Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Neurocomputing 58–60 (2004) 1117 – 1122
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Discrete event simulation in the NEURON environment

M.L. Hines[a,*], N.T. Carnevale[b]

[a]Department of Computer Science, Yale University, 51 Prospect Street, New Haven, CT 06520 8285, USA
[b]Department of Psychology, Yale University, 51 Prospect Street, New Haven, CT 06520 8285, USA

Abstract

The response of many types of integrate and fire cells to synaptic input can be computed analytically and their threshold crossing either computed analytically or approximated to high accuracy via Newton approximation. The NEURON simulation environment simulates networks of such artificial spiking neurons using discrete event simulation techniques in which computations are performed only when events are received. Thus, computation time is proportional only to the number of events delivered and is independent of the number of cells or problem time.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Artificial cells; Spiking nets; Discrete events; Simulation; Integrate and fire

1. Introduction

The NEURON simulation environment is capable of efficient discrete event simulations of networks of “artificial” (integrate and fire) spiking neurons, as well as hybrid simulations of nets whose elements include both artificial neurons and neuron models with membrane currents governed by voltage-gated ionic conductances. In this paper, we discuss how NEURON uses discrete events to implement the classic single time constant integrate and fire neuron and a cell with a natural interval between spikes which is modulated by synaptic input. Readers who wish additional information about the fundamental principles behind the design and implementation of NEURON, especially with regard to its use for continuous system simulations of empirically based models of individual neurons, are referred to [2,3].

* Corresponding author. Tel.: +1-203-737-4232; fax: +1-203-785-6990.
E-mail addresses: michael.hines@yale.edu (M.L. Hines), ted.carnevale@yale.edu (N.T. Carnevale).

0925-2312/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2004.01.175

NEURON's event delivery system draws on ideas previously used by Destexhe et al. [1] and Lytton [4] in their models of networks of "biological" neurons. It opens up a large domain of models in which certain types of "artificial" spiking cells, and networks of them, can be simulated hundreds of times faster than with numerical integration methods. Discrete event simulations are possible when all the state variables of a model cell can be computed analytically from a new set of initial conditions. That is, if an event occurs at time t_1, all state variables must be computable from the state values and time t_0 of the previous event. Since computations are performed only when an event is received, total computation time is proportional to the number of events delivered and independent of the number of cells, number of connections, or problem time. Thus handling 100,000 spikes in 1 h for 100 cells takes the same time as handling 100,000 spikes in 1 s for 1 cell.

The portion of the event delivery system used to define connections between cells is the NetCon (Network Connection) class. A NetCon object typically watches its source cell for the occurrence of a spike, and then, after some time delay, delivers a weighted synaptic input event to a target cell. That is, the NetCon object represents the axonal spike propagation and synaptic delay. A NetCon object can be thought of as a channel on which a stream of events generated at the source is transmitted to the target. Its implementation takes into account the fact that the delay between initiation and delivery of events is different for different streams. Consequently the order in which events are generated by a set of sources is rarely the order in which they are received by a set of targets. Furthermore the delay may have any value from 0 to 10^9.

Handling of incoming events and the calculations necessary to generate outgoing events are specified in the `NET_RECEIVE` block of NEURON's model description language NMODL [3]. Artificial cells are implemented as point processes that serve both as targets and sources for NetCon objects. They are targets because they have a `NET_RECEIVE` block, which handles discrete event input streams through one or more NetCon objects. They are also sources because the `NET_RECEIVE` block also generates discrete output events which are delivered through one or more NetCon objects.

Computer code listings have been edited to remove unnecessary detail, with elisions marked by ellipses and augmented by *italicized pseudocode* as necessary for the sake of clarity. Complete source code for all mechanisms described in this paper are included with NEURON, which is available at no charge from http://www.neuron.yale.edu.

