On The Analysis Of Sigmoid Time Parameters For Dynamic Truncated BPTT Algorithm

V.Scesa, P.Henaff, F.B.Ouezdou, and F.Namoun

***Abstract*—The purpose of the research addressed in this paper concerns a comparative study of two expressions of the time scale parameter for Continuous Time Recurrent Neural Network (CTRNN): a classical time constant expression, and a sigmoid one. Their influence on the stability, the convergence speed and the generalization ability of a BackPropagation Through Time (BPTT) learning algorithm, will be discussed. Firstly, three mathematical conclusions related to the propagation and learning equations are deduced. Then these conclusions are validated on experiments carried out on a real biped robot. Through the identification of the balancing behavior under different robot torso motions, the sigmoid expression will be shown to get the best learning results.**

I. Introduction

THE supporting project of this study was born from the collaboration between a robotic research laboratory (LIRIS) and an industrial company (BIA). This project aims at shaping a smart architecture able to learn how to control non linear multi actuators system including real time constraints. Through the design and the implementation of this controller, we want to evaluate the ability of neural architectures and learning algorithms to achieve non linear control needs in real time. The project also aims at analyzing the capability of such algorithms to be adapted for a particular system, by rising optimal control, but also to be adaptive to changes in its environment, by generalizing the knowledge learned. The experiments will be carried out on two different structures: the ROBIAN biped robot from the LIRIS [1], and the BIA 6 hydraulic axis road simulator. In this paper, the experiments are limited to the ROBIAN robot.

The architecture of the developed controller is based on recurrent neural network with leaky integrator neurons. This allows the focusing of the net dynamics on time constant parameters. Recurrent connections are used for merging the neurons responses in a function able to contain oscillating motions. In order to shape this architecture, a BPTT based algorithm was chosen for its ability to integrate the learning error [2], [3], [4], [5]. The gradient algorithms are mainly used for system modeling, optimization applications, speech recognition [6], or meta learning [7].

The using of recurrent networks with gradient based learning algorithm is not obvious and may be unstable. As time constants modifications have a strong influence over network dynamics [8], it can intensify or eliminate this problem, and the way they are modified needs a specific attention. However, among all the studies based on CTRNNs, only a few are focused on the dynamic analysis of the net parameters [9], [10].

The aim of this paper is to study the influence of the expression of the time parameter over the algorithm stability, on the learning performances and on the evolution of the net dynamics. In a first part, the neural model and the learning algorithm will be detailed and the two time scale parameters expressions will be given. Then, their comparison will be performed, in a mathematic way by focusing on the propagation and learning equations, and on experimental results. An experiment carried out on the identification of the ROBIAN robot balancing will show the performance of the two expressions.

II. Neural Model and Learning Algorithm

A. Neurons and propagation

Following the leaky integrator equation (1), the input data is propagated in the network to generate the neurons outputs. The result belongs to the intrinsic parameters of the neurons and the network: weights, biases and time parameters. Classically, the propagation is written as follows:

$$T_j \cdot \frac{\partial y_j}{\partial t} = -y_j + f\left(\sum_i \left[w_{ij} \cdot y_i\right] + b_j\right) \qquad (1)$$

The corresponding discrete expressions are :

$$x_j(t) = \sum_i \left[w_{ij} \cdot y_i(t - \Delta t)\right] + b_j \qquad (2)$$

$$y_j(t) = \frac{\Delta t}{T_j} \cdot f\left(x_j(t)\right) + \left(1 - \frac{\Delta t}{T_j}\right) \cdot y_j(t - \Delta t) \qquad (3)$$

Where y_j corresponds to the j neuron activity, f is the activation function (tanh) and Δt is the time step. The net parameters w_{ij}, b_j and T_j are respectively the weights, biases and time constants.

In (3), the $\Delta t / T_j$ term is a scale parameter. Its value, given within]0:1], expresses the response speed of the j neuron. This expression is not the only one possible to define

V.Scesa is with the LIRIS laboratory, UVSQ (Versailles university) CNRS, 10,12 av. de l'Europe 78140 Velizy France (phone: (33)139254957, fax: (33)139254985, e-mail: vincent.scesa@liris.uvsq.fr)

P.Henaff is with the LIRIS-UVSQ-CNRS laboratory (e-mail: patrick.henaff@liris.uvsq.fr)

F.B.Ouezdou is with the LIRIS-UVSQ-CNRS laboratory (e-mail: ouezdou@liris.uvsq.fr)

Faycal Namoun is with the BIA company, 8 rue de l'Hautil 78700 Conflans France (e-mail: f.namoun@bia.fr)

the neuron dynamics. In [6] and [11], a sigmoid version of the time scale parameter is proposed. Through this change, the aim is to modify the way the neuron speed varies.

By defining a variable S_j that will be called **time scale parameter**, (3) can be written:

$$y_j(t) = S_j \cdot f(x_j(t)) + (1 - S_j) \cdot y_j(t - \Delta t) \quad (4)$$

In this paper, we will compare the two expressions of the time scale parameter:

$$S_j^1 = \frac{\Delta t}{T_j} \qquad \text{vs} \qquad S_j^2 = \frac{1}{1 + e^{-\lambda_j}} \quad (5)$$

The first one uses the classical **time constant** T_j. The second one uses another parameter called the **sigmoid time parameter** λ_j.

