BRIAN in the Browser: Running Neural Simulations with WebAssembly

Marcel Stimberg
Institut de la Vision, Sorbonne Université/INSERM/CNRS, Paris, France
marcel.stimberg@sorbonne-universite.fr

Who is Brian?

Brian is a **clock-driven spiking neural network simulator** which is **easy to learn and use, highly flexible and easy to extend**. It is written in Python and allows to describe and run **arbitrary neural and synaptic models** without having to write code in any other programming language. Using a **code-generation** technique, it also allows to fully automatically run code in other languages such as C++, leading to faster execution speed.

Stimberg, M., Brette, R., and Goodman, D. F. M. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife*

What is WebAssembly?

WebAssembly (*Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a **portable compilation target** for programming languages, enabling deployment on the web.

webassembly.org

emscripten

Emscripten is a **complete compiler toolchain to WebAssembly**, using LLVM, with a special focus on speed, size, and the Web platform.

emscripten.org

Pyodide

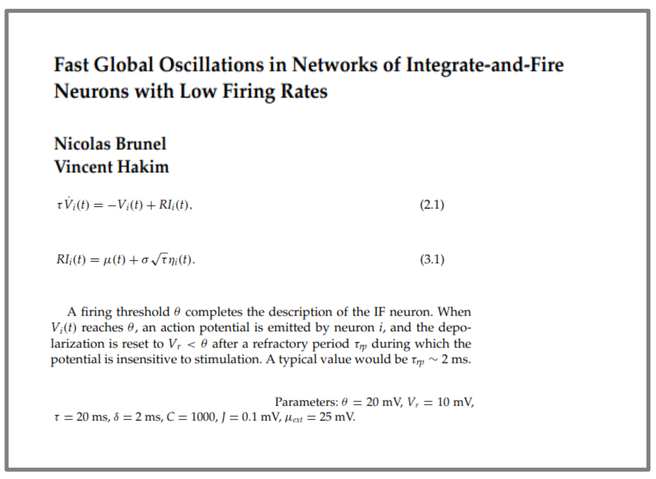
Pyodide is a **Python distribution for the browser** and Node.js based on WebAssembly.

pyodide.org

Brian2Wasm

Brian2Wasm is a new **free and open-source package** for the Brian 2 simulator. It allows users to directly **transform their neural simulation code into WebAssembly code**. The package builds upon the existing **Brian 2 code generation framework**; by using the emscripten framework, Brian2Wasm compiles this C++ code into WebAssembly. Brian2Wasm offers a promising new solution for **simulating neural networks directly in a web browser**. It allows for easy creation of **customized websites** for running simulations and displaying results, with the ability to establish **real-time links between the browser environment and the network simulation** for even greater flexibility and interactivity.

Running a model in the browser



Fast Global Oscillations in Networks of Integrate-and-Fire Neurons with Low Firing Rates

Nicolas Brunel
Vincent Hakim

$$\tau \dot{V}_i(t) = -V_i(t) + RI_i(t), \qquad (2.1)$$

$$RI_i(t) = \mu(t) + \sigma\sqrt{\tau}\eta_i(t). \qquad (3.1)$$

A firing threshold θ completes the description of the IF neuron. When $V_i(t)$ reaches θ, an action potential is emitted by neuron i, and the depolarization is reset to $V_r < \theta$ after a refractory period τ_{rp} during which the potential is insensitive to stimulation. A typical value would be $\tau_{rp} \sim 2$ ms.

Parameters: $\theta = 20$ mV, $V_r = 10$ mV, $\tau = 20$ ms, $\delta = 2$ ms, $C = 1000$, $J = 0.1$ mV, $\mu_{ext} = 25$ mV.

