

Department of Electronic Engineering

**BEng Project Report**

**2020/21**

**Student Name:** Andrew Newman

**Project Title:** Cellular Automaton and

Spiking Cellular Automaton

**Supervisors:** David Halliday

Stuart Porter

Department of Electronic Engineering

University of York

Heslington

York

YO10 5DD

**Abstract**

Biological neurons communicate with each other by sending spikes of voltage that travel along axons, synapses and dendrites. Incoming spikes influence ion transfer across the cell membrane, and when potentials build up past thresholds, outgoing spikes are generated.

Artificial neural networks inspired by their biological counterparts have shown success in recent years at performing tasks that are extremely difficult with more traditional approaches, such as image and speech recognition. These typically use a single real number to represent spike density, but 3rd generation neurons that implement a spiking mechanism have the potential for greater computational power and are an area of active research in both AI and understanding brain function.

A software tool for simulating spiking neural networks using a selection of models is developed in which spiking neurons are connected via regular, repeating patterns of synapses to form cellular automatons. The state of the automaton can be displayed in a variety of ways, but generally as a changing 2D image with each pixel representing the internal potential of a neuron.

There is a focus on arrangements of neurons that are able to sustain activity without external input, while also self-regulating to prevent the spiking behaviour saturating the entire net, and this is related to the concept of wakefulness in hypothetical strong AIs of the future.

**Contents**

[1 Introduction 6](#_Toc71196275)

[1.1 Aim and Objectives 6](#_Toc71196276)

[1.1.1 Aim 6](#_Toc71196277)

[1.1.2 Objectives 6](#_Toc71196278)

[2 Biological Neurons 8](#_Toc71196279)

[2.1 Neurons 8](#_Toc71196280)

[2.2 Spikes 9](#_Toc71196281)

[2.3 Inhibition 10](#_Toc71196282)

[2.4 Noise 10](#_Toc71196283)

[2.5 Brain Structure 11](#_Toc71196284)

[2.6 Consciousness and Wakefulness 11](#_Toc71196285)

[3 Artificial Neural Nets 11](#_Toc71196286)

[3.1 Feed-Forward Networks 11](#_Toc71196287)

[3.2 Recurrent Networks 12](#_Toc71196288)

[3.3 Wakefulness 13](#_Toc71196289)

[3.4 Generations 13](#_Toc71196290)

[3.5 Spiking Neural Net Models 13](#_Toc71196291)

[3.5.1 Leaky Integrate and Fire 14](#_Toc71196292)

[3.5.2 Izhikevich 15](#_Toc71196293)

[3.5.3 Kumar et al 16](#_Toc71196294)

[3.5.4 True North 16](#_Toc71196295)

[3.5.5 Spinnaker 17](#_Toc71196296)

[3.6 Stochastic Behaviour 17](#_Toc71196297)

[3.7 Available Software 17](#_Toc71196298)

[4 Automatons 17](#_Toc71196299)

[4.1 Life 17](#_Toc71196300)

[4.2 Spiking Automatons 18](#_Toc71196301)

[4.3 Noise 19](#_Toc71196302)

[4.4 Learning 19](#_Toc71196303)

[5 Software Design 20](#_Toc71196304)

[5.1 Layers and Synapses 20](#_Toc71196305)

[5.2 Edges 21](#_Toc71196306)

[5.3 NeuronSim 21](#_Toc71196307)

[5.4 Object Orientation, Data Orientation and Threads 22](#_Toc71196308)

[5.5 Performance 23](#_Toc71196309)

[5.6 Event Based Spiking 24](#_Toc71196310)

[5.7 Neuron (GUI) 24](#_Toc71196311)

[5.7.1 Toolbox Window 25](#_Toc71196312)

[5.7.2 Layers Window 25](#_Toc71196313)

[5.7.3 Synapse Window 26](#_Toc71196314)

[5.7.4 Graphics 26](#_Toc71196315)

[5.7.5 Impulse Response Graphs 27](#_Toc71196316)

[5.8 NeuronTest 27](#_Toc71196317)

[5.9 Genetics 27](#_Toc71196318)

[6 Automaton Design 28](#_Toc71196319)

[6.1 LIF Inhibited Feedback Loop 28](#_Toc71196320)

[6.1.1 Hopper 31](#_Toc71196321)

[6.1.2 Jellyfish 31](#_Toc71196322)

[6.1.3 Third Generation Behaviour 31](#_Toc71196323)

[6.2 Izhikevich Reverse Inhibited Network 32](#_Toc71196324)

[7 Testing 35](#_Toc71196325)

[8 Planning and Time Management 36](#_Toc71196326)

[9 Reflection 36](#_Toc71196327)

[9.1 Interface 36](#_Toc71196328)

[9.2 Implementation 37](#_Toc71196329)

[10 Ethics 38](#_Toc71196330)

[11 Conclusions and Further Work 39](#_Toc71196331)

[12 References 40](#_Toc71196332)

[A The First Appendix 41](#_Toc71196333)

[B The Second Appendix 42](#_Toc71196334)

# Introduction

Biological neurons form the brain and its network of sensory inputs and motor control outputs in animals, including humans [1] [2]. Individual neurons communicate with each other by sending spikes of voltage along axons and receiving them on dendrites. Understanding how this system works is one of the keys to diagnosing and treating neurological disorders. Imitating the system has been successful in achieving limited artificial intelligence in fields such as image recognition and behaviour prediction.

Many models of spiking neuron behaviour, from the simple LIF [3] to the biologically representative Hodgkin Huxley model [4], have been developed, and claim various capabilities.

In this project a tool for binding models of spiking neurons into a cellular automaton is developed. A cellular automaton is an arrangement of cells whose behaviour depends on the state of their neighbouring cells. As the state of each cell changes in response to its neighbours, it induces further changes, giving rise to a system that changes over time. A simple set of localized rules can give rise to complex emergent behaviour of the larger system [5]. For this project, each cell of the automaton is modelled as a spiking neuron, and the interaction with its neighbours as the transport of spikes from one to another.

## Aim and Objectives

### Aim

The aim is to provide an environment in which the interactions between spiking neurons can be investigated and learnt from, with a particular focus on the requirements for sustained activity in a network that has no external stimulus driving it.

### Objectives

Conway’s objectives for the Game of Life [5] serve as a starting point (objectives 1 to 3). The rest of the objectives are particular to this project.

1. “There should be no initial pattern for which there is a simple proof that the population can grow without limit.”
2. “There should be initial patterns that apparently do grow without limit.”
3. “There should be simple initial patterns that grow and change for a considerable period of time before coming to end in three possible ways: fading away completely (from overcrowding or becoming too sparse), settling into a stable configuration that remains unchanged thereafter, or entering an oscillating phase in which they repeat an endless cycle of two or more periods.”
4. Information flow between cells must be in the form of spikes similar in nature to those conveyed by axons in biological nervous systems.
5. The behaviour of the automaton should be distinguishable from behaviours which are possible with a 1st or 2nd generation neural net.
6. There will be a deliverable from the project in the form of a piece of software that can be used to experiment with different neuron models and configurations in an automaton.
7. It will be possible to easily configure the software to run an automaton which meets objectives 1 to 5.

Objective 1 is rather vague (especially since the game of life itself has some simple examples that do grow without limit [5]) but its intention serves as a guideline.

There is no obvious proof that objective 5 is possible but an argument that it has been achieved is provided in section 6.

# Biological Neurons

## Neurons

A neuron is a living cell that has many points of input (called dendrites) and a single output (called an axon) [1]. The neuron is affected by a number of ion channels that allow the concentrations of Sodium (NA), Potassium (K), Calcium (Ca) and Chloride (Cl) ions to change over time, at rates which depend not only on their current concentrations, but also influenced by input voltages applied through the dendrites and the presence of chemicals such as neurotransmitters. A neuron responds to changes in the concentrations (often but not always a potential threshold being crossed) by generating spikes of voltage on their axon. The axons connect to the dendrites of other neurons via synapses. Cells in the brain typically have around 10,000 synaptic connections, up to as many as 100,000 in some cases.

A picture containing text, vector graphics

Description automatically generated

Figure Structure of a Typical Neuron

dhp1080 - Creative Commons License - <https://commons.wikimedia.org/wiki/File:Neuron.svg> – accessed 2021-01

In the 1950s a model of the behaviour of a giant squid neuron was developed through experiment [2] [4]. Known as the Hodgkin Huxley Model it is a 4-dimensional family of exponential differential equations, describing the changes in state of the neuron and its potentials over time. These equations have been used to describe other neurons in other species and a wide variety of configurations exists [2, pp. 46-48] leading to different behaviours, both in how a neuron becomes excited enough to trigger a spike, and the shape and pattern of the spikes generated.

The giant squid neuron is also not representative of all the possibilities. “At least a dozen types of ion channel can be involved in the spike generation of human neocortical neurons.” [1, p. 33].

