Spike Time Interval Computational Kernel: Reading and implementation

Antoine Guillot

September 16, 2018

Abstract

The goal of the project is to explore the various possibilities which are offered by the STICK paradigm and to code conventional functions using its neurons and modules.
The presented architectures were implemented with Brian2 and python, the code is available on my Github.

1 STICK

The paper proposes a Spiking Neural Network framework with Turing complete computation ability. Furthermore, the framework does not suffer from the Von Neumann architecture computational bottleneck (the separation of the memory and the processing power).

The framework has two mains component, the neurons and the synapses. The neurons have the following dynamics:

$$\begin{aligned} \tau_m . \frac{dV}{dt} &= g_e + gate.g_f \\ \frac{dg_e}{d_t} &= 0 \\ \tau_f . \frac{dg_f}{dt} &= -g_f \end{aligned} \tag{1}$$

τ_m and τ_f are the characteristic times of the neurons and are respectively set to $100s$ and $20ms$. The other variables are initialized at 0. When V exceeds a threshold $V_t = 10mV$, the neuron emits a spike and all its variables go back to their initial value.

The communication between different neurons is done with the synapses. Similarly to their biological counterparts, they pass a message (i.e. a spike) from one neuron to an other. The amplitude and the sign of the message are defined by the synapse weight w. Four synapses are introduced in the paper:

- V-synapses which modify the neuron potential: $V \leftarrow V + w$
- g_e-synapses which modify the neuron potential: $g_e \leftarrow g_e + w$
- g_f-synapses which modify the neuron potential: $g_f \leftarrow g_f + w$

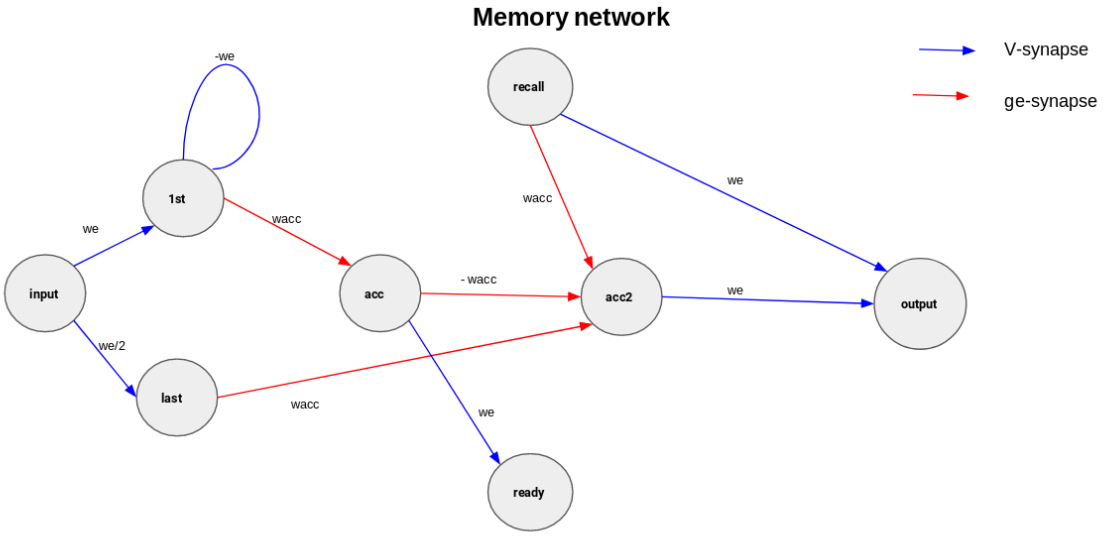
- g_f-synapses which modify the neuron potential: $gate \leftarrow 1_{w=1}$

Standard weights for each synapses are also defined in the paper. In this project, we extend some of the paper results to implement persistent and programmable memory, loop, conditional programs and recursive functions.

2 Persistent and programmable memory

The papers propose the following memory network architecture to store and access a value.

Figure 1: Memory network



The architecture has two main drawbacks:

