8], which gives the outside an overall positive charge [8], whilst on the cytoplasmic side of the membrane, we see concentrations of potassium, and negatively-charged molecules such as proteins and amino acids [8], thus giving it a negative charge. Importantly, it is the movement of ions through ion channels that causes changes in membrane potential [9]. These ion channels are proteins that are found on the membrane surface, which allow the movement of charged ions through the surface; some channels open and close spontaneously, whilst others will open and close in response to a change in membrane potential [8]. We shall now discuss some key properties of the membrane, which helps us better understand the action potential.

Resting Potential

The resting membrane potential is the potential of the membrane whilst there is no activity from other neurons [10], and typically sits at a value of -70mV, although this is not the same for every neuron [10]. The generation of all action potentials starts at the resting membrane potential, and we can see this in figure 1.2, where the potential is constant before the sharp increase. It should be noted that it is possible to have a membrane potential that is lower than the resting potential, something we discuss shortly.

Steps of an Action Potential

An action potential has 3 core steps [9], which are depolarization, repolarization, and hyperpolarization. Depolarization refers to a decrease in magnitude of the membrane potential [8], where the membrane potential moves towards 0mV, although it also refers to a general increase in membrane potential beyond the threshold [8]. Depolarization will occur when the membrane potential exceeds the threshold voltage, which is the potential at which sodium ion channels open to allow movement of sodium ions into the membrane [9]. We describe this as the threshold voltage, because once the potential hits this threshold, it causes the ballistic process of an action potential, as the increase in sodium causes further depolarization, giving way to a positive-feedback loop [9]. Typically, the threshold potential for a neuron is -55mV [2], which once hit, will trigger the feedback loop that raises the potential to 40mV [2], a process that happens over roughly 1ms [9].

After depolarization, repolarization occurs, which is where the membrane returns towards it's resting state [9]. This is caused by the opening of potassium channels, which happens roughly 1ms after the action potential is triggered [9], and it's this flow of potassium ions out of the cytoplasmic side that causes the neuron's potential to move back to it's resting state, and when the potential moves past the threshold (from above), both the movement of sodium and potassium slows to a halt [9]. However, because the kinetics of the potassium channels are slightly slower than the sodium channels, they remain open for longer, which causes a fall below the resting potential for a short period [9], something called hyperpolarization. The progression of an action potential can be seen in figure 1.2, which is a recording of an action potential from a giant squid axon.

Refractory Period

Although already mentioned in section 1.1.2, we should define the refractory period in light of what we have discussed. The refractory period starts at repolarization, due to the movement of potassium decreasing the membrane potential, this makes it very hard for the neuron to generate another action potential [11].

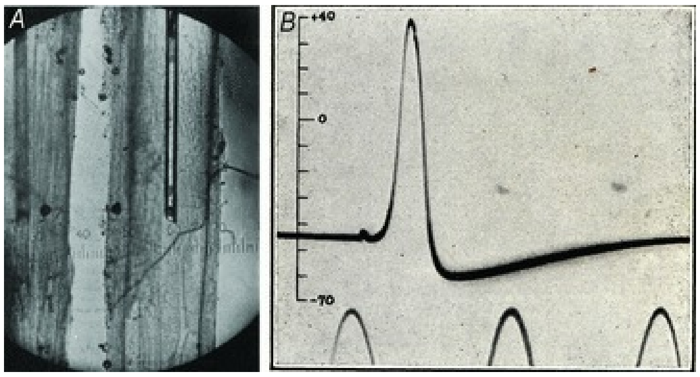

Figure 1.2: Images produced from the famous experiment done by Hodgkin and Huxley in 1939, where they performed the first ever intracellular recording of an action potential in a giant squid axon using an electrode [12]. A) Shows the electrode inserted into the axon membrane, and B) Shows the action potential they recorded, which we would describe as a single spike, where the x-axis depicts time, whilst the y-axis depicts membrane potential in mV. Image obtained from [12]

1.1.4 Synapses

Synapses are the structures used between neurons to communicate, and it is where a neuron's terminal comes close too, but doesn't touch, the dendrites of another neuron [8]. At each synapse, there exists two neurons, one is known as the pre-synaptic neuron, which is the neuron where the action potential meets the synapse, and there is the post-synaptic neuron, which is the neuron to receive a pulse via the synapse [8]. Synapses have no connection between neurons, and instead, rely on the diffusion of neurotransmitters from the pre- to the post-synaptic neuron [8]. Neurotransmitters are chemical agents which operate as 'messengers' between the pre- and post-synaptic neuron[11]. When an action potential reaches a synapse, it will cause the pre-synaptic neuron to release neurotransmitter, which moves towards the post-synaptic neuron. The post-synaptic neuron has receptors on it's surface, and it is the binding of these neurotransmitters to the receptor sites which changes the behaviour of the channels in the post-synaptic cell [11]. This means that the release of certain neurotransmitters can alter the conductance of the post-synaptic cell, either increasing or decreasing the chance that it will fire [11]. This is something we refer to as excitation and inhibition.

1.1.5 Inhibition and Excitation

When we consider inhibition and excitation, it's also important to define the term Postsynaptic Potential, or PSP [11]. A PSP refers to the changes in membrane potential of the post-synaptic cell that moves a cell away from it's resting state [8], and these PSPs can either be excitatory or inhibitory.

An Excitatory PSP, or EPSP, will result in the depolarization of the post-synaptic cell, and results in an excitatory change in the post-synaptic cell, meaning that there is an increased likelihood that it will generate an action potential [8]. This means that on receipt of an EPSP, the potential of the post-synaptic cell will increase. On the contrary, an Inhibitory PSP, or

IPSP, does the opposite, albeit it's dynamics being a little more complicated. An IPSP will decrease the likelihood of an action potential being generated in the post-synaptic cell [8], thus meaning it causes a decrease in potential. Now, for the neurons we shall consider in this report, we describe an IPSP to move the membrane potential towards the minimum potential, which is the lowest possible potential of the neuron (this could be equal to the resting potential). Technically speaking, this is not the true consequence of an IPSP, as due to the dynamics of ions across the membrane, an IPSP can sometimes cause an increase in membrane potential if the current potential is below a certain value [8]. Understanding this behaviour is beyond the scope of this report, and here we shall consider an IPSP to simply decrease the likelihood of an action potential in the post-synaptic neuron.

The chance of an action potential being fired in a neuron is dependent on the summation of EPSPs and IPSPs [8], so, if an EPSP is followed by more EPSPs, the chance of an action potential is greatly increased, and it is generally the case that multiple EPSPs are required to induce an action potential [8]. When EPSPs and IPSPs are summated together, one of two things could happen. Either the depolarizing effect of the EPSP will be weakened, or if the inhibitory signal is strong enough, it will result in the EPSP being cancelled out altogether [8].

Chapter 2

Biological Neuron Models

Here, we shall discuss several important components of the simulation of neurons, looking at the behaviour of both individual neurons, as well as the behaviour of populations of neurons. We shall look at an important model paradigm of a neuron: Integrate-and-Fire, as well as it's famous variant, Leaky-Integrate-and-Fire. We also quantify other key aspects of neural behaviour such as synapses and thresholds. We shall then look at the Poisson spike train, which is used to replicate external stimulus for our simulations. Next, we look at common statistical metrics used to quantify neuron behaviour for individual neurons and populations. Finally, we shall compare techniques used to simulate groups of neurons, more specifically, we look at Monte Carlo simulations and Population methods.

Neuron models are mathematical objects that we use to represent the behaviour of neurons over time. These models will either represent individual neuron behaviour, such as the Integrate-and-Fire model, which simulates neuron behaviour with an evolution process and a method to generate spikes [6]. Alternatively, neuron behaviour can be described on a population-level, where Probability Density Functions can be used to describe the movement of neurons through an N-dimensional state space [13].

2.1 Integrate-and-Fire

The integrate-and-fire (IF) is one of the most famous neuron models, being widely used to study behaviour of neural systems [14]. At it's core, the IF model describes a neurons membrane potential as a function of synaptic inputs [14], but in practise it consists of two components. First, it has an equation which describes the change in membrane potential over time, which can include more than just a simple summation of synaptic input, and a way for the model to generate spikes [6]. Here, we focus on a varient of the model known as the Leaky-Integrate-and-Fire (LIF).

2.1.1 Leaky-Integrate-and-Fire

The LIF model looks to basic electronics to model an individual neuron. It describes a single neuron as a parallel Resistor-Capacitor circuit [15], which mimics two qualities of a neuron membrane. First, a membrane acts as an insulator for our neuron [6], meaning that any charge it receives from the movement of ions, it is able to keep for a short period of time. Given that there is no such thing as a perfect insulator, our cell must start leaking it's charge through the membrane at one point, which gives way to the equivalent of leaky resistance in a capacitor, which is where a capacitor leaks charge as a result of it's insulting material. Hence, we say our neuron has capacitance C and leaky resistance R [6] [15].

Such a circuit has no way to generate an action potential, so, when our membrane capacitor reaches a certain threshold, our model let's it discharge, and we produce an action potential, or spike [15]. The LIF model takes a shortcut with action potentials, since we know each spike produced by a neuron is roughly the same, there's little need to implement any mechanic to replicate this process, thus, we can just reset our membrane capacitor to a resting value [15], which we mark as a spike.

The evolution of an LIF membrane potential is given by the following differential equation [6]

$$\tau_m \frac{du}{dt} = -[u(t) - u_{rest}] + RI(t) \tag{2.1}$$

Where $u(t)$ is the membrane potential at time t, u_{rest} is the resting potential, R is the leaky resistance, $I(t)$ is the input current at time t, and $\tau_m = RC$ is the time constant of the membrane. The threshold of our model is denoted as u_{th}, and when $u(t) \geq u_{th}$, we reset $u(t)$ to equal u_{rest}, which represents an action potential being generated, something we can see in figure 2.1, where a sudden drop in membrane potential represents the firing of a neuron.