The λ_j parameter has no physical meaning, and it has nothing to do with the neuron sigmoid activation function. It expresses the neuron speed in another way, allowing a smoother logarithmic modification of S_j rather than the classical linear one.

B. Learning process

The objective consists in modifying the parameters of the network (weights, biases and time scale parameters) in order to minimize a desired criterion. In a control application, this criterion would be the gap between the desired state of the system and the actual one. In an identification process, the criterion would be the error between the neural model and the taught system. In order to carry out this net parameters adaptation while the system is running, BackPropagation Through Time algorithm, detailed in [3], [4] and [5], is used.

At first, the algorithm computes an error function that corresponds to the criterion to be minimized. This function is the integral of the error expressed as follows:

$$E = \int_{t0}^{t} [e_j(\tau) \cdot d\tau] \quad (6)$$

Where E represents the criterion and is equal to the error integral between t_0 and the current time step t. $e_j(\tau)$ is the output error of the neuron j stored at time τ, and t_0 is the beginning of the integration window.

BPTT algorithm carries out a gradient based learning. Thus, the modification of the parameters is following the opposite value of the error gradient of each parameter:

$$\Delta w_{jk} = -\eta_1 \cdot \frac{\partial E}{\partial w_{jk}} \quad (7)$$

$$\Delta b_j = -\eta_2 \cdot \frac{\partial E}{\partial b_j} \quad (8)$$

and : $\Delta T_j = -\eta_3 \cdot \frac{\partial E}{\partial T_j}$ or $\Delta \lambda_j = -\eta_3 \cdot \frac{\partial E}{\partial \lambda_j}$ (9)

The learning speed is defined by the three different learning rates η_1 for the weights, η_2 for the biases and η_3 for the time parameters.

The use of continuous time neurons implies a dynamic BPTT learning [2]. Hence the gradient values giving the influences of each parameter on the error are derived from gradient equations (11) to (14) written with the backpropagated error (10) :

$$z_j(\tau) = \frac{\partial E}{\partial y_j(\tau)} \quad (10)$$

$$\frac{\partial E}{\partial w_{jk}} = \int_{t0}^{t} [S_k \cdot z_k(\tau) \cdot f'(x_k(\tau - \Delta t)) \cdot y_j(\tau - \Delta t)] \cdot d\tau \quad (11)$$

$$\frac{\partial E}{\partial b_j} = \int_{t0}^{t} [S_j \cdot z_j(\tau) \cdot f'(x_k(\tau - \Delta t))] \cdot d\tau \quad (12)$$

$$\frac{\partial E}{\partial S_j} = \int_{t0}^{t} [z_j(\tau) \cdot (f(x_j(\tau - \Delta t)) - y_j(\tau - \Delta t))] \cdot d\tau \quad (13)$$

and :

$$\frac{\partial E}{\partial T_j} = \frac{\partial E}{\partial S_j} \cdot \frac{\partial S_j}{\partial T_j} \qquad \text{or} \qquad \frac{\partial E}{\partial \lambda_j} = \frac{\partial E}{\partial S_j} \cdot \frac{\partial S_j}{\partial \lambda_j} \quad (14)$$

Where z_j is the backpropagated error of the j neuron, f' is the derivative of the activation function (tanh), and τ is the current time step in the time window.

The backpropagated errors are computed as the classical backpropagation by considering the network states at each time step as successive layers. Thus, the algorithm backpropagates the output errors in the network and in time. Equation (15) details the z_j calculus:

$$z_j(\tau) = z_j^1(\tau) + z_j^2(\tau) + z_j^3(\tau) \quad (15)$$

$$z_j^1(\tau) = \sum_k [z_k(\tau + \Delta t) \cdot S_k \cdot f'(x_k(\tau)) \cdot w_{jk}]$$

$$z_j^2(\tau) = z_j(\tau + \Delta t) \cdot (1 - S_j)$$

$$z_j^3(\tau) = e_j(\tau) \cdot \Delta t$$

In (15), $z_j^1(\tau)$ represents the error coming from the other neurons, while the error due to the dynamic of the neuron j itself is given by $z_j^2(\tau)$, and $z_j^3(\tau)$ is the output error.

In classical BPTT, the z_j value is computed for each neuron and for each time step since the beginning of the learning. Nevertheless, to prevent the memory explosion induced by the storage of every network states ($t_0 = 0$), a truncated BPTT algorithm [12] (BPTT(h)) is used. Thus, the algorithm only keeps in memory the past states included in a time window that follows the current instant ($t_0 = t - h$ with h the time window width). The backpropagated error values are only computed along this sliding time window ($\tau \in [t-h;t]$). Hence only a gradient approximation is computed. This approximation is as close to the exact gradient as the time window is large.