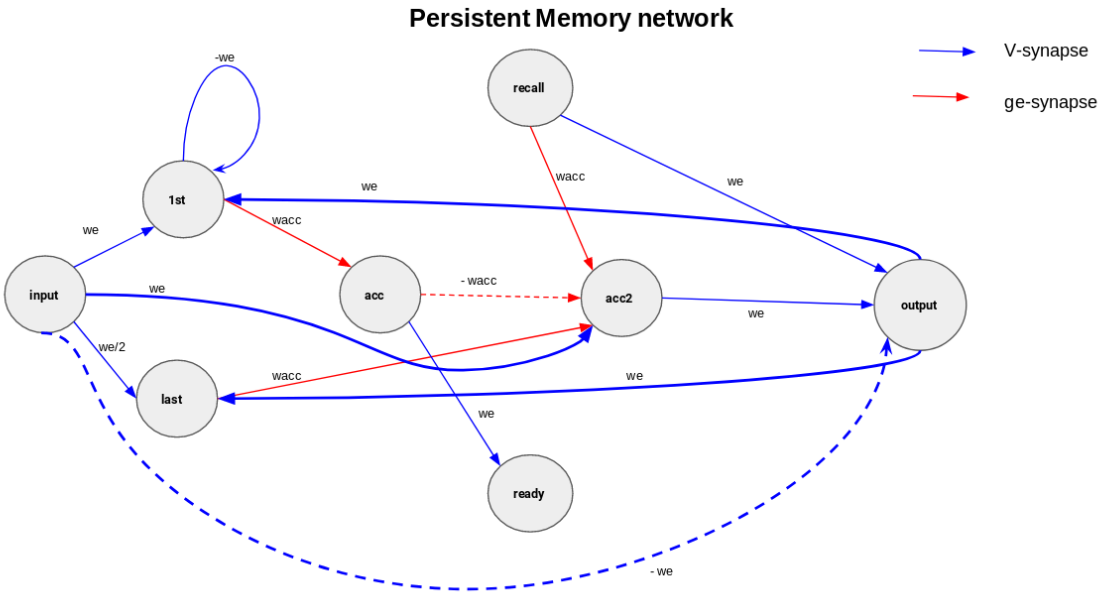
- Reading the stored value destroys it. Once the value has been read, it's lost.
- The architecture does not allow to rewrite the memory with a new value.

These two problems can be solved:

- By connecting the output to the first and last node with a V-synapse, the spike of the output forces the variable to be stored again in the network.
- By connecting the input to acc2 with a V-synapse, we can force the potential of the accumulator to go to 0. To avoid having a spike at the output, we also inhibitate the output with a negative V-synapse.

The persistent and programmable memory has the following architecture:

Figure 2: Persistent Memory network



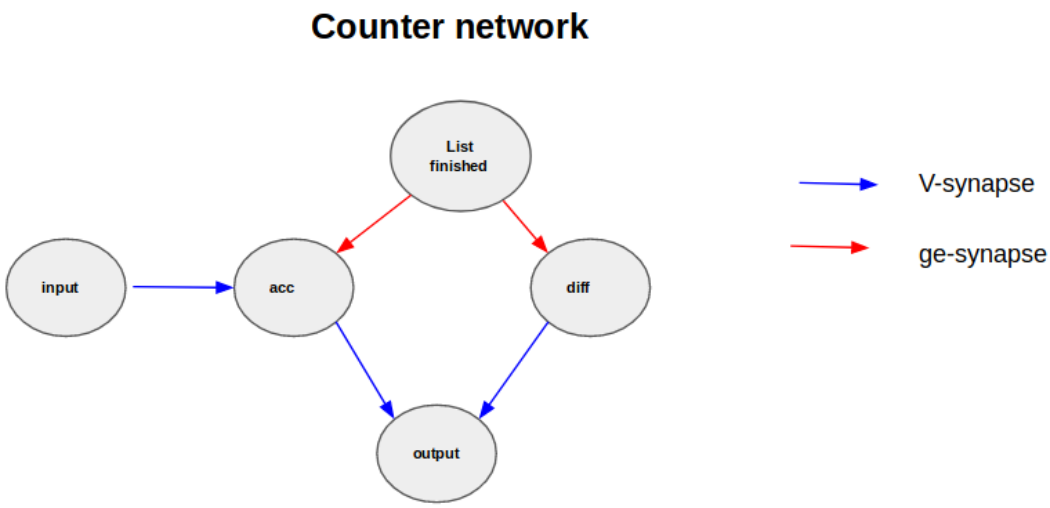
3 Loops

To implement loops, we focused on two examples, counting the length of a list and the sum of its element.