Examples of distinctly different spiking behaviours are neurons which produce a single spike, or a sequence of repeating spikes, or a rapid burst of spikes followed by a quiet period. While many neurons fire in response to a potential threshold being crossed in response to input spikes it is by no means all [2, pp. 238-239], some respond to incoming spikes only at the correct resonant frequency [2, pp. 232-235] and there are others which emit spikes in response to a *decrease* in their inputs [2, pp. 244-246].

Information flows from one neuron to another in the number of spikes transmitted, the time between the spikes [1] [2], and the time taken for them to travel down the axon [6] [7]. To what extent each of these is significant is still open to debate, but all of them have been shown to increase the theoretical computational power of a neuron.

## Spikes

The response of a neuron to an excitatory input is often modelled as a sigmoid curve, which is useful when doing mathematical analysis, but not representative of what is seen, since it doesn’t inherently show the output rapidly falling back to zero. The spikes generated by neurons tend to show a rapid rise in voltage followed by a slower exponential decay [8] but there is quite a bit of variety possible even within a single neuron, depending on the input it receives.

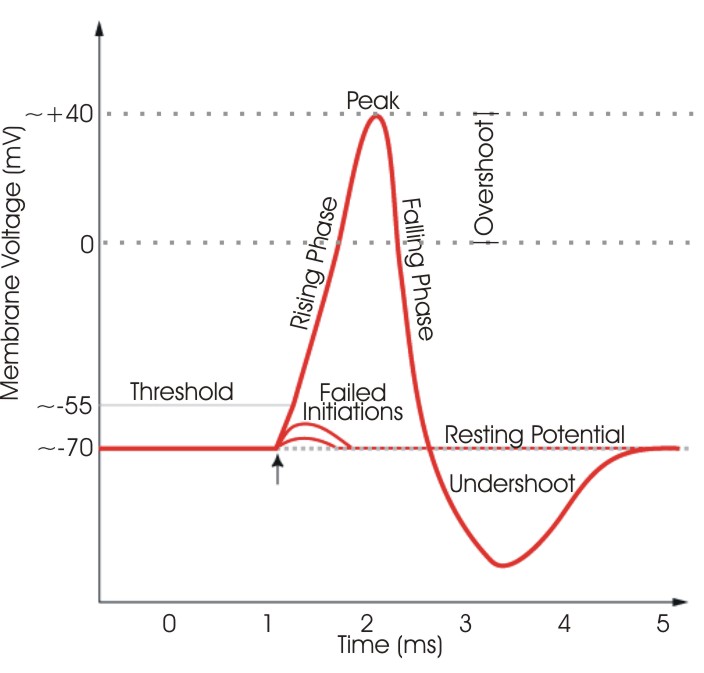


Figure Action Potential [9]

Typical spikes have a peak voltage of the order of a few millivolts and last for up to a few tens of milliseconds. The neuron producing the spike will generally have a refractory period following it during which it will be unable, or at least very reluctant, to produce another spike.

## Inhibition

In addition to neurons which cause excitation of other neurons, there are some which inhibit [1]. Spikes received from inhibitory neurons drive the membrane potential towards its resting state. Inhibitory neurons are less common than excitatory.

Inhibition can also arise from non linear interactions between incoming spikes [1, p. 14], in which the responsiveness of the receiving neuron to other spikes is modulated. Shunting inhibition refers to an input synapse that decreases the effect of excitatory inputs.

## Noise

Nothing physical, including neurons, is ever completely at rest; there is always noise, arising from random variations in chemical distributions and charges. This will only rarely directly result in a neuron firing, because noise is generally quite a bit lower than the threshold voltage (the brain would not function if it was not), but will mean it is rare for a neuron to be at the resting potential. If a group of neurons that respond to a particular input are at rest and receive an excitation some of the neurons in the group are likely to be already partly excited just due to noise, and will respond faster than a non-stochastic analysis would suggest [1, p. 77].

Noise can also act to dislodge a neuron from a locally stable state and cause it to move into a more appropriate one [1, p. 190].

## Brain Structure

The human brain (and many others) is not a homogenous volume of neurons; there are regions associated with specific processing or stimuli, and regions which can be identified by the type of neuron they contain [1, pp. 56-64]. These can then be further subdivided into layers, with the neuron densities and types changing from one layer to the next, and different patterns of connectivity within and between the different layers.

## Consciousness and Wakefulness

An exact definition (or understanding) of consciousness is still elusive, but it can be divided into two concepts: wakefulness and awareness [9]. Awareness is out of scope of this project, but wakefulness and its origin in neurons relates to self-sustaining activity and is relevant here.

The Dynamic Core hypothesis [10] proposes that wakefulness arises from re-entrant behaviour amongst neurons, and relates to a Global Workspace hypothesis that consciousness arises from the interactions involving many different regions of the brain and associated memories.

MRI scans of brain activity in people in vegetative states, and statistical outcomes of brain surgery on different areas of the brain [9], suggest that specific regions are responsible for maintaining wakefulness and driving activity everywhere else. Specifically, trauma and surgery to the posterior cortex is very likely to result in a persistent vegetative state, more so than other regions.

# Artificial Neural Nets

The first artificial neuron was designed in 1943 [11], and since then there has been a great deal of progress, with neural networks now in use in a wide range of fields, from image recognition to self-driving cars.

## Feed-Forward Networks

A common use case of computational neural networks has been data classification (such as image recognition), and this is typically achieved with a feed-forward network in which neurons are arranged in layers with each layer sending data to the next (Figure 3).

Diagram

Description automatically generated

Figure A feed-forward neural net with one hidden layer

The operation of such a system is a single processing event. Inputs are received in the input layer, information flows through the network, and the result is seen in the output layer.

## Recurrent Networks

The human brain clearly does not only act in response to external stimulus. An input triggers processes involving memories of previous inputs that were similar in nature (associative memory [1, p. 147]). This can be mimicked to some degree by adding state to the network. Figure 4 shows the network from Figure 3 expanded to include feedback nodes in the hidden layer.

Diagram

Description automatically generated

Figure A variation on an Elman Recurrent Neural Network

This arrangement would form an IIR LRNN (Infinite Impulse Response Locally Recurrent Neural Network) of a type known as an ENN (Elman Recurrent Network). FIR (Finite Impulse Response) network designs also exist, as do nonlocal recurrent arrangements. IIR RNNs have been shown to be effective at making predictions from time series of discrete inputs [12] and predicting failure of engineered components and systems [13].

## Wakefulness

The neural net described above can be augmented with a new input, one which is always on and feeds into the memory nodes. It would not be appropriate to refer to such a network as conscious (even defining what it means to be conscious in this context is troublesome [14]), but it will have an output even when deprived of all other inputs. With more locally recurrent hidden layers it could even have an output which changed over time, which brings it one very small step in the direction of wakefulness. An artificial neural network which is considered conscious is referred to as a strong AI, and to the authors knowledge no strong AIs exist at this time.

## Generations

The first artificial neurons had a binary state; they were either on or off. A network of such neurons is often referred to as a 1st generation neural net [15]. 2nd generation networks allow for real valued outputs and inputs, which can be thought of as representing the rate of fire of spikes from a biological neuron. Neural networks which use spikes, and thus attempt to mimic the behaviour of biological neurons more closely, are considered 3rd generation. A key difference between 2nd and 3rd generation networks is that timing of spikes becomes important, whether in the form of propagation delays [6] [7] (comparable to axon transmission speeds), delays between spikes [1] [2] (a burst of spikes followed by silence may trigger a response that a regular series of spikes would not, or vice versa) or coincidence (spikes received simultaneously from multiple sources may trigger a response where staggered arrival times would not). This increases the potential for information flow between neurons, and 3rd generation networks have been shown to have more processing power per neuron than 2nd generation, but at the price of greater complexity [1] [2] [6] [7].

## Spiking Neural Net Models

The Hodgkin Huxley model is complicated and correspondingly slow to simulate (Izhikevich approximates it to 1200 FLOPS per neuron per iteration [16]). Faster approaches to it have been proposed [17] and it is interesting from a biological point of view, but for a large and/or fast neural network in software we need to look for alternatives.

There have been many SNN models developed for this purpose, with different goals and advantages [15] [16] [18].

### Leaky Integrate and Fire

Integration is a repeating theme in many spiking network models and refers to summing up incoming spikes over time. This is comparable to the biological process of spikes encouraging the movement of ions across the cell membrane, with the integral mapping to the membrane potential. A leaky integrator is one in which the potential decays over time. A LIF (Leaky Integrate and Fire) neuron [1] [3] is one which integrates incoming spikes into its potential, leaks over time, and fires a spike of its own if its potential exceeds a predetermined threshold.

A LIF neuron with a very slow leak acts mostly as an integrator, which makes it indifferent to the timing of incoming spikes, and as such is difficult to distinguish from a 2nd generation neuron.