III. MATHEMATICAL COMPARISON

A. Domain comparison

In gradient algorithms, each parameter is modified incrementally. For the time scale parameter, the two different expressions S_j^1 and S_j^2, lead to quite different modifications during the learning.

$$S_j^1 \ new = \frac{\Delta t}{T_j \ old + \Delta T_j} \tag{16}$$

$$S_j^2 \ new = \frac{1}{1 + e^{-\lambda_j old + \Delta\lambda_j}} \tag{17}$$

It is obvious that for the S_j^1 expression, the modification may lead to value out of the allowed domain]0:1]. In order to avoid this problem, the variations of T_j must be restricted in a non continuous way (a strong limit on the minimum value). On the other hand, the S_j^2 values are naturally kept in this interval with asymptotic limitations.

B. Temporal responses comparison

The way the neuron dynamic varies depends on the S_j^1 or S_j^2 expressions. Starting from a slow neuron and going to a faster one the speed evolution will be different for a linear evolution on T_j or λ_j.

On Fig.1 the responses evolutions to an Heaviside input for a linear variation of T_j and λ_j are depicted. In the first graph, T_j starts from a slow response (T_j=200s, S_j^1=5.10^{-4}) and varies linearly to a fast one (T_j=0.0101s, S_j^1= 0.99). In the second graph, starting from the same slow response (λ_j=-9.9, S_j^2=5.10^{-4}), λ_j varies linearly to the same faster one (λ_j=4.59, S_j^2=0.99).

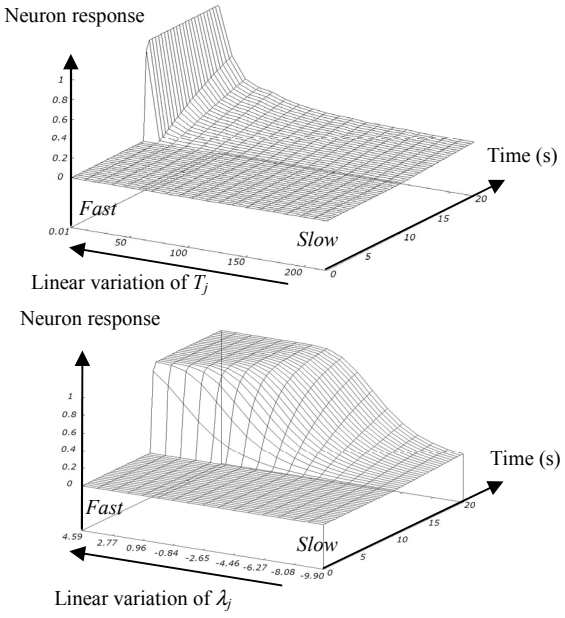

Fig 1 : Neuron responses to an Heaviside input for different speed values varying linearly according to *Tj* (up) and *λj* (down).

While the modification of the neuron behavior is quite rough for S_j^1, it is smoother for S_j^2. Actually, the second one modifies the time scale parameter in a logarithmic plan. In a gradient based algorithm, since the modifications of the parameters are incremental, the use of the sigmoid time parameter allows to expend the S_j evolution on a larger range. Thus, the temporal behavior of each neuron can vary in a slower way.

A smoother and extended evolution is to be preferred because it makes the variation more accurate. The same range of the neuron speed corresponds to a small T_j range with S_j^1, whereas it is contained in a larger λ_j range with S_j^2.

C. Amplifying factor comparison in the learning

In the learning equations (14), the time parameters gradients can be developed as follows :

$$\frac{\partial E}{\partial T_j} = -\frac{\Delta t}{T_j^2} \cdot \frac{\partial E}{\partial S_j} \tag{18}$$

$$\frac{\partial E}{\partial \lambda_j} = \frac{1}{1 + e^{-\lambda_j}} \cdot \left(1 - \frac{1}{1 + e^{-\lambda_j}}\right) \cdot \frac{\partial E}{\partial S_j} \tag{19}$$

For the same value of $\partial E/\partial S_j$ a different multiplicative coefficient amplifies the value of the gradient. This amplification is much more important and bumpy with the S_j^1 expression (maximum=1/Δt), than with the S_j^2 (maximum=1/4). This coefficient increases the rough change of dynamics observed previously on Fig.1. Hence the difference between the smooth evolution with the sigmoid time parameter and the rough one with the time constant is amplified.

D. Comparison of the neuron dynamic modification

The natural way for changing the dynamic of a system governed by a first order equation is to modify its time constant value. The physical meaning of a time constant change, i.e. a change in the response time of the system, makes the interpretation easier of this parameter variation. A higher time constant value leads to a slower system.

Thus, for the neuron, considering a modification of the X_j parameter used, whatever it is (X_j^1=T_j or X_j^2=λ_j), the interpretation will be easier by bringing ΔX_j to the corresponding ΔT_j value.

This transformation is straight for X_j^1 :

$$\Delta T_j = \Delta X_j \tag{19}$$

For X_j^2, the transformation can be identified by equalizing the S_j new variable in (16) and (17):

$$\Delta T_j = \left(T_j - \Delta t\right) \cdot \left(e^{\Delta X_j} - 1\right) \tag{20}$$

The set of the two equations (19) and (20) expresses how

the time constant and so the neuron dynamic are modified in both cases. The differences between the two exhort the contribution of X_j^2 (λ_j) use compared to the X_j^1 (T_j) use.

