Intermediate Lagorce-Benosman Models

Noah Stebbins (`nes2137`)

April 2016

Contents

1 Non-Inverting Memory

This network is based on the inverting memory network. Two accumulator neurons, `acc` and `acc2`, are added to invert the stored value twice.

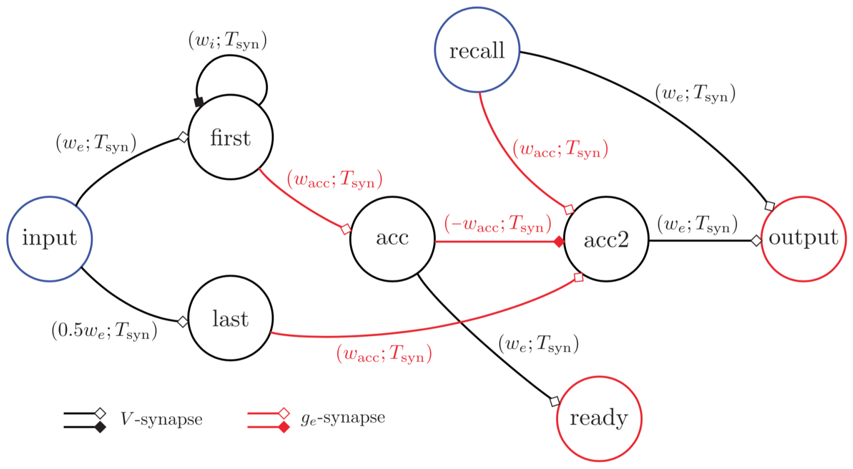


Figure 1: Memory

High Level Description This network outputs a pair of spikes with an interspike interval equivalent to that of two input spikes. There are two `acc` neurons, which allow for integrations beginning at different times. Thus, `acc` completes, causing `acc2`'s integration to halt. The integration is only triggered again when `recall` fires, which causes the final piece of integration in `acc2`; it also causes `output` to spike twice, once at the beginning of this final piece of integration, and once when the threshold is reached.

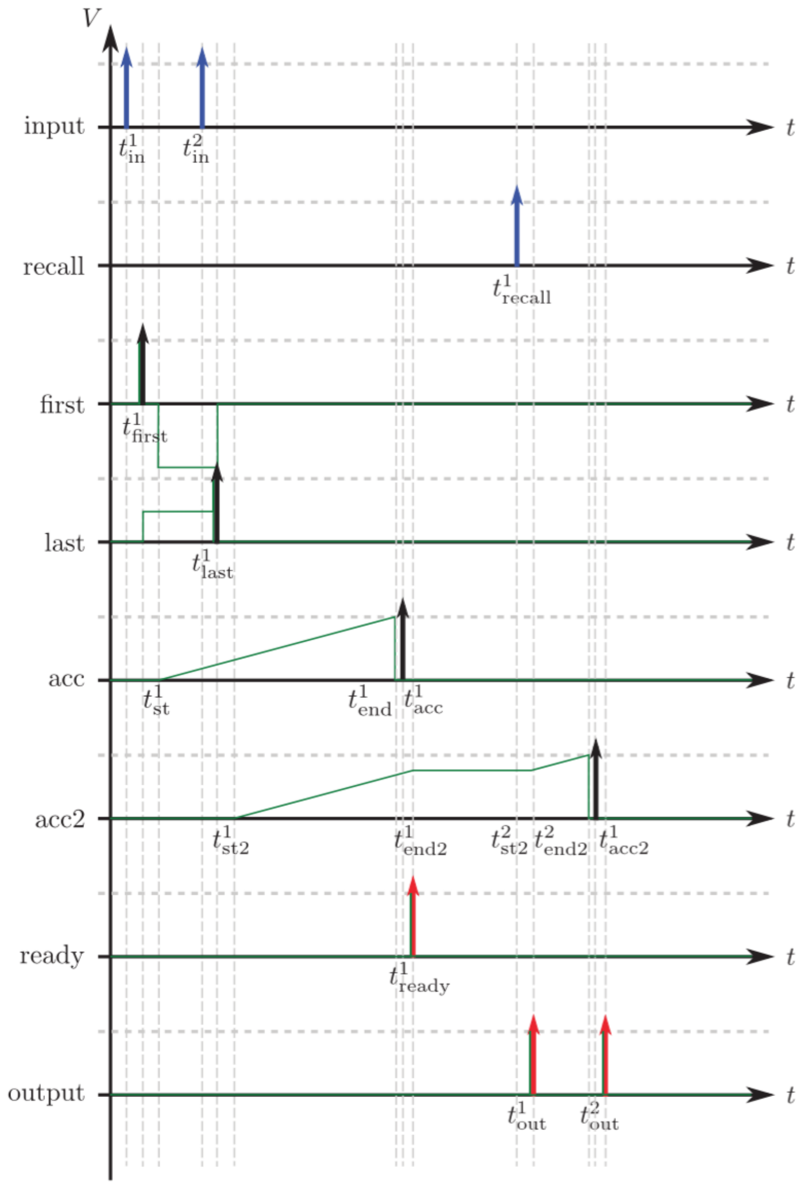


Figure 2: Memory Chronogram

$$t^1_{first} = t^1_{in} + T_{syn} + T_{neu} \tag{1}$$

$$t^1_{last} = t^2_{in} + T_{syn} + T_{neu} \tag{2}$$

$$t^1_{st} = t^1_{first} + T_{syn} = t^1_{in} + 2 \times T_{syn} + T_{neu} \tag{3}$$

