NOTE Communicated by Alain Destexhe

Optimizing Synaptic Conductance Calculation for Network Simulations

William W. Lytton
Department of Neurology, University of Wisconsin, Wm. S. Middleton VA Hospital, 1300 University Ave., MSC 1720, Madison, WI 53706 USA

High computational requirements in realistic neuronal network simulations have led to attempts to realize implementation efficiencies while maintaining as much realism as possible. Since the number of synapses in a network will generally far exceed the number of neurons, simulation of synaptic activation may be a large proportion of total processing time. We present a consolidating algorithm based on a recent biophysically-inspired simplified Markov model of the synapse. Use of a single lumped state variable to represent a large number of converging synaptic inputs results in substantial speed-ups.

1 Introduction

The computational demands of a single synapse in realistic neural simulations can equal the cost of several neuronal units in an artificial neural network. In particular, Markov models of synaptic activation are dynamic systems that may have 10 or more state variables. An alternative, the classical "alpha function" model (Rall 1967), is computationally cheap but lacks obvious biophysical correlates at the channel level (Destexhe, Mainen, and Sejnowski 1994b). Recently, a middle ground has been developed that preserves some major aspects of a biophysically realistic full Markov model at considerably less computational cost (Wang and Rinzel 1992; Destexhe *et al.* 1994a,b). Destexhe and co-workers demonstrated a minimal two-state model with a fundamental biophysical verisimilitude that used a simple implementation practical for network use. We will call this the "DMS model" after the authors' initials.

Individual synapses in neuronal networks are generally represented as distinct entities that alter a conductance in the postsynaptic neuron after detecting some signal, typically voltage or calcium crossing a threshold, in the presynaptic neuron. This representation is widely used in the synaptic packages available with the major realistic neural simulators. All of the synapses of a single type are doing identical, potentially redundant, calculations, albeit at slightly different times. Srinivasan and Chiel (1993) previously demonstrated how multiple alpha functions could be consolidated by representing their summation in an iterated closed form.

We present a similar consolidating algorithm that allows an efficient implementation of large numbers of DMS synapses.

Rather than treating each synapse individually, we will lump all of the synapses of a given type (e.g., GABA_A or AMPA), converging onto a single compartment of one model neuron. These will then be represented by consolidated state variables and a single synaptic conductance and synaptic current. In the following, "single synapse" or "individual synapse" is used to describe the basic two-state DMS model. "Complex synapse" describes the lumped synapse model. Lower-case state variables and conductance (r, g) will be used for the former and upper-case (R, G) for the latter.

2 The DMS Algorithm

The first-order kinetic scheme was introduced by Destexhe *et al.* (1994a). The notation has been slightly modified for simplicity of description of the subsequent algorithms.

$$r_{\text{closed}} \underset{\beta}{\overset{\alpha(C)}{\rightleftharpoons}} r_{\text{open}} \tag{2.1}$$

This model of a ligand-gated ion channel is comparable to the standard Hodgkin–Huxley parameterization for voltage-sensitive ion channels. The difference is that the α and β parameters are not functions of voltage. Instead α is taken as a simple function of transmitter concentration:

$$\alpha(C) = \begin{cases} \alpha_0 & (C = 1) \\ 0 & (C = 0) \end{cases} \tag{2.2}$$

Transmitter concentration C is assumed to be given by a square wave of amplitude 1 and duration C_{dur}. β is a constant.