2. IntFire1: a basic integrate and fire model

The simplest integrate and fire mechanism built into NEURON is IntFire1, which has a "membrane potential" state m which decays towards 0 with time constant τ:

$$\tau \frac{\mathrm{d}m}{\mathrm{d}t} + m = 0. \tag{1}$$

An input event of weight w adds instantaneously to m, and when $m \geqslant 1$ the cell "fires," producing an output event and returning m to 0. Negative weights are inhibitory while positive weights are excitatory. This is analogous to a cell whose membrane time constant τ is very long compared to the time course of individual synaptic conductance changes. Every synaptic input shifts membrane potential to a new level in a time that is much shorter than τ, and each cell firing erases all traces of prior inputs.

An initial implementation of IntFire1 in NMODL is

```
...variable declarations...
INITIAL {m = 0 t0 = 0}

NET_RECEIVE (w) {
    m = m*exp(- (t -  t0)/tau)
    m = m + w
    t0 = t
    if (m >= 1) {
      net_event(t)
      m = 0
    }
}
```

This model lacks the BREAKPOINT and SOLVE blocks, which are used to specify the calculations that are performed during a time step dt [3]. Instead, calculations only take place when a NetCon delivers a new event, and these are performed in the NET_RECEIVE block. When a new event arrives, the present value of m is computed analytically and then incremented by the weight w of the event (specified by the weight field of the NetCon that delivered the event). In this example, the value of m is just the exponential decay from its value at the previous event; therefore the code contains variable t0 which keeps track of the last event time.

If an input event drives m to or above threshold, the net_event(t) statement notifies all NetCon objects that have this point process as their source that it fired a spike at time t (the argument to net_event() can be any time $\geqslant$ the current time t). Then the cell resets m to 0. This particular model has no limit to its firing frequency, but elaborating it with a refractory period is not difficult. However, the system behaves properly with the variable time step method if a NetCon with delay of 0 and a weight of 1.1 has such an artificial cell as both its source and target. In that case, events are generated and delivered without time ever advancing.

Note that this artificial cell does not need a threshold test at every dt. Contrast this with the usual test for membrane potential triggering which is essential at every time step for event generation with "real" cells. Furthermore the event delivery system only places the earliest event to be delivered on the event queue. When that time finally arrives, all targets whose NetCons have the same source and delay get the event delivery, and longer delay streams are put back on the event queue to await their specific delivery time.

The integration state m is difficult to plot in an understandable manner, since the computer value of m remains unchanged in the interval between input events regardless of how many numerical integration steps were performed in that interval. Consequently m always has the value that was calculated after the last event was received, and a plot of m looks like a staircase with no apparent decay pattern or indication of what

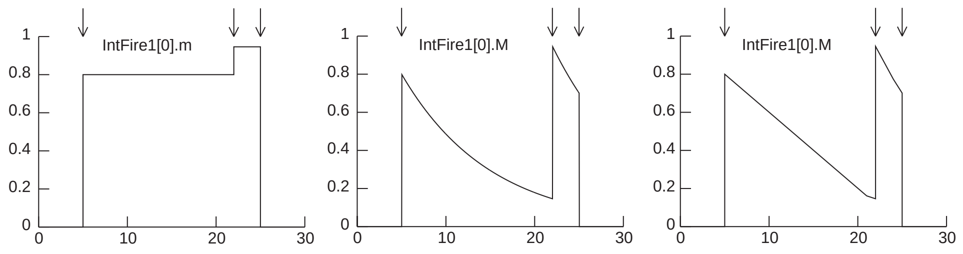

Fig. 1. Response of an IntFire1 cell with $\tau = 10$ ms to input events arriving at $t = 5$, 22 and 25 ms (arrows), each of which has weight $w = 0.8$. The third input triggers a spike. Left: A plot of m looks like a staircase because this variable is evaluated only when a new input event arrives. Center: A function can be included in the mod file that specifies IntFire1 (see text) to give a better indication of the time course of the integration state m. Plotting this function during a simulation with fixed time steps ($\mathrm{d}t = 0.025$ ms here) demonstrates the exponential decay of m between input events. Right: In a simulation with variable time steps, the decay appears to follow a sequence of linear ramps. This is only an artifact of the Graph tool drawing lines between the points computed analytically at the time steps chosen by the integrator.

the value of m was just before the event (Fig. 1 left). This can be partially repaired by providing a function

```
FUNCTION M() {
    M = m*exp(- (t -  t0)/tau)
}
```

which returns the present value of the integration state m. This gives nice trajectories for the fixed step method (Fig. 1 center). However, the natural step with the variable step method is the interspike interval itself, unless intervening events occur in other cells (e.g. 1 ms before the second input event in Fig. 1 right). At least the integration step function `fadvance()` returns 10^{-9} ms before and after the event to properly indicate the discontinuity in `M`.