Neuron `acc` continues to integrate its w_{acc} input until reaching its threshold.

$$V_t = \frac{w_{acc}}{\tau_m} \times (t^1_{end} - t^1_{st}) \tag{4}$$

$$t^1_{end} = t^1_{st} + T_{max} = t^1_{in} + T_{max} + 2 \times T_{syn} + T_{neu} \tag{5}$$

$$t^1_{acc} = t^1_{end} + T_{neu} = t^1_{in} + T_{max} + 2 \times T_{syn} + 2 \times T_{neu} \tag{6}$$

$$t^1_{end2} = t^1_{acc} + T_{syn} = t^1_{in} + T_{max} + 3 \times T_{syn} + 2 \times T_{neu} \tag{7}$$

Now, observing the equations that govern the behavior of `acc2` for the first round of integration.

$$t^1_{st2} = t^1_{last} + T_{syn} = t^2_{in} + 2 \times T_{syn} + T_{neu} \tag{8}$$

$$V_{sto} = \frac{w_{acc}}{\tau_m} \times (t^1_{end2} - t^1_{st2}) = \frac{w_{acc}}{\tau_m} \times (T_{max} + t^1_{in} - t^2_{in} + T_{syn} + T_{neu}) \tag{9}$$

$$V_{sto} = \frac{w_{acc}}{\tau_m} \times (T_{max} - \Delta T_{in} + T_{syn} + T_{neu}) \tag{10}$$

Now, for the second phase of the model, triggered by `acc2`.

$$t^2_{st2} = t^1_{recall} + T_{syn} \tag{11}$$

$$t^1_{out} = t^1_{recall} + T_{syn} + T_{neu} \tag{12}$$

This second integration of `acc2` finishes when its threshold is reached.

$$V_t = \frac{w_{acc}}{\tau_m} \times (t^2_{end2} - t^2_{st2}) + V_{sto} \tag{13}$$

$$T_{max} = t^2_{end2} - t^1_{recall} - T_{syn} + T_{max} - \Delta T_{in} + T_{syn} + T_{neu} \tag{14}$$

$$t^2_{end2} = t^1_{recall} + \Delta T_{in} - T_{neu} \tag{15}$$

Thus, we get a spike from `acc2`.

$$t^1_{acc2} = t^2_{end2} + T_{neu} = t^1_{recall} + \Delta T_{in} \tag{16}$$

$$t^2_{out} = t^1_{acc2} + T_{syn} + T_{neu} = t^1_{recall} + \Delta T_{in} + T_{syn} + T_{neu} \tag{17}$$

$$\Delta T_{out} = t^2_{out} - t^1_{out} = \Delta T_{in} \tag{18}$$

2 Synchronizer

This network is synchronizing a set of input values so that the first spikes encoding every value on its output side happen at the exact same time. It can be used to resynchronize values before networks requiring this condition. It uses a set of of memory networks to store different input values and a `sync` neuron that recalls all these stored values as soon as the last one of them has been stored.

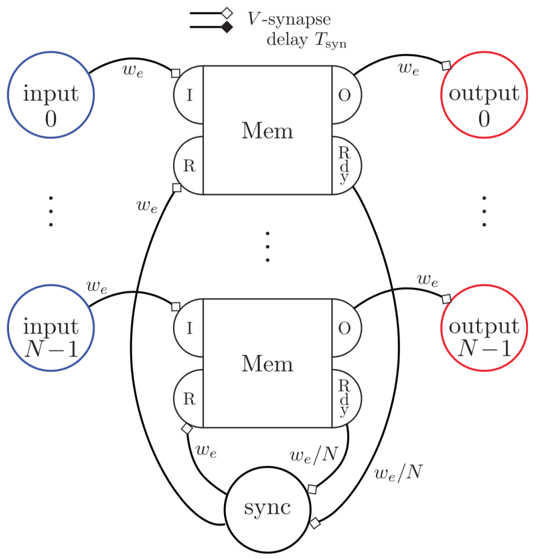


Figure 3: Synchronizer

High Level Description It is implemented using N memory networks. The `sync` neuron keeps track of the number of received inputs. When all the N inputs have been received, this neuron spikes, starting the readout process of the different memories at the same time, thus synchronizing all the outputs.

