Efficient event-driven simulation of spiking neural networks

IOANA MARIAN[1] RONAN G. REILLY[1] DANA MACKEY[2]
[1]Department of Computer Science, [2]Department of Mathematical Physics
National University of Ireland
Belfield, Dublin 4
IRELAND
Ioana.Marian@ucd.ie

Abstract: - We present a general event-driven algorithm for the efficient simulation of spiking neural networks. We focus in this paper on its application to self-organizing maps. Standard event-driven approaches to simulation can significantly reduce computational time, but only when network activity is relatively low. In this article, we propose several strategies to manage efficiently, large numbers of spiking events. The simulation scales well with the increase of the neural activity and is more biologically plausible than competing methods.

Key-Words: - Event-driven simulation, spiking neurons, self-organizing maps.

1. Introduction

In recent years there has been an increasing trend in computational neuroscience towards modeling relatively large networks (e.g., 10^5) of spiking neurons [1, 2]. Consequently, the issue of simulation efficiency has become an increasing problem. Promising work has been done by creating dedicated hardware for spike-processing networks [3] or mapping the simulations onto parallel computers [2]. Performance benchmarks have mainly been applied to large networks, with low activity, regular connections and simple learning procedures [3].

In this paper we address the issue of speeding up spiking neuron simulations, when these entail high frequencies of neural activity, sparse connectivity and plastic spike-driven synapses. Specifically, we address the self-organization process emerging into a pulsed feature map. Assume one implements a simulation of a self-organizing map (SOM) that exhibits high neural activity patterns concentrated within less than 100 ms. Such a learning process entails the management of thousands of events, and together with the need for a large number of training steps, this can lead to very long simulation times. Therefore, one can think of the self-organization process simulated into a neural network as a computational task with particular efficiency issues, risen from the characteristics of the learning process.

Previous research concerned with efficient simulation of large networks of spiking neurons and plastic spike-driven synapses, similar in some respects to our simulation task, has been published in [4]. Mattia & Del Giudice [4] emphasize the additional computational effort required when the synaptic dynamics are taken into account in large-scale networks simulation.Their modality to handle the hierarchy of spikes generated in such a network is simply to discard the noisiness of spike transmission. The resulted algorithm is very efficient and might be acceptable when dealing with models where a fixed number of synaptic delays (e.g., up to 16) can cover the range of simulations aimed. But this method is of no use in the case of neurobiologically plausible large-scale simulations that involve dealing with huge number of noisy synapses. In the following we propose a simple algorithm that assumes not any simplifications of the synaptic or neural dynamics, and still scales very well with the network size and high neural activities.

2. Simulating the Spike Response Model

The nonlinear dynamics of a spiking neuron can be accurately captured by a single variable model, which has a huge computational advantage over comprehensive mathematical models, such as the Hodgkin-Huxley equations. In the formalism known as the Spike Response Method [1], the dynamics of the neuron are encoded in two sets of kernels, representing the effects on a unit of its own spikes (η_i) and those of the other neurons (ϵ_{ij}). The membrane potential V_i of the neuron i is computed at each time moment

t as follows:

$$V_i(t) = \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in F_j} w_{ij} \epsilon_{ij}(t - t_j^{(f)} - d) - \eta_i(t - t_i) \quad (1)$$

where Γ_i denotes the set of neurons presynaptic to i, F_j is the set of all firing times of neuron j and the w_{ij} account for synaptic strengths between cells. If the sum of all excitatory and inhibitory contributions reaches a threshold value, an output spike is generated by the presynaptic neuron at time t_j^f which then travels along the axon and reaches the postsynaptic neuron after a delay d. The postsynaptic response kernel ϵ evolves as a function of the difference $s = t - t_j^{(f)} - d$:

$$\epsilon_{ij}(s) = \exp\left(-s/\tau_m\right)\mathcal{H}(s), \quad (2)$$

where $\mathcal{H}$ is the Heaviside step function and τ_m represents the neural time constant. After emitting the spike a node enters a refractory period described by the kernel η which, in our simulation, is simplified and depends only on the last spiking time t_i of neuron i:

$$\eta_i(t - t_i) = \exp[-(t - t_i)^a + b]\, \mathcal{H}'(t - t_i)\, \theta. \quad (3)$$

Here, θ is the threshold and a and b are constants which give the decay rate of the refractory period. Stochastic behavior was accounted for in the model by adding noise values in the generation of the transmission delays and the refractory period.