Brunel, Nicolas, and Vincent Hakim (1999). Fast Global Oscillations in Networks of Integrate-and-Fire Neurons with Low Firing Rates. *Neural Computation*

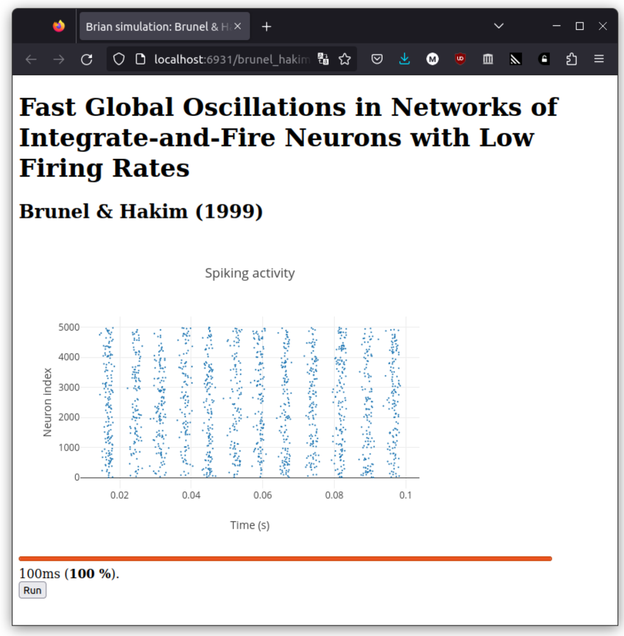
```
from brian2 import *
import brian2wasm
set_device('wasm_standalone')

Vr = 10*mV; theta = 20*mV; tau = 20*ms; delta = 2*ms
N = 5000; C = 1000; sparseness = C/N; J = .1*mV
sigma_ext = 1*mV; mu_ext = 25*mV

eqs = """
dV/dt = (-V+mu_ext + sigma_ext * sqrt(tau) * xi)/tau : volt
"""
group = NeuronGroup(N, eqs, threshold='V>theta',
          reset='V=Vr', refractory=2*ms, method='euler')
group.V = Vr

conn = Synapses(group, group, on_pre='V += -J', delay=delta)
conn.connect(p=sparseness)

M = SpikeMonitor(group)
run(0.1*second, report='text')
```

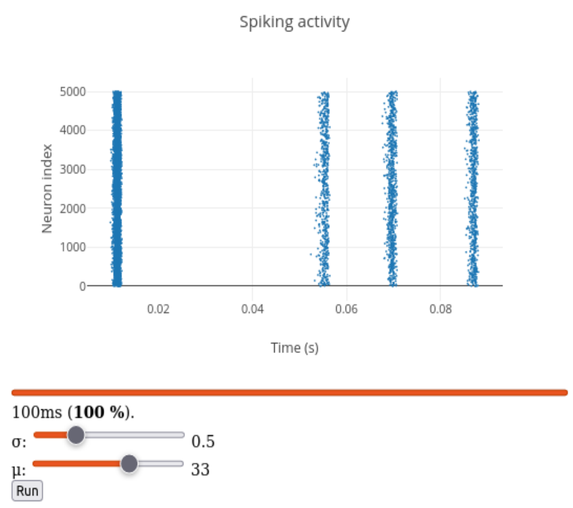


Customizing a simulation run

Change model parameters

```
eqs = """
dV/dt = (-V+mu_ext + sigma_ext * sqrt(tau) * xi)/tau : volt
mu_ext : volt (constant, shared)
sigma_ext : volt (constant, shared)
"""
```

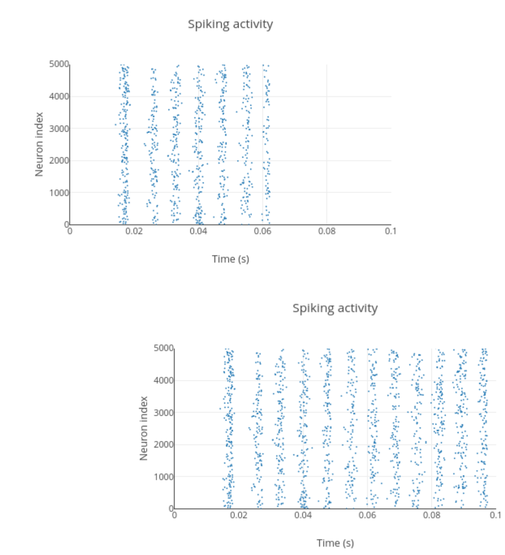
```
<label for="muext">μ: </label>
<input type="range" id="muext"
       min="0" max="50" step="1" value="25"
       oninput="this.nextElementSibling.value = this.value">
<output>25</output><br>
<button type="button" id='brian_run_button'
        onclick="brian_sim.run({
         'neurongroup.mu_ext': document.getElementById('muext').value/1000,
         'neurongroup.sigma_ext': document.getElementById('sigmaext').value/1000
        });">
    Run
</button>
```