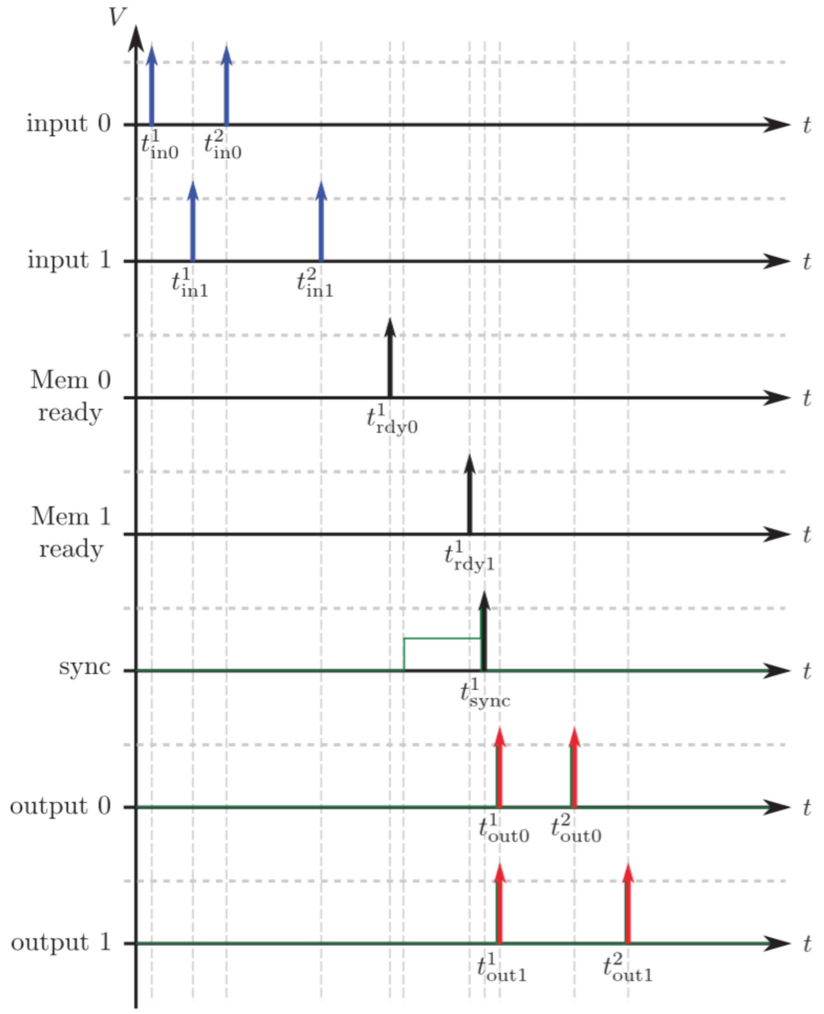


Figure 4: Synchronizer Chronogram

This model is extremely straightforward. The chronogram above has $N = 2$. Every time an internal memory has finished storing an input, its *Rdy* output activates the `sync` neuron with a weight $\frac{w_e}{N}$. Thus, after i inputs have been presented, `sync`'s membrane potential is as follows.

$$V^i_{sync} = \frac{i}{N} \times V_t \tag{19}$$

3 Subtractor (simple)

The simplest iteration of the subtraction network is depicted below and will be expanded upon in the subsequent section. This network computes the difference between `input1` and `input2` and directs the output depending on its sign to either the `output+` or `output-` neuron. If the two inputs are synchronized, the difference between the two is directly given by the interspike between the two second encoding spikes. This is the information that `sync1` and `sync2` neurons are extracting.

When the output sign is known, `sync1` or `sync2` inhibits the pathway to the wrong output neuron such that the output spikes are directed to the correct one.