A few attempts have been made to implement the SRM model [3,5], most of them using a continuous time framework, which guarantees a detailed analysis of the neuron behavior. In the following, we propose an optimal algorithm based on an asynchronous, efficient integration method tuned to the deterministic features of the threshold-fire units.

3. Special features of learning in a self-organizing network

Learning in a self-organizing map is a stochastic process, where the final mapping accuracy depends upon the dimension of the training sample and the number of learning steps performed. The simulation time of a traditional Kohonen network scales poorly. Same observation applies to a pulsed SOM, even when a fast learning mechanism is used, such as proposed in [6].

Let us consider the complexity of an unsupervised learning procedure applied to a pulsed network. In our scenario, the SOM winner is randomly selected among the units with the lowest firing time in a simulation step. A temporal neighborhood of the winner is created, so that only the neurons that fired until a reference time T_{out} are subject to learning. All afferent and lateral synapses of these neurons are then modified according to the learning rules. The afferent weights of the neurons are adapted in such a way as to maximize the similarity between the input postsynaptic potential (PSP) and the connection weight. The synaptic efficacy of a lateral inhibitory or excitatory connection is modified depending on the activity of the two connected neurons and upon the arrival time of the presynaptic spike. For a detailed description of the learning procedure we refer the reader to [7].

In the self-organizing process, the lateral feedback system is used as a basic mechanism for modifying over time the form of the emergent activity pattern. Given the untrained map, the neural activity start out spreading over a large part of the network, that is in our case up to 30% - 50% of the network. But in several iterations of the learning procedure, the network response to one stimulus converges to a stable activity bubble, including a relatively small subpopulation of units. If the network activity is updated just within these active areas and only when they became active, instead of computing the whole network in a time-stepped fashion, than a real speeding up of the simulation can emerge. Grounded on this simple idea, the event-driven approach profits from the focalization of the network activity and provides a suitable implementation for a SOM.

4. Continuous versus time driven protocols

4.1. Design considerations

During a basic computational cycle performed in a neural network one can depict three main phases: 1) apply the input patterns, 2) propagate the activity through the network, 3) apply the learning to the plastic synapses. We illustrate very briefly these phases with a time-driven algorithm implementation. Generally, in a continuous time approach the simulated time is increased in steps of constant size Δt and within each time bin all neurons' activities are computed and occurring spikes are recorded. The algorithm is sketched in Figure 1.

For a network with N neurons, each having S synapses with one filter ϵ and F_j non–negligible firing times, the algorithm complexity estimated for a

```
CT = StartTime;
while CT < TimeOut do
    for all ActiveInputs[$l$] do
        propagate_pattern($l$);
    for all Neurons[$i$] do
        for all InputSynapses[$j$] do
            check_activity($j$) & set_delay($j$);
            for all FiringTimes[$k$] do
                add_PSP($k$) to V($i$);

        if Neuron[$i$] fire then
            record_firing_time($i$);

    CT += TimeStep;
od while
for all SelectedNeurons[$i$] do
    apply_unsupervised_learning($i$);
```

Figure 1: Continuous time algorithm with Δt = 0.1 ms and receiver-oriented connectivity.

```
1  CT = StartTime; SL = nul;
2  while CT < TimeOut do
3      if new_pattern_needed(CT, SL) then
4          for all Inputs[$l$] do apply_pattern($l$);
5      if e = first_event(SL) then
6          if first_integration(e.unit) then
7              apply_first_algorithm(e.unit);
8          else
9              compute_decayed_V(e.unit);
10             add_PSP(e) to V(e.unit);
11         if e.unit fire then
12             for all OutputSynapses[$k$] do
13                 insert_spike_order(SL);
14     CT = e.time;
15 od while
16 for all SelectedNeurons[$i$] do
17     apply_unsupervised_learning($i$);
```