The contribution can be divided into five kinds according to the ΔX_j values. These different contributions can be located by the A, B, C and D areas and the line E depicted on the following graph:

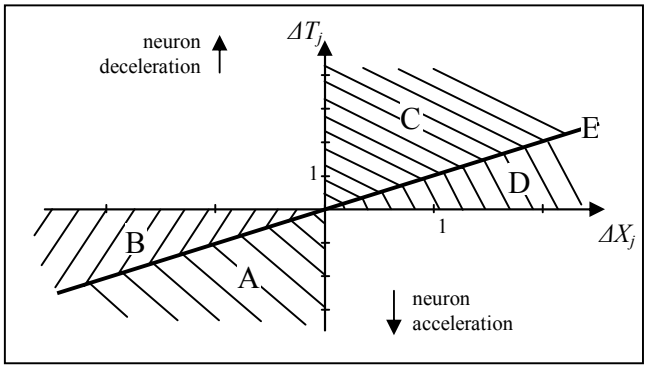

Fig 2 : Localizations of the contributions of X_j^2 use compared to X_j^1 use, in the plan defined by the ΔX_j values and the corresponding ΔT_j.

The five contributions corresponds respectively to :

A - an increase of the asked neuron acceleration

B – a decrease of the asked neuron acceleration

C – an increase of the asked neuron deceleration

D – a decrease of the asked neuron deceleration

E – a null contribution (the use of X_j^2 brings the same modifications than X_j^1)

In the graph Fig.3, (19) and (20) are plotted in the same plan as the previous Fig.2.

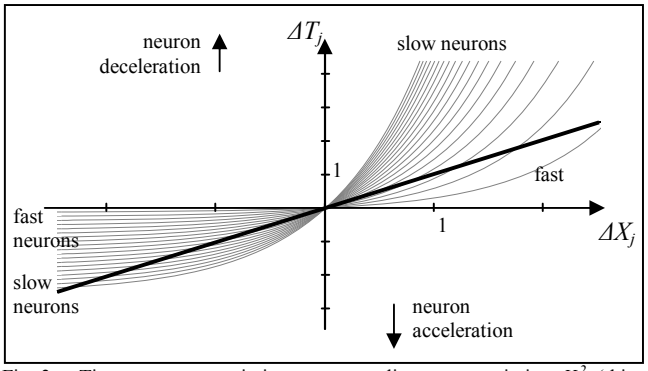

Fig 3 : Time constant variation corresponding to a variation X_j^2 (thin curves) or X_j^1 (bold line). Each curve corresponds to a different initial time constant value.

First, we can notice that, only with X_j^2, the resulting time constant variation is linked with the current T_j value. There are several thin curves but only one bold line. Thus, according to the value of T_j (slow or fast neuron) the modification of the neuron dynamic will be different according to the neuron speed.

Then, with X_j^2 and according to the T_j values, if the neuron is slow, the acceleration or deceleration caused by the learning will be increased (thin curves are in the A and C areas). On the other hand, if the neuron is fast, the variations will be stifled (thin curves are in areas B or D).

Nevertheless, for a fast neuron, if a strong deceleration is asked (great positive ΔX_j *value*), the use of X_j^2 will accentuate this deceleration (the curve goes from area D to area C).

Finally, with X_j^2 and whatever the neuron speed is, if a strong acceleration is asked (great negative ΔX_j *value*), ΔT_j will be asymptotically limited. The limit is $\Delta T_j = \Delta t - T_j$. It leads to an instant response of the neuron ($T_j = \Delta t$).

To sum up, the use of a sigmoid time parameter rather than a classical time constant one brings the 3 following advantages:

- an asymptotic limit to the fastest behavior,
- a smoother and extended evolution of the neuron dynamic,
- a mobility improvement of the time parameter for slow neurons and a reduced mobility for faster ones.

IV. Experimental comparison

A. Experiment description

The main objective of this experiment is to validate our mathematical deductions and to analyze the sigmoid use influence on the learning convergence.

To focus on this point, a model identification rather than a controller shaping will be used. Fig.4 shows how the identification of ROBIAN's torso influence on the ZMP (Zero Moment Point) will be performed.

The ZMP is a key notion in the balance control of a walking robot. It corresponds to the position of the center of pressure on the ground. The ZMP control, introduced thirty five years ago [13], consists in controlling the equilibrium of the biped robot by keeping the ZMP inside the polygon defined by the contact point with the ground.

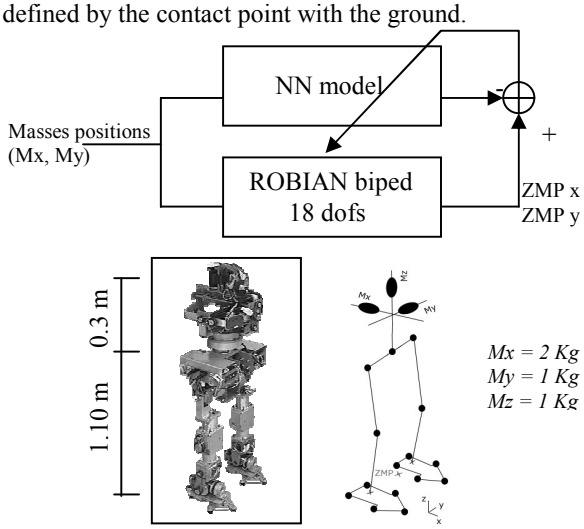

Fig 4: Learning architecture for the identification (up) of ROBIAN robot (down), with 2 legs and a 4 degrees of freedom torso.

