This is an excerpt from my Masters Project, this explains the neural network dynamics and implementation and speedups achieved from using the temporal windowing algorithm.

# Review of Cavalcade Neural Networks

Cavalcade neurons (CNs) are a type of spiking of neuron based on leaky integrate and fire neuron models. The cavalcade neuron’s simple mathematical models are shown in [[22](#Chr10)]. These mathematical models have been expanded on in this project in order to easily reproduce the cavalcade dynamics in a computer simulated environment.

In the event of a pre-synaptic spike arriving at a neuron, the neuron action potential is increased by the spike amplitude the action potential then decays over time depending on a decay constant .

|  |  |
| --- | --- |
|  | (1) |

Equation (1) describes the state of the system at , where represents the index of the discrete spike event.

If additional spikes are presented to the neuron, then the current action potential is added to the incoming spike amplitudes. This value is then the new value to be decayed over time. Figure 2 shows the variation of action potential following the addition of repetitive spikes.

In [[22](#Chr10)] the authors show that when the CN is sent pulses at a fixed frequency, and an infinite amount of time, the action potential can be described using a geometric progression. In this paper an improved analysis is developed, which can cope with spike trains whose amplitude and timings are not fixed.

This analysis builds on [[22](#Chr10)], which shows that spike trains can be described as Dirac-comb functions with varying timings between individual spikes. This allows a concise mathematical description of the cavalcade neuron to be realised.



Figure 2. Fixed amplitude and frequency action potential dynamics of a typical cavalcade neuron

α =0.1, ε=500, t\_f=200.

## Novel mathematical models

For a spike train of fixed frequency and fixed spike amplitude, equation (2) can be used to show the action potential of a neuron at any time after a spike train, described here by a Dirac comb function, is introduced to the neuron.

|  |  |
| --- | --- |
|  | (2) |

However (2) only allows for fixed amplitudes and timing between spikes. In order to modify this equation to allow any spike amplitudes and spike timings, the terms can be replaced by a vector containing spike times (3), and a vector containing corresponding spike amplitudes (4).

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |

Equation (2) can then be split into two separate components. Firstly, the calculated Dirac comb integral is described by (5).

|  |  |
| --- | --- |
|  | (5) |

Secondly the “leak” function, which decays the action potential after each spike, is described by equation (6):

|  |  |
| --- | --- |
|  | (6) |

The value of the action potential at time with the given incoming spike train is then calculated by taking the element-wise product of and the amplitude vector , then multiplying by the transpose of . This is shown in equation (7).

|  |  |
| --- | --- |
|  | (7) |

Calculating the action potential of a neuron at any time, given that it is subject to spikes with random timing and amplitudes, would be time consuming and non-trivial using equations proposed in [[22](#Chr10)]. For example, consider the following situation:

A neuron has incoming synapses with spike trains of varying amplitude and frequency, the equations (8) and (9) below, given in [[22](#Chr10)], only account for fixed spike frequency and amplitude.

|  |  |
| --- | --- |
|  | (8) |

Equation (8) shows the calculation for the neuron’s action potential after the pre-synaptic spike at a fixed time period . This is derived from the fact that at fixed frequency and amplitude, the peaks of action potential of a cavalcade neuron at the time of a pre-synaptic spike is identical to a geometric series.

|  |  |
| --- | --- |
|  | (9) |

Equation (9) is also derived from the geometric progression observable at fixed frequency and amplitude. These two equations are useful at fixed frequency and amplitude, but due to the fact that they are based around geometric progressions, do not work when amplitudes and spike timings are variable.

However the new method given above for calculating action potentials makes analysis of neurons with multiple inputs very simple to implement in mathematics software such as MATLAB.

Also, due to the discrete spiking and a deterministic decay function, implementation of a real-time system is simple for both iterative and event-driven neural network algorithm approaches.

## MATLAB analysis

Here are a few examples of Cavalcade Neural Networks being modelled with the proposed novel mathematics. Source code for these samples can be found in appendix section 13.1.1

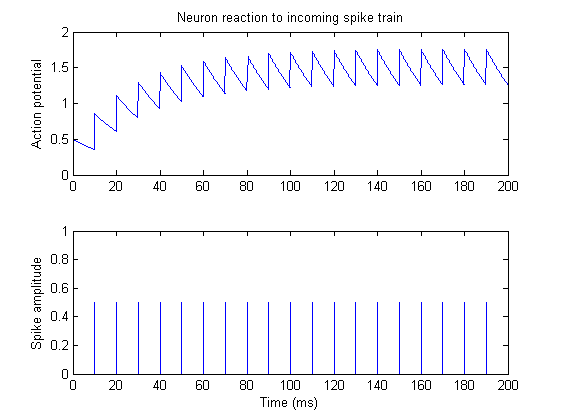


Figure 3. Graph similar to Figure 1, but calculated using the new mathematical model,

, , .

