Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 19 (2006) 1636–1647
Neural Networks
www.elsevier.com/locate/neunet

Fast computation of a gated dipole field

George Mengov[a,*], Kalin Georgiev[b], Stefan Pulov[c], Trifon Trifonov[b], Krassimir Atanassov[d]

[a] *Department of Statistics and Econometrics, Faculty of Economics and Business Administration, Sofia University, 1113 Sofia, Bulgaria*
[b] *Department of Computer Informatics, Faculty of Mathematics and Informatics, Sofia University, Sofia, Bulgaria*
[c] *Object Builder Software – Bulgaria, Sofia, Bulgaria*
[d] *Biomedical Informatics Section, Prof. Ivan Daskalov Centre of Biomedical Engineering, Bulgarian Academy of Sciences, Sofia, Bulgaria*

Received 12 January 2004; accepted 8 May 2006

Abstract

We address the need to develop efficient algorithms for numerical simulation of models, based in part or entirely on adaptive resonance theory. We introduce modifications that speed up the computation of the gated dipole field (GDF) in the Exact ART neural network. The speed increase of our solution amounts to at least an order of magnitude for fields with more than 100 gated dipoles. We adopt a 'divide and rule' approach towards the original GDF differential equations by grouping them into three categories, and modify each category in a separate way. We decouple the slow-dynamics part — the neurotransmitters from the rest of system, solve their equations analytically, and adapt the solution to the remaining fast-dynamics processes. Part of the node activations are integrated by an unsophisticated numerical procedure switched on and off according to rules. The remaining activations are calculated at equilibrium. We implement this logic in a Generalized Net (GN) — a tool for parallel processes simulation which enables a fresh look at developing efficient models. Our software implementation of generalized nets appears to add little computational overhead.

Keywords: Gated dipole field; Adaptive resonance theory; Generalized net

1. Introduction

The continuous-time behaviour of neural circuits is often described with systems of ordinary differential equations, usually integrated numerically as their complexity rules out analytical solutions. Today's software packages that do this are of high quality but require substantial computational resources. Naturally, a demand develops for algorithmic modifications aimed at efficiency.

One example for a set of computationally intensive tasks are the models based on adaptive resonance theory (Grossberg (1976); for an overview on ART see for example Carpenter and Grossberg (2002)). Their implementations have addressed the need for computational economy in a number of ways. One has been to retain differential equations for only the adaptive weights and use equilibrium solutions for all activations as in ART 2 (Carpenter & Grossberg, 1987). ART 2-A (Carpenter, Grossberg, & Rosen, 1991) has excluded the differential equations altogether. In some recent examples (Grossberg & Raizada, 2000; Grossberg & Seitz, 2003; Grossberg & Williamson, 2001) the fastest cell reactions have been computed at steady state, other activity equations have been solved with the Runge–Kutta–Fehlberg 4–5 method, and adaptive weights have been solved at a reduced time scale with Euler's method. However, computational complexity still remains an issue that limits simulations to relatively small neural networks. In the case of a complex model with even moderate dimensionality one may have a situation where "each simulation... takes from a day to a month to run on a 1.4 GHz Athlon processor" (Grossberg & Seitz, 2003).

Some ART implementations solve numerically all differential equations but this approach has worked for relatively small-scale tasks as in Exact ART (Raijmakers & Molenaar, 1994; Raijmakers, van der Maas, & Molenaar, 1996; Raijmakers & Molenaar, 1997). These authors have developed a realistic continuous-time model and have used a software package for stiff problems. Naturally, their implementation requires a lot of computing resources. Raijmakers and Molenaar did not consider it as a problem because their objective had not been

* Corresponding author. Tel.: +359 971 1002x431.
E-mail address: g.mengov@feb.uni-sofia.bg (G. Mengov).

0893-6080/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2006.05.031

an efficient model, but just a correct working model. However, the issue of efficiency quickly comes up when one needs increased task dimensionality. In the present paper we propose a set of modifications reducing the computational load when solving numerically the gated dipole field equations in Exact ART.

The gated dipole was introduced by Grossberg (Grossberg, 1972) to explain complex temporal processes in conditioning and perception. Its main feature is opponent processing whereby disappearance or unexpected absence of a positive (negative) reinforcer can cause negative (positive) emotion. The gated dipole has been modified to serve different purposes. Applications include, among others, modeling Pavlovian conditioning (Grossberg & Schmajuk, 1987), motor control (Gaudiano & Grossberg, 1991), vision (Ögmen, 1993; Ögmen & Gagne, 1990), and consumer decision making (Leven & Levine, 1996).

Neural circuits that model different behaviours may contain arrays of many gated dipoles. In (Grossberg, 1980) a gated dipole field (GDF) was proposed as an implementation of the category layer in ART. Raijmakers and Molenaar (1994, 1997) developed this idea into a working model. Ögmen (1993) proposed a modified array of dipoles to account for retino–cortical dynamics. Couples of interacting dipoles were used in different ways in (Leven & Levine, 1996) and (Gaudiano & Grossberg, 1991).

Our algorithmic modifications are in part or entirely implementable in all of the above cases, should the need for efficient computation arise. The issue there would probably be to devise algorithms for fast computing of the circuitry outside the GDF. In another paper we will show how this can be done for Exact ART.

The processes in a GDF develop at different time scales, which can be separated and computed in parallel. Here we decouple the slow neurotransmitters from the neuron activations. We calculate the former with an analytical formula, and divide the latter into two groups: (1) activations calculated at steady state; (2) activations integrated by unsophisticated numerical procedure, switched on and off. We implement this logic in a Generalized Net, or GN (Atanassov, 1991), a tool for simulating parallel processes of both discrete and continuous nature. A GN software implementation proved the efficiency of our approach. This paper also presents the first real-time simulation of a GN model in neuroscience. It builds upon previous efforts to model information transfer in the brain with generalized nets (Atanassov, in press; Mengov, Pulov, Atanassov, Georgiev, & Trifonov, 2003; Sgurev, Gluhchev, & Atanassov, 2001).

This work deals not with an entire ART architecture but only with the GDF circuit. In the following sections we give an overview on GDF and present modifications for its efficient computing. We compare our simulation procedure with the standard numerical methods built in Mathematica. In an Appendix we discuss generalized nets and outline the particular GDF GN model.

2. Gated dipole field

2.1. The Raijmakers & Molenaar model

An array of gated dipoles could perform the winner-take-all dynamics needed in the category layer of a continuous-time ART network (Grossberg, 1980). Raijmakers and Molenaar (1994, 1997) implemented this idea in their Exact ART system whereby a GDF performs three functions:

1. The dipole with the strongest input signal should win the competition and suppress all other dipoles. In this way its neuron connected to the adaptive weights can stay switched on until reset, and allow weight updating.
2. A reset signal should be able to suppress all dipoles indiscriminately.
3. After a reset, the immediately preceding winner should remain deactivated, and the dipole with the next strongest input should win.