3.1 Spikes train length

We want to compute the number of spikes inputed in the network. When a new spike is inputed, an accumulator is increased by 1 unit (the unit can vary, we used $w_e/1100$).
When we reach the end of the spikes train, a g_e signal is sent to the accumulator and to a neuron with a potential of 0. Both are integrated and the difference of their spiking times is equal to the number of input spikes.The architecture is the following:

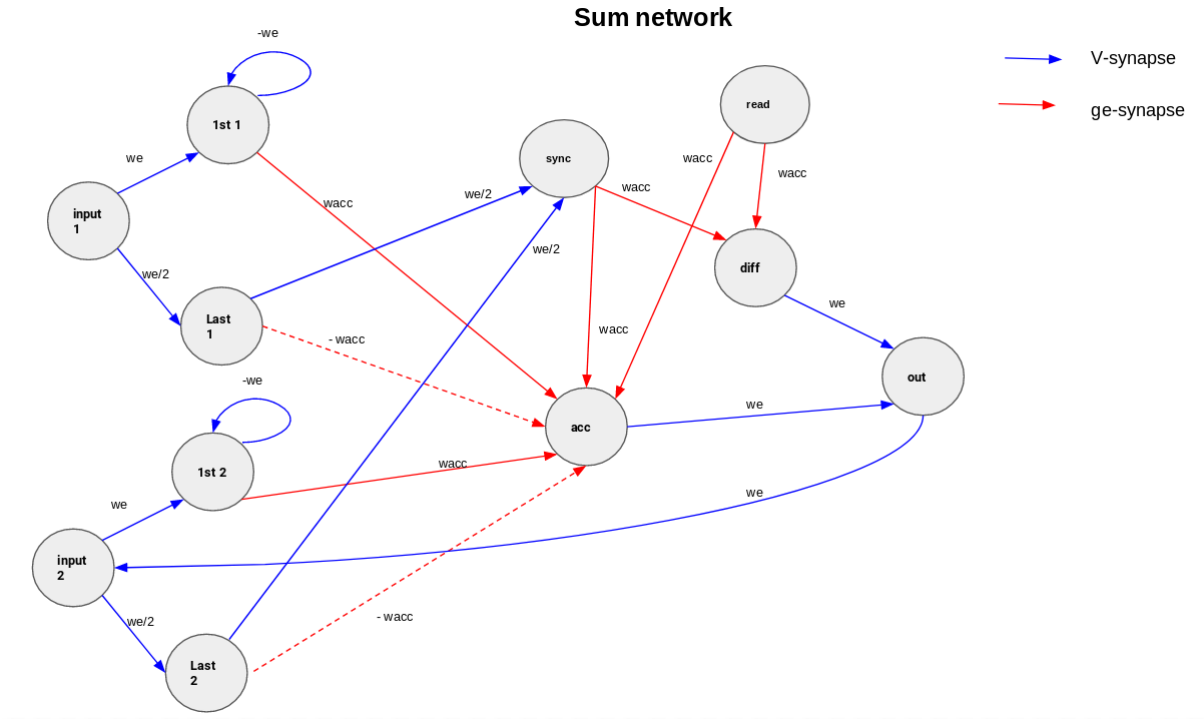
Figure 3: Counter network



3.2 Sum of the elements of a list

Given a sequences a spikes $t_1, t_2, ...t_n$ we want to compute the sum of $(x_k)_{k\geq 1} = t_{2k} - t_{2k-1}$ (i.e the cumulative sums of interspike gaps). The architecture of the sum net is strongly inspired by the linear combination network architecture. One *read* node is added to be able to read the sum once the spikes train is finished. A V-synapse from the *output* to the $input_2$ is added to keep track of the sum. The spikes are fed via the $input_1$.

Figure 4: Sum network



NB: The implementation of the sum is recursive here, given an array a, we have $f(a) = a[0] + f(a[1:])$.

4 Condition

4.1 Modulo

Now, we want to show that the stick paradigm also allows us to implement condition. We will implement a network which implement the following code:

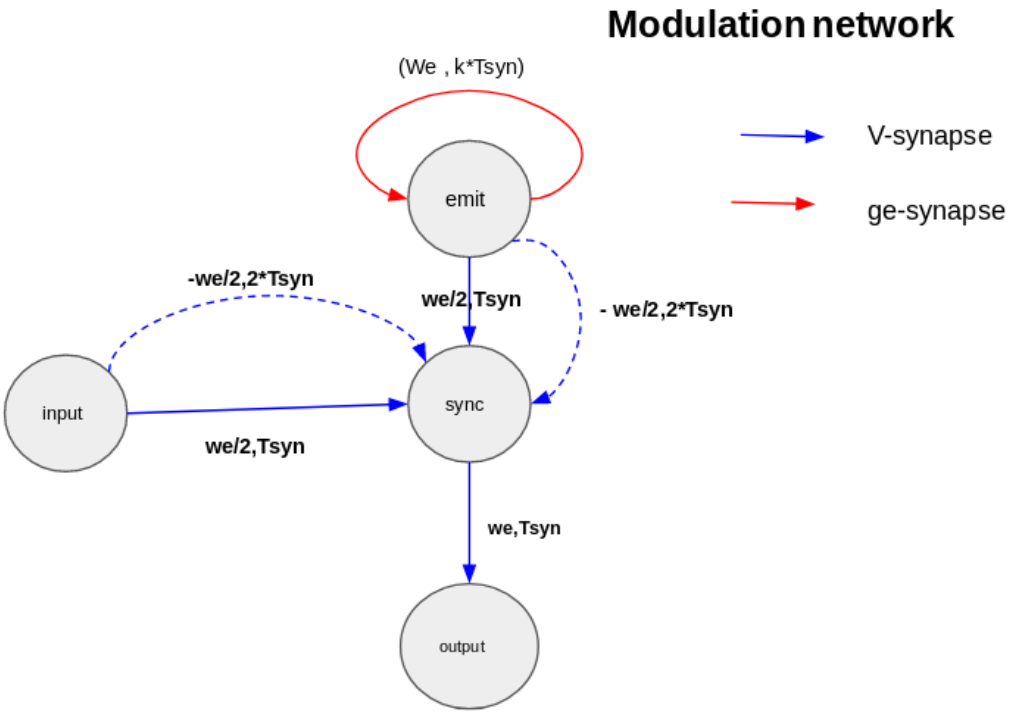
```
def f(x):
  if x mod k == 0:
   yield x
```