Let us define $I(t)$ a little better. Since $I(t)$ represents the incoming current from the synapse, we should formalise the summative nature of the inputs generated by the post-synaptic neuron. When modelling neurons, we also model synapses too, which generate more non-trivial behaviour and allows us to better understand the dynamics of neural systems, and here, we shall use the a basic form of synapse, a static synapse with delta shaped spikes. The former, static synapses, meaning that there is no change to the strength/ efficacy of the synapse over time, and acts as a rely to the post-synaptic neuron. A delta shaped spike means that a reception of a action potential is modelled as an instantaneous jump in membrane potential, that there is an all-or-nothing response in our post-synaptic neuron. This instantaneous jump is represented by the Dirac delta function δ

$$\begin{aligned} \delta(t - t_s) &= 0 \quad && t \neq t_s \\ \delta(t - t_s) &= \infty \quad && t = t_s \end{aligned} \tag{2.2}$$

Where importantly $\int_{-\infty}^{\infty} \delta(t - t_s)dt = 1$, meaning this function will 'fire' when a neuron fires, else it will evaluate to 0. This also means that a neuron can fire at a given time step or it won't, so it can't fire multiple times on a single time step. Using this function, we can define the summation of inputs from pre-synpatic neuron l, where $u_l(t)$ gives the input received from l at time t. We define this input over time, where the set T_l gives the firing times of l

$$u_l(t) = h \cdot \sum_{t_s \in T_l} \delta(t - t_s) \tag{2.3}$$

The constant h represents the synaptic efficacy, which acts as a multiplier to a single synaptic input, and is a value of interest given that the Dirac delta function produces a 1 when it fires, as such, the synaptic efficacy h defines the jump in potential that the post-synaptic neuron experiences. This value is typically lower than 1, which means that many EPSPs will be required in order for the post-synaptic neuron to fire. The synaptic efficacy can also be negative, in which case this makes the resulting spike an IPSP, having an inhibitory affect.

In all that we have mentioned about LIF, delta synapses, and the summation of inputs, we describe this particular type of neuron and synapse model as a LIF neuron with a Delta synapse, and shall be the model which is the basis of this report. Finally, we shall discuss some of the issues and benefits of using this model in a wider context.

This model is far from perfect, and has several flaws, particularly, it doesn't account for the spike history of the neuron, and inherently doesn't change it's parameters based on previous spikes. For example, we have observed a neurons response to incoming action potentials to change as a result of something called it's conductance, which simply means how capable it is of receiving an incoming spike in it's full. If a neuron receives two spikes in a short space of time, say from two different neurons, then it is likely that the second spike will have a lesser affect as the first, due to the relevant ion channels already being open, thus it not being able to make the same difference in potential [6]; this is in contrast with our LIF model, which takes synaptic input with a static efficacy and considers nothing else. In this way, we say that LIF is a simple model and doesn't represent our true understanding of neurons, and is just a representation of it. There are other variants of IF which attempt to deal with this, including a Conductance-based LIF [13] and the Adaptive Exponential Integrate-and-Fire neuron [16]. There are also other models which describe the biological functioning and movement of ions in the membrane, such as the groundbreaking Hodgkin-Huxley model [17].

On the contrary, LIF is still used widely for investigating the behaviour of neural systems [14], and this is because whilst it is simple, it can still simulate a number of core neural behaviours which can be analysed mathematically [14]. Specifically, the LIF neuron is seen to be quite accurate in it's spike generation, and fairly good at producing them at the right times [6], which in itself does not make it a sophisticated neuron model, but has made it 'canonical'[14] and a useful model in computational neuroscience. The LIF is also often used in the analysis of neurons in response to stochastic input [18], both at an individual level, but also at the population level, which is enabled by its simplicity [18]. Finally, the LIF model, and more generally the Integrate-and-Fire model, are seen as a good basis for more complex, and biologically accurate models, such as the previously mentioned Adaptive Exponential Integrate-and-Fire neuron.

Finally, the LIF neuron is considered a good neuron model to test software with, given it's sufficiently complex to simulate core neuron behaviour, whilst being simple enough such that neural simulators can be tested effectively, without taking ages on simulation. This is something seen in the neural simulator NEST [19], who use the LIF neuron model with a selection of different synapses to test their simulator, and specifically use the Delta synapse model to test the consistency of their results throughout updates. This is why this particular model is being used in this report, as it allows us to effectively simulate large populations of neurons, whilst also producing valid data to analyse.

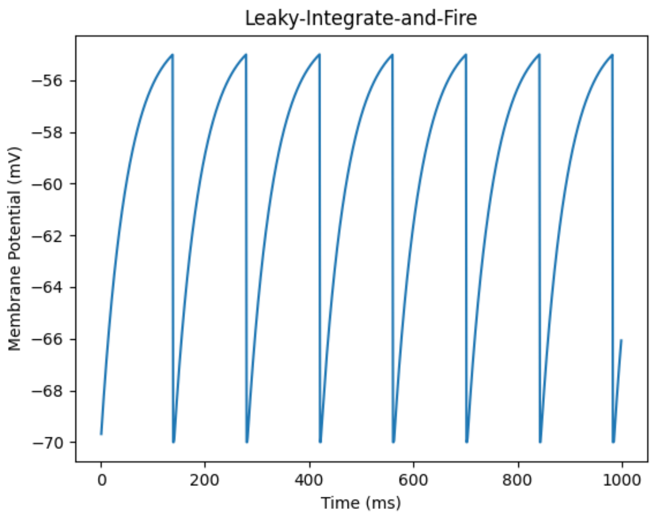

Figure 2.1: The evolution of an LIF neuron over 1 second, receiving a constant input of 80pA. Over this second, we see the LIF neuron fire 7 times, denoted by the vertical drop in potential.

2.2 The Poisson Process

A Poisson Process is used to model a series of discrete events, where the average number of events is known, the average time between events is known, but the exact timing of events is not known [20]. In this sense, we know how many events to expect, but don't know when they will occur. Furthermore, we can't guarantee that the exact number of events we are expecting will happen, as it could be more or less. In mathematical terms, a Poisson process should satisfy the following criteria [20]

- Events are independent of each other, meaning that the happening of one event doesn't affect the probability of a future event.
- The average rate of events (the expected number) is constant.
- Events are discrete, they cannot occur at the same time (they either happen or they don't).

In reality, most real-world phenomena which are modelled with the Poisson process are unable to strictly follow these criteria, however, when we use the Poisson process in neural simulation, we are able to abide by these rules. We shall discuss the Poisson spike train, a widely used form of stimulus in neural simulation, and how the mentioned criteria affects it. The event in our Poisson process for the spike train is a spike, so when we consider a spike, we consider a Poisson event.

Firstly, because each spike needs to be independent of each other, we need to create our train such that the probability of a spike happening at time t doesn't affect the probability that a spike will occur at time $t + \Delta t$, which allows for a massive variety of different spike trains to take shape. Next, for all of our Poisson processes, we must define a firing rate, which doesn't change. This firing rate is used to compute the probability that a spike happens at time t, and naturally, the higher the firing rate, the higher the probability that a spike will happen at time

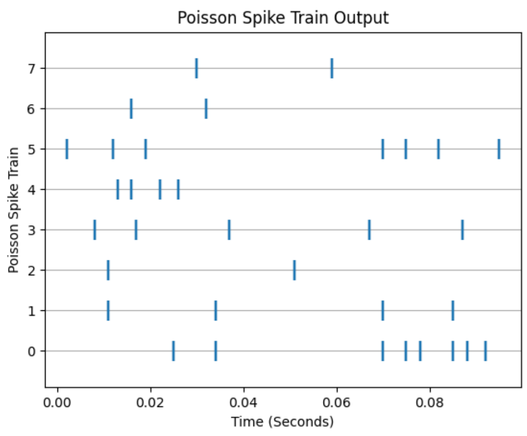

Figure 2.2: 8 different Poisson spike trains, simulated over 100ms with a rate of 50Hz. It can be seen how differently they can vary from trial-to-trial, and why that makes it so useful in simulations.

t. Finally, spikes must be discrete, which means a single Poisson process cannot fire multiple spikes at once, so that on any time step t, a spike will fire, or it doesn't. Figure 2.2 shows us how that materialises.

Poisson processes are important in neural simulations, as they can be used to represent stochastic processes in the nervous system, which is something of interest in neuroscience [21]. We can therefore use Poisson spike trains to represent external input to our system in such a way that we generate slight variability in input each time, however, this is often negated by simulating these trains over a longer period of time. In order to create a spike train which has T time steps, at each step t, we should generate a random number in the range $[0, 1]$, and a threshold value which is λdt, where λ is the expected firing rate of our Poisson spike train, and dt are the individual time steps being taken in our spike train, usually in the order of 1ms. If our random number lies below the threshold, we can count it as a spike, and we won't if it exceeds the threshold. This is the exact process used to generate figure 2.2.

2.3 Common metrics used for describing Neuron behaviour

2.3.1 Rate as a Spike Density and the Peri-Stimulus-Time Histogram

The Peri-Stimulus-Time Histogram is a measurement of neural spike activity over a specified period of time [6]. A PSTH is by name, is a histogram, with bin-width Δt, where t is the time, and each bin represents time $t + \Delta t$, which is typically of order of milliseconds [6]. When we perform a neural simulation, if we have a small population or even a single neuron, we should perform multiple trials in order to calculate the average behaviour of a particular model with slightly varying parameters, which is a principle we can utilise with the PSTH. An experimenter can count the number of spikes that have occurred in a given period, given as $n_K(t; t + \Delta t)$ [6], over K trials of an experiment, which we then divide by to get the average firing rate during this time. Then, because we are only measuring activity in a limited period, we need to alter our calculated value so that it is in Hz, and doing this gives us the overall equation to compute the firing rate. We define the average firing rate between t and $t + \Delta t$ as $\rho(t)$ [6]

$$\rho(t) = \frac{1}{\Delta t} \frac{n_K(t; t + \Delta t)}{K} \tag{2.4}$$

2.3.2 Rate as Population Activity

Another common metric is defining a firing rate of a population of neurons, over a single trial. Now if we consider a set of homogeneous neurons (neurons which all have the same parameters and connection set up), we can use an equation almost identical to equation 2.4, to define the average firing rate of a population in the time period $t + \Delta t$[6], denoted as $A(t)$

$$A(t) = \frac{1}{\Delta t} \frac{n_{act}(t; t + \Delta t)}{N} \tag{2.5}$$

Where N is the number of neurons in the population and $n_{act}(t; t + \Delta t)$ is the number of spikes produced by all neurons in the population. If we simulate a large population of neurons, we are getting the average behaviour of individual neurons, but also their typical interactions within a population, which provides a more comprehensive view to the researcher.

2.4 Methods of modelling Neurons

2.4.1 Monte Carlo

Monte Carlo simulations refer to the modelling and simulation of individual neurons, and the dynamics between them. A Monte Carlo simulation consists of a collection of point neurons, which are neurons that exists as a single point in space, whose dynamics are described by a set of coupled differential equations [22]. Due to the nature of Monte Carlo simulations, they yield several benefits. Firstly, since such a simulation must define the individual neurons, they must also define the individual synapses between them, which allows for simulation of heterogeneous networks [13], where both neuron and synapse parameters can be different, but advanced simulators, such as NEST [23], can simulate interacting neurons of different types, with different synapse models. Point neuron models are also very simple to understand, and if one has a good understanding of the underlying models driving a simulation, then understanding and analysing the behaviour of collections of point neurons is much easier, as their is little abstraction from the models. In addition to this, although this is a relatively simple modelling technique, these models are capable of producing a variety of behaviours, and have been used many times before to produce explanations of brain functioning, including in attention [24], and in working memory [25].