The following equations comprise the Raijmakers & Molenaar model:

$$\frac{dy1_j}{dt} = -y1_j + y5_j + A_E \tag{1}$$

$$\frac{dy2_j}{dt} = -y2_j + [y6_j]^+ + A_E \tag{2}$$

$$\frac{dy3_j}{dt} = -y3_j + z1_j y1_j \tag{3}$$

$$\frac{dy4_j}{dt} = -y4_j + z2_j y2_j \tag{4}$$

$$\frac{dy5_j}{dt} = -Ay5_j + (B - y5_j)((y5_j)^2 + y3_j + eI_j) - y5_j\left(\sum_{l\neq j}^{M}(y5_l)^2 + y4_j\right) \tag{5}$$

$$\frac{dy6_j}{dt} = -y6_j + (y4_j - y3_j) \tag{6}$$

$$\frac{1}{\varepsilon}\frac{dz1_j}{dt} = \beta(\gamma - z1_j) - \delta[y1_j - \Gamma]^+ z1_j \tag{7}$$

$$\frac{1}{\varepsilon}\frac{dz2_j}{dt} = \beta(\gamma - z2_j) - \delta[y2_j - \Gamma]^+ z2_j. \tag{8}$$

Here $[\ldots]^+$ denotes the rectification operator:

$$[\xi]^+ = \max\{\xi, 0\}.$$

The other notation is as follows. Variables $y1_j, \ldots, y6_j$ represent node activities, $z1_j$ and $z2_j$ are chemical transmitters, A_E is arousal signal, I_j is dipole input signal. Constants: A is decay rate of neurons $y5_j$, B is maximum level of neuronal activity, e is input signal strength, β is transmitter increase, γ is maximum transmitter quantity, δ is transmitter decrease, Γ is activity threshold, ε is neurotransmitter speed, M is number of gated dipoles in the field, and $j = 1, \ldots, M$ is index of dipole elements. Index J designates the current winner. Bold characters indicate vectors (**y1**–**y6**, **z1**, **z2**, **I**). This notation coincides with the one introduced in Raijmakers and Molenaar

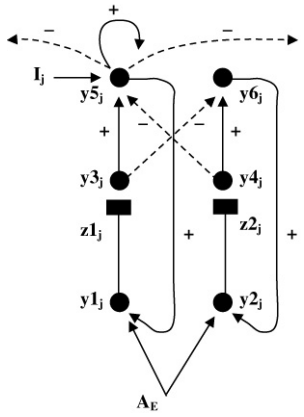

Fig. 1. The Raijmakers & Molenaar model of the gated dipole field.

(1997) to facilitate the understanding of our paper. The model is shown in Fig. 1.

Note that this GDF configuration differs from what Grossberg (1980) proposed — here the transmitter change is also function of activity $y5$ rather than $y1$ only, due to the feedback $y5 \rightarrow y1$ (Fig. 1). Raijmakers and Molenaar (1994) needed this to implement GDF function No 3 from above. Other gated dipole arrays that fulfill the same three functions, for example (Leven & Levine, 1996), may also benefit from this change (Fig. 1) and hence from all the modifications proposed in this paper.

2.2. Modifications

Our approach sought to achieve computational efficiency due to joint action of three modifications to Eqs. (1)–(8), discussed in the following sections. We implemented the GDF in a generalized net operating in a fixed discrete-time scale. Its time step coincided with that of the numerical integration procedure. In our further discussion we rely on the fact that the computation process is stepwise.

2.2.1. Neurotransmitters computed with an analytical formula

We solved the neurotransmitter Eqs. (7) and (8) analytically and adapted the boundary value solution to account for a continuously changing input. When an arousal burst causes GDF reset, all dipoles compete for a short period. One dipole wins and its $y5_J$ stays active long enough to update its connected memory weights. During that period $z1_J$ is consumed, while all other transmitters $z1_j$, $j \neq J$, refill towards equilibrium. Thus, for some time the GDF operates in a regime when its slowest process is continuously changing while its fast node activations stay constant. The signals affecting the release and replenishment of **z1** are constant.

Consider the interaction in the jth pathway $y1_j \rightarrow z1_j$ (j may also be J). The input into $z1_j$ depends on node $y1_j$ (Fig. 1) and is $S1_j = \delta[y1_j - \Gamma]^+$ as per Eq. (7). With this substitution Eq. (7) becomes

$$\frac{1}{\varepsilon}\frac{\mathrm{d}z1_j}{\mathrm{d}t} = \beta(\gamma - z1_j) - S1_j z1_j. \tag{9}$$

Let $S1_j$ change at moment t_0 and stay constant for long enough. Term $S1_j^{\text{old}}$ denotes the input value at time step $t_0 - 1$ and $S1_j^{\text{new}}$ at t_0. While in general $S1_j$ may change abruptly, $z1_j$ cannot, and therefore

$$z1_j(t_0 - 1) \approx z1_j(t_0). \tag{10}$$

For $S1_j = \text{const}$. Eq. (9) has this solution:

$$z1_j(t) = \frac{\beta\gamma}{\beta + S1_j^{\text{old}}} \exp\Big(-(t - t_0)(S1_j^{\text{new}} + \beta)\varepsilon\Big) + \frac{\beta\gamma}{\beta + S1_j^{\text{new}}}\Big[1 - \exp\Big(-(t - t_0)(S1_j^{\text{new}} + \beta)\varepsilon\Big)\Big]. \tag{11}$$

Grossberg (1984) used essentially the same formula to describe transmitter release for a new sustained input. Eq. (11) expresses the gradual shift of $z1_j$ from equilibrium with $S1_j^{\text{old}}$ to equilibrium with $S1_j^{\text{new}}$. If the rate of input change is high the transmitter does not have time to reach its new asymptote $\beta\gamma/(\beta + S1_j^{\text{new}})$, and the formula is inapplicable. However, with a modification, the latter can account for a continuously changing input. Note that Eq. (11) can take $S1_j^{\text{new}}$, corresponding to time step t_0, and cannot take $S1_j^{\text{old}}$ for $t_0 - 1$ as the transmitter could not habituate. We introduce an equivalent hypothetical $\hat{S}1_j^{\text{old}}$ defined as the signal which, had it been maintained for sufficiently long, would have equilibrated the transmitter exactly to its value at $t_0 - 1$. In other words, we consider $z1_j(t_0 - 1)$ being the product of a different history but with the same outcome. In that 'alternative past', a *finished* habituation produced 'mock' equilibrium

$$z1_j(t_0 - 1) = \frac{\beta\gamma}{\beta + \hat{S}1_j^{\text{old}}}.$$

Therefore the needed equivalent value is

$$\hat{S}1_j^{\text{old}} = \frac{\beta\left(\gamma - z1_j(t_0 - 1)\right)}{z1_j(t_0 - 1)}.$$

In summary, at each time moment we calculate the new $z1_j$ in two steps. First, the actual previous $z1_j(t_0 - 1)$ is used to determine an adjusted previous $\hat{S}1_j^{\text{old}}$. Then in the second step the new $z1_j(t_0)$ is computed by Eq. (11) with $\hat{S}1_j^{\text{old}}$ and $S1_j^{\text{new}}$. The same is done with transmitter $z2_j$. This procedure decouples Eqs. (7) and (8) from the rest of the system (Section 2.1). In practice $\hat{S}1_j^{\text{old}}$ can be computed at a slower time step, for example only at moments when the reset signal A_E is activated and then switched off. And even less frequent $\hat{S}1_j^{\text{old}}$calculation can be satisfactory, for example only in the events of winner change.

2.2.2. Fast node activations

We simplify Eqs. (3), (4) and (6) by setting the derivatives to zero. The error thus introduced vanishes very quickly and has no effect on the circuit performance. It is seen from Fig. 1 that nodes $y3_j$ and $y4_j$ receive signals from nodes $y1_j$ and $y2_j$ conveyed by transmitters $z1_j$ and $z2_j$ respectively. Grossberg and Gutowski (1987) and Grossberg and Schmajuk (1987)

have shown that if the transmitters obey equations such as (7) and (8), the mathematical product of transmitter and input can adequately approximate the conveyed signal. Therefore Eqs. (3) and (4) can become

$$y3_j = z1_j y1_j$$
$$y4_j = z2_j y2_j.$$

This simplification is justified because transmitters $z1_j$ and $z2_j$ change at a rate, orders of magnitude slower than all node activities. For example, Raijmakers and Molenaar (1997) simulated Eqs. (7) and (8) with $\varepsilon = 0.001$, thereby introducing speed difference of 2–3 orders of magnitude. In fact much less difference (and greater ε) is still acceptable.