Following the Hodgkin–Huxley notation, the kinetic scheme can be expressed as a first order differential equation that solves for r in terms of R_∞ and τ_R in the usual way:

$$\tau_R \dot{r} = R_\infty - r \tag{2.3a}$$

$$R_\infty = \frac{\alpha}{\alpha + \beta} \tag{2.3b}$$

$$\tau_R = \frac{1}{\alpha + \beta} \tag{2.3c}$$

The update rule derived from the analytic solution for a single time step Δt is

$$r = R_\infty(1 - e^{-\Delta t/\tau_R}) + re^{-\Delta t/\tau_R} \tag{2.4}$$

(Note that this rule defines r in terms of itself, connoting the update step that would be used in software implementation.) The full rule is needed only when transmitter is present, since when $C = 0$, $R_\infty = 0$ and $\tau_R = 1/\beta$. Equation 2.4 can be split into two update rules using $r = r_{\text{on}}$ or $r = r_{\text{off}}$ depending on the presence or absence of transmitter C.

$$\begin{aligned} r_{\text{on}} &= R_\infty(1 - e^{-\Delta t/\tau_R}) + r_{\text{on}} e^{-\Delta t/\tau_R} && (C > 0) && (2.5a)\\ r_{\text{off}} &= r_{\text{off}}\, e^{-\beta \Delta t} && (C = 0) && (2.5b)\end{aligned}$$

Note that r_{on} and r_{off} are not the same as r_{open} and r_{close} from equation 2.1 but are both components of r_{open}. Synaptic conductance g_{syn} and current i_{syn} are defined in the usual way:

$$\begin{aligned} g_{\text{syn}} &= \bar{g}_{\text{syn}} r && (2.6a)\\ i_{\text{syn}} &= g_{\text{syn}} \left(V - E_{\text{syn}}\right) && (2.6b)\end{aligned}$$

3 Summing DMS Synapses

Summing synaptic activations makes it possible to maintain and update two rather than N state variables for the N synapses of a single type converging onto a given cell. An added advantage of this method is that it permits us to maintain a single queue of spike arrival times (now more accurately a heap) instead of N queues. The former improvement results in saving CPU time and the latter in saving both time and memory.

3.1 Separate Summations Required for R_{on} and R_{off}. We cannot use the single update rule of equation 2.4 since this would require summing over different r_{∞_i} and τ_{r_i} depending on the presence or absence of transmitter at the *ith* synapse. However, the two rules 2.5a and 2.5b have known factors that can be precalculated and brought out from under the summation. Again splitting r into r_{on} and r_{off}, we then simply sum across N_{on} and N_{off} synapses, respectively, where the N total synapses have been partitioned depending on their status.

$$\begin{aligned} \sum_{i=1}^{N_{\text{on}}} r_{\text{on}_i} &= (1 - e^{-\Delta t/\tau_R}) \sum_{i=1}^{N_{\text{on}}} R_\infty + e^{-\Delta t/\tau_R} \sum_{i=1}^{N_{\text{on}}} r_{\text{on}_i} && (3.1a)\\ \sum_{i=1}^{N_{\text{off}}} r_{\text{off}_i} &= e^{-\beta \Delta t} \sum_{i=1}^{N_{\text{off}}} r_{\text{off}_i} && (3.1b)\end{aligned}$$

Using $R_{\text{on}} = \sum_{i=1}^{N_{\text{on}}} r_{\text{on}_i}$ and $R_{\text{off}} = \sum_{i=1}^{N_{\text{off}}} r_{\text{off}_i}$ and noting that $\sum_{i=1}^{N_{\text{on}}} R_\infty = N_{\text{on}} R_\infty$, we can simplify 3.1a and 3.1b to produce update rules for unsubscripted Rs:

$$\begin{aligned} R_{\text{on}} &= N_{\text{on}} R_\infty (1 - e^{-\Delta t/\tau_R}) + R_{\text{on}} e^{-\Delta t/\tau_R} && (3.2a)\\ R_{\text{off}} &= R_{\text{off}}\, e^{-\beta \Delta t} && (3.2b)\end{aligned}$$

This form is identical to the single synapse update rules (2.5a and 2.5b) except that the forcing function for $R_{\rm on}$ has been multiplied by $N_{\rm on}$, the number of active synapses. These two update rules 3.2a and 3.2b form a compact two-step inner loop that the complex synapse executes at every time step.