However, there is a big drawback to the use of Monte Carlo methods. It's estimated that the human brain could have up to 85 billion neurons [3][4], or even more, and at this level, the simulation of point neurons doesn't scale well, and proves intractable [22], due to the computational and memory limits of currently available computers [13]. This means we should look for an alternative to solve computational limits, which is exactly what population methods aim to do.

2.4.2 Population Methods

Population methods for simulating neurons involves calculating the overall behaviour and movement of a cluster of neurons through state space. As put in Omurtag et al.(2000), population methods 'approach the task of simulating an ensemble of interacting neurons by means of a statistical description of the population'[26], and very typically this takes the form of Population Density Techniques, or PDTS. With PDTs we use a Probability Density Function (PDF) to model the distribution of neurons in a population, and where they are most likely to be, and during simulation, we calculate how this distribution changes over time in response to incoming noise as well as passive movement (when the population is receiving no input)[27]. Modern research into PDTs aim to model neural populations in more than 1 dimension [28] [27], meaning that multiple aspects of a neuron model can be simulated, allowing for the simulation of models which have coupled dynamics, such as the conductance-based LIF [13], which models both membrane potential and the conductance variable.

We shall now go in-depth and formalise the population method as described by Marc de Kamps and colleagues [29] [28] [13]. These techniques describe a way to model 2-dimensional systems as a PDF, and we will then describe how they can be altered to cater for 1D systems, which are the types of systems we consider in this report. Under a population system, we represent point neurons as vectors $\vec{v}$, which describe a neuron's properties, with the first element traditionally being the membrane potential [28]. This allows point neurons to be represented and plotted in 1-dimensional or higher state space. Now, the movement of a population of neurons through state space is defined by a vector field $\vec{F}$, which itself is defined on an open subset of $\mathbb{R}^2$, and the equation which defines the movement of an individual neuron through state space is given as [28] [13]

$$\tau \frac{d\vec{v}}{dt} = \vec{F}(\vec{v}) \tag{2.6}$$

where τ is the time constant of the neuron. From here, we shall describe the state space of our simulation as M. When we use a threshold potential, the edge of M, denoted as ∂M will overlap with the threshold $V = V_{th}$, and when mass in our density moves into the threshold, these neurons have their potential coordinate reset back to V_{reset}, where it will be clamped there for some refractory period t_{ref}. It is in fact the movement of mass over the threshold which enables population methods to calculate a firing rate [27]. We shall now focus on the PDF and it's evolution over time and state space. Assuming that we have a smooth vector field that is present everywhere on M, we define a PDF $\rho(\vec{v}, t)$ for every $\vec{v} \in M$. Before we start executing a population algorithm, we must discretise M into 'bins', where each bin is of size $d\vec{v}$, and $\rho(\vec{v})d\vec{v}$ is the fraction of neurons in our population whose state lies in $d\vec{v}$. So, if our system if affected by spike trains generated by a Markov process, such as a Poisson process, the evolution of our PDF obeys something known as the differential Chapman-Kolmogorov equation [28] [13]

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial \vec{v}} \cdot (\frac{\vec{F}\rho}{\tau}) = \int_M d\vec{v}' \{W(\vec{v}|\vec{v}')\rho(\vec{v}') - W(\vec{v}'|\vec{v})\rho(\vec{v})\} \tag{2.7}$$

Where $\vec{F}$ and τ are the same components from equation 2.6, and $W(\vec{v}'|\vec{v})$ reflects the probability that at any given time noise will cause a transition from state $d\vec{v}$ to state $d\vec{v}'$ [29]

for any given neuron. If we now consider the synapses in our population of neurons, the simplest choice of synapse is the previously mentioned delta synapses, where the potential of a neuron jumps by efficacy h when receiving an action potential, which consequently means that when the population receives a spike, a portion of our population will jump from $d\vec{v}$ to $d\vec{v}'$. The exact form of this transition actually depends on the noise process, and in this system, we shall be using a Poisson noise process, with rate ν, we can define $W(v'|v)$

$$W(v'|v) = \nu\delta(v' - v - h) - \nu\delta(v - v') \tag{2.8}$$

If we were to consider a fixed value for h, we are able to reduce equation 2.7[28] to

$$\frac{\partial\rho}{\partial t} + \frac{\partial}{\partial\vec{v}} \cdot (\frac{\vec{F}\rho}{\tau}) = \nu(\rho(v - h, v_1, ..., v_{N-1}) - \rho v, v_1, ..., v_{N+1}) \tag{2.9}$$

where the delta functions describe the instantaneous jump that neurons take when receiving input. Importantly, both the right hand side of equations 2.7 and 2.9 state that no mass is ever lost in our PDF, and if one bin loses mass, it can be found in another bin [28]. This covers the dynamics of a population simulation in the presence of input, but what about without the presence of noise, the passive dynamics of the system?

Without the presence of noise in our system, we expect populations of neurons to lose their potential and move towards a more negative state in space. Traditionally, equation 2.9 would have a Taylor expansion on the right-hand side, which leads it to become something called a Fokker-Planck equation, which is a type of equation used to describe the time-evolution of a PDF. This in turn helps us move probability mass according to passive dynamics. This traditional method assumes very small state transitions and bins, something called the diffusion approximation [29], however, the method we describe here doesn't employ this technique. The way that de Kamps and colleagues approach this, an alternative to traditional techniques [28], is to transform equation 2.7 into an ordinary differential equation by introducing a process of coordinate transforms in our state space, which is time-dependent, and creates a new coordinate basis after each time step. When this happens, we obtain a new equation from equation 2.9

$$\frac{d\rho(\vec{v}', t)}{dt} = \nu(\rho(v' - h', v'_1, ..., v'_{N-1}) - \rho(v', v'_1, ..., v'_{N-1})) \tag{2.10}$$

and we now have a coordinate-system that 'co-moves' [28] with the neural dynamics, and accounts for the endogenous behaviour of a population, which has several implications. Firstly, we now have a system that only moves mass in the presence of noise, else, the coordinate system changes to account for our chosen model's behaviour. Secondly, are population simulation process has two interleaved steps: one where we calculate the endogenous behaviour of the population, and one where we numerically solve equation 2.10.

PDTs yield several advantages over typical Monte Carlo methods. The key advantage, especially for 1D simulations, is that they are much faster than simulations of individual neurons [27] [26], and memory usage is much smaller too, as opposed to Monte Carlo simulations which has high memory usage, for processes such as spike buffering and exchange between neurons [13]. Currently, 2D simulations have comparable performance to that of Monte Carlo simulations [28] [13], but importantly still benefits from reduced memory requirements. Furthermore, population methods are enabling better insights into the behaviour of large-scale populations, for example, the neural simulator MIIND was used in research involving populations of neurons in the spinal cord [30], MIIND being a simulator using population density techniques, and used the computed firing rates to compare with real-world EMG data.

Finally, our above explanation covers simulations for 2D models, however in this report, we only consider 1D models. This is a trivial problem to fix. If we consider the second component of the neuron vector $\vec{v}$ to be w, in a 2D model the jumps that this component receives are governed by it's own differential equation, but if we ensure that this equation produces no non-zero values, or just ignore any non-zero values, it enables us to perform a 1D simulation [28]. This is exactly what the simulator MIIND does in this report to simulate LIF neurons (a 1D model), in a 2D environment.

Chapter 3

The Neural Simulators

Here we shall discuss the two simulators that will be the focus on this report: MIIND and NEST. MIIND is a simulator that uses PDTs, whilst NEST performs Monte Carlo simulations.

3.1 MIIND

MIIND, which stands for Multiple Interacting Instantiations of Neural Dynamics, is a neural simulator designed to model the behaviour and interactions of populations using population methods in either a 1- or 2D dynamical system [13]. MIIND is designed such that it is agnostic to the underlying neuron model, meaning an experimenter is able to define custom models, to which MIIND's algorithm will be able to cater for, meaning that it's population techniques can be applied to a variety of different neuron models. It offers much faster performance for 1D simulations compared to the equivalent Monte Carlo [13], and can compete with Monte Carlo for 2D simulations, especially when running it on a GPGPU.

3.1.1 The MIIND Backend

MIIND's backend is written principally in C++ [31], which allows for fast processing of it's core equations and neural dynamics. Specifically, MIIND provides a solution to equation 2.7, for any arbitrary 1- or 2D versions of equation 2.6, with emphasis on the vector field $\vec{F}(\vec{v})$ [13], which determines the endogenous dynamics of a neuron model. When MIIND performs a simulation, it will discretise state space M of the underlying neuron model, where each 'cell' in space is considered to have a uniform distribution of probability mass [13], and will use geometric techniques to move mass. In each iteration of MIIND's computation, it performs 3 steps [13]

1. Probability mass is transferred from a given cell to other cells according to the dynamics of the underlying model, assuming a lack of input.

2. Probability mass is then spread to other cells due to incoming noise. For the delta synapses used in MIIND, this means a simple 'jump' to other cells.

3. If the underlying model uses a threshold-reset mechanism, such as LIF, then any probability mass that has passed the threshold will be removed from the model, and inserted at the cells on the reset potential.

MIIND can use a Monte Carlo version of this algorithm, with a specific amount of point neurons in each cell of it's discretised state space M, but it is best when it employs a geometric method instead.

MIIND employs an XML style simulation set-up, where an experimenter is able to use a single XML file to run large-scale simulations, including the definition of specific neuron models, noise rates, connections between these, and methods of reporting the simulation data. All MIIND simulations use an XML file to define how they behave, however, actually running this experiment can be done using Python. Much like many simulators, MIIND provides a 'non-C++' front-end in order to make it simpler and faster to use. This Python interface is designed to abstract from C++ and will call the relevant functions itself [31]. Importantly, MIIND having a Python interface allows for it to be integrated with other Python-based simulators such as The Virtual Brain [32], giving it greater utility. The Python interface still requires an XML simulation file to be defined.

3.1.2 The Grid and Mesh Algorithms

As previously stated MIIND needs to discretise the neuron state space before performing a simulation, and is has two algorithms which can do that: The Grid Algorithm and the Mesh Algorithm. In this report, we shall be using the Mesh algorithm, but here, we shall briefly discuss the Grid algorithm in order to provide a better understanding of the Mesh algorithm. Both algorithms generate a grid of cells which hold probability mass, but each algorithm defines these grids differently

The Grid Algorithm

Under the grid algorithm, a grid of $M \times N$ identical cells are created, and a transition matrix is used to dictate the movement of probability mass to nearby cells in a single time step according to the underlying model. As mentioned, MIIND's first step in each iteration is to move mass according to underlying dynamics, and in the grid algorithm, it does this by translating the vertices of each cell through state space, which means some cells may not be rectangular anymore. Since the 'new' grid doesn't conform to the Grid algorithms evenly sized rectangle set up, MIIND will then calculate which cells the new quadrilaterals overlap in the grid, and will assign the relevant probability mass to the cells it overlaps. This is done by recursively splitting the quadrilateral into triangles until each triangle covers a single cell, to which the probability mass contained in that triangle is assigned to the overlapping cell.