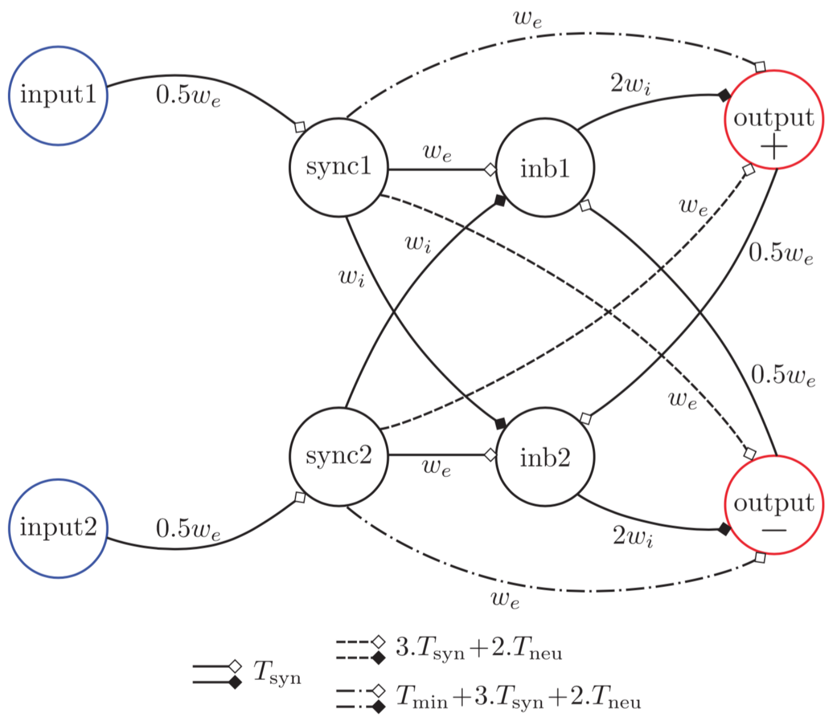


Figure 5: Subtractor (simple)

The subtractor network receives two different inputs (a pair of spikes) from each input neurons `input1` (t^1_{in1} and t^2_{in1}) and `input2` (t^1_{in2} and t^2_{in2}), such that $\Delta T_{in1} = t^2_{in1} - t^1_{in1}$ and $\Delta T_{in2} = t^2_{in2} - t^1_{in2}$ encode its two inputs. Let us consider the case where the first input (`input1`) is larger than the second one. Assuming the inputs to be synchronized, we have

$$t^1_{in1} = t^1_{in2} \tag{20}$$

When `input1` and `input2` are triggered by the first encoding spikes of each input, they activate the `sync1` and `sync2` neurons such that their membrane potentials are now set to $\frac{V_t}{2}$. When the second encoding spike of the smaller input (here, `input2`) is received, the activation from `input2` to `sync2` is sufficient to trigger a spike at time,

$$t^1_{sync2} = t^2_{in2} + T_{syn} + T_{neu} \tag{21}$$

This spike inhibits the `inb1` neuron after a time T_{syn}, moving its membrane potential to $-V_t$ and because the sign of the output is now known, it triggers an output spike on `output+` at time,

$$t^1_{out} = t^1_{sync2} + 3 \times T_{syn} + 2 \times T_{neu} + T_{neu} = T^2_{in2} + 4 \times T_{syn} + 4 \times T_{neu} \tag{22}$$

It also contributes to `output-`'s membrane potential with an activation of w_e at time $t^1_{sync2} + T_{min} + 3 \times T_{syn} + 2 \times T_{neu}$. But before this contribution reaches `output-`, the `inb2` neuron is triggered and produces a spike at time,

$$t^1_{inb2} = t^1_{sync2} + T_{syn} + T_{neu} \tag{23}$$

This inevitably inhibits `output-` with weight $2w_i$ at time $t^1_{inb2} + T_{syn} = t^1_{sync2} + T_{syn} + T_{neu}$. This inhibition thus happens before the direct excitation from `sync2` which leads to `output-` not emitting a spike and its membrane potential to be set to $-V_t$ after receiving these two spikes.

When the second encoding spike of the larger input is received, the spike from `input1` triggers a spike from `sync1` at time,

$$t^1_{sync1} = t^2_{in1} + T_{syn} + T_{neu} \tag{24}$$

This spike leads to the inhibition of `inb2` to a membrane potential of $-V_t$ and an excitation of `inb1` back to its reset potential. It also activates `output+` back to its reset potential and triggers an output spike at time,

$$t^2_{out} = t^1_{sync1} + T_{min} + 3 \times T_{syn} + 2 \times T_{neu} + T_{neu} = t^2_{in1} + T_{min} + 4 \times T_{syn} + 4 \times T_{neu} \tag{25}$$

We thus get a positive output as expected and an output value,

$$\Delta T_{out} = t^2_{out} - t^1_{out} = T_{min} + t^2_{in1} - t^2_{in2} \tag{26}$$

$$\Delta T_{out} = T_{min} + (t^2_{in1} - t^1_{in1}) - (t^2_{in2} - t^1_{in2}) = T_{min} + (\Delta T_{in1} - \Delta T_{in2}) \tag{27}$$

$$\Delta T_{out} = T_{min} + (\Delta T_{in1} - T_{min}) - (\Delta T_{in2} - T_{min}) \tag{28}$$

Knowing that for each value x, we encode it as a time interval $f(x) = T_{min} + x \times T_{cod}$, we have,

$$\Delta T_{out} = T_{min} + x_{in1} \times T_{cod} - x_{in2} \times T_{cod} \tag{29}$$