A LIF neuron with a very fast leak acts as a coincidence detector. Incoming spikes separated by quiet periods will leak away and the potential only reach the threshold if many spikes are received at the same time.

An equation for a simple LIF neuron, excluding tests for the threshold being exceeded, is shown in Equation 1.

Equation LIF Neuron

When a spike is fired the potential is reset. Some models may also apply a refractory period [1], or attempt to mimic one by resetting to a potential below zero.

Variations of the LIF theme have been proposed, including quadratic formulas [16]. A quadratic LIF can display behaviours absent from linear ones due to the presence of an internal threshold where the differential equation becomes positive and accelerates the generation of a spiking potential. The Mihales-Neibur [19] neurons are essentially multiple LIF systems combined into one, and can give rise to bursting, rebound spiking, bistability and a number of other behaviours seen in biology but absent from the basic LIF behaviour described above.

### Izhikevich

In 2003 a neuron model designed to give the spiking patterns seen in biology at a low computational cost was developed [20] and uses two internal variables instead of a single potential. This can be thought of as roughly corresponding to two different ion concentrations in the cell membrane (specifically Sodium and Potassium).

An Izhikevich neuron is able to exhibit 20 different spiking behaviours using two simultaneous differential equations, one quadratic and one linear, and can be an integrator, coincidence detector or resonator, and does so using only 13 FLOPS per neuron per iteration.

The dynamic state of an Izhikevich neuron is given by Equation 2.

Equation Izhikevich Differential Equations

And this is coupled with a firing threshold (V > 30) and a reset behaviour when a spike is fired (Equation 3).

Equation Izhikevich Reset Behaviour

V, u, a, b, c, & d are all stated to be dimensionless, although the spiking threshold of 30 is given in millivolts. The mismatch does not cause any problems with the implementation.

The Izhikevich model can act as an inhibitory neuron and even mimic the output of the biological ones in the brain, but unlike the spike and reset for high potentials, there does not appear to be anything limiting the negative potential a neuron can have. This will not be a problem with shunting inhibitions, but otherwise seems to part of a common theme through all the literature reviewed here - the existence of inhibitors, whether neurons or otherwise, is acknowledged but mostly ignored. Presumably this is because most work on computational neural nets has been geared towards forward connected nets that respond to an input event or data stream, rather than a continuously updating system such as an automaton that needs to self-regulate.

### Kumar et al

Kumar et al [18, pp. 24-30] present an alternative model to Izhikevich, again with only two equations. They refer to Izhikevich as inspiration and provide a similar set of possible output forms. It is not clear why this model might be preferred over Izhikevich, but it demonstrates biologically similar behaviours in response to short pulses of DC current, where Izhikevich seems inclined to use a unit step function as input. This is encouraging in a system that will be using spikes as inputs. The dynamic state is given in Equation 4, and the reset behaviour is identical to that of Izhikevich.

Equation Kumar Differential Equations

Unfortunately, the paper in which this method is described does not provide example values of the parameters used to obtain different output types, which makes it difficult and time consuming to use.

### True North

IBM have developed a True North architecture [3] with the goal of creating a neuron model that can be effectively implemented in custom hardware "sufficient to support useful and interesting cognitive algorithms, while the cost should be no more than necessary in terms of power, area, and speed".

They state "we were able to qualitatively replicate the 20 behaviors of the Izhikevich dynamical neuron model using a small number of elementary neurons" which is interesting, since it implies a simpler model that has the Izhikevich model emergent from it, and emergent behaviour is relevant to this project.

There are some oddities of True North, however, with regards to its response to input spikes. Many of the repeating or bursting behaviours it can exhibit are triggered by a single spike, of any weight at all. A spike with a weight of 0.001 can cause a series of output bursts. This means that it is not acting as an integrator or coincidence detector in these configurations, and problems arising from this are referenced in section 4.

The equations for True North are available in [3] in figure 4. They have not been included here. The implementation used has been modified to remove the optional stochastic features for the reasons given in section 3.6. Essentially, however, True North is a one dimensional leaky integrate and fire (LIF) model with a few quirks, most significantly that it can be set to leak away from the reset potential instead of towards it, which allows it to demonstrate bursting behaviour and bistability and a number of other functions not normally seen in one dimensional systems. Conditional elements of the algorithm are described as multiplication by integers that are either 0 or 1 and some of these have been implemented as if statements in the code for this project. The result is the same. The original is doubtless far easier to implement in custom hardware but is less easy to read in software. Given that the branch statements are completely predictable they are probably the more efficient approach also, but this model was not used a great deal and has not been profiled.

### Spinnaker

The SpiNNaker system [21] approaches the performance problems of large spiking neural nets by simulating them on 1 million arm cores, and by doing so is able to run many spiking models, including Izhikevich, for 1 billion neurons at biological real time. For the Izhikevich model real time means 1 iteration per millisecond [20]. SpiNNaker is not a model in its own right, but a hardware system on which they can be run. Its existence and scale, however, is a strong indicator of the important role efficiency and speed will play in any implementations of SNNs.

## Stochastic Behaviour

Noise is an important part of brain function (see section 2.4), and the True North has several optional features involving random chance. Of particular interest is stochastic integration. If a spike with a weight of 0.2 is received by an integrator the current system will add 0.2 to the input potential, but an alternative is to have a 20% chance of adding 1.0. The average result is the same, but the latter system has the possibility of causing occasional spikes from intermittent inputs that would be never fired under the deterministic approach. Since spike integration is part of the framework of our automaton, rather than the neuron model, any implementation of this should be done in a general way that makes it available to all models, and it would not be difficult to do so.

The stochastic firing threshold, leak, and reset potential would likely need to be model specific but could be translated to work with LIF and Izhikevich without too much imagination.

Section 4.3 goes into more detail about why this has not been done for this project.

## Available Software

Software already exists for simulating neural nets, of all three generations, some commercial and some open source and freely available. IBM make software simulation of the True North architecture an option [3]. Nest provides a python module which can simulate many neuron types (including Izhikevich) [22]. OpenSourceBrain provides an enormous suite of neural net related simulation software [23]. Annarchy makes an impressive attempt at overcoming the difficulties of efficiently simulating spiking neural networks by exposing a scripting interface that automatically generates C++ networks on the users behalf [24]. This is far from an exhaustive list.

What all these solutions have in common, however, is that they cater to the traditional use of neural nets as a device learns (or at least has the capacity to learn) and then applies what it has been taught. The goals of this project are sufficiently unique that a custom software solution has been chosen instead (see section 5).

# Automatons

A cellular automaton is a regular arrangement of discrete cells in which the state of each cell depends on the state of its neighbours, such that the cells change state over time. Large scale complex behaviour can emerge from the simple localized rules.

## Life

An example of a popular cellular automaton is Conway’s Game of Life [5] in which each cell is either ‘alive’ or ‘dead’ and changes between those states depending on the number of live immediate neighbours it has. Life is usually represented graphically as a two-dimensional grid of squares, one per cell.

* A cell with less than 2 living neighbours becomes dead.
* A cell with 4 or more living neighbours becomes dead.
* A cell with 3 living neighbours becomes alive.
* A cell with 2 living neighbours remains in its current state, whether alive or dead.

This can be implemented in terms of neurons, by connecting each cell to all 8 of its neighbours with a weight of 1.0, and to itself with a weight of 0.5, and then setting an activation threshold of 2.25 < V < 3.75.

This can be extended to work in terms of LIF spiking neurons, provided we can weight the middle spike to 0.5, and provided that the neuron has a refractory period of 0 and leaks 100% of its potential in every iteration. This is certainly not a normal configuration for SNNs, but it has been implemented in the software for curiosity and testing purposes.

## Spiking Automatons

A cellular automaton generally uses the term “cell” to refer to its constituent components. A neural network usually uses the terms “neuron” or “perceptron”. In biology the word neuron refers to a special type of cell. In this document the words “neuron” and “cell” have the same meaning when referring to the components of an SNN Automaton.

The goal of this project is to make a cellular automaton in a similar style to Conway’s Game of Life but using a biologically inspired spiking neuron model for the cells. This provides an opportunity to experiment with and investigate the behaviour of the chosen SNN models, but also extends an unusual requirement to them. A feed forward neural net is generally given an input and provides an output. A recurrent network may receive a series of inputs over time and have memory like nodes modifying its behaviour based on its past. An automaton, however, does not necessarily have an obvious input at all, nor output, but is instead expected to act on its own state and to continue changing over time, possibly indefinitely. It is for this reason that a short introduction to wakefulness was presented in section 2.2.6. It would be very unreasonable to suggest that a simple automaton is awake, but achieving continuous activity without an external input is a challenge in its own right, and gives rise to a number of questions that need to be answered if we one day want to construct an artificial intelligence which can be considered conscious.