MIIND will compute the effects of random incoming spikes via a transition matrix M. This matrix dictates how many cells probability mass should move upon receiving a spike, and since the cells in our grid are of equal size, the effect of an individual spike will be the same for all cells: a jump in synaptic efficacy h. When mass moves over the threshold, it is transferred to the bins at the reset potential given there is no refractory period, else, this mass is held in a queue until the refractory period is 'complete'.

The Mesh Algorithm

The Mesh algorithm is similar to the Grid algorithm in many ways, but here, we shall describe how it deviates from it. The Mesh algorithm uses a two dimensional mesh instead of a grid made out of strips which describes the dynamics of the neuron in the absence of input, and follows the so-called characteristic curves of the neuron model [13] [29] [28]. The strip will allow mass to move from any area in state space back towards resting potential. The Mesh for this algorithm is designed such that it's execution is extremely simple, but this means extra time is required for pre-processing. Each strip in the mesh will tend to approach nullclines (areas of straight vertical movement) or stationary points in the field vector, and the width [of the strip] will shrink or expand, meaning they cover less or more of state space respectively, depending on their proximity to such areas.

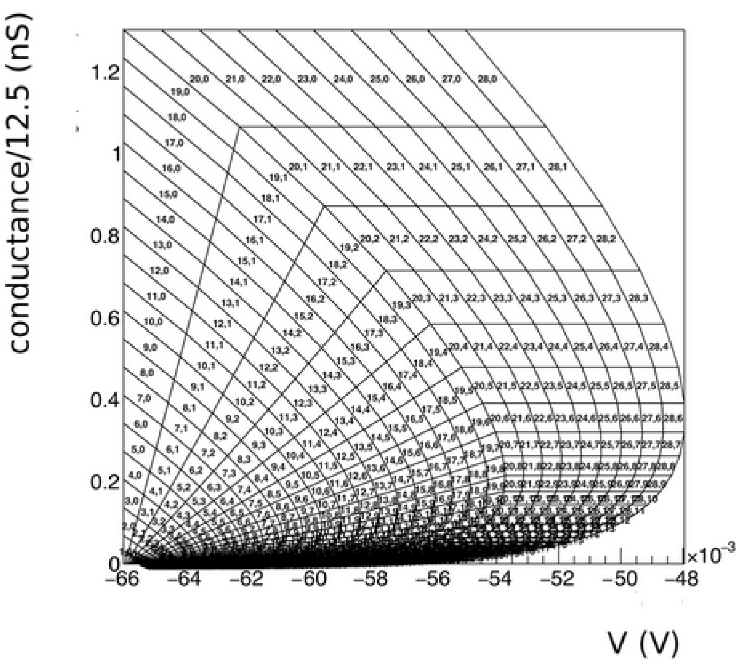

Figure 3.1: A mesh of a two dimensional conductance-based LIF model, where the first number in the cells represents the strip number, whilst the second number is the cell number within the strip [28]. If we look closer, we can see each strip is essentially an array, which probability mass can move across by one each time step, whilst maintaining correct dynamics. Obtained from [28] via [33].

A core difference between the Mesh and Grid algorithm is that each cell in the Mesh algorithm is not necessarily the same size, and this is done such that probability mass only ever needs to move a single cell in a given time step, as the variable size of the cells represents the changes in speed that a population moves through state space, where cells will become larger or smaller depending on how fast or slow the dynamics are respectively. As just mentioned, given that each cell can be variable size, when it comes to simulating the population, the movement of probability mass can be done with simple pointer updates [13]. This is a faster algorithm for solving deterministic dynamics than the Grid algorithm, as the dynamics have been 'pre-solved and baked into the mesh' [13].

3.2 NEST

NEST, which stands for NEural Simulation Tool, is a technology used for the simulation of spiking neural network models, which focuses on the dynamics and structure of neuron networks as opposed to the individual internal makeup and behaviour of a particular neuron model [23]. NEST allows for different neuron models and synapse types to interact in the same simulation. As such, NEST provides the tools necessary to simulate heterogeneous populations. Due to this heterogeneity, NEST uses an adjacency list to maintain neuron connections [23].

NEST has been used in several areas of neurological research, including creating models which mimic brain functions relating to the progression of neurodegenerative diseases [34], and modelling cognitive and sensual processes in the mammalian cortex, where Potjans and Diesmann [35] used NEST to simulate a 1mm^2 patch of neurons in the sensory cortex.

3.2.1 The NEST Kernel

The NEST kernel provides core functionality for NEST [23], and is written in C++. Originally, NEST was controlled using something called SLI, which is a high level, stack-based, scripting language [23], and operated in a command-line environment. NEST's SLI allows a user to create and connect networks of neurons, manipulate arrays and functions, perform mathematical operations, handle exceptions within the network [23], and more. It is important to note that SLI is a language native to NEST, and it is the only way to control the behaviour of a NEST simulation, as this is what the NEST kernel understands. The issue with this is that it difficult to set up simulations and manage them effectively [36], which is why a Python interface for NEST was developed called PyNEST [36]. PyNEST provides a way for researchers to use Python scripts to set up, run, and analyse simulations, which makes using NEST for experiments much easier, as Python is more simple and flexible [36]. This interface provides a layer above the SLI, so whilst using PyNEST, 'under the hood', NEST's SLI is still being used. PyNEST provides an abstraction by sending SLI messages to the NEST kernel from Python functions. In this report, we interact with the NEST kernel via PyNEST.

3.2.2 Key Features and Models

NEST has several different types of models, but the two that we use are Neurons, which are the neuron models, and Devices, which are experimental components for which all the generation of stimulus and the recording of behaviour takes place. In this experiment, we use three different models: a Poisson generator to provide stimulus, a LIF neuron with Delta synapses, and a spike recorder to measure output.

The Poisson Generator

In NEST, a Poisson process, model name `poisson_generator` [37], is designed to mimic a Poisson process, which generates a spike train with Poisson statistics [19], whose rate is defined by the devices property `rate`. The key behaviour to note with `poisson_generator` is that it provides a unique spike train to each neuron is it connected too, which helps provide some variability to a population of neurons and by extension an experiment, something which is especially important with LIF neurons. This is the model we use to provide stimulus in our experiment for NEST.

LIF Neuron with Delta Synapses

The LIF neuron with Delta synapses has the model name `iaf_psc_delta` [38], where `iaf` stands for 'Integrate And Fire', `psc` denotes that we are using a current-based model ('Post Synaptic Current'), and `delta` is the type of synapse being used, which determines how incoming current is treated. This model has standard LIF parameter options, including a reset, resting, and threshold potential, membrane time constant, and a refractory period.

Spike Recorder

Given that we are looking at spiking neuron populations, we want to record the times that neurons in our population spike, which is exactly what the spike recorder does. The spike recorder, model name `spike_recorder` [39], will mark a spike immediately after a neuron produces one, and in our experiment, will output this spike to a file. In order to have our `spike_recorder` write collected spikes to file, we must set it's `record_to` property to `ascii`. When a spike recorder writes to a file (it's name being specified in the PyNEST script) it will write the ID of the neuron that spiked and the time it did so. The spike recorder is a critical instrument in our experiment, as it allows us to track when a neuron in our population fires, which helps us compute a PSTH, which ultimately means we can compare results between simulators.

Parallelism

NEST provides options for both multi-threading and multi-processing with OpenMP and MPI respectively [19]. NEST describes one of it's parallel processing units as a Virtual Process (VP), which specifically references a thread on an MPI processing unit, either local or remote. Whenever NEST performs a simulation in parallel, needs to handle the distribution of neurons, which it does in a round-robin fashion, thus, each VP ends up with a portion of the overall set of neurons used in the simulation. This presents a challenge for NEST, as one neuron could spike on one VP, but the post-synaptic neuron may be on another VP; this is something handled by NEST's global spike exchange mechanism [19], and under this mechanism, if the neurons are on different VPs, then it will be that different VP is responsible for neuron updates and spike generation, whilst the the pre-synaptic neuron's VP will be responsible for delivering it to the correct neuron/ VP.

Given this, we have established that neurons are distributed across VPs and not copied, however, devices are treated differently. When a simulation is set up with a device to record behaviour or to produce a stimulus, these devices get copied across VPs, such that each VP will have its own set of devices. This means that during simulation, we will have dN_{vp} devices, where d is the number of devices defined/ would be present in serial, and N_{vp} is the number of VPs. Due to this, when we run a simulation in parallel, we produce a data file for each copied recording device, so if we had a single spike recorder, and ran our simulation on 8 local threads, that would mean we generate 8 data files containing spikes from each VP's distribution of neurons. It should be noted that if we ran the same simulation in serial, we would get an (almost) identical result, where all the spike data would be contained in a single file, as opposed to 8 in the former example.

Chapter 4

The Method

The aim of this experiment is to find and analyse discrepancies found between the simulation of recurrent-networks between MIIND and NEST, with the context that we want to see how MIIND differs to that of NEST, with NEST being the simulator that produces the 'correct' results. In this chapter, we shall describe benchmarking efforts, the experiment details, and the experiment procedure.

4.1 Benchmarking Simulator Performance

Before we talk about data collection and the experiment procedure, first, we should confirm that MIIND and NEST do in fact produce congruent results for some non-recurrent network types, since we are trying to discern differences between the simulators for recurrent networks. We shall do this by bench-marking these simulators with two benchmark procedures. First, we shall recreate an experiment done by Omurtag et al. (2000) [26], and then we shall then look at a technique of producing Gaussian White Noise using a Poisson distributed input, as seen in de Kamps, M (2013) [29].

4.1.1 Omurtag's experiment

In Omurtag's experiment, there is a specific population set up used to investigate the differences between Monte Carlo and population methods for simulating neurons, and in figure 4.1, we can see the results of this experiment, and the inset is the curve, plotting firing rates, which we aim to generate from both MIIND and NEST. In this experiment, an LIF neuron with delta synapses was used, with a membrane time constant of $\tau = 50$ms, a synaptic efficacy of $h = 0.03$, and an input firing rate of $\nu = 800$Hz, finally, the threshold potential and reset potential were 1.0mV and 0.0mV respectively.