3. FreqFire1: synaptic input modifies the interval between spikes

IntFire1 can be extended to fire at constant frequency (in the absence of synaptic input) by adding a constant bias current. The equation being solved is now

$$\tau \frac{\mathrm{d}m}{\mathrm{d}t} + m = m_\infty. \tag{2}$$

This has analytic solutions for $m(t)$, the firing time (when $m(t) = 1$), and the value for m_∞. These solutions can be stated in terms of a user-defined parameter *invl*, interspike interval in the absence of synaptic input.

The substantive NMODL code for the model description is

```
...variable declarations...
INITIAL {
    minf = 1/(1 - exp(- invl/tau))
    m = 0
    t0 = t
    net_send(firetime(), 1)
}
NET_RECEIVE (w) {
    m = minf + (m - minf)*exp(- (t - t0)/tau)
    t0 = t
    if (flag == 0) {
        m = m + w
        if (m > 1) {
            m = 0
            net_event(t)
        }
        net_move(t+firetime())
    } else {
        net_event(t)
        m = 0
        net_send(firetime(),1)
    }
}
FUNCTION firetime() {:Note- m < 1 < minf
    firetime=tau*log((minf- m)/(minf -  1))
}
```

This exploits the notion of a "self event" that can be placed on the event queue for future delivery to the `NET_RECEIVE` block, and whose delivery time can be adjusted in response to computations in that block. The self event is placed on the queue by the `net_send` statement with a `flag` of 1 to distinguish it from external (synaptic) events, which always have a `flag` of 0. When a synaptic event arrives, the self event is moved to a new firing time by the `net_move` statement. The synaptic event may itself be the trigger for firing, but a returning self event definitely signifies firing; in either case, the membrane state variable `m` is reset to 0.

4. Discussion

NEURON's application domain extends beyond continuous system simulations of models of individual neurons with complex anatomical and biophysical properties, to encompass discrete-event and hybrid simulations that combine "biological" and "artificial" neuronal models. Artificial cell models can have quite elaborate behavior. For example, an excitatory input to NEURON's IntFire4 model generates an exponentially decaying current with time constant τ_e, an inhibitory input generates a biexponential (alpha-function-like) current with time constants τ_1 and τ_2, and the total current is integrated with membrane time constant τ_m. The four cell states are analytic, and the threshold crossing time is computed by a sequence of self events, each triggering a single Newton iteration. Since NEURON's library of mechanisms is extensible through the NMODL programming language [3], users can add new features to existing mechanisms, as well as develop new formulations of artificial neuron models that have analytic solutions. For example, an extension of IntFire1 that implements short-term use-dependent plasticity as described by Varela et al. [5] is available at ModelDB (http://senselab.med.yale.edu/senselab/modeldb/).

Acknowledgements

This work was supported by NIH Grant NS11613.

References

[1] A. Destexhe, Z.F. Mainen, T.J. Sejnowski, An efficient method for computing synaptic conductances based on a kinetic model of receptor binding, Neural Comput. 6 (1994) 14–18.

[2] M.L. Hines, N.T. Carnevale, The NEURON simulation environment, Neural Comput. 9 (1997) 1179–1209.

[3] M.L. Hines, N.T. Carnevale, Expanding NEURON's repertoire of mechanisms with NMODL, Neural Comput. 12 (2000) 839–851.

[4] W.W. Lytton, Optimizing synaptic conductance calculation for network simulations, Neural Comput. 8 (1996) 501–509.

[5] J.A. Varela, K. Sen, J. Gibson, J. Fost, L.F. Abbott, S.B. Nelson, A quantitative description of short-term plasticity at excitatory synapses in layer $\frac{2}{3}$ of rat primary visual cortex, J. Neurosci. 17 (1997) 7926–7940.