3.2 Modifying $R_{\rm on}$ and $R_{\rm off}$ When Individual Synapses Go on or off. Updating the summed synaptic state variables on each time step saves the computational cost associated with updating individual r_is. However, since $R_{\rm on}$ and $R_{\rm off}$ are complementary state variables that follow the respective rise and decay of multiple single synapses, these single synapse r_is are still needed. When a single synapse changes state from off to on, $R_{\rm on}$ must be augmented by the corresponding r_i and $R_{\rm off}$ decremented. Conversely, when an individual synapse turns from on to off, $R_{\rm on}$ must be decremented and $R_{\rm off}$ augmented by the appropriate amount. In addition, the value of $N_{\rm on}$ in equation 3.2a must be incremented by 1 (off$\rightarrow$on) or decremented by 1 (on$\rightarrow$off).

Keeping track of individual r_i values is easily done. Since these state variables are independent of voltage, they can be projected out in time from the last (opposite direction) transition (Destexhe *et al.* 1994a):

$$r_i = R_\infty(1 - e^{-C_{\rm dur}/\tau_R}) + r_i e^{-C_{\rm dur}/\tau_R} \quad \text{(turning off)} \qquad (3.3a)$$

$$r_i = r_i e^{-\beta ISI} \quad \text{(turning on)} \qquad (3.3b)$$

These are identical to equations 2.5a and 2.5b except that time interval Δt has been replaced by $C_{\rm dur}$ (duration of transmitter release) in the first case and by *ISI*, the interspike interval, in the second. Note that while $C_{\rm dur}$ will remain constant during a simulation, $ISI = t - (t_0 + C_{\rm dur})$ where t_0 is the time of last activation. Thus, *ISI* is the interval from the end of synaptic activation to the beginning of the next activation for the same individual synapse.

3.3 Handling Different Maximal Conductances. We now have a way of calculating the state variable $R = R_{\rm on} + R_{\rm off}$. We can calculate a conductance G from this as we did in equation 2.6a or else calculate the components of $G = G_{\rm on} + G_{\rm off}$ individually:

$$G_{\rm syn} = \overline{G}R \qquad (3.4a)$$

$$G_{\rm on} = \overline{G}R_{\rm on} \qquad (3.4b)$$

$$G_{\rm off} = \overline{G}R_{\rm off} \qquad (3.4c)$$

The foregoing analysis assumes that the individual synapses all have identical conductances. This will generally not be the case. To handle varying $\overline{g}_i$s, we need to expand 3.4b and 3.4c in the same manner as previously (cf. equations 3.1a and 3.1b):

$$G_{\text{on}} = \sum_{i=1}^{N_{\text{on}}} g_i r_{\text{on}_i} = (1 - e^{-\Delta t/\tau_R}) R_\infty \sum_{i=1}^{N_{\text{on}}} g_i + e^{-\Delta t/\tau_R} \sum_{i=1}^{N_{\text{on}}} g_i r_{\text{on}_i} \tag{3.5a}$$

$$G_{\text{off}} = \sum_{i=1}^{N_{\text{off}}} g_i r_{\text{off}_i} = e^{-\beta\Delta t} \sum_{i=1}^{N_{\text{off}}} g_i r_{\text{off}_i} \tag{3.5b}$$

We divide through by $\overline{G}$, and change variables to create a new state variable $r_i' = (g_i/\overline{G})r_i$. If we now redefine $R_{\text{on}} = \sum_{i=1}^{N_{\text{on}}} r_{\text{on}_i}'$, $R_{\text{off}} = \sum_{i=1}^{N_{\text{off}}} r_{\text{off}_i}'$ and $N_{\text{on}} = \sum_i (g_i/\overline{G})$, we arrive back at equations 3.2a and 3.2b. The change in the definition of N_{on} is the only one that affects the implementation. The previous $N_{\text{on}} = \sum_i 1.0$ since each individual synapse had identical magnitude 1. Now, instead of incrementing or decrementing by 1 when turning an individual synapse on or off, we simply add or subtract the appropriate $g_i/\overline{G}$.

