Cellular Automaton

And

Spiking Cellular Automaton

Andrew D Newman

University of York

April 2021

3rd Year Project Final Report for degree of BEng in Electronic Engineering

# Introduction

TODO: references

Biological neurons form the brain and its network of sensory inputs and motor control outputs in animals, including humans. 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.

# 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] [7]. 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 [3] [4]. 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.

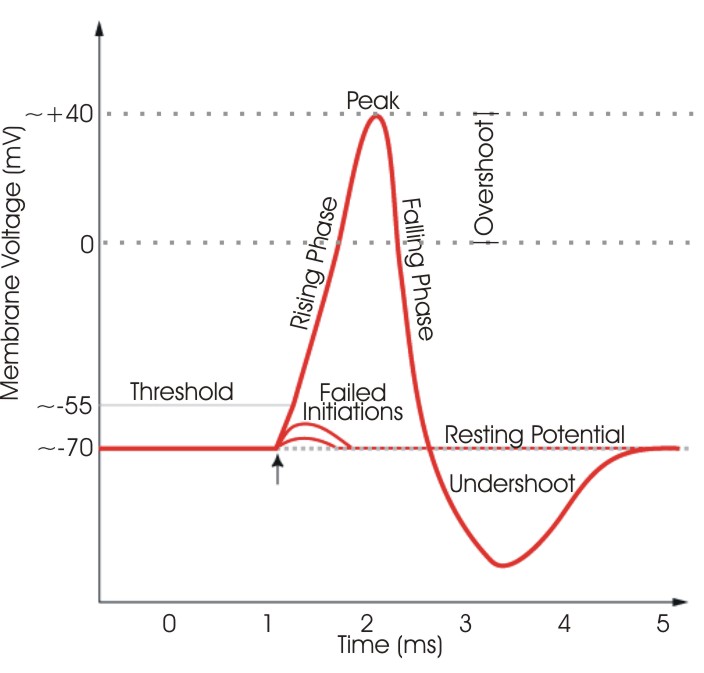


Figure Action Potential

Original author Synaptidude at English Wikipedia. GFDL 1.2, <https://commons.wikimedia.org/w/index.php?curid=6030880>. Accessed 2021-01-31

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

## Noise

## Brain Structure

## Consciousness and Wakefulness

# Artificial Neural Nets

The first artificial neuron was designed in 1943 [7], 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 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 [7] and predicting failure of engineered components and systems [8].

## 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 [9]), 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 right direction. 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 [11]. 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 [4] [5] (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] [4] [5].

## 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 [12]). Faster approaches to it have been proposed [9] 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 [11] [13] [14].

### 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] [15] 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 [12]. The Mihales-Neibur [16] 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 [17] 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.

### Kumar et al

Kumar et al [12, 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 [14] 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 4.2.

### Spinnaker

The SpiNNaker system [18] 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 [17]. 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

## 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 [12]. Nest provides a python module which can simulate many neuron types (including Izhikevich) [13]. OpenSourceBrain provides an enormous suite of neural net related simulation software [14]. 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 [15]. 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 which 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 6).

# Automatons

## Life

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.

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. Complex behaviour emerges from the simple rules.

## Spiking Automatons

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 does not necessarily have an obvious input, 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 2.6. 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.

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 requirement is to have the average value of R be 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 low 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.

One option, similar to that used in life, is to use 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 [14] 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.

A second option is to mimic the behaviour described as the Dynamic Core and Global Workspace hypotheses [23] and outlined in section 2.6. To achieve this a layer of overactive neurons (R > 1) acts as a driver (the dynamic core) and sustains activity in less active layers (R < 1) (the global workspace).

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

The potential of a neuron in most of the described models can grow without limit and is only prevented from doing so by the reset that follows a spike being fired. However, although the Izhikevich model can act as an inhibitory neuron and even mimic the output of the biological ones in the brain, 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. We suspect 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 our automaton that needs to self-regulate.

## Noise

There are many references to stochastic integration (a random chance of synapses being integrated) and noise, with noise being given as an explanation for the speed of the brains response to visual stimuli [1] [2][14]. Stochastic integration can give rise to new behaviours – an occasional spike with a value below the threshold cannot trigger another spike except when random chance has a say in the response, and the TrueNorth architecture [15] relies on this to achieve some of the 20 Izhikevich behaviours. 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, is almost certain to break down under a stochastic system.