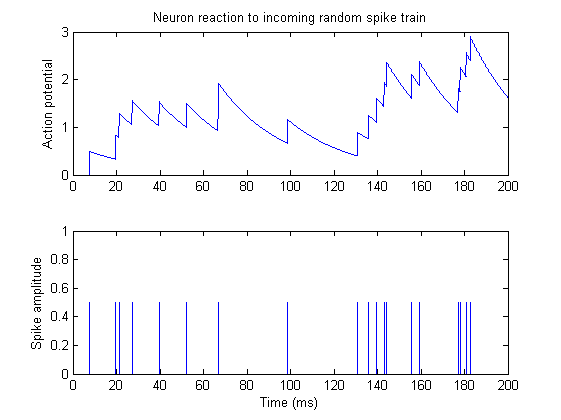


Figure 4. Cavalcade neuron reaction to random input spike train.

, 0

Figure 4 shows the dynamics of the action potential in a Cavalcade neuron with random spike timing. It can be noted that spikes in close temporal proximity cause the action potential to rise fast, whereas spikes separated with large intervals in time allow the action potential to settle. It must also be noted that can be negative (an inhibitory spike). Due to the Cavalcade neuron’s simple mathematical models, Cavalcade neural networks are simple to implement on computers and therefore a good model to use in large artificial neural networks systems.

## Saturation step-response

It can be shown that curve of maximum action potential in a cavalcade neuron with fixed input frequency and amplitude can be modelled as a step response. For example the step response of the neuron saturation is given by the following Laplace domain function:

|  |  |
| --- | --- |
|  | (10) |

And the time domain equivalent:

|  |  |
| --- | --- |
|  | (11) |

Where is given by equation (9).

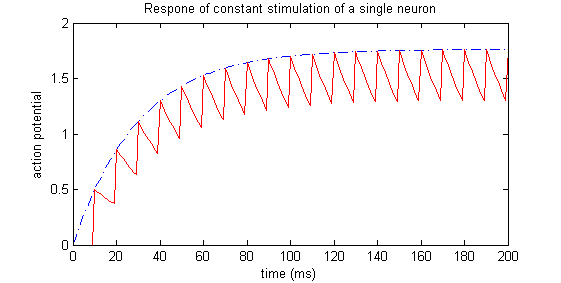


Figure 5. The action potential saturation of a cavalcade neuron modelled as a step response. The blue dotted line shows the output of Equation (11) whereas the red continuous line is the standard cavalcade time response.

, ,

Using this equivalence, the saturation value can be interpreted as the input current to the cavalcade neuron, and values of, and calculated accordingly. This is a very useful method of converting from continuous time inputs to a discrete spiking domain. MATLAB Source code for this example can be found in the appendix section 13.1.2.

# Neuretix Software

Neuretix is a novel set of software libraries and tools designed to simulate millions of neurons in real-time, with any structure and cavalcade dynamics. Neuretix also provides tools for rendering neural networks in 3D in order to visualise and understand excitation in large neural networks.

## Neuretix Architecture

Neuretix uses an object-oriented architecture to describe cavalcade neural networks. Instead of using mathematical modelling such as matrices and indexes containing weights and axon mappings, which is used in many common methods [[23](#Izh03)] [[25](#Mat00)], Neuretix takes advantage of basic memory manipulation techniques such as linked-lists, which have been used to optimise algorithms in computer science since very early computing.

Similar to neurons in biological brains having a location within the brain, the locations of neurons and axons in the Neuretix software have literal locations in memory space (in RAM). Axons are mapped between neurons by using memory location pointers. This reduces the memory usage of methods that use weighting matrices significantly, as there are the same numbers of axon pointers as there are axons. There is no need for square matrices of equal dimensions to the number of neurons.

Figure 6 shows a graphical example of how this architecture is achieved.