The automaton needs to avoid quickly collapsing into a quiescent state. For this to be true the number of neurons which fire as a result of N other neurons firing must average at least N. However, if the average is greater than N then we can expect the automaton to quickly saturate instead, with everything firing as fast as it is able, which is not our intent. This can be expressed for a timestep from T to T+1 as in Equation 5, and our aim is to have the average value of R be close to 1.

Equation R Numbers

Balancing R exactly at 1 is very unlikely, given the chaotic nature we expect from our final system. Life avoids the problem by having a typical R number greater than 1 with sparse populations but killing any cell that has 4 or more neighbours, bringing R down when the cell density goes up. It does not seem appropriate to directly prevent a spiking neuron from firing when it is overstimulated, however.

One option is to use a re-entrant network (akin to the Dynamic Core hypothesis [10] mentioned in section 2.2.6) using inhibitory neurons to maintain balance. These can be simple negatively weighted inputs (subtractive), “shunting” in which a connection decreases the effectiveness of excitatory inputs (divisive) [1, p. 22], or fatigue [1, p. 35]. There are many arguments for combining multiple kinds of neuron [3] and the design used here allows layers of inhibitory neurons to be included. Fatigue has not been implemented, but long-lasting shunting inhibitions can have a similar qualitative effect.

On first glance a second option is that one region of the automaton sustains the activity of the rest (mimicking the posterior cortex [9] mentioned in section 2.2.6). To achieve this a layer of overactive neurons (R > 1) acts as a driver and sustains activity in less active layers (R < 1). Unfortunately, the driving layer generally saturates and consequently induces predictable behaviours with no emergent complexity. The driving layer must self-regulate as a re-entrant network and that brings it back to the previous solution.

Since biological brains appear to contain both of the above mechanism it is also reasonable to construct automatons that do the same.

## Noise

Noise, and its significance in biological neurons, was mentioned in section 2.4. The use of stochastic methods in SNN models was covered in the discussion of the True North model in section 3.5.4.

Unfortunately, any random influence will break repeatability in the automaton. The same pattern of inputs to an automaton would usually result in the same behaviour, regardless of where or when it occurs, and it was decided to respect that rather than take the biologically more realistic stochastic approach. Similarly, a repeating pattern such as the gliders and trains in Life, would be almost certain to break down under a stochastic system, and our automatons would be unable to meet objective 3.

The software used for this project does not implement stochastic behaviour. It does allow random noise to be injected into the system from the GUI should that be desired, however, and this has proved to be the most convenient way to bootstrap a new automaton.

## Learning

A similar argument to that for noise can be made about learning. Individual neurons should not begin changing their associated synapse weights if we want our automaton to have consistent and repeatable behaviour. There is also the problem that the automaton has no output, other than its own state, which makes the usual Hebbian teaching mechanism difficult to imagine. There is no clear “correct” result to propagate through the network and learn from.

The software used for this project does not implement learning. However, there is a small foray into genetic algorithms for tuning purposes which had limited success (section 5.7.4).

# Software Design

This project is entirely software based. Where possible platform independence has been aimed for, but the Microsoft Windows OS on an x64 architecture is the primary target, C++ the language chosen, and Visual Studio the project format and IDE used.

The MVC (Model View Controller [25]) pattern has been used, with a particular focus on keeping the view separate from the rest of the program. This has allowed multiple views to be created, as an automated test framework (NeuronTest), a GUI for editing and executing networks (Neuron), and a simple genetic algorithm (Genetics). All of these link to and use a simulation library (NeuronSim) which does most of the work. NeuronSim is written in pure C++ and has no external dependencies beyond the runtime libraries. NeuronGui uses the Qt library for a cross platform GUI interface and OpenGL for graphics.

All of the source code is available from <https://github.com/adn510/Neuron> but will require Qt and Visual C++ (both available for free) to build.

## Layers and Synapses

All automatons supported by this project are arranged in a regular two-dimensional grid. There can be many layers of these grids, each conceptually stacked on top of the others. All the layers must be using the same model (i.e. Izhikevich) and all the neurons in a given layer have the same configuration (for example, if they are Izhikevich, one layer might be phasic spiking and another layer resonant).

Neurons are connected via synapses to other neurons, and this is configured as a set of synapse matrices which define 2D arrays of synapse weights and transmission delays centred on the source neuron. Every neuron within a layer uses the same synapse matrix, and the connections are relative coordinates of the destination neuron. A synapse matrix can connect neurons to other layers, or to neighbouring neurons in the same layer. A 3x3 matrix mapping from one layer to another is shown in Figure 5.

A picture containing chart

Description automatically generated

Figure Synapse Matrix

This is equivalent (although implemented in reverse) to the behaviour of a convolutional neural network with full weight sharing [27], but applied at every possible set of coordinates instead of being tiled more loosely.

## Edges

The software is designed to handle arbitrarily large synapse matrices, in anticipation of needing many synapses. To match typical neurons in the human brain with 10,000 connections a synapse matrix would need to be 100x100 in size. This sort of size is unrealistic in practice because of performance constraints, but 9x9 matrices were used for a lot of experiments and that is sufficient to bring a large proportion of a layer within reach of the edge of the grid. Rather than have moving patterns dissipate on contact with a dead zone the grid wraps around, from the top to the bottom and from the right to the left.

This can be thought of as long range synapses connecting the edges (akin to the long axons that travel through large amounts of white matter in the human brain [1]), or as the grid being the surface of a torus in three-dimensional space, or as the grid extending to infinity in all directions but in a repeating pattern. The infinite repeating pattern is how the program displays the grid in its default configuration, and it can be scrolled indefinitely in any direction.

## NeuronSim

A simplified UML diagram of the NeuronSim library is shown in Figure 6. The Automaton class is the primary interface, but the layers and synapses it contains can also be created and used in isolation, which is useful when testing and in one or two other situations. The expected use case is that an Automaton is created with a desired width and height and is then used to create layers and synapses. These are configured independently. Running the automaton simply requires the tick() function of the Automaton class to be called repeatedly, once per iteration.

Diagram

Description automatically generated

Figure NeuronSim Class Diagram

The Layer class is primarily an abstract interface allowing different models to be implemented and used polymorphically. The LayerFactory abstracts the creation of the derived types to remove any need for dependency on them from the Automaton and client code.

The Net specialisation of Layer handles the functionality common to all models, but templated with the type of data used (each model has different requirements for the data stored in its neurons, and they are not all the same size). This could also have been achieved with an abstract base type for the neurons and some more polymorphism, but a template allows for some useful optimizations and, perhaps as a curiosity of C++, is neater and easier to follow.

The specializations of Net represent specific neuron models and generally only need to override a small number of simple functions, the most important of which is the tick() function which is called once per iteration and runs the model specific behaviour.

The ConfigItem, ConfigSet and ConfigPresets classes wrap the configuration data for different models in a manner that allows a client (in this case the layer configuration window of the GUI) to adapt as necessary to different settings without needing knowledge of individual specializations.

The SpikeTrain is a circular buffer storing spike data that is in transit from a source layer to a target layer. The shape, duration and delay of spikes is all encompassed in this class, and it has a close relationship with the multithreading methodology described in section 5.4.

## Object Orientation, Data Orientation and Threads

Object-oriented design has been used for most of the software, in the manner common to C++ for most of the code base and augmented with signals and slots for the GUI in keeping with the Qt architecture. However, there is a requirement for speed when dealing with neural net simulations, and the regular grid layout of the automaton provides an opportunity to take a more data-oriented approach to the inner loop.

Neurons for a given layer are stored in a contiguous block of memory, and each neuron has the same configuration as all the rest, and so that data can be moved out of the neuron and into a specialization of the layer. By doing this we save a lot of RAM, but we also hope to reduce the number of cache misses that occur while iterating through the grid, which is a difficult to measure but often significant performance boost. The neurons themselves are simple structs rather than classes and are treated as POD throughout.

Layers interact with each other only via spikes. By separating this interaction into two phases within the tick() function it is possible to process each layer in isolation, the first phase pushing spikes onto layers and the second phase processing layers and outputting spikes. This means a separate thread can be used for each layer (see Figure 7) without any overhead in mutex locks, which is a large step change in performance for multi-layer networks. The program is still conceptually single threaded and only divides temporarily while iterating through layers, which prevents a lot of the complexity associated with threading from permeating the rest of the code base.

Diagram

Description automatically generated

Figure Multithreading Pathways

To facilitate this approach the spike data itself does not belong to either the source or destination layers and is instead written into a spike train that sits between the two. This data is available to only one of its associated layers during each multi-threaded sequence to avoid conflict.

This comes at a price. If spike data were associated with the layer it was being written to then all synapse matrices connecting to that layer would be able to share the same memory, which would save us a lot of RAM and decrease the amount of data being integrated in each iteration step, but would prevent different threads writing spikes simultaneously.

## Performance

An effective automaton needs to be large enough for interesting behaviours to emerge. This means it also needs to be fast, so as not to take an unacceptable amount of time to execute. For the purposes of the configurations used in this project a grid of 256x256 neurons was found to be adequate, but 512x512 better when memory allowed.