The motion of the X and Y masses of the torso (Mx and

My) disturbs the robot equilibrium, leading to variations of the X and Y ZMP positions. The neural network (NN) tries to reproduce these influences by computing the gap between the desired positions and its own outputs. During the learning process, it will try to modify its parameters to vanish this error, by minimizing a cost function. This function is the normalized squared errors sum on the two axis, The output errors can be derived from it as follows:

$$Cost = \left(\frac{ZMP_x - y_x}{maxZMP_x}\right)^2 + \left(\frac{ZMP_y - y_y}{maxZMP_y}\right)^2 \quad (20)$$

$$e_j(t) = \frac{\partial Cost}{\partial y_j(t)} = -\frac{2}{maxZMP_j} \cdot (ZMPj - y_j(t)) \quad (21)$$

Where y_x, y_y are the output activities on the axis and $maxZMP_j$ is the maximum amplitude on the j axis (for normalization).

Two different criteria will be analyzed to quantify the ability of the algorithm to learn the ZMP positions: convergence speed and generalization performance. The observation of the Cost evolution during the learning will inform us about the time needed for convergence and how it occurs. After the learning, a test on another pattern will reveal the learned model ability to generalize its knowledge.

Representative situations should be presented during the learning. They must fully characterize the behavior of the studied system. This implies to give the network patterns that express the dynamic of the ZMP evolution with its oscillations, damping and resonance for the 2 axis.

The learning pattern contains a succession of four different excitations applied to the robot: squared commands with varying amplitudes (0.01m to 0.20m) and with varying frequencies (1Hz-3Hz, containing the resonance of the robot), for both X and Y axis.

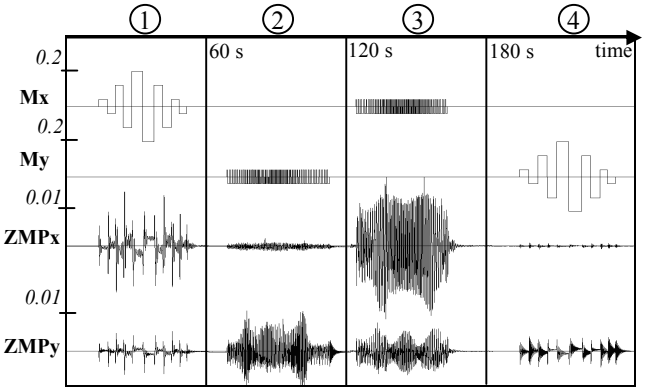

Fig. 5: Masses positions (m) and ZMP positions (m) on the two axis during the learning pattern (60x4 seconds with a sampling rate = 45Hz). In periods 1 and 4, the X mass and then the Y mass, are submitted to successive steps with various amplitude. In the 2 and 3 periods, steps with varying frequency are applied.

In the testing pattern a sine command with a varying frequency that produces the resonance is applied on both X and Y axis.

B. *Learning without time parameters modification*

The Cost evolution is depicted on the Fig.6. In this experiment, the network (composed of 2 inputs 15 hidden neurons and 2 outputs) was taught with one hundred loops on the learning pattern (η_1=η_2=0.025, η_3=0 to fix the S_j values) and a time window width h=40.

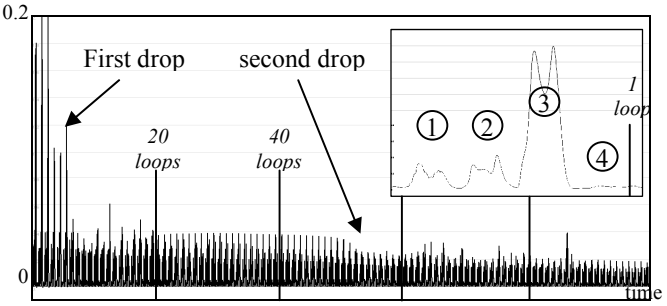

Fig. 6: Cost evolution during the learning without modification of the time scale parameters.

At the beginning of the learning, the network parameters are randomly chosen to generate stable outputs. The time scale parameters (S_j) are taken in the interval [0.5;1], to obtain neurons with speeds close to the dynamic of the system.

Fig.6 shows the evolution of the cost during the learning. In the zoom depicted on the top right corner, the four cost peaks are due to the different kinds of command (amplitude variations: peaks 1 and 4, or frequency variations: peaks 2 and 3) of the learning pattern.

During the experiment, two main drops happened. They are respectively linked to a quick decrease of peak 3 (learning of the X behavior) and peak 2 (learning on Y axis). The instant when the second drop occurred (SDI : Second Drop Instant) is a good factor to quantify the convergence speed. It occurs when the network found a correct answer on the both two axis. Here it takes about 55 loops to arise.

SDI=55

After the learning, a relevant value for estimating the generalization capacity of the learned network is the costs sum during the test pattern (TES : Test Errors Sum). It expresses the difference between an ideal model of the robot and the learned one. With a perfect model, the TES value would be zero. The remaining error is here 59.