Since population methods aim to simulate an arbitrarily large population of neurons, we should also use a large number of neurons for a Monte Carlo comparison. For this benchmark, we use 10^6 neurons for the NEST simulation, as larger numbers of point neurons show behaviour closer to population methods than smaller, and also shall reduce the number of random fluctuations in results, something seen in direct simulations[26]. We then compare this to a MIIND simulation using the Mesh algorithm, with both simulators using the exact same parameters.

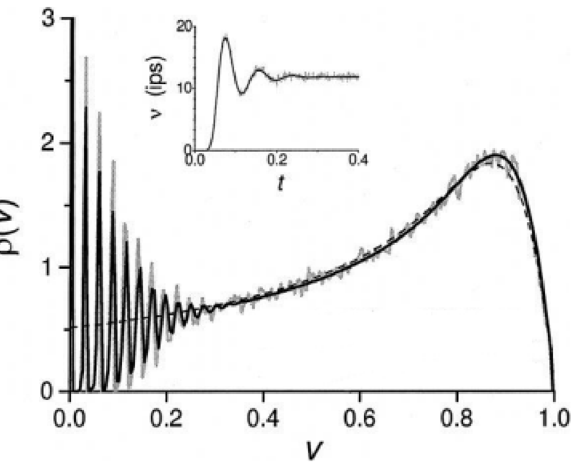

Figure 4.1: Obtained from [26] via [33], we can see the PDF of a population of neurons at equilibrium (where the firing rate stabilises). The smooth black line shows the PDF as computed via population methods, whilst the jagged grey line shows the PDF as computed via direct simulation of 90,000 neurons. Inset: The firing rate of the population under population methods (black line) and under direct simulation (grey line).

The results of this benchmark are discussed in section 5.1.

4.1.2 Gaussian White Noise

Gaussian White Noise (GWN) is a type of random process where each generated noise value is selected from a Gaussian distribution, specifically with a mean of 0 and a finite variance. Here, we talk about about it in a slightly different way, as we attempt to generate a GWN process from neurons using a Poisson noise process, as seen in de Kamps, M (2013) [29]. In this paper, we see how we can set the firing rate and the synaptic efficacy of an input to generate a Gaussian distribution for a neuron population, with mean μ and variance σ^2, whose spikes/ firing rate approximate our desired GWN process. To determine the parameters of our noise process, we can either select an efficacy h and firing rate ν, and use the resulting μ and σ^2, alternatively, and the approach used here, μ and σ^2 can be pre-determined, and the efficacy and firing rate resultant from this. This is done using equation 4.2, reworked from equation 4.1

$$\begin{aligned} \mu &= \tau\nu h \\ \sigma^2 &= \tau\nu h^2 \end{aligned} \tag{4.1}$$

Which can be rewritten into

$$\begin{aligned} h &= \frac{\sigma^2}{\mu} \\ \nu &= \frac{\mu^2}{\tau\sigma^2} \end{aligned} \tag{4.2}$$

where τ is the membrane time constant. We can choose a μ and σ^2 to find our efficacies and input firing rate, and in fact, the previously mentioned Omurtag experiment is a member of this set of populations ($\mu = 1.2$, $\sigma^2 = 0.036$). Now, equation 4.2 works for small σ and large μ, which results in a small h and a large ν, however, when σ becomes larger, and the difference between μ and σ decreases, equation 4.2 starts to break down. This is a result of the synaptic efficacy being too large for any diffusion regime (the ability for a population of neurons to distribute itself across state space) to work, since the neurons make large jumps and hit the threshold too quickly, preventing any Gaussian distribution from forming. As such, when σ becomes large, we must use equation 4.3

$$\begin{aligned}\mu &= \tau J(\nu_e - \nu_i) \\ \sigma^2 &= \tau J(\nu_e + \nu_i)\end{aligned} \tag{4.3}$$

Which can be rewritten into

$$\begin{aligned}\nu_e &= \frac{J\mu + \sigma^2}{2\tau J^2} \\ \nu_i &= \frac{J\mu - \sigma^2}{2\tau J^2}\end{aligned} \tag{4.4}$$

where J is a fixed synaptic efficacy, ν_e is an excitatory input rate, and ν_i is an inhibitory input rate. We use J in place of h from equation 4.1. Given that we have a fixed J, this just leaves the input rates to be determined, and the use both an excitatory and inhibitory input allows a Gaussian profile to be maintained whilst accommodating for larger variance in our population distribution. Technically, it is possible to use equation 4.4 for all values of σ, where for low σ we set $\nu_i = 0$, but in reality using equation 4.4 will result in an illegal negative inhibitory rate if used incorrectly. To note, the value of σ is crucial in determining the firing rate of a GWN process, as it determines how much of the population will cross the threshold potential, should we have no variance in our population, then all neurons will sit at the value of μ, meaning they have no chance of firing, or will fire at very high rates.

In this benchmark, we test the values of μ over different σ's, using both the methods mentioned above, if $\sigma < 0.5$, then we use equation 4.2, and if $\sigma \geq 0.5$, then we use equation 4.4 instead. For this benchmark, we use membrane time constant $\tau = 50$ms, and for $\sigma \geq 0.5$, we use $J = 0.1$. The results of this benchmark are discussed in section 5.1.

4.2 Experiment Procedure

For this experiment, we aim to collect data from different population set-ups, primarily focused on the number of connections present in the system, but also on the type of system and the synaptic efficacies used. This experiment was written in Python, as this is the language that the interface for both MIIND and NEST is written in. Our experiment program was executed on the ARC3 systems at the University of Leeds [40].

4.2.1 Experiment Parameters

Network Type

When we refer to the network type, we describe how our populations are organised in terms of their connections between each other. In this experiment, we shall test on two different network types: Balanced-EI, and a self-connected population. A self-connected population is a single population with something called a feedback loop. This is where it connects to itself, such that it's spikes should then induce more spikes in the population, thus creating positive-feedback. We use this network as a simple test for the simulators.

Balanced E-I (or Balanced Excitation-Inhibition) is a more interesting concept in neuroscience and computational neuroscience alike, commonly used when modelling neural activities [41], and seen often in different areas of brain function, such as working memory [41]; it even suggested to be a contributing factor in neurodivergence [42]. Here, we focus on the Balanced E-I network, which is a network of two populations, one excitatory, one inhibitory. Both populations are connected to each other and themselves, which means that both populations receive excitatory and inhibitory feedback, either from themselves or the other population. Finally, both populations will be connected to the same stimulus. We use this network as a more complex test for our simulators. In this experiment, we don't use a 'true' balanced E-I network, instead we use it's structure as a template to test connection parameters on it.

Connections

In reference to connections, during the experiment, we change the number of connections that each network has. When we refer to the 'number of connections', we talk about the number of incoming connections each neuron has. This means that all neurons in the populations will have incoming connections from other neurons. In addition, each neuron has a single incoming connection from a Poisson input, which provides external stimulus for our networks.

For the balanced E-I network, we loop through a range of excitatory and inhibitory connections, such that for each number of excitatory connections, we test all possible numbers of inhibitory connections. The number of excitatory connections refers to the in-degree for neurons in the inhibitory population from the excitatory population, and the in-degree of connections for the excitatory population's feedback loop. The same principle applies to the inhibitory connections. For the balanced E-I network type, we test for excitatory and inhibitory connections in the range of $[0, 250]$, with an interval of 10 for both. For the self-connected network, we simply loop through each possible number of connections in the feedback loop, testing each number of connections. For this network, we test feedback connections in range $[0, 250]$, with intervals of 10.

Synaptic Efficacy

Synaptic efficacy is an important parameter to alter, as it allows us to test these networks with a variety of different input strengths, which gives a better overall picture of the networks' behaviour. In this experiment, we don't loop over synaptic efficacies, instead, we randomise them over a series of trials, and then average the results. For both types of networks, we randomise the efficacy of the connection between the Poisson input and the populations (for the balanced E-I network, both populations use the same input efficacy), whilst all other efficacies are fixed at 1. This was done as the Poisson input will be the dominating factor in determining the firing rate of the population, thus, it seem logical to alter this to give a broader variety of results. Furthermore, it was originally planned to randomise the value of all of the connections, however, this would've meant to many points of randomness in our data, making it unreliable, and needing an intractable number of repetitions for each set-up. The synaptic efficacies for each test shall be selected between the interval $[0.9, 1.0]$, and the number of repetitions will be 5, which was selected in order to have efficacies which produced high enough firing rates for spike compilation, but also in terms of making sure the experiment could be done in tractable time.

Another decision that was made regarding efficacies was whether to use heterogeneous efficacies or homogeneous efficacies throughout the population, as originally, we used a feature of NEST's interface to create many connections all with different efficacies. However, it was decided that this created too many points of randomness, thus requiring further trials. Of higher precedence however, was the fact that MIIND only allowed a single synaptic efficacy to be defined between populations, meaning the connections between two populations in MIIND were homogeneous. This means that in order to perform a fair comparison between the simulators, NEST also had to be using homogeneous efficacies.

Control Variables

This experiment has several control variables which we shall briefly discuss. Firstly, we control the number of neurons used in each NEST simulation, and on each trial and set-up, the number of neurons for NEST is 10^5, for MIIND, we don't specify a population size, but we will use the Mesh algorithm for all tests. Next, the firing rate for all tests is 800Hz, which was chosen because it elicits enough spikes to generate reliable firing rates in NEST, but also being small enough such that each simulation takes a tractable time, based on the rates magnitude. The neuron model we shall use will be the LIF with Delta synapses with parameters $\tau = 50$ms, $V_{th} = -55$mV, $V_{rest} = -70$mV, $V_{reset} = -70$mV, and $V_{min} = -75$mV (the minimal potential). Lastly, the simulation time for each experiment will be 500ms.

Firing Rates

The variable which we shall record during this experiment is the firing rate of the populations simulated. For MIIND, this is trivial, as the process of calculating a firing rate is abstracted from the user, however for NEST, we have to calculate the firing rate ourselves. This is done using both techniques mentioned in section 2.3, where first we calculate the population firing rate using equation 2.5, and then 'take inspiration' from equation 2.4 to then calculate an average over several runs (in practise, this just involves calculating a mean average over a list). We can then generate a PSTH, and plot a curve over it, which allows us to see the firing rates of populations in NEST over time. We see this in equation 4.5, which describes how the average firing rate for a specific population set up is computed

$$A(t) = \sum_{k=1}^{K} \frac{1}{K} \frac{1}{\Delta t} \frac{n_k(t; t + \Delta t)}{N} \tag{4.5}$$

Where $A(t)$ is the average firing rate at time t, K is the number of trials being performed, and k is the trial number. The function $n_k(t; t + \Delta t)$ is the number of spikes generated by the population during trial k between times t and $t + \Delta t$. Finally, N is the size of the neuron population.