Izhikevich claims a 1ms simulation time for 10,000 neurons with 1,000,000 synapses on a 1GHz PC [20]. That is only a 100x100 grid of neurons, which is rather small for an automaton and would limit the size of feature we could hope to detect. It is also only 100 synapses per neuron, rather than the 10,000 typical of the brain [1]. There is a balance to be made between designing for speed and premature optimization, but it was obvious even without profiling that efficiency would be an important factor here [15] [16].

Note that an automaton of this nature is extremely parallelizable and any attempt to fully optimize it should almost certainly be targeting a GPGPU or custom hardware. This project needed to be flexible and therefore targeted the CPU, and consequently only aimed to be fast enough to be usable, rather than as fast as possible. An update speed equal to the 60Hz update rate of a typical monitor would be ideal, but 30Hz is more common with large networks and is acceptable.

When tested on a i7 4 core 2.4GHz PC with 8 Gb of RAM using a LIF automaton consisting of 8 layers, 524,288 neurons and 17,694,720 synapses the simulation managed 1000 iterations in 9052 ms. This works out at approximately 0.8 synapses per clock cycle. With graphics enabled a 31ms frame time is achieved, which corresponds to the 30Hz target.

## Event Based Spiking

An early design stored spikes as individual events. As it now stands, all layers have incoming spikes on all neurons, but many of them are zero much of the time. For a finished product, if we were to try to use a specific automaton to achieve some task, an event based system would almost certainly be faster and use less RAM, and it is the mechanism used by TrueNorth [3] for the same reasons. However, this software is designed for investigating different configurations and storing distinct events leaves no meaningful upper bound to memory and CPU usage. Out of memory situations were far too common during experiments and events were discarded in favour of the flat memory model.

The cost of this is not insignificant. Performance tests show approximately 33% of the CPU time being spent integrating spike data, most of which is zeroes.

## Neuron (GUI)

The GUI for the main application built on the NeuronSim library was implemented using C++ and Qt and is intended to make every data item that might need to be regularly edited available in one place. There are very few rarely used settings that could be conveniently hidden behind a menu, and as a result it can look a little cluttered on first glance (Figure 8). In this case a single layer of True North neurons in their bistable configuration is shown.

A picture containing text

Description automatically generated

Figure The main GUI in single window form, showing a single layer / single synapse matrix True North configuration

There are 4 areas to the application window, and three of these are docking windows that can be moved to another monitor to make more room for the automaton if desired.

### Toolbox Window

The toolbox contains controls that affect the entire automaton and the view of it, and by default is docked on the left of the main window. The model of neuron to use and the size of the layers is decided here. The simulation can be zoomed in and out, scrolled around, stopped, started and stepped, and limited to a fixed frame rate if needed.

There is also an editing section which allows spikes of a given weight and pattern to be injected into a chosen layer. New spike patterns can be defined in the form of png images saved into a specific folder and will be automatically added to the list, but most often it has been used to inject random noise to bootstrap a new automaton.

### Layers Window

The layers window is a docking container that contains configuration settings for each layer. All layers have a spike shape and duration, and an associated colour to allow them to be distinguished in the graphics view, but also a set of parameters that depends on the neuron model being used. In this case the model is True North which requires more fields than most. These are added dynamically at run time to allow them to adapt. See the ConfigSet and ConfigItem classes for implementation details.

There is also a preset drop down available at the top of this window. Presets are saved in a plain text format and can be created manually with an external text editor and are a significant time saver when configuring many layers.

### Synapse Window

Synapse matrices are loaded from disk in the form of greyscale png files. A black pixel represents a weight of 0 and a white pixel a weight of 1. This limits the weights to an 8 bit resolution, but the most common use case is to define a pattern of synapses in which all weights are 1 or 0. In this case a ring shape of weights has been defined and each neuron will be sending spikes to more distant neighbours but not its immediate ones.

Delays can be specified as a function of the length of each synapse, but in this case the delays have been set to zero.

### Graphics

The main area of the main window is an OpenGL rendering buffer. The rendering style defaults to an infinite (repeating) plane which can be zoomed and scrolled but can also display the data in other formats such as a 3D stack (interpreting the layers as the 3rd dimension). The direct use of OpenGL for this is a compromise between the limitations of Qt’s own built in graphics and a desire to avoid unnecessary dependencies on 3rd party libraries.

Layers can be colour coded and are drawn additively. There are limits to how much information a single rgb pixel on a screen can convey and this approach gave the best balance of flexibility and ease of use. An earlier system in which neuron models defined their own colours (for example, red and green channels for the U and V variables in an Izhikevich neuron) was very effective if there was only a single layer, but it became apparent that viable automatons generally need more than one and this method was discarded.

A lot of the work involved in the graphics happens in parallel. The graphics can be set to update on every iteration (in which case the simulation waits for the screen update) or to update when ready (multiple iterations will occur between updates if the simulation is fast enough). The screen can also be set to stop updating at all, which can simplify some debugging or optimization tasks.

### Impulse Response Graphs

Within the layer configuration panels there is a button labelled ‘graph’. This displays a graph of the spikes generated by the current configuration in response to an input signal of configurable weight and duration. This is achieved by creating a 1x1 layer and running it as though in the simulation, except with any spikes fired redirected into a data array. Example graphs are shown in Figure 12 and Figure 13 on page 33.

This graphing capability was useful as a manual test of neuron behaviour during development, but also convenient occasionally while designing automatons.

## Genetics

The automatons this program creates cannot learn. Not only do they fully share synapse weights with their neighbours, they also have no objectives. There is no output which can be said to be correct or incorrect or provide an error term to propagate through the network. However, a crude measure of the automaton as a whole can be defined. An average number of spikes fired per iteration of around 10% of the maximum would suggest that it has neither died out nor saturated, for example, or an R number close to 1 means one of the objectives has been met.

A set of scripts were created using methods such as this to create automatons by making random numbers of random changes to the configuration and then keeping changes that led to a better score. This is a very slow way to develop anything, but it is automated and can be left running in the background for long periods. One such script remains in the project called Genetics in the neuron repository, but mostly only as an example.

Genetics were not particularly effective. In addition to taking a long time to run because of the low probability of a random change being good, they also tend to interpret their goal rather literally. A completely dead automaton with no spikes at all, for example, has an R number of exactly 1. An automaton with 1 extremely active layer and 5 dead layers has about 15% activity and looks quite promising to an unsophisticated scoring function. Ultimately the decision was made to abandon them in favour of manually configuring everything.

# Automaton Design

Many automatons were tested during this project. Most of them either died out very quickly or rapidly saturated the entire grid, which is indicative of the difficulty in finding a balance.

Four neuron models have been implemented (not counting a spiking version of Conway’s Game of Life). These are Linear LIF (a basic linear leaky integrate and fire model with exponentially decaying potentials), Izhikevich, Kumar and TrueNorth.

The Kumar model [18] proved to be unusable without more information regarding configuration parameters. This model also seems inclined to continue to spike indefinitely after the inputs have been removed (this is clear, in retrospect, from the graphs presented in the original paper). A neuron which continues to spike even when deprived of inputs renders shunting inhibition meaningless, and they also seem to be very resistant to direct inhibition.

The TrueNorth model [3] is far too responsive (triggering spikes, and even repeating bursts of spikes, in response to any input at all, no matter how small, leads to huge R values that are pretty much impossible to balance out). It also means it suffers from the same inhibition resistance as the Kumar model, and shunting inhibitions are, again, almost useless.

## LIF Inhibited Feedback Loop

LIF is limited in what it can do - there are many behaviours it can’t exhibit [16] – but it can be easily understood and inhibitory neurons are effective at maintaining R values close to 1. This makes LIF by far the easiest model as a base for viable automatons.

A successful automaton using the LIF model is available as a saved state in the repository, under the name of “LIF\_white\_matter”. This consists of a stack of 6 excitatory layers, 1 direct inhibition layer and 1 shunting inhibition layer. The excitatory layers (shown in green in Figure 9) each connect via a 3x3 synapse matrix to the next layer in the stack. The last one connects back to the first, which can be considered analogous to long distance axons moving from deep in the brain back up to near the surface (alternatively it can be seen as a torus shape). The two inhibitory layers (the direct inhibition is in blue and the shunting inhibition is in red) are excited by every layer in the stack, and act to inhibit every layer in the stack, again with 3x3 synapse matrices.

The green layers have a spiking threshold of 3 and emit square spikes with a weight of 1, duration of 2, and no delay. They are coincidence detectors (they leak 50% of their potential in every iteration) and are limited to (and reset to) a minimum potential of zero.

The blue layer is configured almost the same, except that the spikes it emits have a weight of -1 and its threshold is set to 30.5. Although it is excited by 6 times as many layers as the green ones, it also has ~10 times the threshold, which causes it to act as an inhibitor only in times of excessive spiking.