TES=59

C. *Learning with time constants variation S_j^1*

The learning is started again with the same initial values for the network parameters and the same learning rate η_1 and η_2. Two different values of learning rate η_3 are compared for the modification of the time constant. The first one will be η_3=0.0001, chosen for generating a slow learning on the time constant. Then, a rate η_3=0.001, will be used for faster learning.

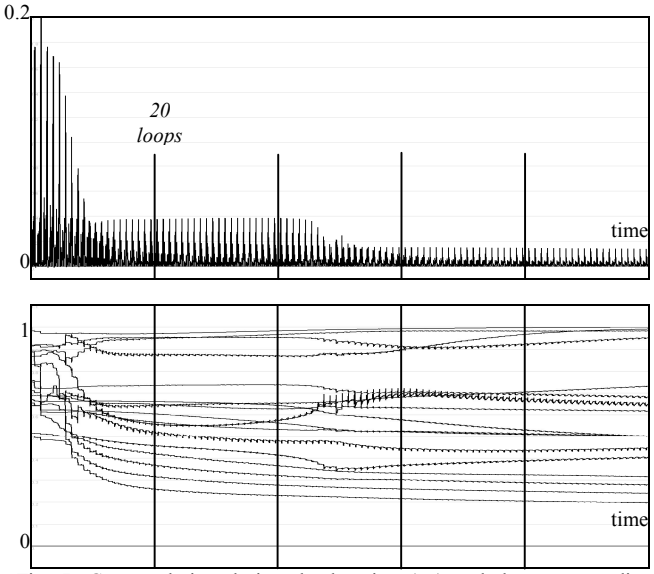

Fig. 7: Cost evolution during the learning (up) and the corresponding variations of the S_j^1 values (down) through slow modification of the time constants : η_3=0.0001.

In this first learning, the modification of the time constant is quite slow. The convergence speed obtained is approximately the same as the previous learning, but the generalization capacity at the end is better:

SDI = 50 TES = 35.1

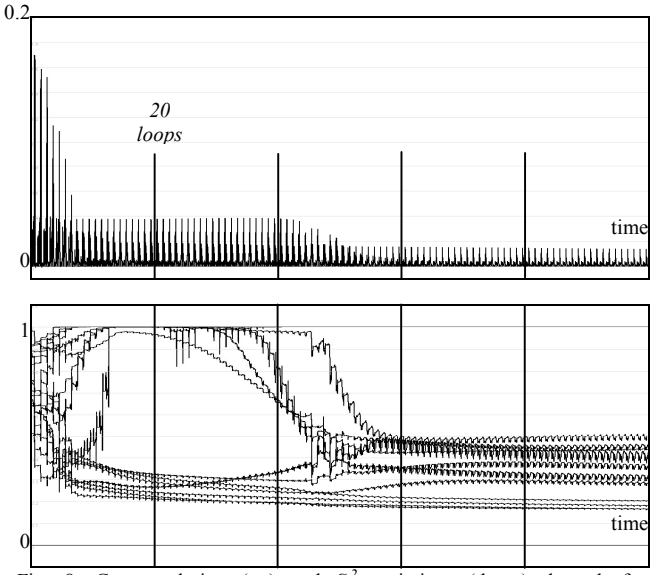

Fig. 8: Cost evolution (up) and S_j^2 variations (down) through fast modification of the time constants : η_3=0.001.

In a second learning, the modification of the time constant is faster. The convergence speed obtained is, again, approximately the same, but the learned network shows a better generalization capacity:

SDI = 51 TES = 32.9

D. Learning with sigmoid time parameter variation S_j^2

For this experiment, the learning is modifying the time scale values through the modification of the sigmoid time parameters instead of the time constants. It starts with the initial sigmoid time parameters that correspond to the previous initial time constants. The same weights and biases initial values are also used. The previous learning rates η_1 and η_2 are kept. Two η_3 values are used: η_3=1.25 and η_3=0.5. They were experimentally chosen to bring the same time scale variation speed than in the previous section.

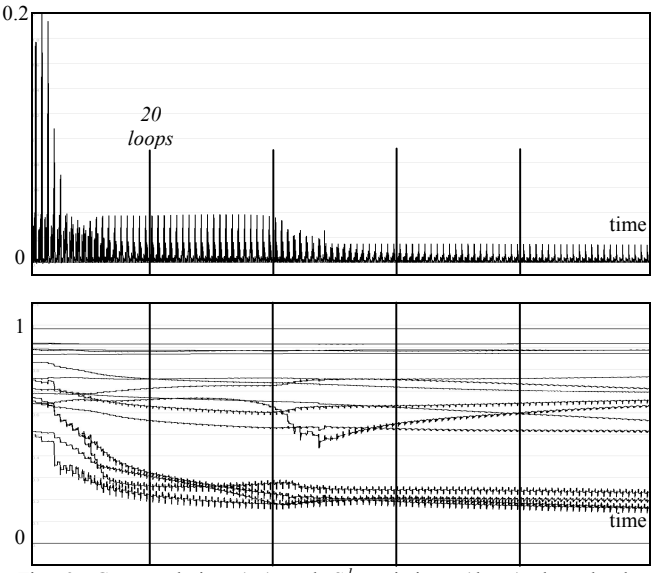

Fig. 9: Cost evolution (up) and S_j^1 variations (down) through slow modification of the time constants : η_3=0.5.