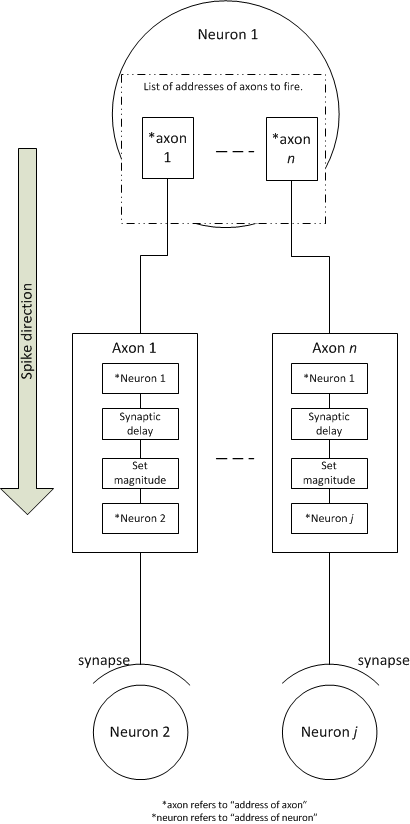


Figure 6. Diagram of software implementation of neurons and axon in the Neuretix software.

Due to the fact that neurons and axons are implemented in discrete object-oriented classes, there is the ability to implement complex functionality such as refractory periods and learning. It is also useful in the temporal windowing algorithm described later.

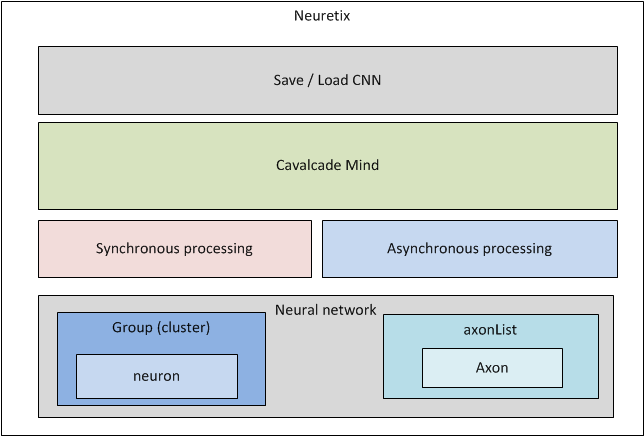


Figure 7. Diagram of Neuretix Software stack

Another advantage of the memory structure of the Neuretix system is that by modifying the list of axon pointers in any neuron, new connections can be made or broken on-line and in real-time. This is similar to the method proposed in [[22](#Chr10)] for axonal genesis.

Figure 7 shows an abstracted and simplified version of the Neuretix software stack. It would be very difficult to hard code the memory parameters described by Figure 6 therefore Neuretix has built in API and memory management systems to reproduce optimal neural networks with ease.

## Network XML Representation

Neuretix allows neural networks to either be designed programmatically using inbuilt methods, or by using XML schemas. In the Neuretix system, a neural network is defined by two files: the cluster, and the axon list.

The cluster defines the layout and cavalcade dynamics of various groups of neurons and the axon list defines any axons that exist within that cluster.

|  |
| --- |
| Example clusters.xml |
| <ClusterData>  <Group GID="0" x="-0.3" y="0.0" z="0" r="1" g="0" b="0">  <Neuron NID="0" x="-0.3" y="0.3" z="0.0" Epsilon="500" Threshold="0.7" Refractory="10"/>  <Neuron NID="1" x="-0.3" y="-0.3" z="0.0" Epsilon="500" Threshold="0.7" Refractory="10"/>  </Group>  <Group GID="1" x="0.3" y="0" z="0" r="0" g="1" b="0">  <Neuron NID="0" x="0.3" y="0.3" z="0.0" Epsilon="500" Threshold="0.7" Refractory="10"/>  <Neuron NID="1" x="0.3" y="-0.3" z="0.0" Epsilon="500" Threshold="0.7" Refractory="10"/>  </Group>  </ClusterData> |

|  |
| --- |
| Example axons.xml |
| <AxonData>  <Axon AID="0" NsID="0" NdID="1" GsID="0" GdID="1" SynDelay="0" Mag="0.7"/>  <Axon AID="1" NsID="1" NdID="0" GsID="0" GdID="1" SynDelay="0" Mag="0.7"/>  </AxonData> |

The neural network described in XML above consists of four neurons separated into two groups and two axons. The geometric locations of the neurons and colour of neuron groups are encoded into the Neuretix system in order to distinguish between neuron groups when viewing simulations. The neurons and groups within a cluster are also assigned ID numbers which are used in decoding the neural network from the XML form to memory space.