$$\Delta T_{out} = T_{min} + (x_{in1} - x_{in2}) \times T_{cod} \tag{30}$$

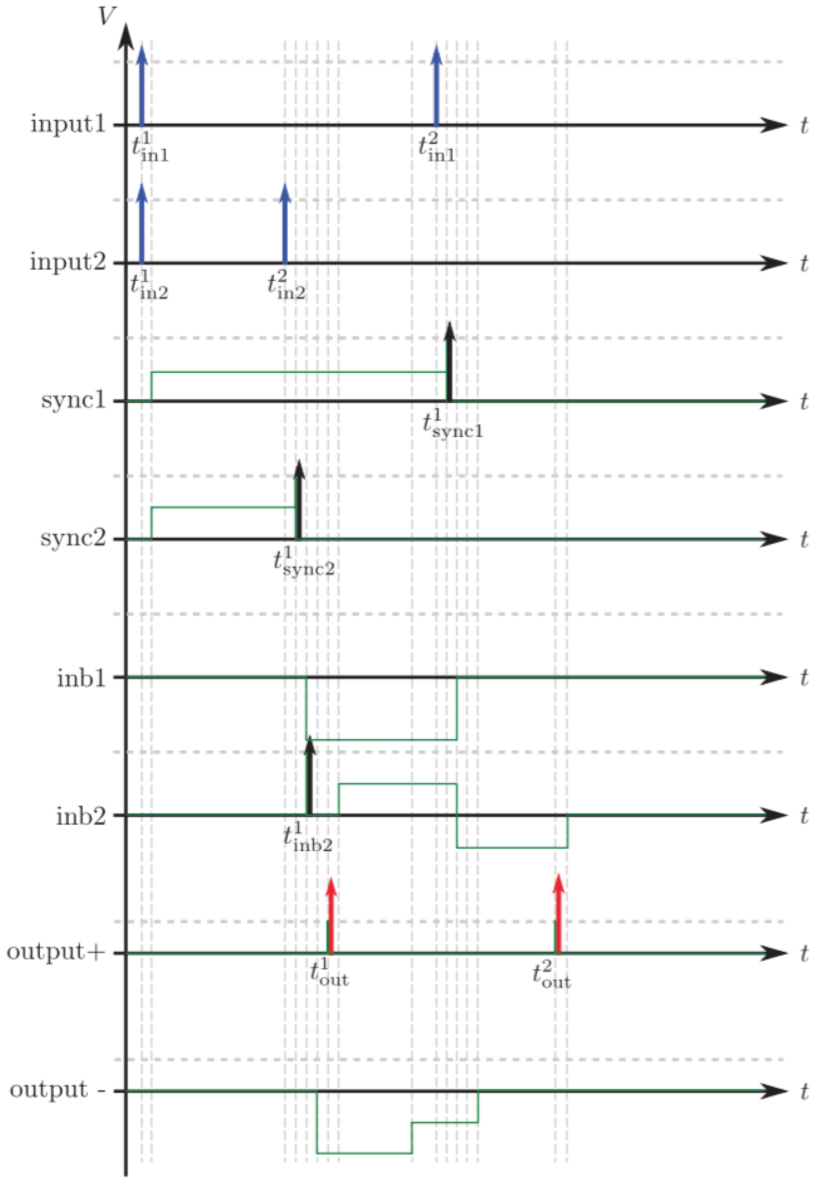


Figure 6: Subtractor Chronogram

Understanding this Model This is a highly efficient yet complex model. No neuron nor synapse is wasted here and is expressed compactly. I will explicitly look at the `inb` neurons here, since they are the crux of this model along with the `sync` and `output` pairs. In the case of the `inb` neurons, the first one to get fired will "burn" it's restoring V_t, since it already fired. Thus, *the edges from the other output neuron back to this inb neuron are necessary to allow this neuron to restore, since you can take advantage of the knowledge that their are guaranteed two spikes on that particular output to weight that specific edge by half.* Obviously, the inhibition arises because you want to allow the second spike to happen on that corresponding output neuron. In this case, I am referring to the synapse between `sync` and its opposite `inb` counterpart, which governs output spiking.

4 Subtractor (full)

In this network, we add the `zero` neuron and its connections. In the previous simple implementation, when both inputs are equal, the two parallel pathways of the network are triggered at the same time and the lateral inhibition has no time to select a winning pathway. In that case, the output is emitted on both `output+` and `output-`, which can be problematic for the following networks expecting only one of the two pathways to be activated.

To solve this problem, we add the `zero` neuron with a set of fast synaptic connections. They allow the detection of the case of equality between the two inputs. In that case, the `output-` pathway is quickly inhibited to produce spikes coding for the zero output only on the `output+` neuron.