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.

## 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 mixed success.

# Objectives

*Most of this section is copied from the Initial Report for this project.*

Conway’s objectives for the Game of Life [5] serve as a starting point for the objectives for 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.”

Additional objectives have been added, relating to the use of SNNs:

1. Information flow between cells must be in the form of spikes similar in nature to those conveyed by axons in biological nervous systems.
2. The behaviour of the automaton should be distinguishable from behaviours which are possible with a 1st or 2nd generation neural net.
3. 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.
4. 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 a tentative argument that it has been achieved is provided in section 7.

# 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 [23]) 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 each other. 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 synapse matrix which defines a 2D array 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

## 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.

## NeuronSim

A simplified UML diagram of the NeuronSim library is shown in Figure 5. 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 places. 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

## 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 memory, but we also hope to reduce the number of cache misses that occur while iterating through the grid. 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 the layer and outputting spikes. This means a separate thread can be used for each layer (see Figure 6) without any overhead in mutex locks, which is a significant performance improvement for multi-layer networks. The program is still basically 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.

## 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 [15]. 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 and it was 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.

# 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 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 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 is limited in what it can do - there are many behaviours it can’t exhibit [12] – 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.

Izhikevich can demonstrate a number of interesting and usable behaviours (an example is shown in Figure 8) 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 9). 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.

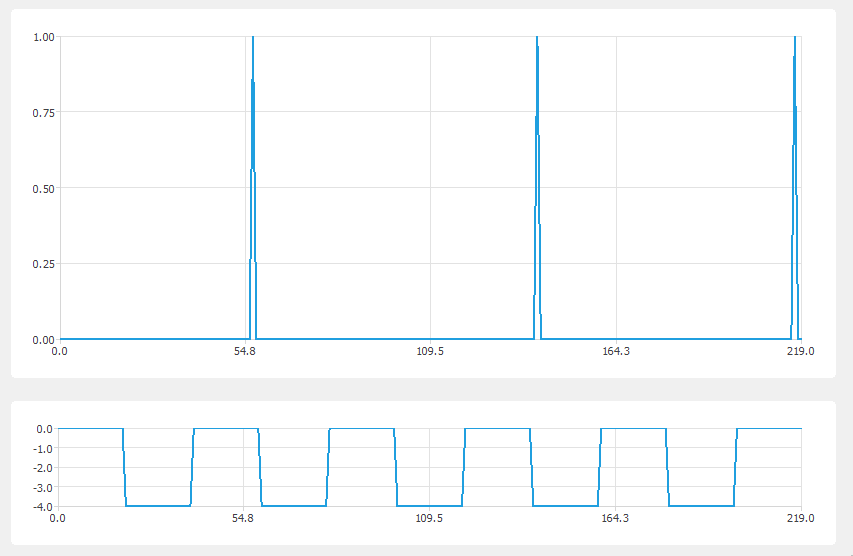


Figure Izhikevich Resonant Neuron Rebound Spiking. The bottom graph shows the input spikes and the top graph the output response of the neuron.

## LIF Inhibited Feedback Loop

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 10) 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 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 – 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.

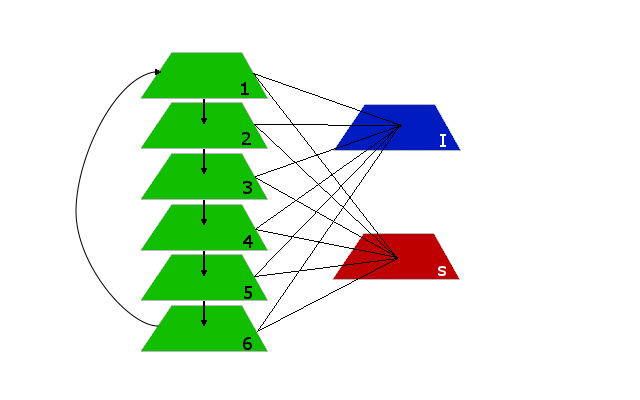


Figure LIF White Matter Synapse Flow

Figure 11 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 10. 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.

