Department of Electrical and Electronic Engineering
Imperial College London
Final Year Project Interim Report

Randomised time step methods for neural network simulation

Author: Fabio Deo
Course: EIE4
CID: 01338063
Academic Supervisor: Dr. Daniel Goodman
Second Marker: Dr. Krystian Mikolajszyk

Contents

1 Introduction

1.1 Spike-Based Neural Processing

Spiking Neural Networks (SNN) make use of bio-mimetic algorithms to attempt to replicate more closely the biological learning process undertaken by the human brain. In classical Artificial Neural Networks (ANN) neurons forward information to one another at each iteration, so that they can learn by processing this flow of data and update their weight through back-propagation. However, the human brain consists of interconnected neurons that communicate through electrical impulses called action potentials, which are triggered only when certain environmental conditions arise. In particular, each neuron has a membrane voltage, which is dynamically modified by the incoming electrical impulses. When the voltage exceeds a threshold called firing threshold, a spike is triggered and the membrane voltage is immediately reset to its initial value. The connections among neurons that allow them to communicate to each other are the synapses. This spiking model is commonly known as Integrate and Fire (IF) model. Leaking IF models are also extremely common, as they also mimic the leakage in membrane voltage present in biological neurons. Their behaviour over time follows a pattern similar to the one visualised in Figure 1.

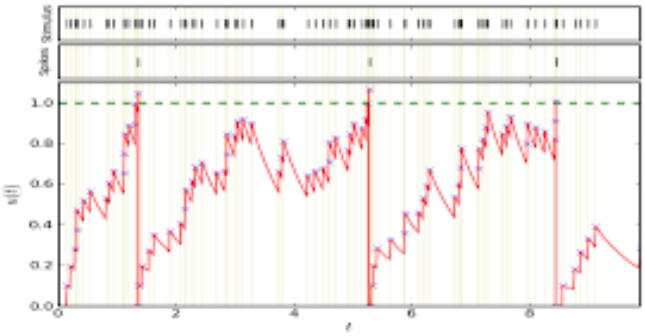

Figure 1: Leaking Integrate and Fire Neuron Behaviour from (Kabasov et al. [2]).

1.2 The BRIAN Simulator

One of the most widely used simulators for spiking neural network simulations is BRIAN. BRIAN is a free, open source clock driven simulator (Goodman, Brette [3]). It can easily be downloaded, imported and used in the Python programming language. Its clock-driven behaviour implies that a time-step dt is defined to discretise events happening during the simulation. Neurons behaviour is mapped using differential equations, which can be easily defined in BRIAN and automatically interpreted by the tool. BRIAN does not currently support a randomised time step methods to execute neural network simulations, therefore one of the objectives of the project is to integrate such feature in the simulator, if proven to be computationally more efficient than known alternatives, as further discussed in Chapter 5.

2 Background

2.1 Time-Step Analysis

The research of a suitable time-step is fundamental, as numerical integrations are the main bottleneck in large network simulations efficiency. This is because large dt lead to simulation artifacts, as observed (Hansel, Mato et al. [4]). In simulations adopting large time-steps, neurons eventually stabilised to asymptotic states. This behaviour has been identified as a consequence of the inaccuracy in resetting the potential following a spike. Small differences amongst various neurons are eventually discarded, resulting in a synchronisation of their membrane potential curves. This issue can be mitigated by performing denser simulation, i.e. reducing the time-step magnitude. Nonetheless, such a solution often becomes nonviable due to a large increase in computational complexity.

2.2 Efficient Time-Stepping Numerical Methods

The easiest approach to define a more efficient time-stepping function is to build more accurate time integration schemes, leveraging higher order numerical methods to achieve better approximations. While this course of actions can delay the origin of synchronisation artifacts, it retains an intrinsic error margin, derivable from the synaptic-induced conductance equation, as explained in (Shelley, et al. [5]). Figure 2, shows to what extent the new numerical methods investigated by Shelley and Tao overcompensate the choice of a larger dt.