Figure 2: Basic event-driven algorithm implementing a sender-oriented strategy.

time bin is $O_1(N \cdot S \cdot F_j)$. If we work at time resolution Δt, than computing the whole network in a time interval T entails a complexity of $O_1 \cdot T/\Delta t$. This basic line complexity can be decreased in two ways. We begin with reducing the computation time of a single unit by carefully choosing the neuron connectivity scheme and using an efficient integration method. If this condition is fulfilled, then we aim to reduce the number of neuron states that are computed during a simulation step. This goal is usually achieved using an event-driven approach [8] and is discussed in the next section of the paper.

The local connectivity of the network has a significant effect on the unit integration time. We implemented 10% probabilistic connectivity, with short range excitatory connections and long range inhibition. Using such a sparse scheme, we need a dedicated structure to specify the list of connections. As has been pointed out in [2,3], for pulsed networks the sender-oriented method, holding the values of the neuron output synapses, e.g., weights and delays, proves to be most efficient.

Furthermore, the integration method can be optimized by exploiting the deterministic nature of the neural model. Between any two firing moments, the neuron's depolarization has a deterministic evolution. Therefore, rather than computing the sum over all the presynaptic inputs at each time step, the sum of the past spikes is stored, and decayed every time when the current PSP contribution has to be added.

4.2. A basic event-driven algorithm

The main lines of the algorithm are outlined in Figure 2. In the event-driven approach the integration of a unit activity is performed in an asynchronous way, triggered by the receival of one or several spike-events (see line 5 Fig. 2). The core of the algorithm consists of processing the spiking events from a chronologically ordered list SL (lines 5, 13 Fig. 2). Each new spike is fully characterized by a time stamp representing the delivery moment and the index of the target unit. The input patterns are applied when the event list becomes empty or when the current time of the simulation exceeds the next pattern time stamp (lines 3, 4 Fig. 2).

The SOM network is presented with the input patterns in such a way that it favors an accumulation of noisy input signals at the beginning of each training pattern. These spikes accumulate and are computed during a first integration step (lines 6, 7 Fig. 2) using equations (1) & (2). The membrane potential from time t_i is stored and on the arrival of a new spike, this value is decayed by the formula $V_i(t) = V_i(t_i) \cdot exp[-(t-t_i)/\tau_m]$ and added to the postsynaptic spike contribution. Apart from decreasing the computational time, this method has the advantage of minimizing the memory load. Instead of storing the last F_j firing times for all j presynaptic nodes and the corresponding transmission delays, we only keep four values: the target unit and the delivery time t stored in the spike structure, and the last decayed sum $V_i(t_i)$ together with the corresponding integration moment t_i, values kept by each active unit.

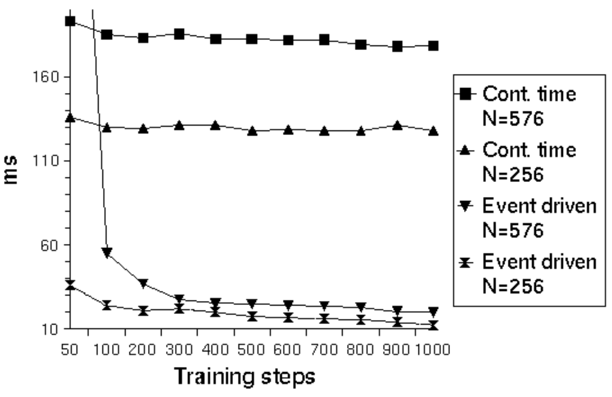

Figure 3: Computational times per cycle, for the event and time driven algorithms, for N = 256, 576 units.

An approximation of this algorithm complexity for the simulation of one time bin is given by $O_2(aN \cdot S \cdot [1 + \log_2 length(SL)])$, where a is the average network activity in a time slice. The first term designates the computational effort employed by the integration of all units that receive spikes at a certain moment of the simulation (lines 5-10 Fig. 2). The second term of the sum represents the additional effort required to insert spikes in order in the event list (lines 11, 13 Fig. 2). Simulation of the whole network for a time period T entails the complexity $O_2 \cdot T / \delta t$. Here δt represents the time resolution used in the generation of noisy delays and input signals.