Chapter 5

Results

5.1 Benchmark Results

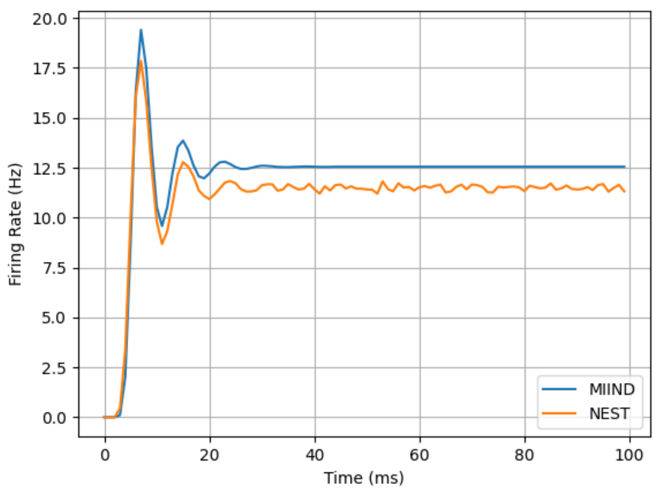

Figure 5.1: The Omurtag benchmark

After simulation of the Omurtag experiment, we see the results shown in figure 5.1. We see at time $t = 20$ that the population starts to reach equilibrium for both simulators, and before $t = 20$, both simulators have the same movement of their populations through state space, peaking just before $t = 10$ and hitting it's minima just after. We can also see the variations that the NEST population experiences after stabilising, which is also seen in figure 4.1, and that MIIND's population movement is much smoother, again, an observation we can make from Omurtag et al. (2000) [26]. However, it should be noted that MIIND's population methods leads to a consistently higher firing rate than NEST, of roughly 0.5Hz throughout the simulation, and although this might seem like a discrepancy of concern, in reality it's trivial, as it is something which can be solved by increasing the resolution/ bin count used in MIIND. When comparing the outcome of this setup to figure 4.1, we see that NEST produces the accurate plot, but crucially, both MIIND and NEST show the same trend in behaviour through time.

In relation to GWN, we see good agreement between the two simulators, especially as the value of σ increases to $\sigma = 0.9$, and below this the simulators have a good comparison between curves, with the main exception being at $\sigma = 0.1$ for $\mu = 0.8, \mu = 0.9$. However, we can attribute this to the differences between the Monte Carlo and population methods, in particular given that both set-ups have lower firing rates, where they would be sensitive to discrepancies and minor changes.

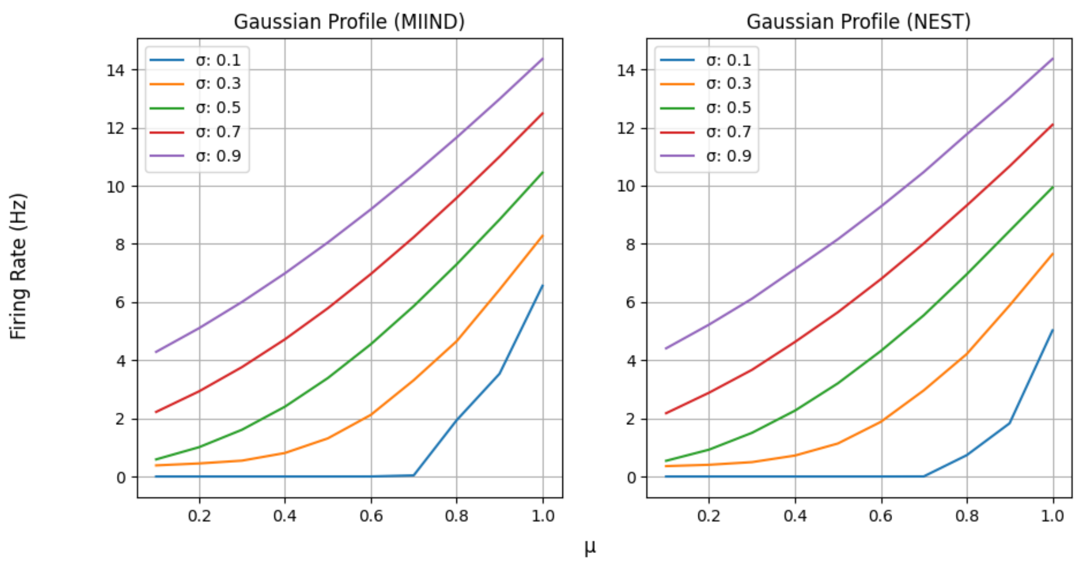

Figure 5.2: The Gaussian White Noise benchmark

5.2 Discussion of Results

In summary, clear discrepancies can be seen between self-recurrent networks in MIIND and NEST. In this discussion, we shall be using two types of visualisations to display and explain our findings. First, we shall be using something called a gain curve, used in figure 5.3, which plots the firing rate of a population against another parameter in the network. The aim of a gain curve is to plot how the firing rate of a population increases as we change another factor, for the self-connected population, we use the number of connections in the feedback loop as this parameter. The other visualisation we shall be using is in figure 5.4, which is called a heat-map, and plot how a 3rd variable varies against two other factors. In this case, we plot how the average firing rate changes as the number of excitatory and inhibitory connections change. For each network, we shall discuss the results from the NEST simulator (what we are deeming the 'correct' result), and then seeing how MIIND differs.

The results of the experiment on self-connected networks can be seen in figure 5.3. If we look at the gain curve generated from NEST (right), we see how initially there is a sharp rise in firing rate gained from the increase in connections, which we shall refer to as n. When $n = 50$, it starts to level off, but the overlay-ed trend line shows us that as n increases beyond $n = 50$, we see a gradual increase of firing rate and eventually stops increasing altogether, and maintaining a zero-gradient.

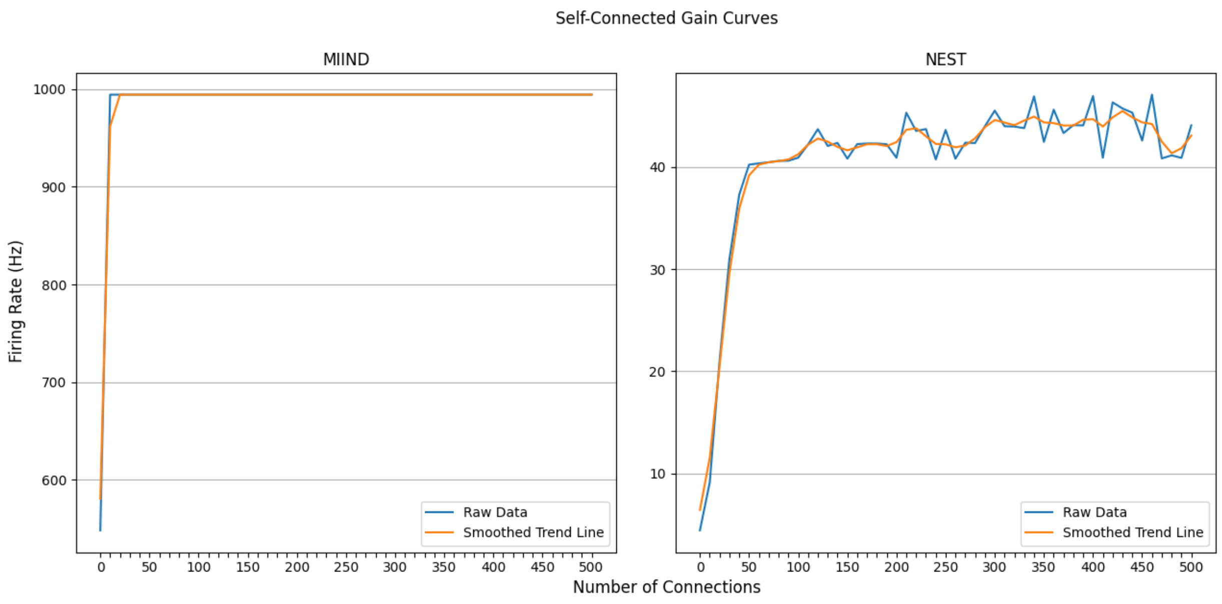

Figure 5.3: Results from the trials on self-connected populations. It should be noted that the 'Smoothed Trend Line' is present to give a better idea of the progression of the curve, but isn't truly faithful to the original results.

In comparison, the results from MIIND's curve (left) show something very different, and although it maintains a steep rise at the beginning, this is where the similarity end's with NEST's results, as MIIND sees the steep rise after $n = 10$. After this, all gain stops, and each population continues to average just below 1000Hz. Now although it was to be expected that MIIND would produce a 'smoother' result, as mentioned in section 4.1.1, because population methods tend to have less variation in there results compared to Monte Carlo methods, MIIND still shows unusual behaviour in that there is absolutely no change in firing rate as n increases, by either variation or upwards trends. Another key observation that can be made on MIIND's results is that they are almost two orders to magnitude higher than NEST's, where the populations under NEST had never exceed 50Hz, but MIIND's populations always go above 500Hz approaching 1000Hz. In continuation, almost every set-up in MIIND, the first computed firing rate that was above 0Hz was 550.671Hz, which is actually the average firing rate of the population with $n = 0$, which is the population with no feedback at all. So even when there is no feedback in our populations, MIIND still shows a significance different to that of NEST, where the average firing rate for NEST's no feedback population is 4.438Hz.

The results for the Balanced E-I network show something quite similar to that of the self-connected networks: NEST shows a clear developing pattern, whilst MIIND shows a constant value. The results are visualised in figure 5.4 and we can see some very clear patterns in the NEST results. The overarching trend is how the firing rate rapidly decreases as the number of inhibitory connections increases, and how more inhibitory connections are required to reduce the firing rate of both populations. Indeed, we see what can be described as a 'terminal' line in the heatmap, almost perfectly dividing the image in half. The average firing rate of the population (either excitatory or inhibitory) rapidly drops as the the number of inhibitory connections increases towards the number of excitatory connections. When the number of inhibitory connections is equal to the number of excitatory connections, the rate that the average firing rate decreases slows, but still gradually decreases as the number of inhibitory connections continues to grow. This would suggest that the relationship between the number of inhibitory connections and the average firing rate is inversely proportional. It would also seem that when the number of excitatory connections is much higher than the number of inhibitory connections, the average firing rate is more variant, which can be seen starting from 110 excitatory connections and above. Although, it would be the case that in this area, the average firing rate grows as the number of inhibitory connections increases, but it quickly starts to decrease when the number of inhibitory connections is close to the number of excitatory connections. Finally, an important feature to note between the excitatory and inhibitory populations is how they 'mimic' each other, as the number of connections change, the average firing rates of the populations increase and decrease together.