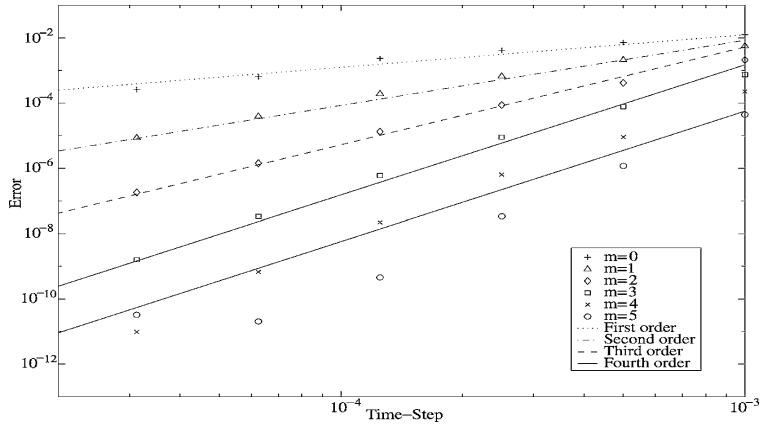

Figure 2: Comparison of Numerical Methods Performance from (Shelley, et al. [5]).

(Hansel, Mato et al. [4]) performed a similar study on a model that will be replicated in this project for testing purposes. In the interest of brevity and clearness, this report will further address it as as the "original model". In the original study, second order Runge-Kutta with interpolation are proven to be at least a degree of magnitude faster than other numerical methods such as Standard RK2, Standard Euler and Euler with interpolation. Therefore, although higher-order schemes do not provide a full solution to synchronisation artifacts in neuronal networks, they radically improve the performance of the simulation. Without them, the computational time needed would rapidly become unfeasible as the complexity of the model architecture increases.

2.3 Variable Time-Stepping Methods

This project aims to investigate if varying the time step throughout the simulation mitigates the synchronisation activity introduced in Chapter 2 . This theory has been explored by (William Lytton, Micheal Hines [6]). They defined a new variable time-stepping integration method called *lvardt*, that leverages the differences in spiking activity to tailor neuron-specific time-steps. Neurons that show an active synaptic behaviour are re-initialised with smaller time-steps upon firing, while the ones that undergo longer idle periods are progressively re-initialised with a larger dt. This dynamic modification of each neuron's individual time-step integrator produces very good results in networks with significant synaptic activity, as the computational power spared on idle neurons is used on denser numerical iterations on highly active neurons, improving the overall results and reducing the risk of synchronisation artifacts. However, it must be taken into account the large overhead complexity carried by the re-instantiation of the time-steps based on each neurons recent behavior and the computational cost of having a integrator synchroniser to make sure that state-variables stay up-to-date throughout the simulation. Nonetheless, the results presented by (William Lytton, Micheal Hines) suggest that this innovative variable time-stepping method outperforms fixed alternatives for reasonably complex and active networks. The network analysed in the research has been coded in NEURON, a spiking neural network simulator that provides variable time-step integration functionalities (M. L. Hines, N. T. Carnevale [11]).

3 Test Model

3.1 Model Architecture

For this project, the original model will be replicated in BRIAN2 compatible code and used for efficiency and accuracy analysis. The original network encompasses 128 identical excitatory neurons coupled all-to-all. The following differential equation describes the evolution of the membrane potential over time:

$$C * \frac{dV}{dt} = g_l(V - V_l) + I_{syn}(t) + I_0 \tag{3.1}$$

Table 1 summarises the set of parameters used in the original model. Some of them have not been specified by (Hansel, Mato et al.). This lack of information can cause small differences in the future in terms of results between the original model and the Brian2 replicated version. Therefore, the choice of such parameters will be appropriately discussed and tuned to try to reproduce as similar results as possible.

Parameter	Description	**Value**
N	Number of neurons	128
g_l	Conductance of the voltage-independent leak current	0.1 $mSiemens/cm^2$
V_l	Reversal potential of the voltage-independent leak current	-60 $mVolt$
V_{rest}	Resting membrane potential	-60 $mVolt$
C	Membrane capacitance	1 $\mu Farad/cm^2$
τ	Passive membrane time constant	$C/g_l = 10$ $msecond$
τ_1	First time constant for the biexponential synaptic current	3 $msecond$
τ_2	Second time constant for the biexponential synaptic current	1 $msecond$
θ	Membrane potential threshold	-40 $mVolt$
I_0	Constant external drive	non specified