The red (shunting) layer is an integrator, but not a perfect one – it retains 90% of its potential between iterations. Spikes emitted have a weight of 1 and a duration of 5 and the threshold is 35.5. Although this threshold is high, relative to the other layers, the integrator behaviour means it triggers far more easily. Its effect is also far less pronounced than the direct inhibition.

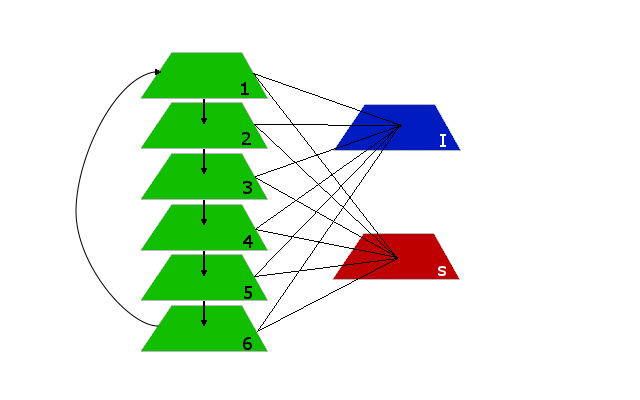


Figure LIF White Matter Synapse Flow

Figure 10 shows the internal state of the neurons around 100 iterations after being initiated with a series of random spikes. The colour coding is the same as for Figure 9. Red areas are slowly fading as a result of the integrator neurons only leaking very slowly and show where recent activity has died out. Green neurons can be seen in excited states and blue regions are often seen inside clusters of green, where the activity has grown enough to trigger the inhibition layer.

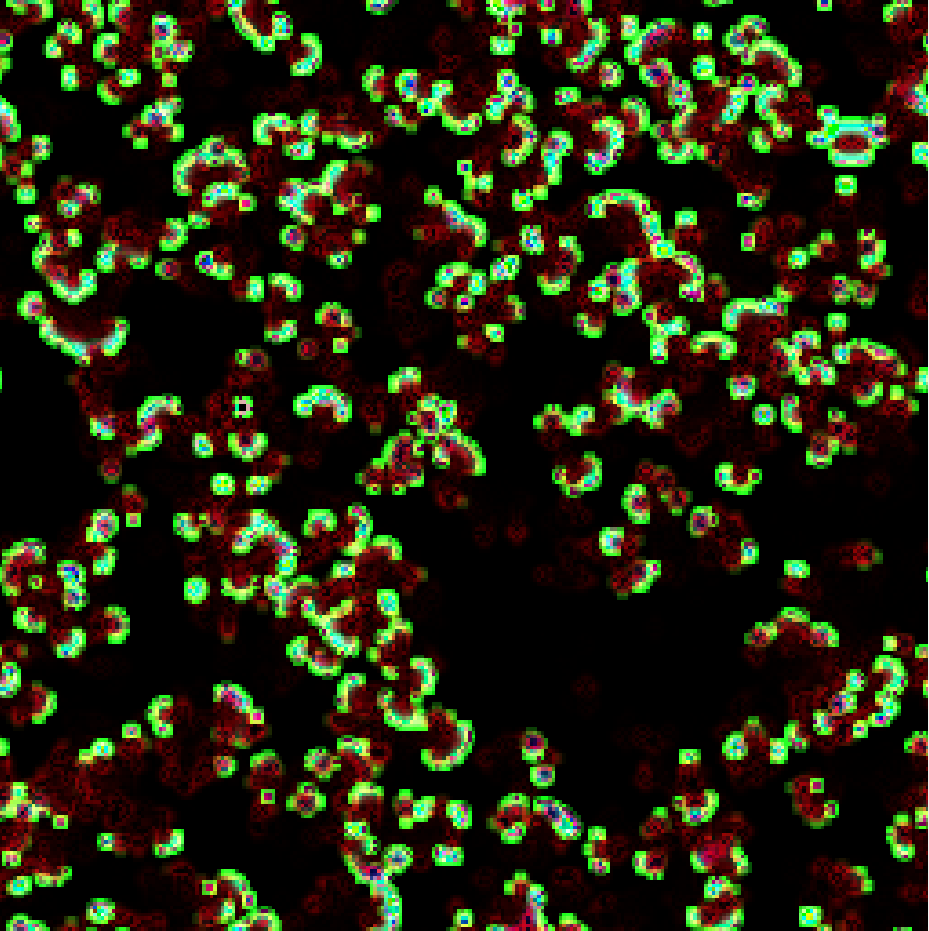


Figure LIF White Matter Automaton. A typical state within a hundred iterations of being triggered by random noise.

The R number of this automaton is a bit hard to state precisely. During times of high activity it is a little less than 1, resulting in it dying down slowly, but it produces two common repeating patterns (and possibly more that haven’t been identified). These travel slowly through the grid and never expire, so over a long period of time the R number is exactly 1.

### Hopper

One of the repeating patterns is small enough to fit in a 10x10 square, and hops from side to side while travelling diagonally. It is reminiscent of the gliders from Life, except with very exaggerated side to side movement.

A video which includes a hopper pattern (it is near the middle and travelling up and left) is included alongside this report (hopper.mp4).

### Jellyfish

A pattern which frequently appears, and which we will refer to as the jellyfish for the rest of this document, goes through a sequence of over 100 states before repeating itself, and travels slowly along one axis of the grid as it does so. It exists in some form in every layer at once and varies in size from around 40 neurons across to a single neuron firing by itself, although clearly a single firing neuron only sustains activity because surrounding neurons have a partially excited state already.

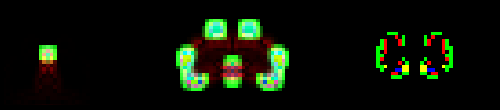


Figure Jellyfish Patterns

Figure 11 shows three stages of the jellyfish. The first is a travelling rectangle that is very nearly stable and covers quite some distance before expanding and unfolding into a much larger pattern, one step of which is shown in the middle. The last image is at a similar point in time to the second, except instead of showing the internal state of each neuron it shows only those which are in the process of firing spikes. Colours are additive, so yellow squares are showing at least one excitatory neuron and the shunting neuron firing together.

A video of the jellyfish in action is included alongside this report (jellyfish.mp4).

### Third Generation Behaviour

Objective 5 for this project requires behaviour which is not present in 2nd generation neurons. All of the spikes used in this automaton have integer weights, and although floating point accuracy means the spiking thresholds might not be exactly at an integer value the sensitivity of the thresholds should have a resolution of 1, if the weight of the spikes is all that matters.

The threshold for the shunting inhibition layer is 35.5. If this is decreased to 35.0 the jellyfish shape reduces to a travelling rectangle. It survives, but no longer goes through the expand and collapse phase.

If the threshold is increased to 36.0 the jellyfish shape expands, continues to expand, and then dies out completely. Meanwhile the rest of the automaton goes from R < 1 to R > 1 and begins to slowly fill the screen. The hopper still exists and becomes a lot more common.

Changing the direct inhibition layer threshold by 0.5 in either direction also has visible effects.

For the behaviour of the system to change twice in a threshold range smaller than the spike weight there must be an additional effect in play, and there are two likely sources. The first is that non integer inputs are being received due to shunting inhibition, but this is the shunting layer that is being manipulated and it has no shunting inputs of its own. The alternative explanation is that the decay of the potential between spikes is giving rise to an increased resolution. This is a temporal effect and therefore it seems likely that there is 3rd generation behaviour present and that objective 5 has been met.

## Izhikevich Reverse Inhibited Network

Izhikevich can demonstrate a number of interesting and usable behaviours (an example is shown in Figure 12) but also presents some inconvenient properties. An example is the resonant neuron configuration. All resonant neurons are also coincidence detectors, but a resonant Izhikevich neuron also triggers when treated as an integrator and shows a rebound spike shortly after being inhibited (see Figure 13). It is good that the model can effect 20 different types of biological spiking patterns, but less helpful that many of them are demonstrated at the same time from the same neuron. Rebound spiking makes it particularly difficult to inhibit a network to keep the R number down. Shunting inhibitions work, but direct inhibition causes even more spikes and tends to fail spectacularly.

Graphical user interface, chart, bar chart, histogram

Description automatically generated

Figure Izhikevich Chattering Neuron. The bottom graph shows a constant input for 150 iterations, and the top graph shows chattering behaviour as an output and that it stops when the input is removed.