During this slow learning, the convergence speed seems to be a little faster, and the learned network answers correctly on the test pattern:

SDI = 49 TES = 37.6

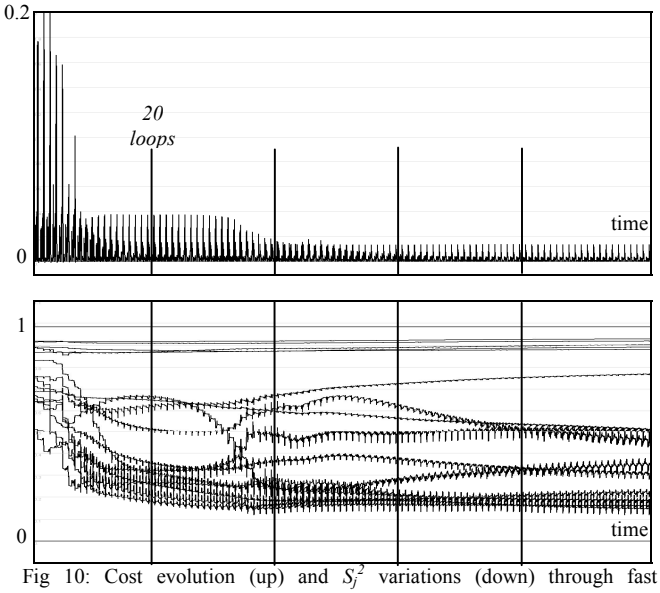

Fig 10: Cost evolution (up) and S_j^2 variations (down) through fast modification of the sigmoid time parameters : η_3=1.25.

In this experiment, the convergence speed reaches its fastest value and the generalization test shows the best result.

SDI = 32 TES = 31.7

V. DISCUSSION

A. Validation of the 3 mathematic deductions

By observing the time scale modifications graphs, it is possible to validate the mathematical deductions given in section III.

1) Asymptotic limit to fastest behaviors

A neuron reaches its fastest dynamic when its time scale value is 1. During the S_j^1 learning (Fig.7 and Fig.8), the algorithm drives several S_j^1 parameters to this maximum value. One neuron answers instantaneously for slow learning, and 6 neurons do so for fast learning (superposed on the graph). Actually, the algorithm hardly blocks the modification of the T_j parameters to this limit. The consequence can easily be seen on graph Fig.8. The time scale values, and, as a consequence, the neurons speed, are not slowed down while approaching the instant limit. When they reached it, they get stuck on it (after 20 loops, 9 neurons are stuck and only 3 of them managed to slow down).

This phenomenon is troublesome because the range of possible network dynamic behaviors decreases with the number of neurons stuck. The problem is not the instant responses (that could be useful) but the attracting power of this limit that confines the network's dynamic evolution.

This problem does not occur in the S_j^2 (Fig.9 and Fig.10). Some time scale parameters approach the instant limit but it does not behave as an attractor. The first mathematical deduction is thus experimentally confirmed.

2) Smoother and extended evolution of the dynamic

The network dynamic is strongly linked with the time scale values. If all neurons are fast (time scale parameters close to 1), the network will generate a fast response to its input excitation. In the same way, if they all are slow, the global response will be slow. But, if the speeds are homogeneously distributed, the network could arise slow or fast responses. The possible behaviors contained are more varied. Thus, during the learning, it is important to allow the S_j to evolve in the same way for full possible range]0:1]. If not, some ranges should be kept aside and some dynamic behavior would be difficult to arise.

However it clearly appears in Fig.8 that S_j^1 learning brutally jumps between a slow and a fast range (where the parameters are getting stuck). The algorithm avoids the interval [0.5:0.9[, because in this range, the speed variation are very sensitive to little modifications of T_j.

With the S_j^2 modification, this not occurs (Fig.10). Although this experiment presents approximately the same modification speed, the time scale values are varying in an homogeneous way along the whole range.

Thus, this remark meets the second mathematical deduction. The time constant variation induces a brutal modification of the time scale parameters whereas the sigmoid time parameter modification is smoother and extended on the whole range.

3) Mobility improvement or reduction

In the backpropagated error equation (15), the second term behaves as a first order along the time window [2]. So, the faster the neuron, the more the backpropagated error (z_j) variation on this neuron will be important. Moreover, if the S_j value of this neuron also varies, this will induce disturbing variations of the weights and biases parameters (in (11) and (12), z_j and S_j are multiplied). The convergence stability and speed could be affected by this phenomenon.

The evolution of the time scale values of Fig.8 and Fig.10 shows that the compared methods have opposite behaviors. On the one hand, the S_j^1 use implies oscillating values for fast neurons and slow variation for slow neurons. On the other hand, the S_j^2 expression induces a varying behavior for slow neuron and smooth variations for fast neurons. This observation validates to the third mathematical deduction.

B. Consequences

The study of the cost evolution graphs and the SDI and TES values provides the experimental consequences of these deductions.