We can see immediately that the basic line complexity $O_1(N \cdot S \cdot F_j)$ of the continuous time algorithm, has been decreased by computing the activity of only a percent aN of the whole network, and eliminating the complexity entailed by checking for each synapse all firing times F_j.

Figure 3 illustrates a comparison of the two algorithms when used to train self-organizing networks with different sizes. Relatively small networks have been used mainly because, due to the low dimension of the input patterns set, the self-organization of a larger SOM would fail. Moreover, a network with 0.6 k units, 10% connectivity and an activity $a = 30\%$ entails a computational effort equivalent to the simulation of a 16 k units network with same connectivity and 0.5% activity. An input set of 12x20 time-coded patterns was used to train the SOM to represent 12 different directions of movement. For the details of learning procedure see [7].

The results in Figure 4 illustrate the good scaling of the event-driven simulation with the change of the activity pattern in SOM. In contrast to this behavior, the time spent in the synchronous simulation (first two graphics in Fig. 3) is proportional only to the network dimension and is independent of its activity. Note that during the first learning cycles when the network exhibits a high activity (e.g., 100 Hz) this event-driven algorithm applied to the larger network (576 units) scales more poorly than the continuous approach. This is due to the additional complexity $O(\log_2 length(SL))$ entailed by the management of the spike-event list. These observations are consistent with those of other authors [3, 4], which assume that only spiking neural networks with low activity might be suited to an event-driven simulation. Despite of these theoretical predictions, we present in the next section an algorithm with a very good scalability in the case of high neural activity and more important which does not require any simplification of the neural model.

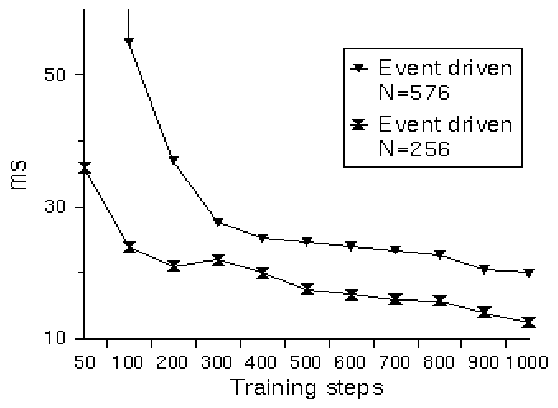

Figure 4: Illustration of event-driven algorithm scaling with the change in the network activity. Computational times per cycle are presented for first 1000 training steps, when activity decreases from 100 HZ to 33 Hz (not shown in the picture).

5. How can we deal with high neural activity patterns ?

Two strategies are proposed to reach the goal of efficient simulation of a neural network that exhibits firing rates of 100 HZ. Both of them address the most time- expensive process in the event-driven policy, namely the management of the event list.

Multiple spikes. A straight-forward asynchronous simulation can generate a maximum number of $aN \cdot S$ action potentials in each time step, where aN gives the number of firing units and S the average number of synapses per neuron. Instead of creating a specific event to handle each of these spikes, several PSPs can be accumulated in a single structure and delivered together to the target unit, but only if their time stamp matches. A similar concept was previously formulated by Schoenauer [3] and defined as weight caching. We utilize the concept of a $multiple$ $spike$, to store the list of all m synapses weights that deliver a PSP to a certain neuron i, at a given time moment t. Therefore, the computational load per time step is reduced to $aN \cdot S/m$. In the most favorable scenario, m can equal S and consequently the simulation scales very well when the network activity increases. On the other hand, when m represents just a very low percentage of S, this method will not bring a significant performance improvement. The main parameters affecting this value are the topology of local connections and the time resolution used. A possible trade-off might result from using a larger time step (e.g., 1 ms), which would increase the probability of aggregating spikes and hence, decrease the length of the list.