Eq. (6) can be simplified similarly:

$$y6_j = y4_j - y3_j. \tag{12}$$

Grossberg and Gutowski (1987) used exactly the above equation for $y6_j$. Node $y6_j$ emits a positive signal only after arousal A_E had come and is gone. Eq. (12) produces adequately the antagonistic rebound (important for analyses outside the scope of this paper). The introduced small error has absolutely no effect on the performance of the three GDF functions (Section 2.1). In the Raijmakers and Molenaar (1994) analysis $y6_j$ could be removed entirely.

2.2.3. Numerical integration switched on and off

The third modification affects neurons that function in alternating regimes of transient competition and equilibrium. We implement rules for switching on and off the numerical integration. It is these rules — and not the numerical method, that account for the abrupt variable changes when the transients begin and cease. Therefore the numerics could be unsophisticated — Runge–Kutta 4 was acceptable, which greatly reduced the overall computational load.

Raijmakers and Molenaar (1997) have shown that after competition has finished, vectors **y1**, **y2**, and **y5** simultaneously reach equilibrium maintained indefinitely until new A_E reset occurs. This is in accord with the ART 2 design principle "stable choice until reset" (Carpenter & Grossberg, 1987). Then due to A_E, new competition takes place and is eventually won by a new node $y5_J$, which approaches 1, while all other $y5_j$, $j \neq J$, are quenched to almost 0. From that moment t onward numerical integration is not needed and we switch it off. The winner $y5_J$ obtains value 1, and the rest $y5_j$ obtain 0.

In practice we set a threshold, θ_H of 0.95, which must be exceeded by a $y5_j$ for this switch to happen. At the same time all other $y5_j$ must fall below another threshold θ_L, equal to 0.05. The exact rule for switching the numerical procedure off is a bit more complex, because it must also account for the $y1_j$ and $y2_j$ behaviour:

$$\text{IF } \begin{array}{l} \forall j (y1_j, y5_j \in [0; \theta_L] \cup [\theta_H; 1] \,\&\, y2_j \in [0; \theta_L]) \,\& \\ \exists J (y1_J \in [\theta_H; 1] \,\&\, y5_J \in [\theta_H; 1]) \end{array}$$
$$\text{THEN switch numerical procedure off.} \tag{13}$$

The above rule is valid for $y1_j$, $y2_j$, and $y5_j$ activations in the interval [0, 1], which is achieved with the choice of simulation parameters (Appendix B). Numerical integration is resumed when there is a reset signal $A_E > 0$ released indiscriminately towards all dipoles:

IF $A_E > 0$ THEN switch numerical procedure on,

ELSE keep the current **y1**, **y2**, and **y5** values.

As A_E increases abruptly from 0 to 1 all signals along pathways $y1_j - y3_j - y5_j$ also increase and cause interaction among the $y5_j$ nodes via the inhibitory and excitatory connections (Fig. 1). While all dipoles compete, the last winner is at a disadvantage due to its exhausted $z1_J$. Hence a dipole with the next largest eI_j in Eq. (5) wins (function No 3 from Section 2.1).

A_E must be strong enough to ensure that $y5_J$ would fall below another $y5_j$, $j \neq J$, the next winner. Raijmakers and Molenaar used a gaussian pulse with sufficient A_E 'energy'. We preferred a rectangular pulse that ceased only after the current winner had lost the competition, i.e., $y5_J < y5_j$ (GDF function No 2). Our choice introduced a flexibility needed in experiments with higher GDF dimensionality where the on-centre off-surround interactions require increased A_E 'energy' (and duration) to overturn the winner. A rectangular signal was also more convenient for implementation in a mixed logical–numerical system.

3. A generalized net model of the gated dipole field

As our goal was to speed up the GDF computation we employed Generalized Nets in a software simulator. The process of rethinking and reformulating the task in GN terms enabled us to take a different look at the neural interactions and to develop our modifications. The latter could also be implemented without the GN framework, however, we retained it for two reasons. First, GNs can be useful in other neural modeling tasks (Atanassov, in press; Sgurev et al., 2001) and therefore it was important to have some estimate of their performance in general, and of their computational overhead in particular. Second, GNs are suitable for parallel machine algorithms (Atanassov, 1991), and though we used an ordinary PC in this case, hardware with parallel processors remains an option. We give a description of the GN implementation of GDF in Appendix A. Source code and GDF GN demo are available from http://www.clbme.bas.bg/Projects/gnifs/gn/GNsoftware.html.

3.1. Model simulation

The simulation of our GN model showed accurate performance of the three GDF functions. Fig. 2 presents the time course of **y5** as computed by Mathematica's Runge–Kutta–Fehlberg 4–5 method (top), and by GN with intermittent activation of the Runge–Kutta 4 procedure (middle and bottom). We simulated a field of 100 gated dipoles receiving constant input $\mathbf{I} = [0, 0.01, \ldots, 0.99]^T$. The field was subjected to a sequence of A_E bursts, which allowed the different $y5_J$ winners to equilibrate at value 1 as per condition (13), and then remain stable for some time. The other $y5_j$ nodes competed, but were eventually quenched. All winners

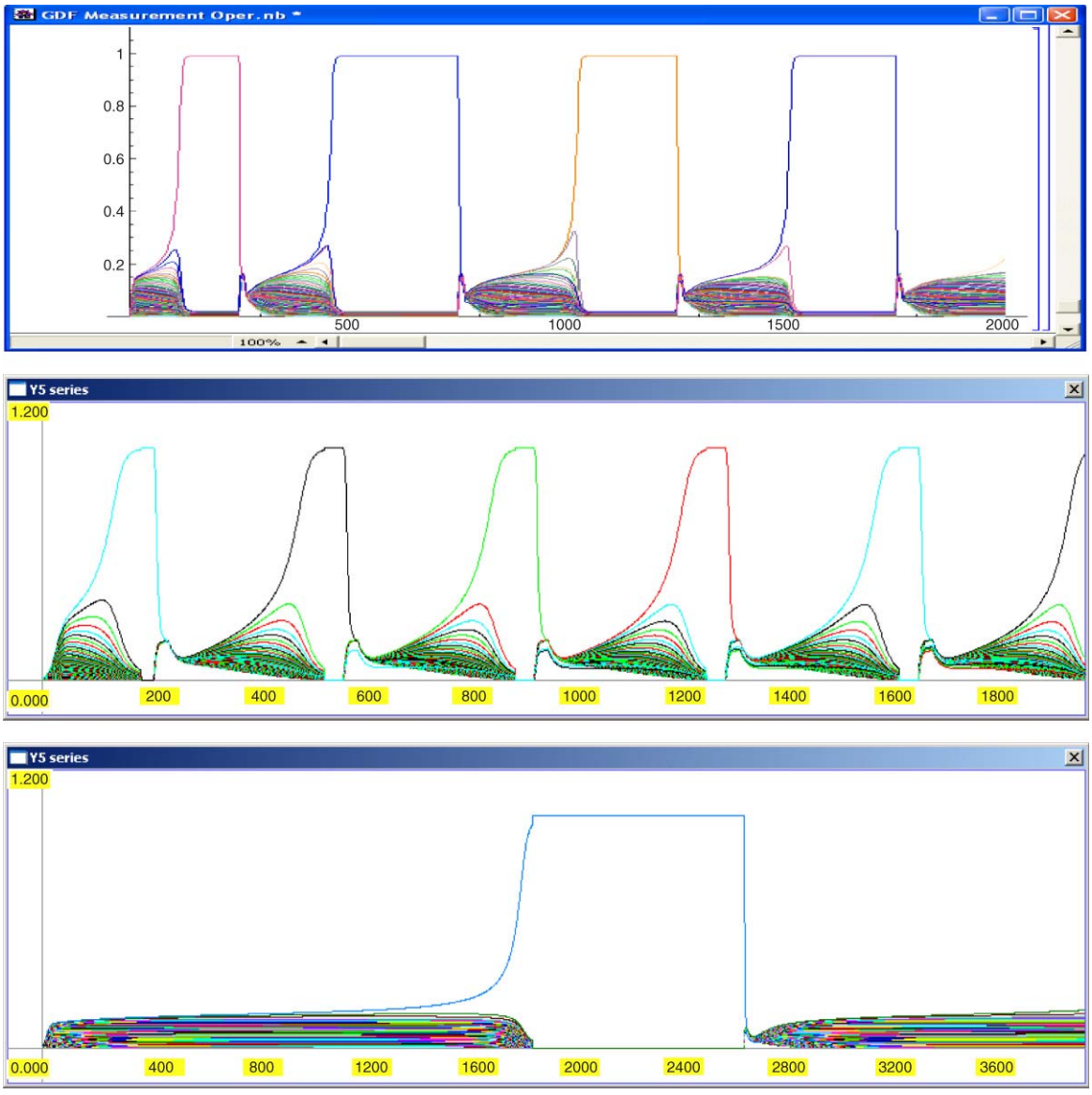

Fig. 2. Time course of **y5** in gated dipole competition. Top: Mathematica simulation with 100 dipoles. Middle: Generalized net simulation with 100 dipoles. Bottom: Generalized net simulation with 600 dipoles.