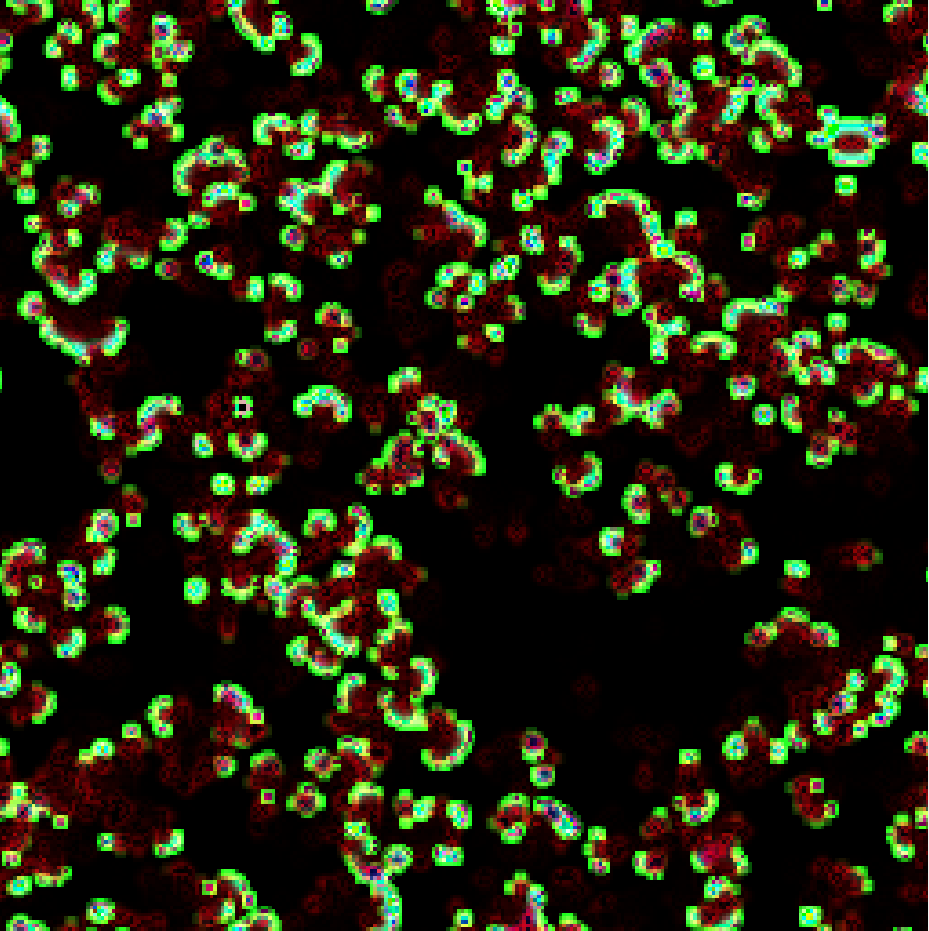


Figure LIF White Matter Automaton. A typical state within a hundred iterations of being triggered by random (white) noise.

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.

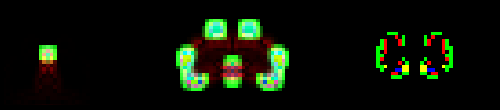


Figure Jellyfish Patterns

Figure 12 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 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 although it probably could be made to happen in a 2nd generation recurrent network, with a bit of effort, it seems likely that there is 3rd generation behaviour present.

# Ethics

We do not anticipate any ethical issues arising in 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 [6]). Purpose built hardware should mitigate this effect to some extent.

# Further Research

## Phase Diagrams and Bifurcations

If the state of a neuron, or indeed anything, can be represented with N variables, then it can be considered as a point in an N dimensional space of possible states. If the behaviour can be represented as a rate of change over time of the N variables then the behaviour can be considered as a set of N dimensional vectors within that space, each indicating the direction and magnitude of the differentials at a point [2, pp. 8-20]. A diagram of these arrows is called a Phase Space plot.

If all the arrows on the plot near a point tend towards it then that is a stable point of equilibrium. Small perturbations to the state will not result in lasting changes. If the arrows all tend away from a point it is an unstable equilibrium. If the arrows tend towards a point along one arbitrary axis but away from it along another it is called a saddle. A set of arrows forming a loop is called a limit cycle attractor.

We will consider two-dimensional phase spaces here, for simplicities sake, but remain aware that biological neurons have more (Hodgkin Huxley has 4 for example). A diagram a 2D phase space is shown in Figure 3, showing a saddle-node bifurcation. The system will rest at the stable equilibrium, and small inputs will have no lasting effect, but any change to the state moving it to the right of the saddle node will cause a significant change in the behaviour, possibly resulting in the limit cycle being reached.