Firstly, comparing the cost evolution graphs of Fig.7, 8, 9 and 10 to the one obtained during the no modification experiment Fig.6, it is obvious to say that the time scale modification brings a stabilization of the convergence. Without modification of the neuron dynamic, the cost meets a lot of difficulties to clearly converge to a minimum. It takes more loops for the second drop : SDI = 55, and the convergence is less stable. More than 90 loops are needed to reach the cost range obtained just after the SDI point in the other experiments.

The convergence speed, expressed by the SDI values, is approximately the same for all experiments except for the last one where it's better. There, the homogeneous modifications help the weights modification to reach the convergence in a better way. Whereas, in the other experiments, it does not help enough or perturbs it.

Finally, the quality of the created nets can be evaluated through the TES values. Although the TES values obtained are not strongly different, it is possible to extract some remarks from these results. The worst is the one obtained with no modification of neurons dynamics. The best result is achieved with strong modifications of the sigmoid time parameters. This is probably due to the way the time scale values vary. They explore a wide range of time scale values

smoothly and homogenously. As far as the two low rates learning are concerned, the modification only seems able to help the convergence stabilization but doesn't manage to find a good set of time scale value for decreasing the cost.

The experiments carried out validated the mathematical deductions. It also highlights their influence on the learning behavior. Actually, the use of a sigmoid time parameter rather than a classical time constant one shows a faster learning convergence, and more accurate neural networks.

VI. CONCLUSIONS

In this study, a comparison between two different ways of expressing the time scale parameter in a leaky integrator neural network (classical time constant and sigmoid time parameter) was carried out. The approach focused on identifying the one that fits best to the adopted learning method based on truncated dynamic backpropagation though time algorithm.

At first, a mathematical comparison of the influence on the neuron behavior when time scale changes occur was made. Next, an experimental study based on a real system identification (a biped robot) was carried out.

It appears that the sigmoid expression brought the following advantages:

- an asymptotic limit to an instantaneous response of the neurons, which is important to avoid stuck behaviors,
- a smoother and extended evolution of the neuron dynamic, that provides a more homogeneous path in the whole time scale range]0:1],
- a mobility improvement of the time parameter for slow neuron and a stiffen mobility for faster one, which brings a less oscillating convergence.

These advantages lead a faster convergence of the learning algorithm and more accurate neural networks.

As a further development, the adopted learning algorithm will be tried for controlling the equilibrium of the ROBIAN biped robot and for generating force trajectories on the BIA road simulator. Thus, the stability and speed of our learning method will be tested on two different systems (electric and hydraulic). Thus its limits will be shown, as far as real time control, robustness and generalization are concerned.

REFERENCES

[1] B. Mohamed, F. Gravez, F.B. Ouezdou, "Emulation of the dynamic effects of human torso during walking gait", in Journal of Mechanical Design, vol. 126, Issue 5, pp 830-841, Sept 2004.

[2] B.A. Pearlmutter, "Gradient calculation for dynamic recurrent neural networks: a survey", in Transactions on Neural Networks, 6(5):1212-1228, 1995.

[3] P.J. Werbos, "Backpropagation through time: what it does and how to do it", Proceedings of the IEEE, vol. 78, no. 10, pp. 1550 1560, 1990.

[4] A.J. Robinson and F. Fallside, "Static and dynamic error propagation networks with application to speech coding", In Anderson pp. 632 – 641, 1987.

[5] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning internal representations by error propagation Parallel distributed processing: explorations in the microstructure of cognition", eds D E Rumelhart, J L Mc- Clelland and the PDP Research Group (MIT Press, Cambridge MA) pp 318–362, 1986.

[6] Nguyen, M.H. and G.W. Cottrell, "Tau Net: A neural network for modeling temporal variability", in Neurocomputing 15 pp. 249-271, 1997.

[7] S. Hochreiter, A.S. Younger, and P.R. Conwell, "Learning to learn using gradient descent", in lecture notes on Comp. Sci. 2130, proc. Intl. Conf. on Artificial Neural Networks (ICANN-2001), pages 87-94. Springer: Berlin, Heidelberg, 2001.

[8] K. Doya. "Bifurcations of recurrent neural networks in gradient descent learning". Submitted to IEEE Transactions on Neural Networks, 1993.

[9] Draye, J., Pavisic, D., Libert, G, "Dynamic recurrent neural networks: a dynamical analysis", IEEE Trans. on Systems Man and Cybernetics, Part B, vol 26, n 5, pp. 692-706. 1996

[10] R.D. Beer, "Parameter space structure of continuous-time recurrent neural networks", submitted, the supplementary Mathematica notebook, 2005.

[11] F-S. Tsung. "Modeling Dynamical Systems with Recurrent Neural Networks". PhD thesis, Department of Computer Science. University of California, San Diego, 1994.

[12] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent connectionist networks", In Y. Chauvin and D. E. Rumelhart, editors, Backpropagation: Theory, Architectures, and Applications, Erlbaum, Hillsdale, NJ, 1990.

[13] Vukobratovic, M. and Borovac, B.. "Zero-moment point – thirty five years of its life" in International Journal of Humanoid Robotics. 1(1): 157-173. 2004.