The axon schema shown in the example axon list *axons.xml* uses the ID numbers from the groups and neurons in order to identify the pre and post-synaptic neurons:

* *NsID* - refers to source neuron (pre-synaptic)
* *NdID -* destination neuron (post-synaptic)
* *GsID* - the group to which the pre-synaptic neuron belongs.
* *GdID -* the group to which the post-synaptic neuron belongs.

Axons description also contains a value of a synaptic delay *SynDelay* and a fire magnitude *Mag* which is the value (the spike amplitude) at the post-synaptic neuron.

Figure 8 shows a diagram of the network that would be produced by the Neuretix software when presented with the XML files given above. This is only an example of a very simple system, but using this XML representation as a base level, any network of any structure can be created. Various other parameters could also be added to these schemas if more complexity is required, for example if the neural networks implemented spike timing dependant plasticity, the parameters for learning rates would be added to the *<Axon>* elements.

In the Neuretix 3D simulation, the neurons are rendered as cubes and axons as lines. Figure 8 is simply an example representation of how the neural networks are decoded in the Neuretix software.

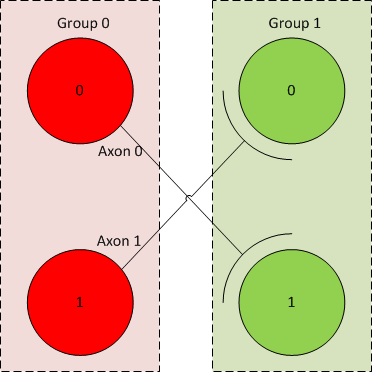


Figure 8. diagram showing the neural network structure produced based on the previous XML neural network files.

Neuretix can also handle multiple implementations of Cavalcade mind instances (shown in Figure 7). This means many different separately running neural networks can be created and run independently or in parallel with a multi-core processor.

## Neuretix neural network processing algorithms

Neuretix is able to process neural networks using synchronous and asynchronous (event-driven) methods.

### Synchronous

The synchronous method simply iterates though all neurons calculating their current values of action potential dependent on the last spike times (equation (1)), and then subsequently all axons, processing any axons that need to induce post-synaptic spikes due to their synaptic delays elapsing.

Using this method, neurons and axons are iterated over even if they have had no input changes in the last iteration.

Using a weight matrix [[23](#Izh03)], to update neural networks at each time step incurs a time complexity of roughly where is the number of neurons. This is mainly due to the fact that if there is a large amount of wasted calculations where the weights of the matrix are 0 (no axon exists).

Neuretix’s synchronous algorithm reduces the computational complexity in synchronous mode to where is the number of axons. By separating the definition of axons from neurons, there is no need for nested loops calculating all values inside matrices. This is a very useful improvement and increases neural network performance. It is especially useful for viewing neural networks in real-time in the 3d environment, as the route of excitation through networks can be observed.

However, using a novel event-driven method called “Temporal Windowing” the time complexity can be reduced to a function that is invariant of network and axon list size.

### Asynchronous (Temporal Windowing)

Neuretix’s asynchronous temporal windowing (TW) algorithm makes the time complexity of a single time-step or iteration of the entire network where S is the number of axons that cause post-synaptic spikes to occur at that point in time.

The TW algorithm works by storing a set number of lists, corresponding to each possible synaptic delay value, the TW algorithm assumes that synaptic delays can be discretised to multiples of the time step of the neural network algorithm.

TW also takes advantage of the fact that the cavalcade dynamics are formed from deterministic mathematical functions and instead of iteratively updating action potentials at each time step, action potentials are updated by the differential of timestamps corresponding to the spike timings. For example when a neuron receives a spike , it records the time that the spike was received, and at the next spike , calculates the difference in time in order to calculate the action potential decay up to that point. is then set as . There is no need to calculate the action potential between spikes.

TW also dynamically allocates memory in order to reduce the consumption of memory in the cavalcade mind.

Figure 9 shows a detailed example of the architecture of the TW algorithm.