Communicate during a simulation

```
from brian2wasm.functions import send_spike
group = NeuronGroup(N, eqs, threshold='V > theta',
                    reset='V = Vr; dummy = send_spike(i, t)',
                    refractory=2*ms, method='euler')
```

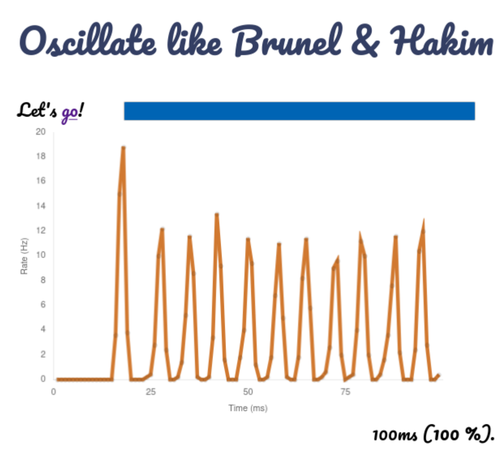
```
brian_sim.worker.onmessage = (e) => {
let index = e.data.index;
let t = e.data.time;
spikes[0].x.push(t);
spikes[0].y.push(index);
// Update plot for every 2ms of simulated time
if (t > brian_sim.last_plot_time + 0.002) {
    layout.datarevision += 1;
    Plotly.react('brian_canvas', spikes, layout)
    brian_sim.last_plot_time = t;
}
```



Simulating with style

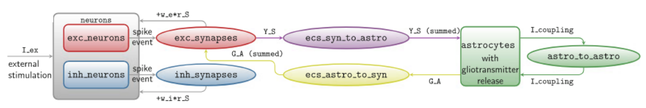
```
<style>
html {
    font-family: 'Pacifico', cursive;
}
…
</style>
```

```
const brian_results = event.data.results;
const t = brian_results['ratemonitor'].t;
const rates = brian_results['ratemonitor'].rate;
let data = …;  // gather data and preprocess
const ctx = document.getElementById('brian_canvas');
new Chart(ctx,
          {type: 'line',
           data: {labels: t.map(x => x*1000),
                  datasets: [{label: 'Rate',
                              data: data
                             }]
                 },
          …}); // configure plot
```



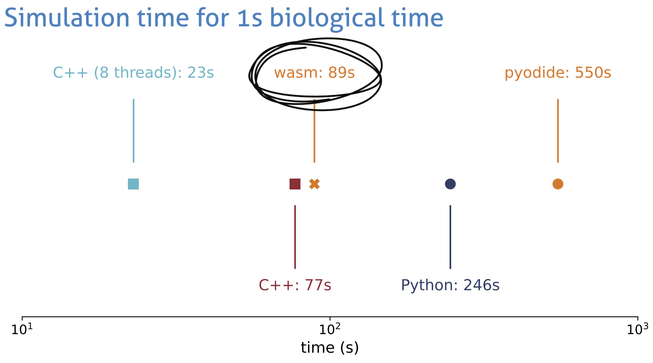
Browsers can be fast!

A more complex example: a network of 4000 LIF neurons, ~1.2M synapses with short-term plasticity, and 3200 neurons modulating the synapses,



Stimberg, M., Goodman, D. F. M., Brette, R., & Pittà, M. D. (2019). Modeling Neuron–Glia Interactions with the Brian2 Simulator. In M. De Pittà & H. Berry (Eds.), Computational Glioscience (pp. 471–505).

Simulation time for 1s biological time



Open issues and next steps

The only automatic plot at the moment is a spike raster plot with Plotly, we'd like to support **more plot types** and **more plotting libraries**
Transferring data from the WebAssembly module to the JavaScript is currently inefficient; large models therefore need the user to select the necessary data manually.
Connecting models to **real-time input from the browser** (e.g. camera, microphone) is already possible but cumbersome – we'd like to provide convencience functions to make this easier.

Plotting libraries

Raster plots: *Plotly* (https://plotly.com/javascript/)
Population rate: *chart.js* (https://chartjs.org)

Try it out

brian-team.github.io/brian2wasm

Download the poster

briansimulator.org
github.com/brian-team/brian2wasm