Institute of Neuromorphic Engineering

THE NEUROMORPHIC ENGINEER
A PUBLICATION OF INE-WEB.ORG

10.2417/1200907.1659

Brian: a simple and flexible simulator for spiking neural networks

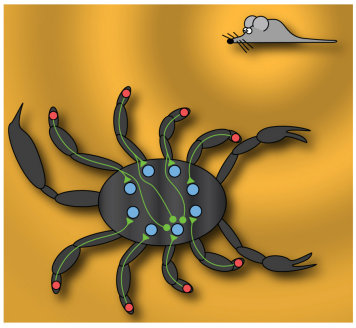
Romain Brette and Dan Goodman

New neural-simulation technology makes spiking neuron models more accessible to systems neuroscience and neuromorphic engineering.

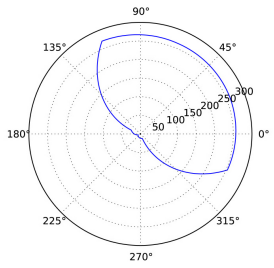
Neurons communicate with stereotypical electrical impulses called action potentials or spikes. This fact has been known for about 150 years, but most neural network theories in the 20th century have described neural activity in terms of firing rates. Spikes have been seen as simply the biological elements that the brain used in order to communicate and compute with these analog rates, in the same way as the intensity of light is encoded by the arrival rate of photons. However, over the last 15 years, neurons have been found to be more deterministic than previously thought and the coordination of spikes now seems to play a major role in sensory coding and computation. Consequently, many neuroscientists are now interested in modeling neural computation at the spike level. Unfortunately, simulating spiking models is much more difficult and time consuming than standard analog models, which has caused many scientists to refrain from making this switch.

Several systems have been developed in the past to simulate neuron models efficiently.[1] Some of them have been successful in the computational neuroscience community, but their complexities and steep learning curves have limited their audience. In many practical cases, it takes considerably more time to write the code than to run the simulations. While previous simulators addressed the issue of computational efficiency, we developed the Brian neural network simulator[2] with a different goal in mind: to minimize the time users spend on writing the code of neuron models so that they can spend more time on the science. The motto of the Brian project is 'a simulator should not only save the time of processors, but also the time of scientists'. Minimizing development time rather than simulation time implies different choices, putting more emphasis on flexibility, simplicity and readability.

A.



B.



C.

```
wave = lambda t:.2*sin(2*pi*300*Hz*t)*cos(2*pi*25*Hz*t)
eqs_legs = '''                                          # Leg mechanoreceptors
dv/dt = (1+wave(t-d)-v)/tau+sigma*(2/tau)**.5*xi : 1  # xi is white noise
d : second                                              # wave delay
'''
legs = NeuronGroup(8, eqs_legs, threshold='v>1', reset='v=0', refractory=1*ms)
legs.d = radius/wave_speed*(1-cos(prey_angle-legs_angle))
eqs_neuron = '''        # Command neurons
dv/dt = (x-v)/tau : 1
dx/dt = -x/tau    : 1  # PSPs are alpha functions
'''
neurons = NeuronGroup(8, eqs_neuron, threshold='v>1', reset='v=0')
synapses_ex = IdentityConnection(legs, neurons, 'x', weight=4)
synapses_inh = Connection(legs, neurons, 'x', delay=.7*ms)
for i in range(8):
    synapses_inh[i, (4+i-1)%8] = -1
    synapses_inh[i, (4+i)%8] = -1
    synapses_inh[i, (4+i+1)%8] = -1
run(300*ms)
```

Figure 1. *Brian implementation of a model of prey localization in the sand scorpion.[3] See details in the text. A) Architecture of the neural system. B) Firing rates for the eight command neurons in a polar plot for a prey at an angle of 45° relative to the scorpion. C) Corresponding Brian script, with parameter definitions and graphical commands omitted.*

These goals led us to make two choices: firstly to use a standard programming language that is both intuitive and well established, and secondly to let users define models in a form that is as close as possible to their mathematical definitions. The Brian simulator is written in Python, a well-established language

Continued on next page

that is intuitive, easy to learn and benefits from a large user community and many extension packages (in particular for scientific computing and visualization). Models are defined directly by providing their mathematical definition, consisting of differential equations and discrete events (the effect of spikes). This original approach has crucial benefits: it makes the code easy to read and share, it minimises the amount of simulator-specific syntax that needs to be learned, and makes simulating custom models as simple as simulating standard models.