Table 1: Network Parameters

A neuron triggers an impulse when its membrane voltage V reaches the membrane potential threshold θ. After the spike, the membrane voltage of the spiking neuron is instantaneously reset to the resting voltage as follows:

$$V(t_0^+) = V_{rest} \quad if \quad V(T_0^-) = \theta \tag{3.2}$$

No refractory period is considered in the original model definition; this design choice will be retained in the BRIAN Simulator replication. Furthermore, the synaptic current $I_{syn}(t)$ evolves over time as modeled by the following equation:

$$I_{syn}(t) = \frac{\overline{I_{syn}}}{N} \sum_{neurons} \sum_{spikes} f(t - t_{spike}) \tag{3.3}$$

where:

$$f(t) = \frac{1}{\tau_1 - \tau_2}(e^{-\frac{t}{\tau_1}} - e^{-\frac{t}{\tau_2}}) \tag{3.4}$$

Section 3.3 further explains how to translate the model architecture in Brian2 compatible differential equations that can be fed as input to the *NeuronGroup* and *Synapses* classes to generate the model in BRIAN2 code.

3.2 Neurons Initialisation

In the original model at $t = 0$ all synaptic currents I_syn are zero and the neurons voltages are chosen in the range $-60\ mV$ to $-40\ mV$ using:

$$V_i(0) = \tau I_0 + (V_{rest} - \tau I_0)e^{-c\frac{(i-1)T}{N\tau}} \tag{3.5}$$

where $i = 1, ..., N$, $0 < c < 1$ and

$$T = -\tau * ln(\frac{\tau I_0 - \theta}{\tau I_0 - V_{rest}}) \tag{3.6}$$

T defines how long the neuron will take to fire without interactions. Finally the coefficient c is the degree of initial synchrony. When c is set to 1, the membrane voltages of the neurons will spread out following a uniform distribution in the range $-60\ mV$ to $-40\ mV$.

3.3 BRIAN Translation

To replicate the model architecture presented in Section 3.1 the differential equations that describe both the membrane voltage and synaptic current must be derived from the original equations. The former is already given by Hansel, Mato et al. in Equation 3.1 and needs very little manipulation to be transformed in a Brian2-compliant form. Dividing both sides by the membrane capacitance C yields the following:

$$\frac{dV}{dt} = \frac{1}{C}[g_l(V - V_l) + I_{syn}(t) + I_0] \tag{3.7}$$

This final form complies to the standards accepted by Brian2 interpreter, hence can be included in the code in string form as follows (assuming that the other parameters used in the equation have been previously defined):

```
eqs = '''
dV/dt = (-gl*(V - Vl) + I_syn + Io) / C : volt
'''
```

Listing 1: Simple example of how to define differential equations in Brian2

The synaptic current, instead, in the original paper is defined as shown in Equation 3.3. Hence, it must be transformed from integral form to ODE form, before being fed into Brian2 classes. By recognising the current used in the original model as a bi-exponential synaptic current, it is trivial to leverage Brian2 documentation to transform Equation 3.3 and 3.4 into the following pair of differential equations:

$$\frac{dI_{syn}}{dt} = \frac{(I_{peak} * s - I_{syn})}{\tau_1} \tag{3.8}$$

$$\frac{ds}{dt} = \frac{-s}{\tau_2} \tag{3.9}$$

where the scalar I_{peak} is used to rescale the curve to exactly replicate the behaviour of a single synaptic current. Figure 3 shows a single synaptic impulse as defined by equation 3.4 as a function of time. To find the current peak, equate the first derivative of the synaptic current to zero as follows:

$$\frac{df}{dt} = \frac{1}{\tau_1 - \tau_2}(-\frac{e^{-\frac{t}{\tau_1}}}{\tau_1} + \frac{e^{-\frac{t}{\tau_2}}}{\tau_2}) = 0 \tag{3.10}$$

that, by substituting $\tau_1 = 3\ ms$ and $\tau_2 = 1\ ms$, yields to:

$$e^{\frac{2000t}{3}} - 3 = 0 \tag{3.11}$$

therefore:

$$t_{peak} = \frac{3 * ln(3)}{2000} = 1.648\ ms \tag{3.12}$$

At $t = t_{peak}$, the synaptic current of a single neuron $f(t_{peak})$ reaches the value of 192.45.