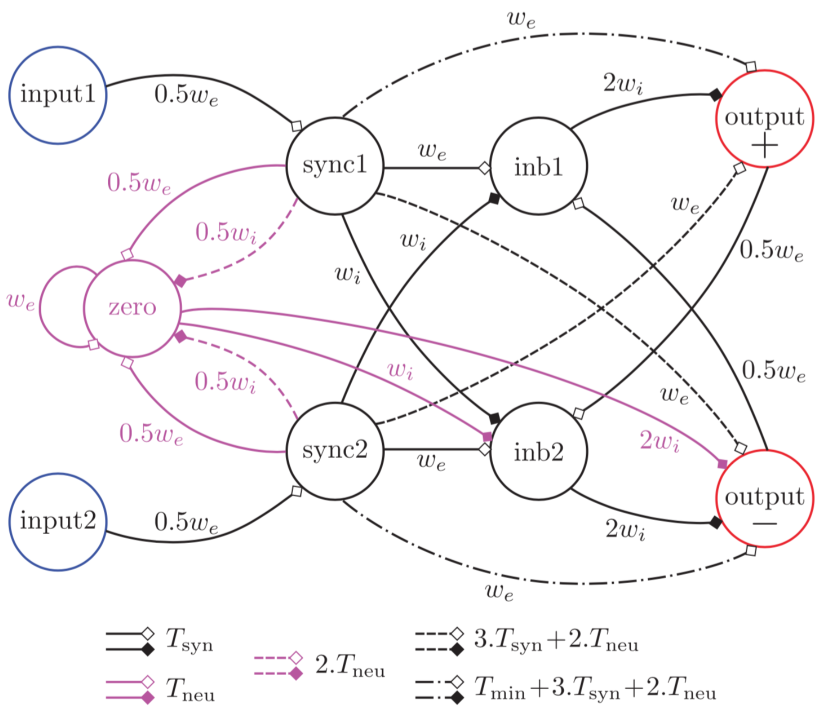


Figure 7: Subtractor (full)

5 Linear Combination

The figure below computes the linear combination of N signed inputs with arbitrary coefficients $\alpha_0 \dots \alpha_{N-1}$,

$$s = \sum_{i=0}^{N-1} \alpha_i x_i \tag{31}$$

where $x_i, i \in 0, \dots, N-1$ are the different inputs of the network.

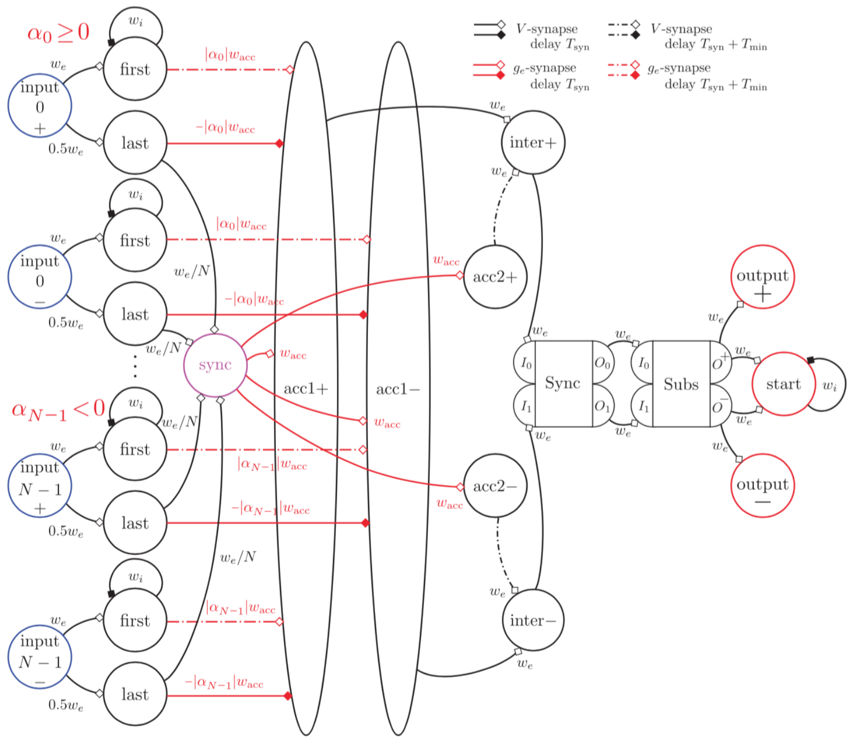


Figure 8: Linear Combination

High Level Description The linear combination network uses the same principle as the memory network to store values in an accumulator. To implement the coefficients of the sum, we multiply the synaptic weight of the accumulation current by the coefficient corresponding to the input. To handle the signs of the inputs and coefficients, we use two accumulators. The first one is storing intermediate results, which are positives (i.e. the sign of the input is the same as the one of its associated coefficient), while the second stores negative values (i.e. the sign of the input is different from the one of its associated coefficient).

When all the inputs have been fed into the network, the `sync` network is triggered, causing the readout process of these accumulators. Their content are inverted as for the memory network and then synced before entering a subtractor network. This last network computes the difference between the positive and negative contributions of the different inputs and produces a signed output. A `start` neuron is then triggered to spike to indicate that the computation has ended. This signal can be used to trigger further networks.