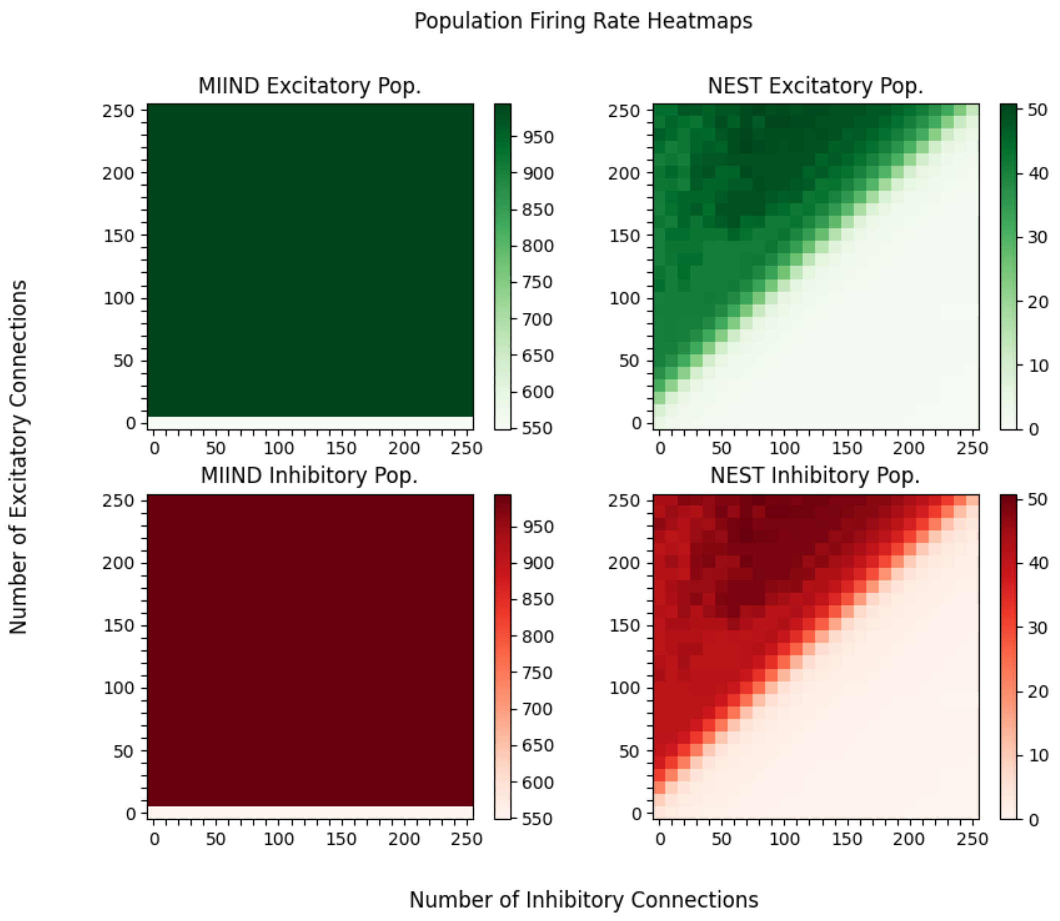

Figure 5.4: The results from the Balanced E-I experiments, where each heatmap represents the average firing rate of either the excitatory or inhibitory population. The colour intensity represents the average firing rate in Hz from that particular experiment, and the green heatmaps show the excitatory populations, whilst the red show the inhibitory populations.

In contrast, MIIND's balanced E-I simulation shows something very similar to that of the self-connected experiment. When there is no feedback, the population fires at constant 550.671Hz, but when some form of feedback is introduced, the firing rate immediately increases towards 1000Hz. MIIND's simulation shows no resemblance at all to the results produced in NEST, however, it does still have an important quality that NEST's does, which is the fact that both populations also mimic each other, abet nearly two orders of magnitude above NEST's.

One would assume that due to feedback in the population causing firing rates to almost double, that the presence of feedback is causing too much probability mass to move over the threshold voltage, a problem NEST does not have to content with. However, without further testing, the actual source of the problem cannot be determined.

5.3 Conclusion

To conclude, we can see clear discrepancies between the computed firing rates for recurrent networks between MIIND and NEST, where NEST shows clear trends and patterns as the number of connections change, whilst MIIND consistently produces a much higher firing rate when feedback is introduced into the population. Furthermore, MIIND fires at a fairly consistent rate when any feedback is introduced, at just below 1000Hz, which means that connection parameter changes in the MIIND simulation does not produce any disernable difference.

5.4 Future Work following this

A productive and informative task following this work would be to redo the experiment with different parameters, including different pre-set values and parameter ranges. Two particular aspects of this would be changing the ranges of synaptic efficacies used to a wider range, and especially towards a range with a lower bound closer to 0. In combination with this, altering the firing rate used to generate these results would also allow for a greater range of results to help solve this issue. It may also give a different set of MIIND results, and potentially we may see more details trends on MIIND's part. It would also be of interest to perform the balanced E-I experiment again with a larger range of connections, which would help to better understand it's behaviour.

Furthermore, it may be worth performing this experiment on other types of recurrent networks, and even multiple recurrent populations in tandem to gain a better understanding of how their behaviour deviates in MIIND from NEST. This experiment could also be used as a framework to help solve issues with other models that show discrepancies for recurrent networks in MIIND, such as with a conductance-based LIF or other variants of the IF model paradigm.

References

[1] Q. B. Institute, “What is a neuron?.” https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron, 2019. Accessed: 2022-03-07.

[2] E. Kandel, J. Koester, S. Mack, and S. Siegelbaum, *Principles of Neural Science, Sixth Edition.* McGraw Hill, 2021.

[3] S. Herculano-Houzel, “The human brain in numbers: a linearly scaled-up primate brain,” *Frontiers in Human Neuroscience*, vol. 3, 2009.

[4] R. Williams and K. Herrup, “The control of neuron number,” *Annual Review of Neuroscience*, vol. 11, pp. 423–453, 1988.

[5] T. Bekinschtein, D. Bor, C. Jarrett, R. Kanai, M. O’Shea, A. Seth, and J. Ward, *30-SECOND BRAIN.* Quarto Publishing plc, 2017.

[6] W. Gerstner, K. Werner, R. Naud, and L. Paninski, *Neuronal Dynamics.* Cambridge University Press, 2014. Accessed 2021-10-25.

[7] A. Chandra, “Mcculloch-pitts neuron - mankind’s first mathematical model of a biological neuron.” https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1, 2018. Accessed: 2022-04-25.

[8] C. Henley PhD, *Foundations of Neuroscience.* Michigan State University, 2021. Accessed: 2022-03-15.

[9] M. Grider, R. Jessu, and R. Kabir, “Physiology, action potential.” https://www.ncbi.nlm.nih.gov/books/NBK538143/, 2021. Accessed: 2022-03-15.

[10] J. Moini MD, N. Avgeropoulos MD, and M. Samsam MD, PhD, *Epidemiology of Brain and Spinal Tumors.* Academic Press, 2021. Accessed: 2022-03-17.

[11] D. Purves, G. Augustine, D. Fitzpatrick, and et al., *Neuroscience, 2nd edition.* Sinaer Associates, 2001. Accessed 2022-03-17.

[12] A. Hodgkin and A. Huxley, “Action potentials recorded from inside a nerve fibre,” *Nature*, vol. 144, pp. 710–711, 1939.

[13] H. Osborne, Y. Lai, M. Lepperød, D. Sichau, L. Deutz, and M. de Kamps, “Miind: A model-agnostic simulator of neural populations,” *Frontiers in Neuroinformatics*, vol. 15, 2021.

[14] A. Burkitt, “A review of the integrate-and-fire neuron model: I. homogeneous synaptic input,” *Biol Cybnern*, vol. 95(1), pp. 1–19, 2006.

[15] L. Abbott, “Lapicque’s introduction to the integrate-and-fire model neuron (1907),” *Brain Research Journal*, vol. 50, pp. 303–304, 1999.

[16] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of Neurophysiology*, vol. 94(5), pp. 3637–3642, 2005.

[17] M. Häusser, "The hodgkin-huxley theory of the action potential," *Nature Neuroscience*, vol. 3, p. 1165, 2000.

[18] E. De Schutter and N. Brunel, *Computational Modelling Methods for Neuroscientists.* The MIT Press, 2010.

[19] "Nest simulator documentation." `https://nest-simulator.readthedocs.io/en/stable/index.html`, 2021. Accessed 2021-12-15.

[20] Koehrsen, "The poisson distribution and poisson process." `https://towardsdatascience.com/the-poisson-distribution-and-poisson-process-explained-4e2cb17d459`, 2019. Accessed: 2021-11-03.

[21] A. Aldo Faisal, L. Selen, and D. Wolpert, "Noise in the nervous system," *Nature Reviews Neuroscience*, vol. 9, pp. 292–303, 2008.

[22] P. Martínez-Cañada, T. Ness, G. Einevoll, T. Fellin, and S. Panzeri, "Computation of the electroencephalogram (eeg) from network models of point neurons," *PLOS Computational Biology*, vol. 17, 2021.

[23] M. Gewaltig and M. Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[24] G. Deco and A. Thiele, "Cholinergic control of cortical network interactions enables feedback-mediated attentional modulation," *European Journal of Neuroscience*, vol. 34, pp. 146–157, 2011.

[25] G. Mongillo, O. Barak, and M. Tsodyks, "Synaptic theory of working memory," *SCIENCE*, vol. 319, pp. 1543–1546, 2008.

[26] A. Omurtag, B. Knight, and L. Sirovich, "On the simulation of large populations of neurons," *Journal of Computational Neuroscience*, vol. 8, pp. 51–63, 2000.

[27] E. Haskell, D. Nykamp, and D. Tranchina, "Population density methods for large-scale modelling of neuronal networks with realistic synaptic kinetics: cutting the dimension down to size," *Network: Computation in Neural Systems*, vol. 12, pp. 141–174, 2001.

[28] M. de Kamps, M. Lepperød, and Y. Lai, "Computational geometry for modeling neural populations: From visualization to simulation," *PLOS Computational Biology*, vol. 15, 2019.

[29] M. de Kamps, "A generic approach to solving jump diffusion equations with applications to neural populations," 2013. Accessed: 2022-03-20.

[30] G. York, H. Osborne, P. Sriya, S. Astill, M. de Kamps, and S. Chakrabarty, “A neural circuit model of proprioceptive feedback from muscles recruited during an isometric knee extension,” *bioRxiv*, 2021. Accessed: 2022-04-11.

[31] H. Osborne and M. de Kamps, “Miind: a population level simulator..” `https://github.com/dekamps/miind`, 2022. Accessed: 2022-04-12.