If this diagram were to represent a neuron with the potential on the horizontal axis then we can see we have a coincidence detector that ignores small increases in potential but, on receiving one large enough to cross an internal threshold, begins firing spikes repeatedly until inhibited by an external influence.

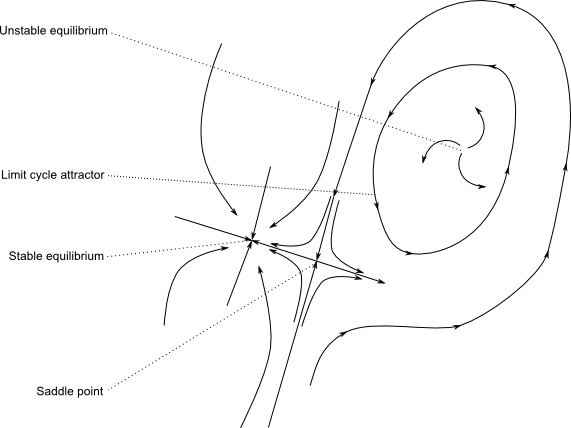


Figure Features of a 2D Dynamical System (recreated in the style of Izhikevich [2, pp 17])

There are four possible kinds of bifurcation in such a system [2, p. 11], depending on whether there exists a saddle node, and whether there exists a limit cycle. For a single model of neuron behaviour to be able to exhibit a good range of different types of behaviour it needs to be configurable to act as any of the four. Those without limit cycles fire only one spike in response to excitation, and those with no saddle node act as resonators instead of integrators.

## Performance

An effective automaton needs to be fairly large, in order to provide enough space for interesting behaviours to emerge. This means it also needs to be fairly fast, so as not to take an unacceptable amount of time to execute. No precise definition of “fairly large” or “fairly fast” is available.

Izhikevich claims a 1ms simulation time for 10,000 neurons with 1,000,000 synapses on a 1GHz PC [13]. 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 and we will try to be reasonable, but it is obvious even without profiling that efficiency will be an important factor here [10] [11].

Note that an automaton of this nature is extremely parallelizable and any attempt to fully optimize it should almost certainly be targeting a GPGPU. This project needs to be flexible and will therefore target the CPU, and consequently the investigative phase will only aim to be fast enough to be usable, rather than as fast as possible.

# 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] | W.Mass, “Networks of Spiking Neurons: The Third Generation of Neural Network Models,” Elsevier Science Ltd., Graz, 1996. |
| [4] | A. Taherkhani and etal, “A review of learning in biologically plausible spiking neural networks,” *Neural Networks,* vol. 122, pp. 253-272, 2020. |
| [5] | M. Gardner, “MATHEMATICAL GAMESThe fantastic combinations of John Conway's new solitaire game "life",” *Scientific American,* pp. 120-123, 10 1970. |
| [6] | 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]. |
| [7] | 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. |
| [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] | A. M. Nagya and N. H. Sweilamb, “An efficient method for solving fractional Hodgkin–Huxley model,” *Physics Letters,* vol. 378, pp. 1980-1984, 2014. |
| [10] | Eugene.M.Izhikevich, “Which Model to Use for Cortical Spiking Neurons?,” *IEEE Transactions on Neural Networks,* vol. 15, no. 5, 2004. |
| [11] | S. Ghosh-Dastidar and H. Adeli, “Spiking Neural Networks,” *International Journal of Neural Systems,* vol. 19, no. 4, pp. 295-308, 2009. |
| [12] | 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. |
| [13] | Eugene.M.Izhikevich, “Simple Model of Spiking Neurons,” in *IEEE Transactions on Neural Networks*, 2003. |
| [14] | A. S. Cassidy and etal, “Cognitive Computing Building Block: A Versatile and Efficient,” IBM, 2013. |
| [15] | GIMP, [Online]. Available: https://docs.gimp.org/2.8/en/plug-in-convmatrix.html. [Accessed 12 2020]. |
| [16] | F. Ponulak and A. Kasinski, “Introduction to spiking neural networks: Information processing, learning and applications,” *Acta Neurobiol Exp,* vol. 71, pp. 409-433, 2011. |