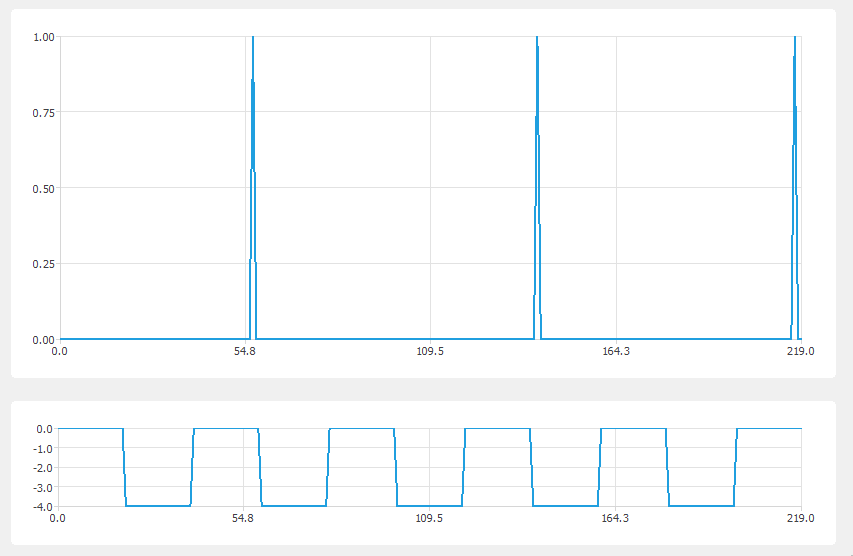


Figure Izhikevich Resonant Neuron Rebound Spiking. The bottom graph shows the input spikes and the top graph the output response of the neuron.

The internal state of one of these neurons often reaches a point at which a spike is inevitable, but the spike itself doesn’t happen until several iterations into the future. During this time interval attempts to inhibit the neuron are ineffective, but also the neuron itself is not triggering inhibiting neurons itself yet and won’t until it finally fires. This means the spikes themselves are trailing behind the activity causing the spikes. Inhibiting neurons are already a delayed reaction, since a spike must be received and another sent before they affect the excitatory layer. This additional delay from the slow spiking behaviour means that by the time an inhibitory spike is received it is generally too late to do anything useful.

As a result of their own inherent refractory periods, however, Izhikevich automatons tend to resist becoming saturated. Almost any configuration (that doesn’t die out) will end up producing a never ending series of waves, which technically gives it an R number of 1 with almost no effort at all, but instead of complex behaviour emerging from simple localized interactions between cells (the goal of the automaton) there is simple behaviour emerging from quite complex interactions.

A picture containing text

Description automatically generated

Figure Izhikevich waves

Figure 14 shows the spikes being fired during a single iteration of a fairly typical Izhikevich wave automaton. There are two sets of waves present, which have been indicated in red and green. Small waves (indicated in red) travel extremely quickly along the spiral, while the spiral itself forms a much slower wave (indicated in green) travelling outwards. The center of the spiral is sometimes stable, and sometimes moves around, depending on how the layers have been configured. A video of this particular automaton is included with this report as Izhikevich\_waves.mp4. The video shows a small number of single iteration steps before switching to full speed. This automaton does not meet the objectives – on an infinite plane it very clearly would grow without limit and fails objective 1.

A better Izhikevich based automaton is included with the project under the title of “Izhikevich\_reverse\_inhibition.neuron”. This consists of three layers, coloured red, green, and blue (Figure 14). Instead of specialized inhibitory neurons, the layers are arranged in a loop, with neurons in each layer exciting the next layer in the loop and inhibiting the previous one. All spikes have a duration of 1, a magnitude of 1 (or -1) and are connected via synapses to a circular region of neurons in a 9x9 area. Two of the layers are fast spiking neurons with a slightly modified b values (changed from 0.2 to 0.25). The other layer contains chattering neurons, although the automaton still functions if an unmodified fast spiking layer is used instead.

A picture containing chart

Description automatically generated

Figure Izhikevich Reverse Inhibition. Each layer excites the one below and inhibits the one above.

No stable, repeating, or travelling patterns have been identified, but the automaton remains active and chaotic for long periods, possibly indefinitely, after being given random noise as a trigger (see Figure 16).

Sending inhibitive spikes backwards is equivalent to sending them 2 layers forwards, and also 5 or 8 layers forwards, because of the looping nature of the design. This appears to overcome the problems with inhibition lagging the excitation that needs to be inhibited. Similar designs using loops of up to 8 layers have also been shown to work, the key to stability being the arrival of inhibitory spikes ahead of the excitation from the same source.

A picture containing light

Description automatically generated

Figure Izhikevich Reverse Inhibition Automaton

A video of this automaton in action is included with the report as reverse\_inhibition.mp4.

Alternative approaches involving longer synapses (or in biological terms longer axons) for the inhibition, or placing delays on transmission of excitation, were far less successful, and this may be a matter of scale; a delay of three iterations on the spike is absorbed in a much longer time periods within the differential equations governing each neuron.

# Testing

Both automated and manual testing were used for this project. Automated testing came in the form of the NeuronTest project, and manual testing was ad hoc and made considerable use of the impulse response graph functionality.

## Automated Testing

There is a test project (NeuronTest) which links with NeuronSim. This is not a GUI project and does not test the user interface, being more concerned with the framework and simulation.

Tests exist for each of the main classes of the framework and, where a neuron model was required, generally use the Life model since that one has a well-established correct behaviour.

In addition to the tests for specific areas of functionality there is also a performance test (which is essential for getting consistent measurements during profiling in the chaotic environments being simulated) and a stability test which creates and destroys layers and synapse matrices randomly, regularly stepping the simulation itself, in an attempt to crash the program. The random seed is logged, so if it does cause a crash the run can be repeated to verify a fix. It has not managed to cause a crash since before the first phase completed but was very useful when it did.

## Manual Testing

Almost all the functionality exposed by the GUI is used regularly and gets tested on an ad hoc basis as a result. The GUI itself has been manually tested by individually going through every button and setting, but is a relatively thin view layer on top of the simulation library and doesn’t have a great deal that needs to be tested.

The user interface does provide an impulse response graphing window, however, and this proved very useful for verifying that neuron models behaved in the manner intended. The Izhikevich, True North and Kumar models were all checked in this way, and the results for the first two were qualitatively what would be expected for the configurations used. The Kumar model never gave a reasonable response and while it seems unlikely it is incorrect (because it is a very simple algorithm that can be checked by eye) it cannot really be said to have been tested successfully.

# Planning and Time Management

The initial project plan placed a deliverable of a working system for simulating automatons on 2021-02-25, working SNN models by 2021-03-25 and a finalized automaton and its specification on 2021-04-20. Approximately 1 month for each major phase.

The last major change to the code base was the multithreading implementation, which was committed on 2021-02-23, which is a very good match to the original plan. Ongoing improvements continued for another two months in small incremental changes, but that was not unexpected.

The neuron models were implemented far faster than planned and were all available by 2021-02-27. With the underlying framework in place, implementing a new neuron model involves copying an existing one and changing a single function, and that function is usually not complicated. This has been taken as a good sign regarding the design.

Obtaining a workable automaton took longer than expected but was still completed on schedule due to beginning the third phase early. The initial plan was to come up with a single set of rules and this project has ended with two automatons, one based on LIF and one based on Izhikevich.

Overall, the project ran to schedule, met its objectives, and delivered on time.

# Reflection

## Interface

The interface is sufficient for the task at hand and has the flexibility to acquire more viewing styles or control systems as needed but suffers a little in the layer and synapse windows. They were placed vertically at the side of the main window to leave the graphics viewing area as close to the optimum square shape as possible on a typical wide screen display.

The principle is sound, but the design anticipated only two or three layers and matrices. The automaton with the jellyfish pattern, as an example, has 8 layers and 40 synapse matrices, resulting in very tall thin scrolling regions. Qt’s docking system is rather limiting, and it would probably have been better to insist on using two windows all the time, drop the docking capability, and place the layer and synapse configuration panels horizontally to make better use of screen space.

It would have been interesting to be able to view the internal state of a single neuron in greater detail, and to watch it change over time as the simulation progressed, but it is not clear how that could be generalized for arbitrary neuron models, and the effort required would not be justified by the reward.

The use of OpenGL was probably a good decision. Using OpenGL directly adds a lot of function calls and a third-party library could have saved the trouble, but avoiding dependencies is also valuable, and left as much room as possible for optimization in case it proved necessary. OpenGL made additive blending and 3D modes available, which a basic image viewer widget would be unlikely to provide.

## Implementation

Several refactors of the code base occurred during development, mostly relating to the way spike information is passed through synapses. The need to move to a flat memory model and the decision to go with a multithreaded simulation step both had significant knock on effects. It was to be expected, for an investigative project of this nature, and has not caused major problems. Insisting on separation of view from controller and controller from model helped to keep the disruption to manageable levels. Having an automated test suite helped instil confidence that the refactors hadn’t broken anything.

The amount of CPU time spent adding zero weight spikes to neuron potentials (see section 5.6) is excessive and, given the constraints, probably unavoidable, which means further attempts to optimize this code will have a poor return on investment. Dropping the flat memory model would help, if an alternative way of dealing with excessive activity could be found, but a GPGPU based implementation of a specific automaton that was known not to excessively spike would be a better next step if larger or faster simulations are required.