are clearly discernible in Fig. 2 (top and middle) due to their different hue (or colour in the demo). It is evident that both in Mathematica and GN simulations a new winner emerges after each competition.

The essential difference between these two approaches was in the behaviour of the quenched $y5_j$ nodes. In uninterrupted numerical integration (Fig. 2, top) some of them never reached zero. The GN, in contrast, used switching off rule (13), which blocked the integration procedure and automatically assigned '1' to the winner and '0' to all other **y5** elements (Fig. 2, middle).

4. Speed comparison

Because the ART model of Raijmakers and Molenaar (1994, 1997) consists entirely of differential equations it may be considered plausible and, in that sense, realistic. Our GDF was designed to be its computationally efficient variant with respect to the three GDF functions (Section 2.1). Therefore it is important to compare integration procedures with regard to the speed with which a task is finished.

With all modifications (Section 2.2) it was reasonable to expect that the GN would be much faster than traditional numerical procedures. It is worth noting that our modifications are invariant of the programming language used. Had we worked directly in a C++ environment rather than GN environment, we may have achieved even faster computation. The interesting issues concerned algorithm performance with regard to GDF dimensionality, task difficulty, and computational cost. We chose as criterion in the dimensionality experiments the CPU time needed for computation of a given task. The GN workload was best measured by CPU time needed for each time step. The task difficulty was changed in a number of ways in the context of different experiments. We used a personal computer with a 1.7 GHz Pentium 4 processor.

4.1. Comparing GN with standard numerical integration procedures

We compared the proposed system with the following standard numerical methods implemented in Mathematica 4 (Wolfram, 1999): Runge–Kutta–Fehlberg 4–5, Adams, Gear, and Adams–Gear. The CPU time needed for computing a specific task was the main performance criterion. This time was measured by the C++ run-time library function clock in the GN system, and by the Mathematica Timing function applied to the command NDSolve for solving differential equations numerically.

We consider our comparison of such different software solutions as a standard Mathematica procedure and the proposed system as satisfactory. CPU time was the most objective measure at our disposal. Note that the Mathematica Timing procedure measures exactly the CPU time, which is very different from the actual time needed by the computer to solve a problem. We observed that with large-scale GDFs the latter was 4–5 times longer than the former. We found that for our tasks the CPU time consumed by Mathematica was comparable with other similar packages.

We took special care to ensure that in the Mathematica sessions our experiments were run under the same conditions. This involved starting a new session before each run of Timing[NDSolve[·]] and keeping constant as much as possible all other influential factors described in the Mathematica Book (Wolfram, 1999).

5. Results

5.1. Stable choice, maintained for long and short periods

We were interested in two regimes of the large-scale GDF: First, enduring periods of equilibrium, occasionally interrupted by resets and competition; second, frequent resets coming shortly after each new winner had stabilized. We studied these regimes with a field of 100 gated dipoles. In our experiments the A_E bursts came at regular intervals. We varied the ratio between duration of the whole period (transient process followed by equilibrium with a new winner), and duration of transient process alone.

Figs. 3a and 3b show the system performance with transient process lasting for 10% and 90% of the period respectively. The top panels present the time course of all dipoles' **y5** signals, while the mid panels show the depletion and replenishment of **z1**. These plots confirm the correctness of the GN model, which not only changed the winner after each reset, but also depleted more neurotransmitter in the longer equilibrium periods.

The lowest two plots in Fig. 3 show simulation workload in real time as measured by CPU time needed for GN computation at each step. Fig. 3a (bottom) delineates periods of equilibrium when workload was approximately constant from periods with activated numerical procedure when the CPU needed increasingly more time to calculate every step. Each switching off of the numerical procedure was accompanied by a momentary jump of CPU time, while no such jump appeared at switching on. This can be explained by the complexity of the activation rules: Detecting a new A_E pulse is enough for the numerical integration to begin. In contrast, stopping it requires checks of conditions about all the elements of **y1**, **y2**, and **y5**.

Fig. 3a (bottom) also shows that on average, CPU time during numerical integration did not even double as compared to the equilibrium periods. This fact surprised us because we expected a much larger difference. Apparently the use of Runge–Kutta 4 instead of a more sophisticated procedure caused this effect. GN performed in the same way when equilibria were short and the numerical procedure was active during 90% of the time (Fig. 3b).

Fig. 4 highlights the point that too frequent computation of the adjusted signal $\hat{S}1_j^{\text{old}}$ (Section 2.2.1) may not be necessary. If, in response to abrupt input changes (Fig. 4, middle), adjusted $\hat{S}1_j^{\text{old}}$ is computed at each time step (Fig. 4, top), then the transmitter vector **z1** is computed precisely as in Raijmakers and Molenaar (1997). This solution (Fig. 4, bottom) accounts for the tiny little jumps in neurotransmitter consumption during the competition transients (see the arrows in Fig. 4).

Virtually the same picture was obtained when $\hat{S}1_j^{\text{old}}$ was updated only at the moments of A_E onset and switching off. With even less frequent update of $\hat{S}1_j^{\text{old}}$, i.e., only at the moments of winner change, the general picture remained intact, but the little jumps disappeared (Fig. 3a, middle; 3b, middle).

5.2. Increasing the gated dipole field dimensionality

The main question in this research was how much better could the GN system perform when compared to the Mathematica numerical procedures with regard to large-scale gated dipole fields. To answer it, we performed a number of experiments with increasing dimensionality. We present the results in Fig. 5a.

In the first experiment a GDF received linear input signal

$$\mathbf{I} = \left[0, \frac{1}{M}, \frac{2}{M}, \ldots, \frac{M-1}{M}\right]^T. \tag{14}$$

This **I** was presented for 13 000 time steps, signal A_E lasted 10% of each period (Section 5.1), and the number of gated dipoles M was gradually increased. Apparently the task difficulty increased not only due to the rise of M, but also due to the reduced differences between the elements of **I** — an increase in dipole field size decreases the resolution of **I** among neighbours and thus leads to increased competition. As seen from Fig. 5a, with the traditional numerical methods the CPU time grew rapidly with even moderate dimensionality increase. The GN, in contrast, coped with this task quite well.

Fig. 5a also shows that Runge–Kutta–Fehlberg 4–5 was computationally the most economical method among the standard ones, and was useful for much bigger GDFs. The former was also tested with A_E lasting 90% of the time (Section 5.1). Fig. 5b shows that in this case the standard method needed much more time while GN was again very efficient.