Making our new state variable a proportion rather than a conductance is done for convenience and to maintain the Hodgkin–Huxley tradition of dimensionless state variables. The new state variable is described by a slight variation in equation 2.1. In the usual convention, $r_{\text{closed}} = 1 - r_{\text{open}}$. With this modification $r_{\text{closed}} = (g_i/\overline{G}) - r_{\text{open}}$. The dimensionless state variable is also useful for managing simulations. With $\overline{G}$ treated as a simulation-wide global parameter, equation 3.4a gives the user the ability to globally alter the strength of a particular neurotransmitter by reducing the corresponding $\overline{G}$. This is analogous to the common experimental practice of dumping transmitter antagonists into the bath *in vitro* or giving antagonists systemically *in vivo*.

4 Maintaining a Single Queue

Simulating delay is necessary because most simulations do not include axons. Therefore the delay encompasses both the time taken for an action potential to proceed down the axon (axonal delay) as well as the typically shorter time required for transmitter to diffuse across the synaptic cleft and bind. Handling delays requires maintenance of a queue, a data structure that always disgorges its oldest element (first-in, first-out). Typically, the time of a presynaptic activation is added to the appropriate delay and then stored on a queue. When this time is reached in the simulation, the item is removed from the queue, and the postsynaptic element is activated.

Since many individual synapses are now maintained as a single complex synapse, it is natural to consider maintaining a single queue instead of N queues. The queue must now store not only the times of synaptic activation but also an index indicating the specific individual synapse.

4.1 Managing the Queue from the Presynaptic Side. In the direct object-oriented approach to the synapse, the synapse manages its own initiation by constantly checking the presynaptic cell for a signal, generally the passage of voltage above a predetermined threshold. The consolidation of signal management in a single queue would require an array of such presynaptic pointers. The alternative, maintaining a presynaptic array of postsynaptic pointers, is far more efficient: access across the pointer is required only when spikes occur instead of on each time step (Bower and Beeman 1994). Such forward pointers are particularly important in implementations on multiprocessor computers where pointer access between different CPUs is relatively slow (Niebur *et al.* 1991).

Using forward pointers, the queue receives its input from a structure associated with the presynaptic neuron. When triggered, this structure writes a time stamp equal to current time plus the appropriate delay. Presynaptic identity is also written in the form of an index. The queue is read postsynaptically when time reaches the value of the next queue time. The postsynaptic mechanism is then altered by moving the corresponding r_i from R_{off} to R_{on}. Because a complex synapse has a single C_{dur}, the queue can serve double duty and signal not only the initiation of the synapse but also its termination. For this reason, the queue time is not removed but is instead incremented by C_{dur}. The queue is implemented with two heads: the first head gives the time for initiating another synapse while the second head gives the time for terminating one. Each time is associated with an index that indicates exactly which individual synapse is being started or stopped.

4.2 A Heap Qua Queue Handles Differing Synaptic Delays. Individual synapses may have different delays. If these synapses share the same queue, an individual synapse with a relatively long delay could activate presynaptically shortly before one with a relatively short delay. This would put a later time on the queue in front of an earlier time. The synapse associated with the earlier time would be activated only after the later time was removed from the queue. To avoid this problem, a heap is used in lieu of a queue. Items in a heap are maintained in numerical order. A traditional heap implementation involves a binary tree (Knuth 1973). In the present case, items arrive out of order relatively rarely and are usually not very far out of order, making a binary tree unnecessary. Instead, the item is checked when it arrives, with the appropriate heap location readily found by forward search when needed.