Detailed Explanation The first part of the linear combination presented in the figure can be decomposed in a series of simpler circuits to what has been presented before for the inverting memory. For each of the N inputs, one branch is managing positive inputs while the other is managing negative inputs (only one of these two can be activated in any computation). The architecture of each of these branches can be seen as inverting memory storing a value in either `acc1+` or `acc1-`. The targeted accumulator is chosen depending on the sign of the input and the sign of its associated coefficient α_i to represent the sign of the input's contribution to the result. `acc1+` is accumulating all positive contributions (positive inputs with positive coefficients and negative inputs with negative coefficients). `acc1-` is accumulating all negative contributions (negative inputs with positive coefficients and positive inputs with negative coefficients). With the same reasoning leading to equation A.7, we can get the contribution of an input i to its associated accumulator, if ΔT_{in}^i is the input interspike associated with the considered input,

$$V_{sto}^i = |\alpha_i| \frac{w_{acc}}{\tau_m} (\Delta T_{in}^i - T_{min}) \tag{32}$$

The integration in the accumulators being linear, we obtain the membrane potential stored in the two accumulators after all inputs have been fed into the network,

$$V_{sto}^{acc1+} = \sum_{i \in X^+} V_{sto}^i = \sum_{i \in X^+} |\alpha_i| \frac{w_{acc}}{\tau_m} (\Delta T_{in}^i - T_{min}) \tag{33}$$

$$V_{sto}^{acc1-} = \sum_{i \in X^-} V_{sto}^i = \sum_{i \in X^-} |\alpha_i| \frac{w_{acc}}{\tau_m} (\Delta T_{in}^i - T_{min}) \tag{34}$$

where X^+ is the set of inputs contributing positively to the output and X^- is the set of inputs contributing negatively to the output. When the N inputs have been fed into the network, the `sync` neuron finally receives enough excitation to produce a spike at time t_{sync}^1. This spike triggers the readout process of `acc1+` and `acc1-` and, at the same time, starts integrating in neurons `acc2+` and `acc2-`. This process is similar to the one used in the memory network. If we consider the positive accumulator, we obtain spikes from `acc1+` and `acc2+` at time t_{acc1+}^1 and t_{acc2+}^1, respectively, with the following conditions, where $t_{st}^1 = t_{sync}^1 + T_{syn}$ is the time at which the integration begins.

$$V_t = \frac{w_{acc}}{\tau_m}(t_{acc1+} - t_{st}^1) + V_{sto}^{acc1+} \tag{35}$$

and

$$V_t = \frac{w_{acc}}{\tau_m}(t_{acc2+} - t_{st}^1) \tag{36}$$

By definition of w_{acc}, we get,

$$t_{acc2+} = t_{st}^1 + T_{max} \tag{37}$$

and

$$T_{max} = t_{acc1+} - t_{st}^1 + \sum_{i \in X^+} |\alpha_i|(\Delta T_{in}^i - T_{min}) \tag{38}$$

$$t_{acc1+} = t_{st}^1 + T_{max} - \sum_{i \in X^+} |\alpha_i|(\Delta T_{in}^i - T_{min}) \tag{39}$$

Neuron `inter+` is thus producing two spikes with an interspike ΔT_{inter}^+ such that,

$$\begin{aligned} \Delta T_{inter}^+ &= t_{acc2+} + T_{min} + T_{syn} + T_{neu} - (t_{acc1+} + T_{syn} + T_{neu}) \\ &= \sum_{i \in X^+} |\alpha_i|(\Delta T_{in}^i - T_{min}) + T_{min} \end{aligned} \tag{40}$$

The same reasoning on `acc1-` and `acc2-` leads to a pair of spikes on neuron `inter-` with an interspike ΔT_{inter}^-,

$$\Delta T_{inter}^- = \sum_{i \in X^-} |\alpha_i|(\Delta T_{in}^i - T_{min}) + T_{min} \tag{41}$$

These two values are then synchronized by a synchronizer network described previously and subtracted from one another such that the output is,

$$\begin{aligned} \Delta T_{out} &= \Delta T^{+}_{inter} - \Delta T^{-}_{inter} + T_{min} \\ &= \sum_{i\in X^{+}} |\alpha_i|(\Delta T^{i}_{in} - T_{min}) - \sum_{i\in X^{-}} |\alpha_i|(\Delta T^{i}_{in} - T_{min}) + T_{min} \\ &= \sum_{i=0}^{N-1} \epsilon_i \alpha_i (\Delta T^{i}_{in} - T_{min}) + T_{min} \end{aligned} \tag{42}$$

where ϵ_i is +1 if input i is positive and -1 otherwise. This is the expected result of the linear combination.