Finally we compared three cases of GN performance — each with a different type of input **I**. Those types were: First, linear

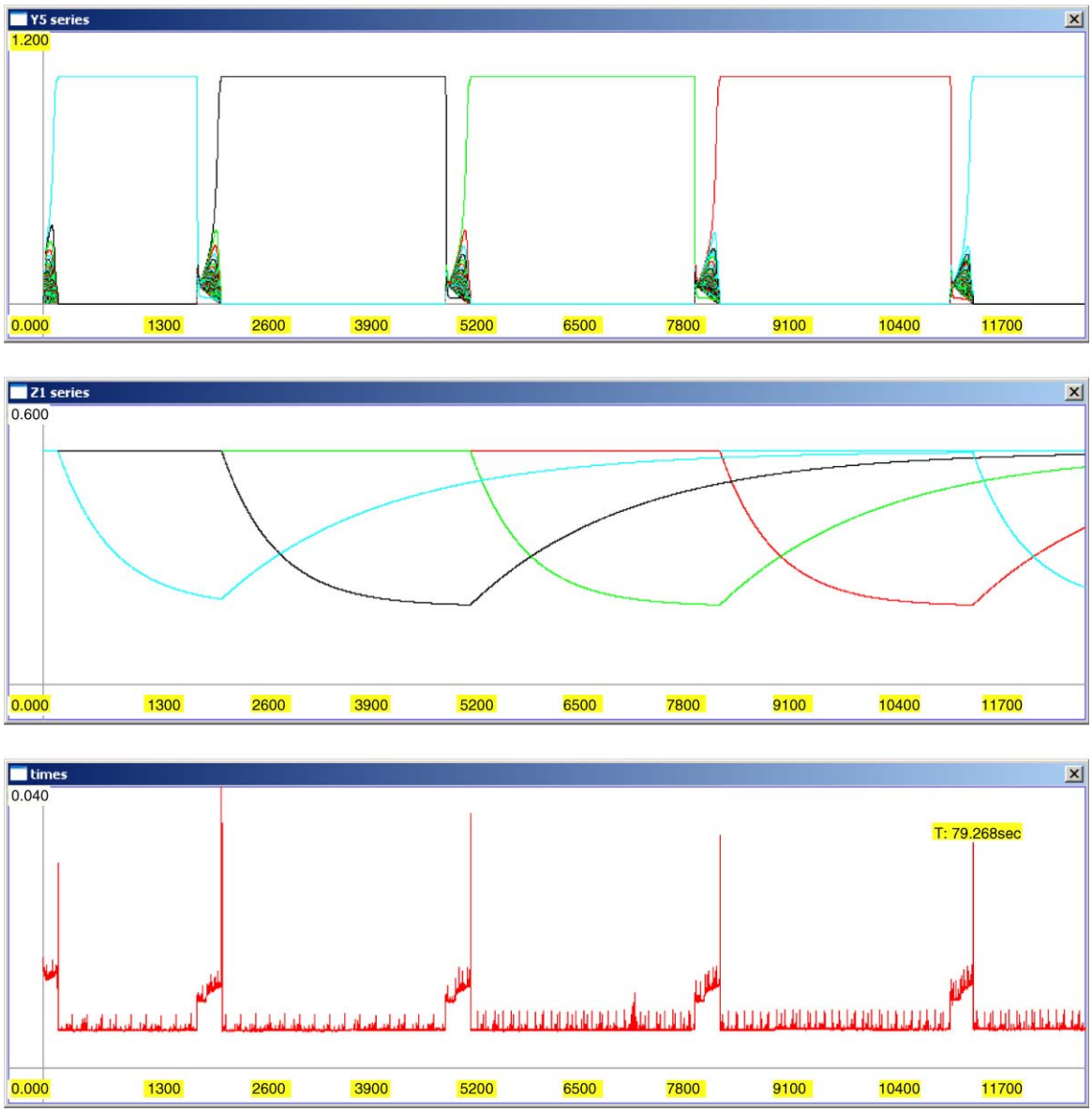

Fig. 3a. Performance of the generalized net in a regime with numerical method activated for 10% of the time. Top: Time course of **y5**. Middle: Time course of **z1**. Bottom: CPU time needed at each time step. In the upper right-hand corner we show total CPU time needed to compute the behaviour of 100 dipoles for 13 000 time steps in GN with 1.7 GHz processor.

input as per Eq. (13); second, logistic curve that greatly reduced the differences among the most competitive elements of **I** — those close to 1; third, exponential curve that enhanced the differences among these largest elements of **I** and thus eased competition. The effect of input variation became noticeable with 50 dipoles and more (Fig. 5c). As expected, the difficult sigmoid input needed more CPU time than the other two, but still, up to 100 dipoles this difference was not huge.

It should be noted that the Runge–Kutta 4 procedure used in GN was good enough for large-scale GDFs as well. Fig. 2 (bottom) shows the time course of **y5** in 600 gated dipoles receiving exponential curve as input. In this case competition lasted for quite long but was eventually won by a dipole.

6. Discussion and conclusions

In this work we achieved efficient computation of large-scale gated dipole fields while fully preserving the essential system dynamics. In fact the speed increase due to our modifications amounted to at least an order of magnitude for fields of 100 dipoles and more. We adopted a ‘divide and rule’ approach, as we clustered the eight dipole equations into three categories, and modified each category in a separate way. Thus we ended up needing a simple numerical procedure, for only three equations, and for only part of the time. Initially we expected our controlled use of numerical procedure to account for most of the gain, as a GDF in ART operates in short transients and long equilibrium periods. It turned out,