[32] P. Sanz Leon, S. Knock, M. Woodman, L. Domide, J. Mersmann, A. McIntosh, and V. Jirsa, “The virtual brain: a simulator of primate brain network dynamics,” *Frontiers in Neuroinformatics*, vol. 7, 2013.

[33] T. Willis, “Modelling neural circuits with synaptic plasticity.” Unpublished, 2019. Submitted as part of BSc Computer Science (Industrial) degree program at the University of Leeds.

[34] C. Bachmann, T. Tetzlaff, S. Kunkel, P. Bamberger, and A. Morrison, “Effect of alzheimer’s disease on the dynamical and computational characteristics of recurrent neural networks,” *BMC Neuroscience*, vol. 14, 2013.

[35] T. Potjans and M. Diesmann, “The cell-type specific cortical microcircuit: Relating structure and activity in a full-scale spiking network model,” *Cerebral Cortex*, vol. 24, no. 3, pp. 785–806.

[36] J. Eppler, M. Helias, E. Muller, M. Diesmann, and M. Gewaltig, “Pynest: A convenient interface to the nest simulator,” *Frontiers in Neuroinformatics*, vol. 2, p. 12, 2009.

[37] “poisson_generator – generate spikes with poisson process statistics.” https://nest-simulator.readthedocs.io/en/v3.3/models/poisson_generator.html, 2021. Accessed: 27-04-2022.

[38] “iaf_psc_delta – current-based leaky integrate-and-fire neuron model with delta-shaped postsynaptic currents.” https://nest-simulator.readthedocs.io/en/v3.3/models/iaf_psc_delta.html, 2021. Accessed: 27-04-2022.

[39] “spike_recorder – collecting spikes from neurons.” https://nest-simulator.readthedocs.io/en/v3.3/models/spike_recorder.html, 2021. Accessed: 27-04-2022.

[40] U. of Leeds Research Computing Team, “Arc documentation.” `https://arcdocs.leeds.ac.uk/welcome.html`. Accessed: 2022-04-21.

[41] G. Tian, S. Li, T. Huang, and S. Wu, “Excitation-inhibition balanced neural networks for fast signal detection,” *Frontiers in Computational Neuroscience*, vol. 14, 2020.

[42] V. Sohal and J. Rubenstein, “Excitation-inhibition balance as a framework for investigating mechanisms in neuropsychiatric disorders,” *Mol Psychiatry*, vol. 24(9), pp. 1248–1257, 2019.

[43] O. Meijboom, “Ethical issues associated with the use of animal experimentation in behavioural neuroscience research,” *Current topics in behavioral neurosciences*, vol. 19, pp. 3–15, 2015.

[44] H. Algahtani, M. Bajunaid, and B. Shirah, “Unethical human research in the field of neuroscience: a historical review,” *Neurological sciences: official journal of the Italian Neurological Society and of the Italian Society of Clinical Neurophysiology*, vol. 39, pp. 829–834, 2018.

[45] L. Squire, “The legacy of patient h.m. for neuroscience,” *Neuron*, vol. 61, pp. 6–9, 2009.

[46] F. Miller and T. Kaptchuk, “Deception of subjects in neuroscience: An ethical analysis,” *The Journal of neuroscience: the official journal of the Society for Neuroscience*, vol. 28(19), pp. 4841–4843, 2008.

[47] “The ethical neuroscientist,” *Nature Neuroscience*, vol. 11, p. 239, 2008.

Appendix A

Self-appraisal

Here, we shall now switch to the use of first-person pronouns where appropriate.

A.1 Critical self-evaluation

This project had several things that went well, but also things that could've been improved upon. The areas of this research that went well were developing a core understanding of the subject, the experiment supporting a positive result, and the data-management throughout the project.

Firstly, given that this project involved an area is not taught on the Leeds Computing Curriculum, at least for the BSc program, it was clear that a large part of this work was going to be researching and synthesizing information in order to be able to proceed with developing a project aim. I believe that this was done with a good degree of success, and as shown in chapters 1-3, where I display a solid intuition and understanding to not only the mathematical processes being used to simulate neuron behaviour, but also the biology which underpins it. In continuation, this understanding has led me to developing an experiment designed such that it shows clear discrepancies between the MIIND and NEST simulators, and I believe that the experiment was for the best part, well designed. This is because through the process of developing it, I had to make several decisions, such as reducing the areas of randomness, and decreasing the number of altered parameters, to ensure that it could be considered scientific as possible. On top of this, I had to ensure that both simulators were performing the same task, using the same set up of parameters, and a good example of this is where I had to change the NEST set-up regarding the heterogeneity and number of connections to match MIIND's way of defining connections.

Finally, I would argue that for someone who has never done a large-scale data project, that the data-management, generation scripts, and file-structure was very good. From the get-go, I aimed to ensure that data was organised such that I could easily see the results of a specific experiment, and that it was also easy for my analysis program to extract the data files with minimal debugging and time required to build and execute the program. This is something that I believed made this project run much smoother than it would've done without good data handling. This was also helped by the fact that I understood what visualisations I wanted to create before running the experiment, which meant I could tailor the experiment such that it produced understandable visualisations.

On the contrary, there were several things that definitely held the project back can be considered areas of improvement. These areas are the management of the experiment execution, managing project progress, and the issues I faced at the start of this project.

In regards to experiment execution, the big issue I had was managing all of the jobs which needed to be ran on the ARC3 systems, especially when considering the time I had left to be able to execute them. Personally, I think I didn't utilise the full set of tools and options available to me for running my programs, which led to a very disjointed process of generating my final data. If I were to deal with this again, I would run more tests on smaller programs and data parameter ranges in order to test the limits of the computing cluster and what I was allowed to do, as opposed to panicking about how long the programs were taking to execute.

Furthermore, I believe a key aspect of my workflow that hindered this project was my organisation. I took quite an adhoc approach to developing the experiment which often left me second-guessing myself. This was arguably a result of two things. First, although I planned the core premise and aims of the experiment, I didn't plan the fine details of it, which left many parameter values ambiguous, causing them to be chosen based on precedence as opposed deriving them via a logical process. A good example of this would be the choice of firing rate in the final experiment, where I chose 800Hz to be rate of the external stimulus because I had used it in a previous benchmark, and I got it from testing which order of magnitude that I wanted the stimulus, but didn't think to decrease or increase the value within the magnitude. Finally, there were issues I faced getting started with this project, as I felt very overwhelmed by the amount of content that I had to read and understand before I could decide what direction I wanted to take. This is a realisation I had once I started collecting references for the final report, where I found many resources which would've been helpful at the time, or delved further into papers that I had seen which clarified many points of confusion I had at the time, but didn't read then because I felt drowned in content.

A.2 Personal reflection and lessons learned

I believe that doing this project has taught me many valuable lessons which I shall employ in future projects, either academic or industrial. Firstly, this project for me was great exposure to the world of academia, as it was a novel challenge for me to need to find information not from a single textbook, but instead from a collection of literature. Being able to synthesize this in a way that not only I understand but also for someone else to understand helped me improve my ability to use resources other than textbooks to acquire information. However, as someone who does struggle to read large passages of text, a key lesson I have learnt is that academia may not be an area where I would thrive in the future. Where instead, I may enjoy a more practical project with more testing and programming than reading and investigation.

Another big issue I knew was going to be a problem when I came into this project was my mathematical skills. I wouldn't regard my maths as poor, but at the same time, having not done an A-level qualification I understand that it is not up to the standard required to navigate computational neuroscience. So whilst I learnt the necessary maths and understand it now, I believe a lot more project progress could've been made earlier if I had better initial understanding.

In continuation, I would say that the type of communication I had with my supervisor, Marc de Kamps, could've been better. I often posed my questions either not as direct questions or will ask them before I have spent good time trying to find the answer myself. This meant that sometimes I didn't understand what Marc was talking about, and had to have it re-clarified later, and phrasing questions correctly for them - especially over email - sometimes lacked. However, after the assessor meeting with Abdulrahman Altahhan, this was a problem that sorted itself out after we started having more in-person discussions, something which I greatly benefited from.

A.3 Legal, social, ethical and professional issues

A.3.1 Legal issues

Typically, with research projects involving data analysis such as this, data may be obtained off human participants, to which we have to have proper handling of personal data to ensure data confidentiality and anonymity, amongst other things. The data acquired in this report does not come from human participants but instead was collected from software modelling abstract mathematical processes, meaning that there are no legal issues with data.

Furthermore, both the simulators used in this report have public repositories on GitHub and are not considered proprietary. This means that there are no legal issues with using others intellectual property without permission, as both simulators have been referenced and links to the relevant websites for more information can be found in Appendix B and the references section. Finally, I do not claim to have contributed or have any ownership over either simulator, they have been developed and belong to others.

A.3.2 Social issues

This project has no direct, or indirect, social ramifications, or issues attributing to it's creation. Since the aim of this report is to help solve an issue with software which simulates an abstract mathematical process, the proper functioning of this software wouldn't have any problematic social repercussions. It could be argued that since this software is used to model human biological processes, should it be used in future research which could have social ramifications, then one might consider that a social problem for this research. However, I would argue that this is unrealistic, and in practise, the most impacting this report will be is to solving an issue with recurrent networks in MIIND.

A.3.3 Ethical issues

Although this report directly does not incur any ethical issues, due to it discussing abstract mathematical models, it may be worth addressing that much of the reading done for this report is based off centuries of research done by the neuroscience community, a lot of which were performed on animal [43] and human subjects, which have rigorous ethical standards, due to horrific experiments being done in the past [44]. Some knowledge in neuroscience also comes from case studies, such as the famous case of H.M. [45], who had part of his hippocampus removed during surgery, and lost the ability to form new memories, who couldn't possibly have had the ability to give informed consent [46] for the research that was done on him. To cite relevant work, in this report we mentioned the Hodgkin-Huxley model and their initial experiment in figure 1.2, showing an action potential from a giant squid axon [12], which clearly they had to have obtained from a squid. Now although we have highlighted some examples of unethical experiments in Neuroscience, it should be stated that considerable efforts have gone into making it more ethical, and these efforts are taken seriously [47]. So whilst this project incurs no ethical issues, it is worth noting the distant relation with them.

A.3.4 Professional issues

There are little professional issues with this project, as good conduct was followed when created this report and obtaining data, and no data was fabricated or falsified during this either. One could argue that given my supervisor for this project is a co-developer for the MIIND simulator, that there could be a conflict of interest with the writing of this report, however this is simply not the case, and my supervisor shares the aim of finding the issues with MIIND that this report highlights.

Appendix B

External Material

Although the majority of material in this report was of my own creation, it is important to note that I haven't contributed to either MIIND or NEST, and the simulators themselves don't constitute my own work.

MIIND materials can be found at [13] and [31].

NEST materials can be found at [23] and [19].

Finally, the documentation for ARC3 can be found at [40].