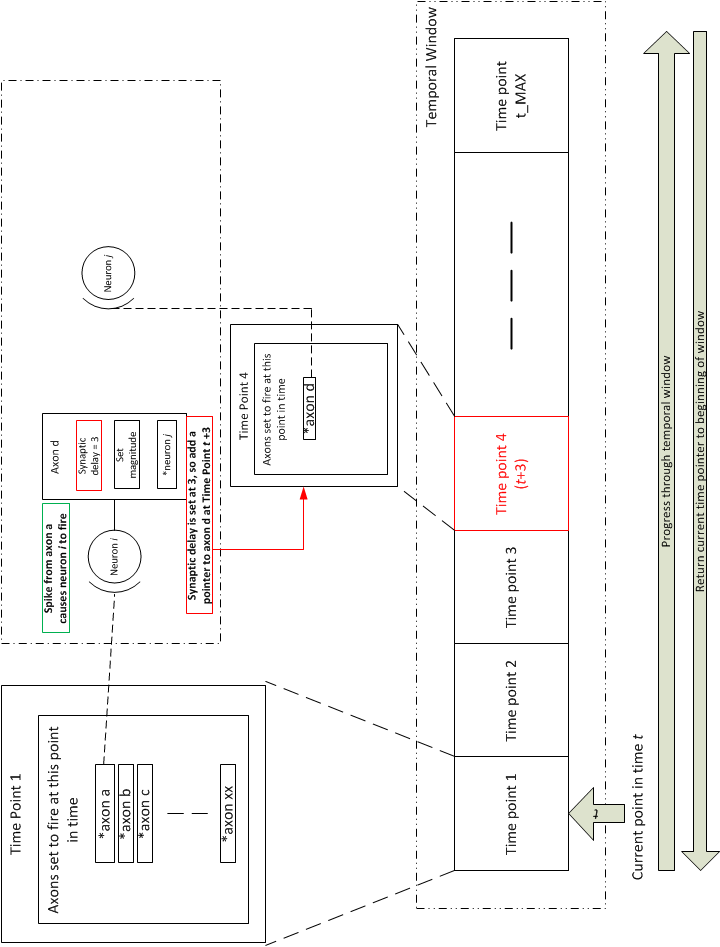


Figure 9. Temporal windowing algorithm diagram that shows the basic premise of operation and how this ties into the Neuretix neural network architecture.

TW works on a basis that there is a memory structure that holds information about all axons that fire at specific points in time. Each point in time is defined as a “time point.” Each time point contains a linked-list of pointers to axons that are set to cause post-synaptic excitation or inhibition at that time. The time points are initially allocated a fixed number of list items in memory. This number increases dynamically at run-time if required.

It would not be possible or practical to have a time point structure for every moment in time, as that would require infinite memory and there would be no point in holding onto memory of past spikes in previous time points. To manage this issue, the temporal window consists of a set number of time points, which effectively acts as a Mobius strip of memory, that is, the memory allocation effectively wraps around on itself. As long as the number of time points is larger than the maximum synaptic delay of any axon, the TW algorithm can overwrite past experiences.

TW implements a pseudo program counter that iterates through all the time points in the entire window, when it gets to the last time point of the window, it returns to the first time point

In Figure 9, time point 1 contains references to three axons that have been set to fire at this point, axon a,b and c. In this example, the effects of the firing of b and c are ignored.

The following steps explain how the temporal window algorithm manages these axon firing events:

* Axon a, is connected to a post-synaptic neuron *i*.
* The spike from axon a causes neuron *i* to fire.
* Neuron *i* is connected to axon d which is in turn connected to neuron *j*
* On firing, neuron *i* causes a reference to axon d to be placed at time point *t* + 3 (synaptic delay of axon d is 3)
* Assuming time point 2 and 3 have no axon fire events registered, the algorithm does nothing in these two steps.
* When the current time pointer reaches time point 4 the synaptic delay of axon d has elapsed.
* Axon d’s spike with its corresponding amplitude is sent to neuron *j*.

It must be noted that as the temporal window only has a finite number of time points, if the current time step t plus the synaptic delay of any axon registering to fire is larger than t\_MAX, the axon must be registered in the “wrapped” time point of the temporal window.

Although this algorithm is far less intuitive than using mathematical methods such as weight matrixes and iteration through all components of the neural network, this method dramatically speeds up neural network processing.

### TW Benchmarks

In order to test the efficiency of the TW algorithm a set of benchmark problems were set and performance was compared between synchronous and TW algorithms.



Figure 10. Neuron chain benchmark test network diagram

Figure 10 shows a neural network diagram for a benchmark test to test the performance of the TW algorithm against synchronous methods. neurons are connected in a chain, each neuron has identical dynamics as do the axons between each neuron. The axons have a fixed synaptic delay of 1 time step and are guaranteed to cause a post-synaptic spike.