Quick sorting an unordered pool. As we noticed from our first experiments, the length of the spike list can grow tremendously in the case of high network activity, even for a small network with N = 576 units. Therefore, we have to deal with an ordered insertion in the growing list, of complexity $O(\log_2 length(SL))$ multiplied by a factor of $aN \cdot S \cdot T/\delta t$. One immediate improvement, resulting in a speed-up of up to 4 times, can be achieved if we search the list selectively from the start or from the end, depending on the new spike time stamp.

A solution to this problem was proposed in [4], using not only one event list, but several FIFO queues associated with a set of ordered delays. Even if this proposal gives rise to a very efficient algorithm, it assumes an important computational simplification, namely the existence of a limited set of fixed values for the axonal delays. In [7] we have pointed out that fine tuning of the spiking neurons weights to the input stimuli features can be achieved only with a high randomness and noisy factors, including the existence of noisy delays. The solution proposed in [4] is acceptable only for a limited set of applications, where noisiness can be discarded, but we believe that majority of biologically plausible large-scale simulations cannot get into this frame.

Hence we consider that an efficient algorithm has to deal with noise factors and we propose here such a method with a very good scalability. Instead of inserting each new event in the right position in the list, with the corresponding complexity of $O(\log_2 length(SL))$, we just add it to an unordered pool of spikes, entailing complexity $O(1)$. Since the events have to be processed in chronological order, at equal intervals of length T_{window}, the main process stops from processing spikes, and selects those events from the pool that have to be computed in the next period, corresponding to $t + T_{\text{window}}$. The T_{window} value is similar to the safe window concept used in parallel simulations and guarantees the temporal correctness of the algorithm. The selection of the spikes is performed using a quick sort algorithm. Most importantly, we run the sorting algorithm only on a small percentage q of elements in the pool, namely those who have the time mark within the next processing interval. Thus, instead of dealing with the insertion complexity of $O(aN \cdot S \cdot \log_2 length(SL) \cdot T/\delta t)$, our quick sorting method reduces it to $O(qN_p \cdot \log_2 qN_p)$ multiplied by T/T_{window}. The T_{window} interval can be set at 1 or 2 ms which means 20 times bigger than the time resolution δt. Given the low value of q, the sorting computational effort remains low and is almost independent of the pool size N_p.

6. Evaluation of the algorithm

For an evaluation of the strategies discussed above, we use a measure of the computational effort, given by the time required to compute N_k units, e.g., 1,000 neurons in our case.

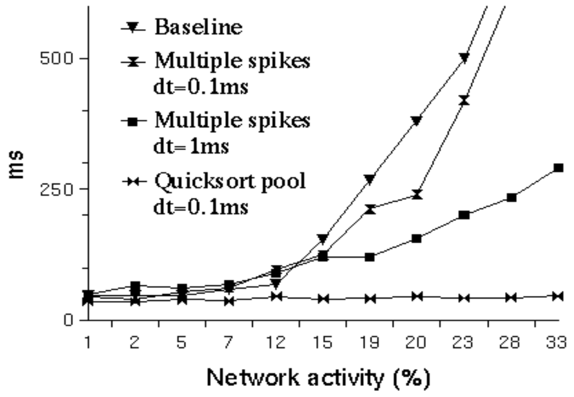

Figure 5: Computation times for updating 1000 units vs. levels of network activity, with different event handling methods applied. The network activity is measured as the average number of spikes in one ms divided by the total number of neurons for N=576.

Our findings, presented in Figure 5, reveal a gradual increase in the performance of up to 20 times, when the above strategies are added one by one to the implementation. Note that, if the multiple spikes strategy is applied to a series of events generated with a fine time resolution (0.1 ms) the probability of spikes accumulating is still low, leading to an average improvement of 30%. Only when we increase the time step to 1 ms does the method start to prove really efficient. If one needs to maintain a high time resolution, than this method together with the quick sorting of the pool is recommended.

Combining these two strategies we obtain the minimal time complexity

$$O_3(aN \cdot \frac{S}{m} \cdot \frac{T}{\delta t} + qN_p \cdot \log_2 qN_p \cdot \frac{T}{T_{\text{window}}})$$

with q taking values from 7% to 25% of the total number of events N_p in pool. One can see in Figure 5 (graphic 4) that through this means the simulation time scales very well with the increasing of network activity.