Further consolidation is possible by creating a single master heap for all synapses of a given type. Each heap entry must then include not only an index for the presynaptic mechanism but also one for the postsynaptic mechanism. The algorithm must take account of postsynaptic mechanism number as well as time in maintaining heap order.

5 Simulation Results

Benchmark simulations were run in NEURON (Hines 1993) on Sun SPARC10s under SunOS 4.1.3 and Intel Pentiums under Linux. Figure 1 shows results comparing individual single synapse evaluation with the complex summated synapse. In Figure 1B the summation of DMS state variables ($\sum r_i$, dashed line) is compared to the single R_{on} calculated using the present algorithm (solid line). The lines do not coincide because the complex synapse algorithm includes weighting for the different $\bar{g}$s. Figure 1C compares conductances for the two schemes. Benchmarking demonstrated a 3-fold speed-up with the current algorithm. Extending the simulation to 200 fully interconnected mutually excitatory neurons receiving similar input and spiking at approximately 12 Hz demonstrated a 45-fold speed-up.

A more complex simulation with 225 excitatory and inhibitory neurons was also benchmarked. Individual neurons had two compartments and 8–9 voltage- and/or calcium-sensitive conductances. Connectivity was nearly complete with a total of 50,400 synapses and the average firing rate was approximately 20 Hz. No attempt was made to optimize either the old or new synapse model by determining ideal queue lengths; instead, conservative values were used. With the synapse model presented here, core memory usage was reduced 38% from 8 to 5 Mb. CPU time was reduced by 96% from 38 hr 50 min to 1 hr 35 min.

6 Discussion

Simulations of neuronal networks quickly fall victim to the perils of combinatorics. While the calculation time required for simulating individual neurons increases proportionally with number of neurons n, the number of synapses can increase up to n^2 depending on convergence. Specifically, the number of synapses S can be given either by the product of convergence and number of postsynaptic cells $S = C \cdot Post$ or by the product of divergence and number of presynaptic cells $S = D \cdot Pre$. Percent convergence, C/Pre, expressed in terms of number of synapses is $S/(Post \cdot Pre)$. This is equal to percent divergence: $D/Post = S/(Pre \cdot Post)$ (Traub *et al.* 1987). Calling this term percent connectivity (p_{ij}), a network with N cell types and n_i cells of each type will have number of synapses S given by

$$S = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} n_i n_j \tag{6.1}$$

Depending on the complexity of the single neuron model chosen, time spent in synaptic computations can readily outrun the time spent modeling the neurons themselves. This will be particularly true in parallel implementations if pointers are not managed carefully, as noted above.