The benchmark is to time how long it takes a spike, starting from neuron 0 to reach neuron using both algorithms. The results of this test can be seen in Figure 11.

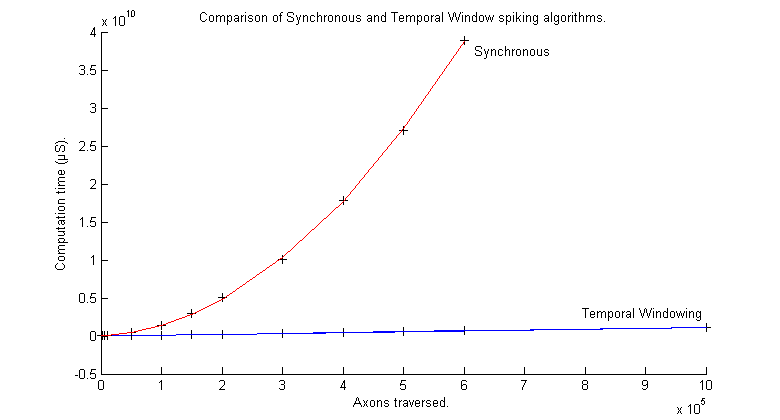


Figure 11. Comparison of TW and Synchronous algorithms on a neuron chain test shown in Figure 10

Figure 11 clearly shows that at low network activity (in this case a single spike) the TW algorithm has a time complexity of as there is only a single spike to process at each time step of TW. The Synchronous method however has a time complexity of as expected. It can clearly be seen that as the number of neurons increase in the network, the computation time of the chain test increases exponentially for the synchronous method, and linearly for the TW method.

Furthermore, with high network activity the TW algorithm still outperforms the synchronous method by a fairly large factor. This can be explained by the simple analysis of the time complexities of the two algorithms shown in Table 3.

Table 3. Analysis of time complexity of neural network algorithms

|  |  |
| --- | --- |
|  | Time complexity |
| Synchronous |  |
| Temporal Window  (with respect to number of spikes) |  |
| Temporal Window  (50% activity) |  |
| Temporal Window  (100% activity) |  |

Even when the temporal window algorithm is processing a neural network with 100% activity, its time complexity is still reduced due to the fact that it does not iterate through the parameters of the neurons. Updates of neuron parameters are processed only when spikes reach the neurons.

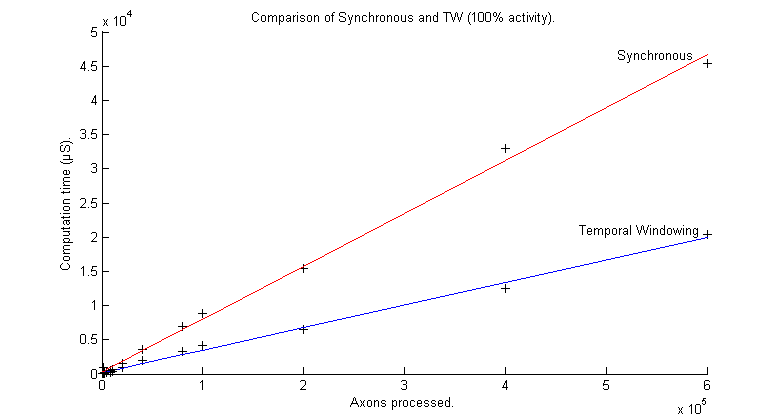


Figure 12. Completion time for a single iteration of a network, arranged as in Figure 10 with 100% network activity

It is unlikely that a spiking neural network of a very large size will ever require 100% network activity at any point in time, therefore TW is a very good algorithm for fast processing of spiking neural networks.

The Neuretix libraries have Synchronous and temporal windowing methods implemented and tested within them, and can be imported and used in any projects using a set of simple API tools.

### Memory requirements

A final point about the temporal windowing algorithm is an analysis of the memory usage. Due to each time point containing a linked-list of axon pointers, potentially each time point structure could be filled with every axon in the network. This could potentially take up a large amount of memory.

To analyse this memory usage, a worst case scenario was implemented using the following parameters:

|  |  |
| --- | --- |
|  | (12) |

|  |  |
| --- | --- |
| 1.28 GB | (13) |

It can be seen in equation (12) that memory usage can become large if a high time resolution and large networks are required. However, reducing the time resolution of the problem reduces memory usage by a factor of 10. It also needs to be noted that this is the worst case scenario, where network activity is constantly 20,000.