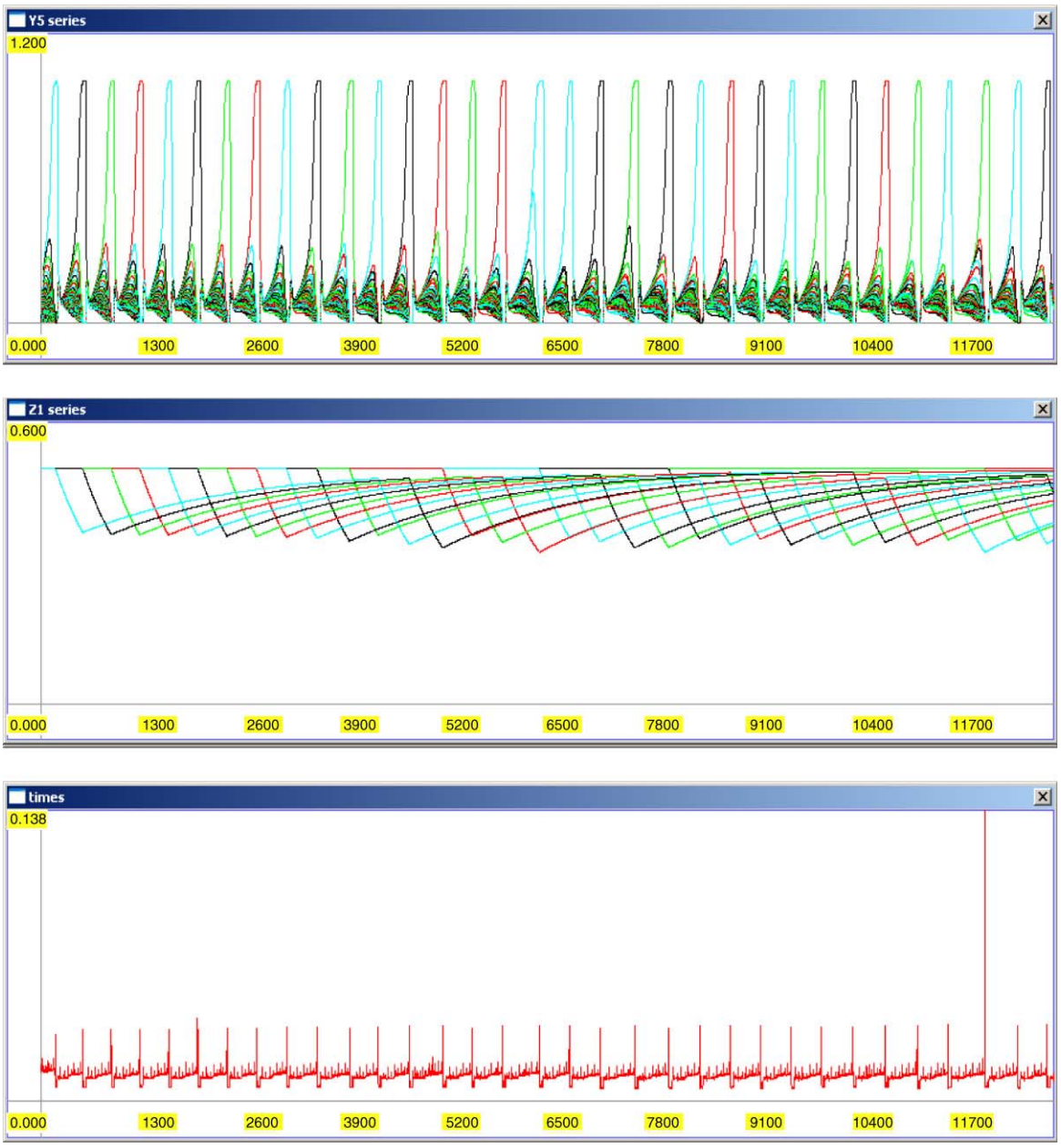

Fig. 3b. Performance of the generalized net in a regime with numerical method activated for 90% of the time. Top: Time course of **y5**. Middle: Time course of **z1**. Bottom: CPU time needed at each time step.

however, that our logical–numerical system was much less sensitive to the transient/equilibrium ratio than the standard Runge–Kutta–Fehlberg 4–5 method. This was desirable but unexpected and we concluded that the simplicity of the procedure had greater impact than the short duration of its activation.

We used Generalized Nets as a programming language — in contrast to their more popular use as a modeling method. We found this solution to be convenient because it added very little computational overhead. We believe that in a different development environment (e.g. C++) the overall speed might be even higher, but the results as shown in Fig. 5a would be essentially the same. With GNs in hand we freed our thinking from more established approaches, and adopted a fresh look towards developing an efficient model. Of course techniques more or less similar to ours have been used in discrete-event simulation in general, and in neural modeling in particular (Bower & Beeman, 1998; Hines & Carnevale, 1997; Lee & Farhat, 2001).

Our approach can be extended to a full continuous-time ART system and this is a topic of future research. Such a logical–numerical system would in its simplest variant probably not detect bifurcation points as in Raijmakers et al. (1996) because of the particular rule-based numerical integration. However, once the system dynamics are known they would easily be translated into rules for activating sets of $y5$ units. In

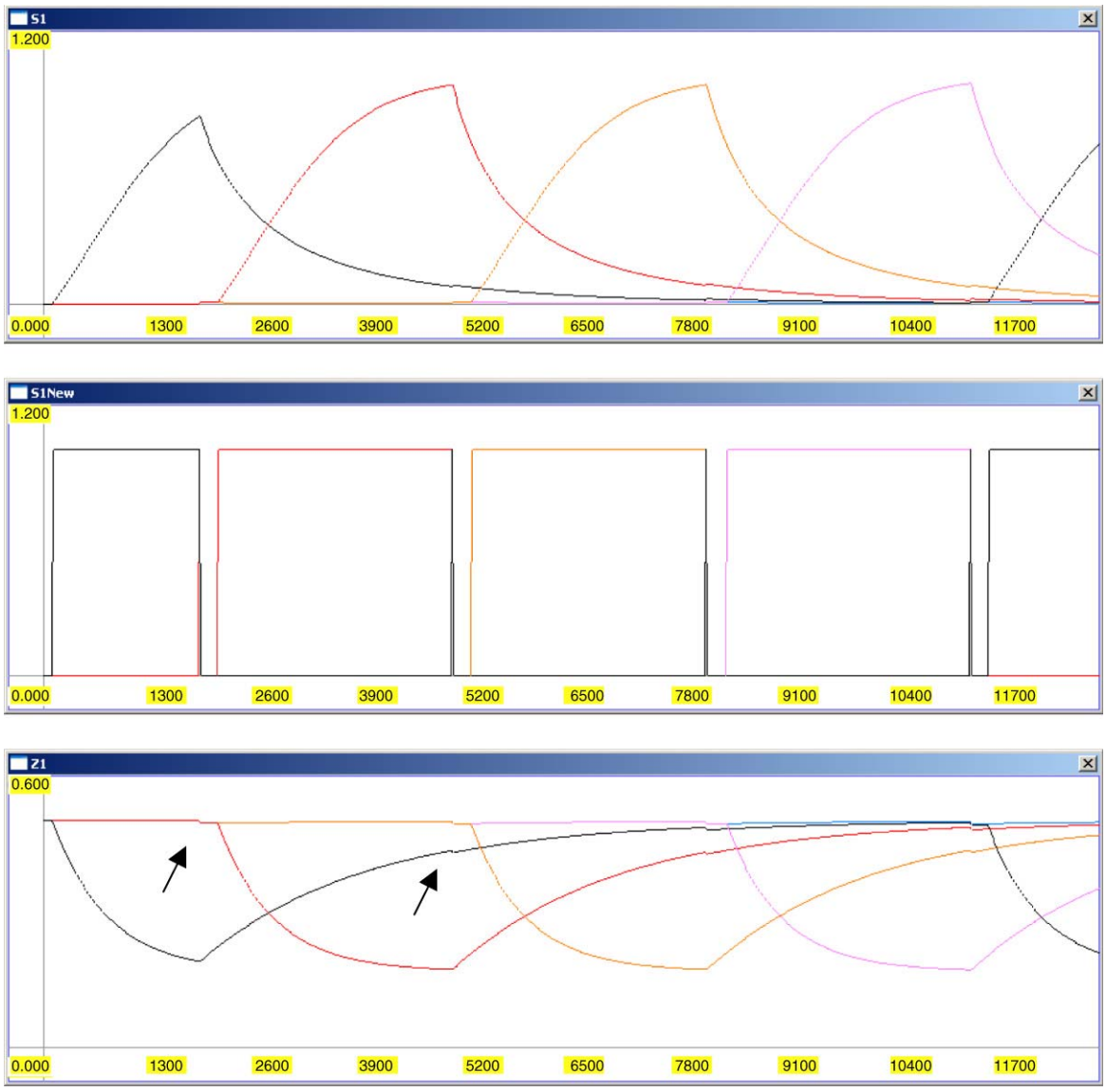

Fig. 4. Effect of a frequent $\hat{S}1_j^{\text{old}}$ update on neurotransmitter computation. Top: Time course of the adjusted equivalent $\hat{S}1_j^{\text{old}}$, computed at each time step. Middle: Time course of $S1_j^{\text{new}}$. Bottom: Time course of **z1**. The arrows highlight little jumps in transmitter consumption during transient processes.

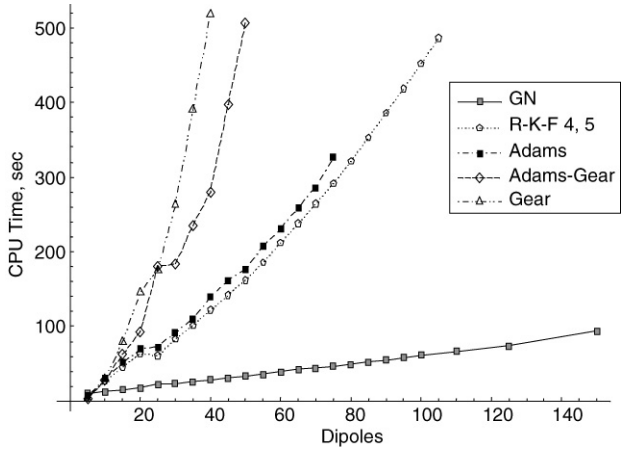

Fig. 5a. CPU time needed by standard numerical integration methods and generalized net for a task with growing dimensionality.