The network should only yield the spikes which have been emitted at a time (given an t_init) which is a multiple of k (i.e. t mod k = 0 ms). The network has two main components:

- An emitter, a neuron which emits a signal every k ms.
- A synchronizer, which spikes iff it receives a spike both from the input and the emitter. The synchronizer also needs to be set to zero at each timestep.

The architecture of the network is below.

Figure 5: Modulation network

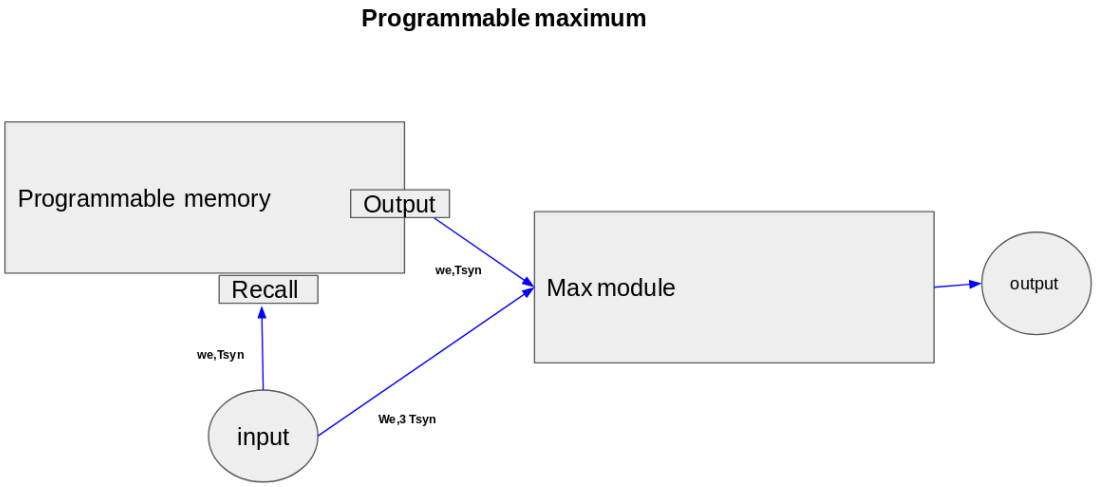


Each k time-step, the emitter spikes and the synchronizer get its potential to $V_t/2$.If at the same time, a spike is inputed, the synchronizer increase its potential to v_t and spikes. Whether it spiked or not, the synchronizer potential is set to zero again on the next time step.

4.2 Programmable maximum

Here is a second example which implements a programmable maximum. A value z is stored in a programmable memory and the network implements $f_z(x) = max(x, z)$ (which can also be seen as an if/else).
The network is simple and composed of a programmable memory and a maximum module. The architecture is the following:

Figure 6: Programmable maximum



5 Recurrent functions

We have already implemented a recurrent function to compute the sum of the elements of a list. We can also use a recurrent function to compute its maximum.

Given a sequences a spikes $t_1, t_2, ...t_n$ we want to compute the maximum of $(x_k)_{k\geq 1} = t_{2k} - t_{2k-1}$ (i.e the maximum of inter-spike gaps) The maximum of the list can be computed by using the programmable maximum and by feeding the output of the maximum to the input of the memory. Then, each time two spikes are inputed with a gap x_t (its an element of the list), the value of the memory is set to $max(x_t, max_{i\leq t-1}(x_i)$. Once the list has completely been fed, the maximum can be read on the programmable memory.

Figure 7: Maximum of a list