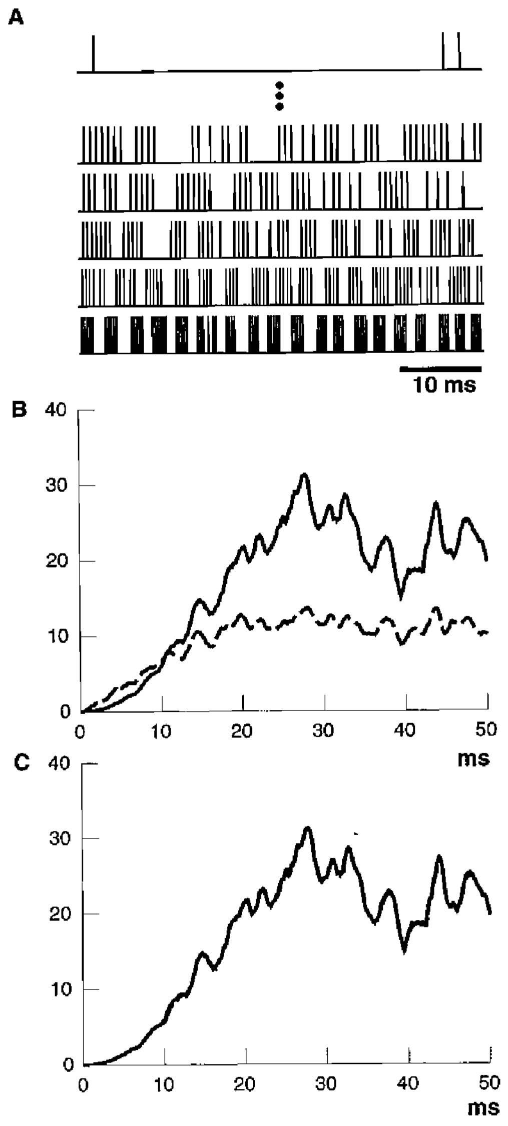

Figure 1:

Figure 1: Comparison of DMS model synapse (dashed line) with the complex synapse (solid line). Randomized synaptic input was used to drive both synapse models. Individual $\bar{g}$ values ranged systematically from 0.1 to 5 μS while delays ranged from 0 to 25 msec. (A) Six of the 50 presynaptic spike trains used as input to the two synapse models. The bottom 5 traces are the most rapidly spiking and the top 1 trace is the least rapidly spiking cell. Spike trains were produced with a Poisson generator using the gen.mod presynaptic spike generator written by Zach Mainen. (B) Comparison of summed state variables for the two models: $\sum r_i$ (dashed line) vs. R (solid line). The former is dimensionless while the latter is in μS. (C) Comparison of summed conductance (in μS) for the two models: $\sum r_i g_i$ (dashed line) vs. $R\overline{G}$ (solid line). The curve for the complex synapse is identical to that in B since $\overline{G} = 1$. Although not apparent here, the superposition is imperfect due to time-step round-off differences between the two implementations.

The consolidated implementation presented here extends the value of the original DMS synapses in reducing this computational load.

Acknowledgments

I would like to thank Jack Wathey, Mike Hines, and Alain Destexhe for many helpful virtual discussions and the two anonymous reviewers for useful comments and corrections. Scott Deyo ran some of the benchmarks. This work was done using the NEURON simulator with support from NINDS and the Veterans Administration.

References

Bower, J., and Beeman, D. 1994. *The Book of Genesis*. Springer-Verlag, New York.

Destexhe, A., Mainen, Z. F., and Sejnowski, T. J. 1994a. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Comp.* **6**, 14–18.

Destexhe, A., Mainen, Z. F., and Sejnowski, T. J. 1994b. Synthesis of models for excitable membranes, synaptic transmission and neuromodulation using a common kinetic formalism. *J. Comp. Neurosci.* **1**, 195–230.

Hines, M. 1993. NEURON—A program for simulation of nerve equations. In *Neural Systems: Analysis and Modeling*, F. H. Eeckman, ed., pp. 127–136. Kluwer Academic Press, Boston, MA.

Knuth, D. 1973. *The Art of Computer Programming Vol. 3: Sorting and Searching*. Addison-Wesley, New York.

Niebur, E., Kammen, D. M., Koch, C., Ruderman, D., and Schuster, H. G. 1991. Phase coupling in two-dimensional networks of interacting oscillators. In *Neural Information Processing Systems* 3, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, eds., pp. 123–129. Morgan Kaufmann, San Mateo, CA.

Rall, W. 1967. Distinguishing theoretical synaptic potentials computed for different somadendritic distributions of synaptic inputs. *J. Neurophys.* **30**, 1138–1168.

Srinivasan, R., and Chiel, H. J. 1993. Fast calculation of synaptic conductances. *Neural Comp.* **5**, 200–204.

Traub, R. D., Miles, R., and Wong, R. K. S. 1987. Models of synchronized hippocampal bursts in the presence of inhibition. I. Single population events. *J. Neurophys.* **58**, 739–751.

Wang, X. J., and Rinzel, J. 1992. Alternating and synchronous rhythms in reciprocally inhibitory model neurons. *Neural Comp.* **4**, 84–97.

Received May 26, 1995; accepted August 15, 1995.