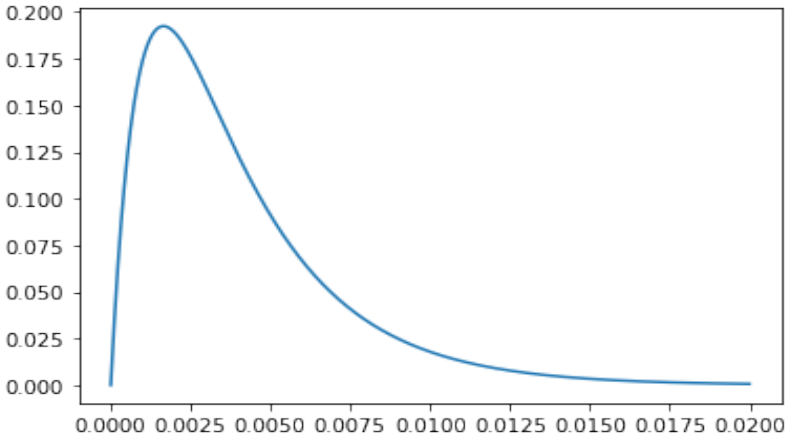

Figure 3: Single neuron synaptic bi-exponential current.

4 Experiment Methodology

Fixed time-stepping methods, as the name suggest, consist of a global static time-step, that will not change throughout the simulation of the network. Fixed time-steps are extremely easy to implement (in Brian2 just reassign the variable *defaultclock.dt*) thus they do not carry any computational overhead. However due to their lack of flexibility, fixed *dt* methods are also prone to incur in the synchronisation artifacts discussed in Section 2.2. For this reason a more complex variable approach has been investigated by (William Lytton, Micheal Hines [6]). As introduced in Section 2.3, its significant performance improvement has been attributed to the lack of unnecessary numerical integration computations for largely inactive neurons. This computational power "gain" is dynamically redistributed to fast-spiking neurons via initialising them with smaller time-steps, hence increasing the calculation accuracy. This project will not address such a complex methodology in choosing the time-stepping size, instead focusing on researching the effects of random time-step allocation on the network's performance. This design choice comes from the assumption that fixed time-step methods greatly suffer from synchronisation artifacts due to the fact that all neurons are updated in a predictable and synchronous fashion. Therefore, introducing randomness should result in more synchrony-averting neurons. Two randomised time-step methods will be integrated in the replicated version of the original model and their effects on the simulation performance will be thoroughly analysed. The two chosen designs for randomised time-step method analysis are presented in Section 4.1 and 4.2 respectively.

4.1 Shared Random Time Stepping

The first approach to be investigated is shared random time stepping (SRTS). In this method the *dt* remains a global parameter shared by all neurons, but it is recomputed randomly at every iteration. Figure 4 visualises an example of neurons sharing a dynamically and randomly generated time-step.

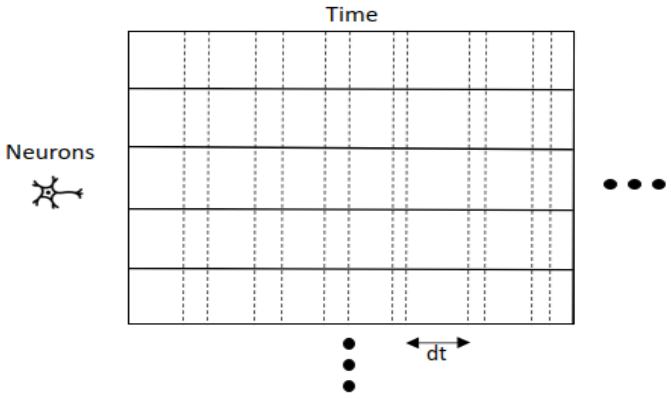

Figure 4: Shared random time-stepping method visualisation.

This method does not carry large computational overheads, given that it encompasses just one extra random number calculation each iteration. However, the lack of extra complexity makes it theoretically less likely to produce significant improvements in the network performance, as numerical integrations are still performed at the same exact moment in time for each neuron, hence retaining the risk of eventual synchronisations among neurons.