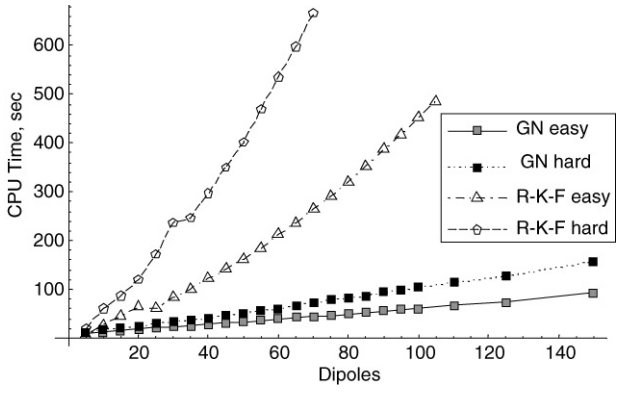

Fig. 5b. CPU time needed by the Runge–Kutta–Fehlberg 4–5 procedure and the generalized net for an easy task (transient process lasted 10% of the time), and a hard task (transient process lasted 90% of the time).

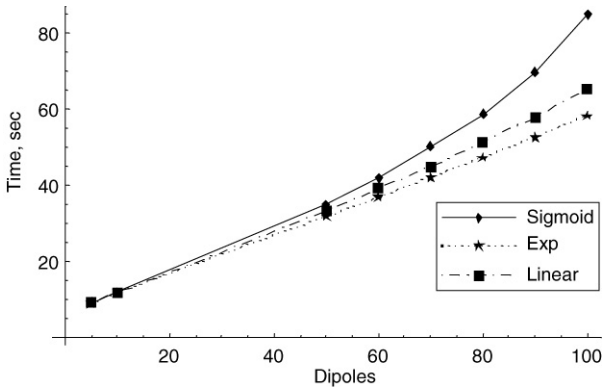

Fig. 5c. CPU time needed by the generalized net for easy (Exp), medium (Linear), and difficult (Sigmoid) tasks.

that case the model classification properties would be eligible for comparison with dART (Carpenter, 1997) and other similar systems. In the present paper we needed a subset of GN components and simulation capabilities excluding bifurcation analysis. However, GN theory could suggest other approaches for efficient numerical integration in the bifurcation analysis task, exactly as it did for this paper's GDF computation.

Our work focused on GDF in Exact ART. Because the core of adaptive resonance theory is not a set of equations but a set of principles, we did not need to deal with issues of numerical accuracy in the way other studies of neural systems did (cf. Lee & Farhat, 2001). We developed an efficient system that could correctly fulfill the GDF functions.

Appendix A. Generalized nets and GDF model

The Petri Net has seen many variants with different modeling capabilities. One example is the Generalized Net or GN (Atanassov, 1991, in press), which includes many previously known modifications as partial cases. GN is suitable for modeling parallel processes with concurrencies and synchronization in production and other systems. It has dynamical elements, temporal parameters, and memory. It operates on a fixed discrete-time scale and at each step changes its state. In various implementations some components may be omitted to give a reduced GN. The reader is referred to (Atanassov, 1991; Radeva, Krawczak, & Choy, 2002) for a comprehensive treatment of the topic.

For the GDF task we used a reduced GN with computable predicates and floating-point characteristic functions. Because the gated dipole dynamics have been well studied they are easy to model with GN. In general a GN consists of *transitions* with *places* and *tokens*. Transitions are like 'crossroads' for the information moving through the network places. These places are the 'parking lots', and the tokens are 'moving vehicles'. Each token contains its own *characteristic* — the 'useful load' that a vehicle carries. Fig. 6 presents the complete GN model of the gated dipole field. Since its thorough treatment would

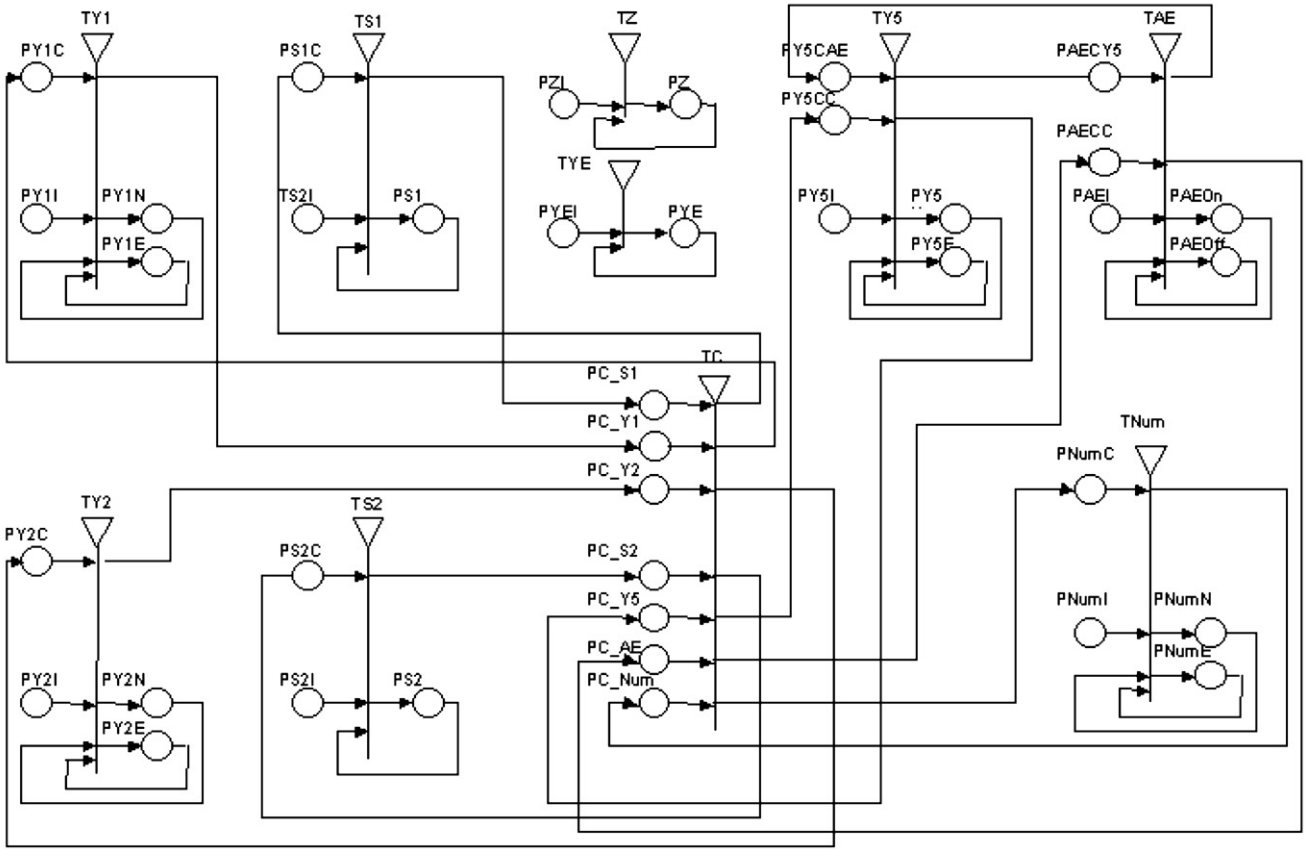

Fig. 6. Generalized net model of the gated dipole field.

be tedious we give only a general overview. Then we discuss in detail an illustrative example with neurons **y1**, which are among those computed by switching on and off the numerical integration procedure.