## Graphics

The graphics changed many times during development as more things were learnt about the kind of automaton that would end up being used. The final system is functionally good but has room for improvement.

The image of the neurons for each layer is obtained as an ABGR 32 bit texture, which was necessary at first, but the alpha channel has been replaced with additive blending modes and the red, green, and blue channels are now always the same to accommodate per-layer colouring. A single 8-bit greyscale texture would serve the same purpose, would cost slightly less time to write, and use significantly less memory. Perhaps more importantly, it would result in much less data being uploaded to the graphics card in each iteration. What impact that would have on performance is unknown, and probably depends on the hardware the program is running on.

Similarly, the synapse matrices are loaded as 32 bit image files because they originally encoded both weights and delays, but delays are now handled as a set of simple functions in a drop down box and the images could be made greyscale and smaller.

## File Formats

The difficulty in making automatons balance with R = 1 led to many of the attempts using far more layers than originally anticipated. Often, these would be identical to each other, and connected in a pattern of almost identical synapse matrices. It would have been convenient to be able to edit these manually as text files on some occasions, or to be able to run through them with a shell script.

The save file format for everything except layer configuration data is binary, which makes a lot of sense for the internal neuron states but makes manipulating them outside of the application itself close to impossible. Storing all but the neurons states and spike train information in an editable format would have been a better decision. It would also have made debugging certain problems easier.

# Ethics

No ethical issues are anticipated arising from this project, beyond those typical of any other software project. This specific research will make only sporadic use of one computer.

From an environmental point of view it may be worth noting that neural nets are very processor intensive and that if projects of this nature became common place we would expect them to drive a similar energy consumption pattern to that seen in bit coin mining (which is more energy intensive than actual mining [26]). Purpose built hardware should mitigate this effect to some extent.

This project makes some references to the concept of consciousness in artificial intelligence. There are an enormous number of ethical issues that arise when considering this subject but none of them arise from this project specifically. The automatons are not intelligent, aware or conscious.

# Conclusions and Further Work

The software created in this project is fast, flexible and effective for its intended purpose, and can be easily extended with new SNN models and new patterns of connectivity between neurons. Various SNN models have been experimented with in a large variety of arrangements, and stable self-sustaining activity meeting the objectives has been achieved with the LIF and Izhikevich models.

## Learning and Automatons

The automatons discussed here cannot learn or adapt and while they provide an instructive environment for investigating SNN behaviour they appear to be a dead end as far as application goes. It is certainly possible to add more models to the framework provided, and by doing so to search for methods of self-regulation and self-stimulation that achieve the desired results more effectively, but the act of making the automatons here capable of learning would be the act of converting them into a deep, re-entrant but sparsely connected neural network with inputs and outputs and they would cease to be automatons in the way that word has been used in this document. This is not to say it would not work, but it is a purpose better served by existing software or hardware solutions.

## Inhibition

A strong AI does not have to sustain its own activity with a region of re-entrant neural net – there is no reason why it cannot be driven by an external input of some other form entirely. The human brain, however, presumably does [10]. Whether to further understand brains or to create conscious AIs, the network will still need to prevent itself saturating, and doing so appears to be reasonably challenging. It also appears to require inhibitive behaviours.

Inhibitive neurons and spikes are referenced in a lot of literature [1][2] and many spiking neuron models take into account behaviours such as rebound spiking [3] [19] [21]. However, the absence of a minimum potential in most references to LIF [1] [2][3] and Izhikevich [21] suggests that they see minimal actual use. Additionally, none of the models referenced in this project specify how a shunting inhibition might be implemented. Does a shunting inhibition spike only apply a divisive or multiplicative effect during the iteration in which it is received, or should it persist, decaying over time like LIF? Does it have more complicated interactions and require more than one state variable like Izhikevich? There does not appear to be a simplified computational model available for this, yet it seems likely that it is essential for any attempts at building a strong AI. It also seems reasonable to assume that the presence of shunting interactions between neurons could increase their computational power. Further research into this area may be rewarding.

# References

|  |  |
| --- | --- |
| [1] | T. P. Trappenberg, Fundamentals of Computational Neuroscience, Truro: Oxford University Press, 2002. |
| [2] | E. M. Izhikevich, Dynamical Systems in Neuroscience, The MIT Press, 2007. |
| [3] | A. S. Cassidy and etal, “Cognitive Computing Building Block: A Versatile and Efficient,” IBM, 2013. |
| [4] | A. L. Hdgkin and A. F. Huxley, “A QUANTITATIVE DESCRIPTION OF MEMBRANE CURRENT AND ITS APPLICATION TO CONDUCTION AND EXCITATION IN NERVE,” *Journal of Physiology,* vol. 117, pp. 500-544, 1952. |
| [5] | M. Gardner, “MATHEMATICAL GAMESThe fantastic combinations of John Conway's new solitaire game "life",” *Scientific American,* pp. 120-123, 10 1970. |
| [6] | W.Mass, “Networks of Spiking Neurons: The Third Generation of Neural Network Models,” Elsevier Science Ltd., Graz, 1996. |
| [7] | A. Taherkhani and etal, “A review of learning in biologically plausible spiking neural networks,” *Neural Networks,* vol. 122, pp. 253-272, 2020. |
| [8] | G. Eyal and etal, “frontiers in cellular neuroscience,” 29 June 2018. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncel.2018.00181/full. [Accessed 01 2021]. |
| [9] | Synaptitude, “wikipedia,” [Online]. Available: https://commons.wikipedia.org/w/index.php?curid=6030880. [Accessed 31 01 2021]. |
| [10] | T. e. a. Zhao, “Consciousness: New concepts and Neural Networks,” *Frontiers in Cellular Neuroscience,* vol. 13, 2019. |
| [11] | G. M. Edelman, J. A. Gally and B. J. Baars, “Biology of Consciousness,” *Frontiers in Psychology,* vol. 2, 2011. |
| [12] | T. H. Abraham, “(Physio)logical circuits: the intellectual origins of the McCulloch-Pitts neural networks,” *Journal of the History of Behavioural Sciences,* 2002. |
| [13] | Y. Cao, Z. Zeng and T. Huang, “A Modified Elman Neural Network with a New Learning Rate Scheme,” *Neurocomputing,* April 2018. |
| [14] | E. Zio and etal, “Failure and Reliability Predictions by Infinite Impulse,” *Chemical engineering Transactions,* vol. 26, 2012. |
| [15] | . E. Hildt, “Artificial Intelligence: Does Consciousness Matter?,” *Frontiers in Psychology,* vol. 10, p. 1535, 2019. |
| [16] | S. Ghosh-Dastidar and H. Adeli, “Spiking Neural Networks,” *International Journal of Neural Systems,* vol. 19, no. 4, pp. 295-308, 2009. |
| [17] | Eugene.M.Izhikevich, “Which Model to Use for Cortical Spiking Neurons?,” *IEEE Transactions on Neural Networks,* vol. 15, no. 5, 2004. |
| [18] | A. M. Nagya and N. H. Sweilamb, “An efficient method for solving fractional Hodgkin–Huxley model,” *Physics Letters,* vol. 378, pp. 1980-1984, 2014. |
| [19] | A. Kumar, S. Kansai and M. Hanmandlu, “Classification of different neuron behavior by designing spiking neuron model,” in *2013 IEEE International Conference ON Emerging Trends in Computing, Communication and Nanotechnology (ICECCN)*, Tirunelveli, 2013. |
| [20] | Ş. Mihalaş and E. Neibur, “MIT Press Direct,” 01 March 2009. [Online]. Available: https://direct.mit.edu/neco/article/21/3/704/7389/A-Generalized-Linear-Integrate-and-Fire-Neural. [Accessed 2021]. |
| [21] | Eugene.M.Izhikevich, “Simple Model of Spiking Neurons,” in *IEEE Transactions on Neural Networks*, 2003. |
| [22] | X. e. a. Jin, “Modeling Spiking Neural Networks on Spinnaker,” *Computing in Science and Engineering,* pp. 91-97, September/October 2010. |
| [23] | “nest-simulator,” Nest Initiative, [Online]. Available: https://nest-simulator.readthedocs.io/en/latest/contents.html#. [Accessed 2021]. |
| [24] | “OpenSourceBrain,” [Online]. Available: https://www.opensourcebrain.org/. [Accessed 2021]. |
| [25] | J. e. a. Vitay, “ANNarchy: a code generation approach to neural simulations on parallel hardware,” *Frontiers in Neuroinformatics,* vol. 9, 2015. |
| [26] | S. Burbeck, “Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC),” ParcPlace Systems, Inc., [Online]. Available: https://web.archive.org/web/20120729161926/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html. [Accessed 20 04 2021]. |
| [27] | M. J. Krause and T. Tolaymat, “nature.com,” 05 11 2018. [Online]. Available: https://www.nature.com/articles/s41893-018-0152-7. [Accessed 12 2020]. |

# The First Appendix