4.2 Neuron-Specific Random Time-Stepping

The second approach that will be tested is a neuron-specific randomised time-stepping method (NSRTS). Oppositely to SRTS, this proposition carries a significant complexity overhead of $O(N)$ where N is the number of neurons in the network. This is because a new random time-step must be computed for each neuron on every iteration. However, this additional layer of complexity can theoretically bring significant improvements in avoiding synchronisation artifacts during simulation. In fact, in this case numerical integrations on the membrane voltage and the synaptic current of each neuron are performed asynchronously. Should this method have an important effect on avoiding synchronisation, the random dt could be increased in magnitude, reducing the simulation density, hence its computational cost. If the latter produces an efficiency boost more significant than the initial complexity overhead, the overall performance of the network improves. Figure 5 visualises an example of neuron-specific random time-stepping network.

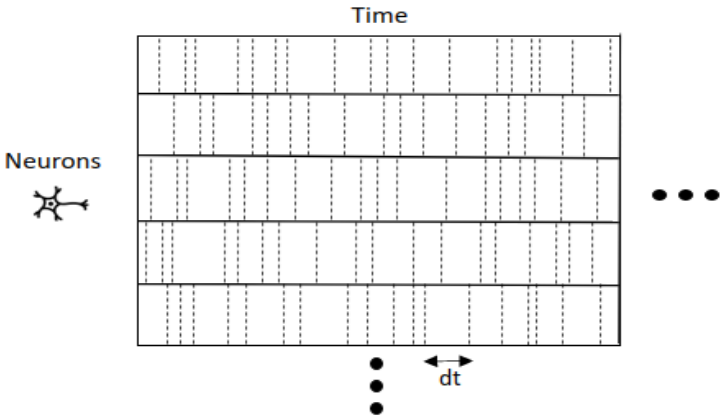

Figure 5: Neuron-specific random time-stepping method visualisation.

5 Future Work

Replicating the original model in BRIAN2 code as outlined in Section 3.3, is the first deliverable of the project. The result will be used in the experimentation stage, where both the shared randomised time-stepping and neuron-specific randomised time-stepping methods will be integrated into the original model architecture. To do so in BRIAN2, the source code must be modified first, in order to account for a dynamic dt. Therefore, the second project deliverable includes modifications of the C++ BRIAN2 codebase, rather than on the Python replicated model. When an adequate support for dynamic time-stepping is added to the simulator, it can be utilised in the Python replicated model to integrate the two aforementioned random time-stepping algorithms in the original model. Both methods' performances will be thoroughly analysed by using the results displayed by (Hansel, Mato et al. [4]) in the original paper as benchmark. Should any of the two approaches prove to be a significant improvement in the efficiency of the model, it will be considered to integrate this new dynamic time-step functionality in a future release of BRIAN Simulator, alongside a publication of the experiment. Figure 6 summarises in a GANTT chart the core and stretch deliverables of the project, as well as the future timeline and deadlines for each one of them.

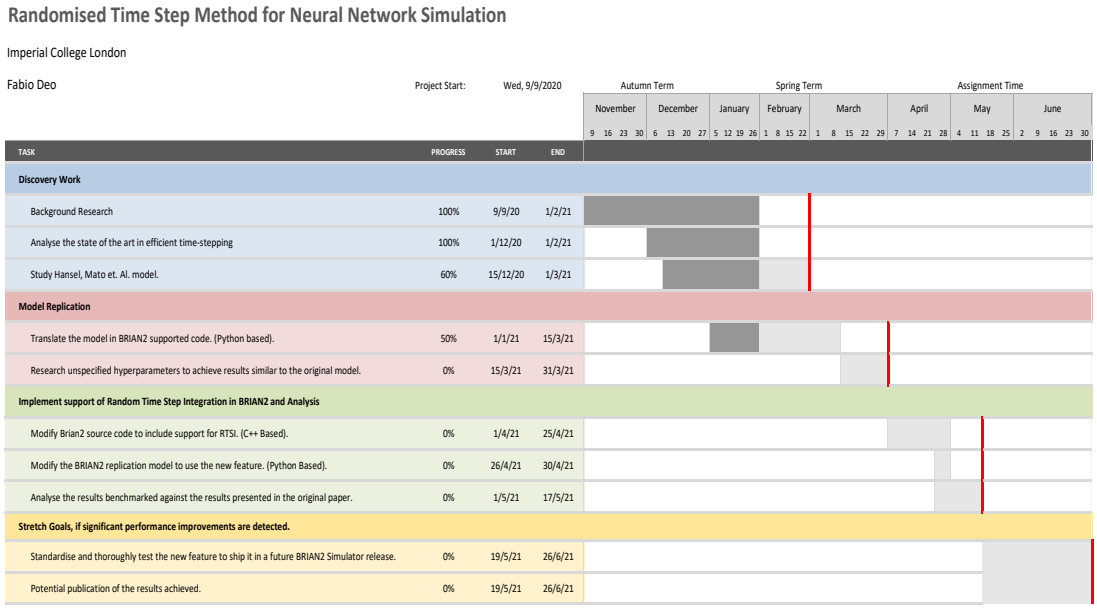

Figure 6: Gantt Chart showing the project deliverables and deadlines.