A simulation using Brian is a Python program executed either from a script or interactively from a Python shell. Figure 1 shows a Brian script that implements a model of prey localization in the sand scorpion.[3] Only the definition of the parameters and plotting commands are not shown. The movement of the prey causes a surface wave (function wave in the code in panel C) which is detected by mechanoreceptors (the red points in panel A) at the ends of each of the scorpion's legs. The mechanoreceptors are modelled by noisy leaky integrate-and-fire neurons with an input current defined by the surface wave displacement at the ends of the legs (object legs in the code, defined by the equations eqs_legs). These neurons send an excitatory signal (the object synapses_ex) to corresponding command neurons (the blue points) modeled by leaky integrate-and-fire neurons (object neurons with equations eqs_neuron), which also receive delayed inhibitory signals (the object synapses_inh) from the three legs on the other side (the for loop). A wave arriving first at one side of the scorpion will cause an excitatory signal to be sent to the command neurons on that side causing them to fire, and an inhibitory signal to the command neurons on the other side, stopping them from firing when the wave reaches the other side. The result is that command neurons are associated with the directions of the corresponding legs, firing at a high rate if the prey is in that direction. This fairly complex model can be simulated with only about 20 lines of Brian code.

Python is an interpreted language, and although it is very fast there is an overhead for every Python operation. Brian can achieve very good performance by using the technique of vectorisation, similar to that familiar to Matlab users. The idea is to replace loops by operations on large vectors, so that the interpretation overhead becomes negligible. Brian uses vectorisation for both the simulation and the construction of the model (e.g., initialisation of synaptic weights), and for large networks it can achieve speeds comparable to that of code written directly in C.[4]

To sum up, Brian is a convenient simulation tool for exploring new spiking neural models, modeling complex neural models at the systems level, and teaching computational neuroscience. We are currently working on distributed simulation technologies for Brian, focusing on using graphics processing units, an inexpensive piece of hardware consisting of a large number of parallel processing cores. Finally, we would like to mention that Brian is an open source project and we warmly welcome external contributions.

This work was partially supported by the French ANR, the CNRS and the Ecole Normale Supérieure. The authors would like to thank all those who tested early versions of Brian and made suggestions for improving it.

Author Information

Romain Brette and Dan Goodman
CNRS, Ecole Normale Supérieure
Paris, France

Romain Brette is an assistant professor in computational neuroscience. His research interests include spike-based neural computation (especially in the auditory system), theory and simulation of spiking neuron models, and intracellular recording techniques.

Dan Goodman is a postdoctoral researcher in computational neuroscience. His research interests include the role of spike-timing based coding and computation, and neural simulation technologies.

References

1. R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris Jr., M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. El Boustani, and A. Destexhe, *Simulation of networks of spiking neurons: a review of tools and strategies*, **J. Comp. Neurosci. 23** (3), pp. 349–98, 2007.
2. http://www.briansimulator.org
3. W. Stürzl, R. Kempter, and J. L. van Hemmen, *Theory of arachnid prey localization*, **Phys. Rev. Lett. 84** (24), pp. 5668–5671, 2000.
4. D. Goodman and R. Brette, *Brian: a simulator for spiking neural networks in Python*, **Frontiers in Neuroinformatics 2** (5), 2008. doi:10.3389/neuro.11.005.2008