The table in Figure 6 presents a rough comparison of our results with the times reported in [4]. For similar networks, with respect to size, connectivity and learning complexity, we compare the computational effort required to simulate all operations entailed by the firing, at different frequencies, of N_f neurons.

N (1k units)	N$_f$ (1k units)	Layered delays (ms)	QuickSort alg. (ms)
0.6	1.0	0.4	1.0
1.1	2.4	0.9	2.6
2.2	5.0	1.7	3.3
3.0	7.0	2.1	4.3
4.0	9.5	2.8	5.2
5.0	12.0	3.5	6.3

Figure 6: Execution times per neuron vs. size of the network N, when N$_f$ neurons fire. For the layered-delays algorithm we refer the reader to [4]. Note that the average firing rate in the layered delays simulation is 3 Hz, whereas the execution times for Quick sorting algorithm are recorded when a neural activity of 100 Hz is present.

The main strength of the Mattia & Del Giudice algorithm resides in the layered structure, with 4 to 16 transmission delay values, specifically designed for computational simplicity, although this was done at the expense of biological plausibility. The times reported in [4] are obtained for an average spiking frequency of 3 Hz. The key features that distinguish our algorithm from the one reported in [4] reside in: dealing with noisy synaptic transmission, without any reduction of the noisiness in the model and the simulation of high activity patterns up to 100 Hz. As an example of the complexity entailed, for $N = 4$k units a spiking frequency of 100 Hz generates a pool of events in the order of 220,000 elements. Our algorithm scales linearly with $2{\cdot}N$ and it manages to keep the simulation time approximately twice as big as in [4], in the conditions of a 30 times higher frequency.

7. Discussion

We have shown that significant improvements in the simulation of a pulsed SOM network can be achieved by using an event-driven framework.

The asynchronous simulation scales very well with the change in activity level and benefits from the decrease in the firing frequency of the neurons. Our findings are consistent with other event-driven simulations performed for spiking neurons [2,3].

Furthermore we address the specific case of high neural activity patterns occurring in certain stages of the SOM simulation. We present several strategies (multiple spikes, spike integration method and quick sorting pool) that reduce the simulation time by up to 20 times. Is proposed a simple method to manage and update the events structure, without incurring the normal insertion overhead. Unlike the algorithm described in [4], our method does not assume any simplifications of the network dynamics and parameters. The resulting algorithm scales linearly with the network size, when activity increases up to 33%. We believe that these findings will support efficient event-driven simulation of spiking neural networks when the generated events are in the range of 10^6.

References:

[1] W. Gerstner, Spiking neurons, *Pulsed Neural Networks*, W. Maas & C.M.Bishop (Ed.), Cambridge, MA: MIT Press, 1999, pp. 4-48.

[2] A. Jahnke, U. Roth & H. Klar, Towards efficient hardware for spike-processing neural networks, *Proceedings WCNN' 95*, Washington, USA, 1995.

[3] T. Schoenauer, S. Atasoy, N. Mehrtash & H. Klar, MASPINN: Novel Concepts for a Neuro-Accelerator for Spiking Neural Networks, *Proc. VIDYNN' 98*, Stockholm, June 1998.

[4] M. Mattia & P. Del Giudice, Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses, *Neural Comp.*, 12, 2000, pp. 2305-2329.

[5] C. Fohlmeister, W. Gerstner, R. Ritz & J. van Hemmen, Spontaneous excitations in the visual cortex: stripes, spirals, rings and collective bursts, *Neural Comp.*, 7: 905-914, 1995.

[6] B. Ruf & M. Schmitt, Self-organization of spiking neurons using action potential timing, *IEEE Transactions on Neural Networks* 9:3, 1998, pp. 575-578.

[7] I. Marian & R. G. Reilly, Self-organization of neurons coding directional selectivity in motor cortex, *Proc. AICS 2001*, NUIM, Ireland, Sept. 2001.

[8] A. Ferscha & S. Tripathi, Parallel and distributed simulation of discrete event systems, *Tech. Rep. CS-TR-3336*, University of Maryland, 1994.