References

[1] Filip Ponulak, Andrzej Kasiński (2011) Introduction to spiking neural networks: Information processing, learning and applications.

[2] Nikola Kasabov, FIEEE, FRSNZ (2012) Evolving spiking neural networks and neurogenetic systems for spatio-and spectro-temporal data modelling and pattern recognition.

[3] Daniel Goodman, Romain Brette (2008) Brian: a simulator for spiking neural networks in Python

[4] D.Hansel, G.Mato, et al. (1998) On numerical simulations of integrate-and-fire neural networks

[5] Micheal J. Shelley, Louis Tao (2001) Efficient and Accurate Time-Stepping Schemes for Integrate-and-Fire Neuronal Networks

[6] William Lytton, Micheal Hines (2005) Independent variable timestep integration of individual neurons for network simulations

[7] Romain Brette et al. (2007) Simulation of networks of spiking neurons: A review of tools and strategies

[8] Michiel D'Haene, Benjamin Schrauwen et al. (2009) Accelerating Event-Driven Simulation of Spiking Neurons with Multiple Synaptic Time Constants

[9] Maurizio Mattia, Paolo Del Giudice (2000) Efficient Event-Driven Simulation of Large Networks of Spiking Neurons and Dynamical Synapses

[10] Jan Reutimann, Michele Giugliano, Stefano Fusi (2003) Event-Driven Simulation of Spiking Neurons with Stochastic Dynamics

[11] M. L. Hines, N. T. Carnevale (2001) NEURON: A Tool for Neuroscientists

[12] S.D. Cohen, A.C. Hindmarsh, P.F.Dubois (2000) CVODE, A Stiff/Nonstiff ODE Solver in C

A
Python Model

```
from brian2 import *
start_scope()

# Parameters
N      =      128                          # number of neurons
gl     =      0.1    *     msiemens        # conductance of the voltage-independent
     leak current
Vl     =      -60    *        mV           # reversal potential of the voltage-
    independent leak current
C      =        1    *        ufarad       # membrane capacitance
tau    =         C / gl / joule            # passive membrane time constant
theta  =      -40    *        mV           # membrane potential threshold
tau1   =        3    *        ms           # first time constant of bi-exp
tau2   =        1    *        ms           # second time constant of bi-exp
Io     =      0.1    *        mA           # DELIBERATE constant current drive

# FOR INITIAL CONDITION
T = -tau*log((1/(tau*Io)- theta)/(1/(tau*Io) - Vl)) # MODIFIED TO MAKE IT COMPILE
c=1 # in range (0, 1). 1 for uniform distribution.

########### MODEL ##########################
defaultclock.dt = 0.01*ms
mx = log(3) * 3 / 2000 * second
invpeak = 1/(tau1-tau2)*(exp(-mx/tau1)-exp(-mx/tau2)) * second
eqs = '''
dV/dt = (-gl*(V - Vl) + I_syn + Io) / C : volt
'''

syn_eqs = '''
dI_syn/dt = (invpeak * s - I_syn) / tau1      : amp
ds/dt     =  -s /  tau2                       : amp
'''

eqs = Equations(eqs + syn_eqs)

########### NEURONS ########################
G = NeuronGroup(N, eqs, threshold='V > theta', reset='V = Vl', method='exact')

## Init Voltages and currents.
for neu in range (N):
    G.V[neu] = 1/(tau * Io) + (Vl - 1/(tau*Io)) * exp(-c*(neu)*T/(N*tau))
    G.I_Syn[neu] = 0

########### SYNAPSES ########################
S = Synapses(G, G, 'w: amp', on_pre='I_syn += w')

## Connect
S.connect(condition='i!=j')

## Init weight and parameter.
S.w = 0.03 * mA # DELIBERATE
S.s = 1 * amp
```