The vertical lines with triangles on top (Fig. 6) are GN transitions. For example, TY1 and TY2 in the left upper and lower parts of the figure correspond to node vectors **y1** and **y2** respectively. TS1 and TS2 represent the signals causing transmitter release as discussed in Section 2.2.1. TZ up in the middle is **z1** and **z2** transmitter transition, while TYE stands for three node vectors: **y3**, **y4**, and **y6**. From GN point of view it is convenient to combine the latter three in one transition because they are computed with the same simplification technique (Section 2.2.2). Note that we omit some connections in Fig. 6 just for visual clarity — they still exist in the real implementation. For example transition TAE sends signals not only to TC and TY5 as shown in the picture, but also to TY1, TY2, and TYE. Transitions TY5 and TAE (up to the right) do the **y5** and A_E computations respectively. Transition TC in the middle does not correspond to any gated dipole element, but coordinates the whole system computation. Through its many input places TC receives data about the other transition states, and emits two kinds of signals to them. The first kind is commands to switch on or off the numerical procedure (to transition TNum) or to produce the next A_E pulse (to TAE). The signals of the second kind inform other transitions about various new events, such as an input change from outside the GDF, appearance of a new A_E pulse, emergence of a new winner etc. Transition TC is also responsible for communicating inputs **I** and A_E to the system. The latter two are external for the GDF. The experimenter sets up the values of **I** and the frequency of A_E before simulation.

How does the GN work? Let us focus on the **y1** neurons represented by TY1 transition. Imagine that a new A_E reset signal has arrived and has begun influencing all $y1_j$ and $y2_j$ nodes (Fig. 1). The GN activities may be mapped onto the GDF interactions in the following way. First, the TY1 transition (Fig. 6) receives a message from the controlling transition TC that an input change has occurred. The carrier of that information is a token that leaves TC and enters place PY1C (deciphered as 'position of the control token of **y1**') of TY1. Simultaneously another token registers in place PY1I (deciphered as 'input position of the token representing **y1**'). The latter token has as its characteristic the current **y1** values. The token's presence in PY1I is detected by coordinating transition TC, which then orders switching the numerical integration on. Now the token leaves PY1I and enters PY1N (N stands for numerical procedure). This 'crossing' of the transition by the token leads to calculation of all **y1** values for the next time step. The token moves along the arc of the TY1 transition and comes back to position PY1N, calculating **y1** in the next time step. Transition TY1 sends the **y1** values via another token to place PC_Y1, which is an entry place for transition TC.

Simultaneously all other neurons and transmitters are calculated, and their values sent to TC, which checks if a $y5_j$ node has won the competition. This circulation goes on until TC finally detects a new winner and orders the numerical procedure to switch off. Then the token we have followed enters place PY1E (E stands for equilibrium) and circulates for the next time steps until new events make coordinator TC give another order. Thus we have explained the interaction of TC and TY1 with regard to **y1** computing. The remaining **y2**–**y6**, **z1**, and **z2** can be discussed in a similar way.

Appendix B

The following parameter values were used in the simulations. Constants: $A = 0.001$, $B = 1.0$, $e = 0.01$, $\beta = 0.5$, $\gamma = 0.5$, $\delta = 5.0$, $\Gamma = 0.1$, $\varepsilon = 0.001$. Initial values:

$$y1_j(0) = y2_j(0) = \cdots = y6_j(0) = 0, z1_j(0) = z2_j(0) = 0.5,$$
$$j = 1, \ldots, M; A_E(0) = 0.$$

References

Atanassov, K. (1991). *Generalized nets*. Singapore: World Scientific.

Atanassov, K. (2006). *Theory and applications of generalized nets*, Sofia: Pensoft (in press).

Bower, J. M., & Beeman, D. (1998). *The Book of genesis: exploring realistic neural models with the general simulation system*. New York: Springer Verlag.

Carpenter, G. A. (1997). Distributed learning, recognition, and prediction by ART and ARTMAP neural networks. *Neural Networks*, *10*, 1473–1494.

Carpenter, G. A., & Grossberg, S. (1987). ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, *26*(23), 4919–4930.

Carpenter, G. A., & Grossberg, S. (2002). Adaptive resonance theory. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (2nd ed.). Cambridge, MA: MIT Press.

Carpenter, G. A., Grossberg, S., & Rosen, D. B. (1991). ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks*, *4*, 493–504.

Gaudiano, P., & Grossberg, S. (1991). Vector associative maps: Unsupervised real-time error-based learning and control of movement trajectories. *Neural Networks*, *4*, 147–183.

Grossberg, S. (1972). A neural theory of punishment and avoidance, II: Quantitative theory. *Mathematical Biosciences*, *15*, 253–285.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors & II: Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, *23*, 121–134, 187–202.

Grossberg, S. (1980). How does a brain build a cognitive code? *Psychological Review*, *87*, 1–51.

Grossberg, S. (1984). Some psychophysiological and pharmacological correlates of a developmental, cognitive, and motivational theory. In R. Karrer, & J. Cohen (Eds.), *Brain and information: Event related potentials*. New York: New York Academy of Sciences.

Grossberg, S., & Gutowski, W. (1987). Neural dynamics of decision making under risk: Affective balance and cognitive-emotional interactions. *Psychological Review*, *94*(3), 300–318.

Grossberg, S., & Raizada, R. (2000). Contrast-sensitive perceptual grouping and object-based attention in the laminar circuits of primary visual cortex. *Vision Research*, *40*, 1413–1432.

Grossberg, S., & Schmajuk, N. (1987). Neural dynamics of attentionally modulated Pavlovian conditioning: Conditioned reinforcement, inhibition, and opponent processing. *Psychobiology*, *15*(3), 195–240.

Grossberg, S., & Seitz, A. (2003). Laminar development of receptive fields, maps, and columns in visual cortex: The coordinating role of the subplate. *Cerebral Cortex*, *13*, 852–863.

Grossberg, S., & Williamson, J. R. (2001). A neural model of how horizontal and interlaminar connections of visual cortex develop into adult circuits that carry out perceptual groupings and learning. *Cerebral Cortex*, *11*, 37–58.

Hines, M. L., & Carnevale, N. T. (1997). The neuron simulation environment. *Neural Computation*, *9*, 1179–1209.

Lee, G., & Farhat, N. H. (2001). The double queue method: A numerical method for integrate-and-fire neuron networks. *Neural Networks*, *14*, 921–932.

Leven, S. J., & Levine, D. S. (1996). Multiattribute decision making in context: A dynamic neural network methodology. *Cognitive Science*, *20*, 271–299.

Mengov, G., Pulov, S. V., Atanassov, K., Georgiev, K., & Trifonov, T. A. (2003). Modeling neural signals with a generalized net. *Advanced Studies in Contemporary Mathematics*, *7*(2), 155–166.

Ögmen, H. (1993). A neural theory of retino–cortical dynamics. *Neural Networks*, *6*, 245–273.

Ögmen, H., & Gagne, S. (1990). Neural models for sustained and on–off units of insect lamina. *Biological Cybernetics*, *63*, 51–60.

Radeva, V., Krawczak, M., & Choy, E. (2002). A review and bibliography on generalized nets theory and applications. *Advanced Studies in Contemporary Mathematics*, *4*(2), 173–199.

Raijmakers, M. E. J., & Molenaar, P. C. M. (1994). Exact ART: A complete implementation of an ART network, including all regulatory and logical functions, as a system of differential equations capable of stand-alone running in real time. In *Internal report*. Amsterdam: Department of Developmental Psychology, University of Amsterdam.

Raijmakers, M. E. J., van der Maas, H., & Molenaar, P. C. M. (1996). Numerical bifurcation analysis of distance-dependent on-center off-surround shunting neural networks. *Biological Cybernetics*, *75*, 495–507.

Raijmakers, M. E. J., & Molenaar, P. C. M. (1997). Exact ART: A complete implementation of an ART network. *Neural Networks*, *10*(4), 649–669.

Sgurev, V., Gluhchev, G., & Atanassov, K. (2001). Generalized net interpretation of information processing in the brain. In *Proceedings of the international conference 'automatics and informatics'*.

Wolfram, S. (1999). *The mathematica book* (4th ed.). Cambridge: Cambridge University Press.