6 Integration

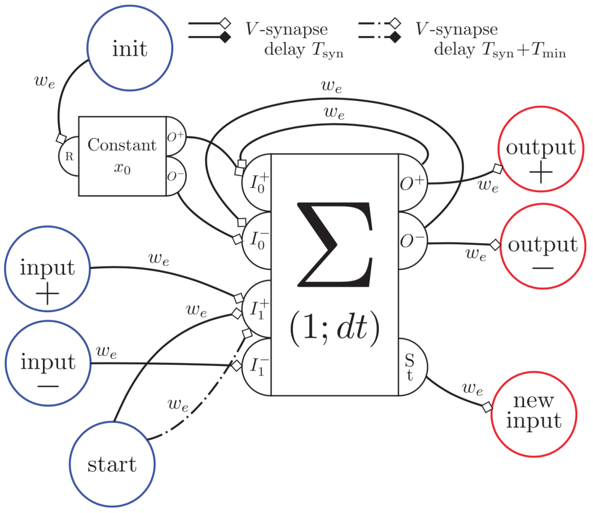


Figure 9: Integrator

High Level Description This network integrates an input over time. The `init` and `start` neurons are used to initialize the inner state of the integrator. The network is then driven by its input. Every time an input is received, it is integrated and added to the latest value of the output. When a new value is output, the `new_input` neuron spikes, requesting a new input.

Detailed Explanation The integrator network allows reconstruction of a signal from its derivative fed into its input. It is using a multiplier and an accumulation network with a linear combination network. The output of this accumulator network is looped into its first input with a unit gain. The input, composed of `input+` and `input-`, is fed into this accumulator with a gain dt corresponding to the chosen integration time step. Each time an output is produced on `output+` and `output-`, the indicator neuron `new_input` is triggered to notify that the integrator is ready to receive its next input. This system is thus driven by its input. Every time an input is provided, the corresponding output is computed. Two auxiliary input neurons are also provided. The `init` neuron loads the integrator with its initial value. This allows the internal state of the integrator (through the linear combination network) to be set to an initialization value. The `start` neuron feeds a zero into the input of the integrator, thus computing its first output.

7 First-Order System

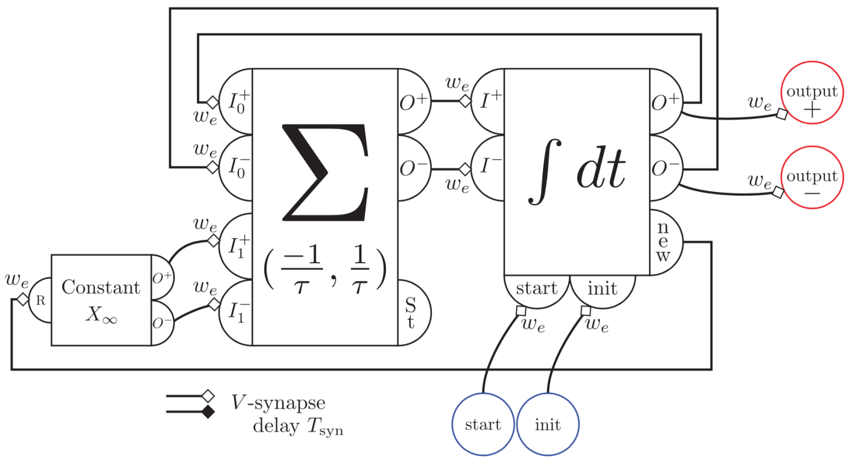


Figure 10: First-Order Differential System

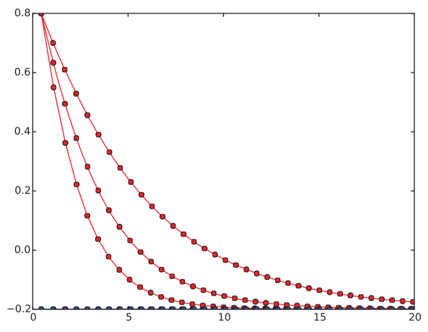
A first-order differential system solves the following equation:

$$\tau \frac{dX}{dt} + X(t) = X_\infty \tag{43}$$

We implement this network with three of the networks described in the previous section:

- a constant network is providing the input X_∞ to the system
- a linear combination network is computing $\frac{dX}{dt}$
- an integrator network is computing X from its derivative

The `init` and `start` neurons initialize and start the integration process. `init` has to be triggered before the integration process can take place to load the initial value of the integrator network. `start` has to be triggered to output the first value from the integrator network. When an output is provided by the integrator network, the constant network is activated using the `new_input` neuron of the integrator, hence feeding two values into the linear combination computing the new derivative of the output. With the implementations presented in the previous sections, this network requires 118 neurons. Results of its simulation with different set of parameters for τ and X_∞ and for $dt = 0.5$ are presented in the following figure.


(a) Different values of τ

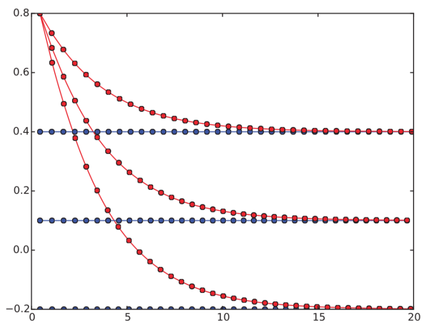

(b) Different values of X_∞

Figure 11: Neural Implementation of a First-Order Filter (Different Parameters)