Time Window Width Influence On Dynamic BPTT(h) Learning Algorithm Performances: Experimental Study

Vincent Scesa, Patrick Henaff, Fethi Ben Ouezdou, Faycal Namoun

Time Window Width Influence On Dynamic BPTT(h) Learning Algorithm Performances: Experimental Study

V. Scesa[1], P. Henaff[1], F.B. Ouezdou[1], F. Namoun[2]

[1] LISV, Université de Versailles St Quentin,
10, 12 avenue de l'Europe, 78140 Vélizy, France.
[2] BIA company,
8 rue de l'Hautil, 78730 Conflans Ste Honorine, France.

Abstract. The purpose of the research addressed in this paper is to study the influence of the time window width in dynamic truncated BackPropagation Through Time BPTT(h) learning algorithms. Statistical experiments based on the identification of a real biped robot balancing mechanism are carried out to raise the link between the window width and the stability, the speed and the accuracy of the learning. The time window width choice is shown to be crucial for the convergence speed of the learning process and the generalization ability of the network. Although, a particular attention is brought to a divergence problem (gradient blow up) observed with the assumption where the net parameters are constant along the window. The limit of this assumption is demonstrated and parameters evolution storage, used as a solution for this problem, is detailed.

1 Introduction

Born from the collaboration between a robotic research laboratory (LISV) and an industrial company (BIA), the supporting project of this study aims to shape a smart architecture able to learn to control non linear multi actuators system under real time constraints. Through the design and the implementation of this controller, we want to evaluate the ability of neural architectures to achieve the non linear control needs in real time. The project experiments are carried out on two structures: the ROBIAN biped robot from the LISV [1], and the BIA road simulator (www.bia.fr). The architecture of the developed algorithm is based on continuous time recurrent neural networks (CTRNN). To shape this architecture, a truncated BPTT algorithm was chosen for its ability of integrating the learning error and for its simplicity to be implemented for real time online applications. This well known gradient algorithm is widely used for system modeling and control [2], optimization applications, speech recognition [3], or meta learning [4]. A detailed description of this algorithm is given in [5], [6] and [7]. Nevertheless, as far as we know, no study aimed to extract the important role of the truncation width on the success of the learning. Obviously, the truncation width influences deeply the learning speed and quality. Our real time experimentations on ROBIAN biped also show us that, when the net parameters are considered as constant along a large window, the learning stability could be hardly spoiled.

Thus, through a statistical analysis of learning results, the links between the width of the time window, the stability of the learning, the "constant parameters" assumption, the convergence speed and the generalization abilities of the learned networks, have to be raised. To focus on this influence, a first experiment dealing with direct model identification rather than a controller shaping will be considered.
The next section details the neural model and the BPTT(h) learning algorithm used. The experimental plant is described in section 3. In section 4, the influence of the window width will be studied. The "constant parameters" assumption with its influence on the learning stability will be discussed. A stabilizing modification of the learning equation, to take into account the parameters evolution history, will be proposed. It will allow raising the dependencies between the window width and the learning results. Finally, discussions about the parameters evolution storage and the window choice to find a compromise between learning speed and nets accuracy will be given.

2 Dynamic truncated Backpropagation Through Time algorithm

2.1 Neural Model

Following the classical CTRNN equation (1), the input data are propagated in the network to generate the neurons outputs. The net activities belong to the intrinsic neurons and network parameters : weights (w_{ij}), biases (b_j) and time parameters (T_j).

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

Where y_j corresponds to the j neuron activity, f is the activation function (tanh). Using a time scale parameter S_j (classically $S_j = \Delta t / T_j$, Δt is the time step), the discrete corresponding equation can be written as follows:

$$y_j(t) = S_j \cdot f\left(\sum_i \left[w_{ij} \cdot y_i(t - \Delta t) \right] + b_j \right) + (1 - S_j) \cdot y_j(t - \Delta t) \qquad (2)$$

As the network is fully recurrent, each neuron receives the outputs of all the other ones. The weights, biases and time constants shape the temporal response of the net.

2.2 Learning algorithm

The objective of the parameters modification consists in minimizing a desired criterion. For an identification process, the criterion would be the gap between the neural model and the taught system. To carry out the adaptation of network parameters, BackPropagation Through Time algorithm, detailed in [5], [6] and [7], can be used.
This algorithm is computing an error function that corresponds to the criterion to be minimized. The error function (E) is defined as the integral of each net output errors. The parameters modification ($\Delta param$) is leaded by the error gradient inverse value. The following equations give the error function and the parameters modification law:

$$E = \int_{t_0}^{t} [e_j(\tau) \cdot d\tau] \qquad (3) \qquad\qquad \Delta param = -\eta \cdot \frac{\partial E}{\partial param} \qquad (4)$$

Where $e_j(\tau)$ is the output error of the neuron j stored at the current time step in the time window (τ). During the learning, to minimize E, the algorithm modifies the net parameters following the gradient descent (4), where *param* is w_{ij}, b_j or S_j and η is the corresponding learning rate. To compute these "delta" values for continuous time neurons, a dynamic BPTT learning algorithm [5] is needed. The gradient values giving the influence of each parameter are computed with equations (5), (6) and (7):

$$\frac{\partial E}{\partial w_{jk}} = \int_{t_0}^{t} [S_k \cdot z_k(\tau) \cdot f'(x_k(\tau - \Delta t)) \cdot y_j(\tau - \Delta t)] \cdot d\tau \qquad (5)$$

$$\frac{\partial E}{\partial b_j} = \int_{t_0}^{t} [S_j \cdot z_j(\tau) \cdot f'(x_k(\tau - \Delta t))] \cdot d\tau \qquad (6)$$

$$\frac{\partial E}{\partial S_j} = \int_{t_0}^{t} [z_j(\tau) \cdot (f(x_j(\tau - \Delta t)) - y_j(\tau - \Delta t))] \cdot d\tau \qquad (7)$$

The backpropagated errors ($z_j(\tau)$) for a j^{th} neuron can be written as following:

$$z_j(\tau) = \frac{\partial E}{\partial y_j(\tau)} = \sum_k [z_k(\tau + \Delta t) \cdot S_k \cdot f'(x_k(\tau)) \cdot w_{jk}] + z_j(\tau + \Delta t) \cdot (1 - S_j) + e_j(\tau) \cdot \Delta t \qquad (8)$$

Where *f'* is the derivative of the activation function (tanh). For each neuron, the z_j is computed by considering the network states at each time step as successive layers. Thus, the output errors are backpropagated in the network and in time. In BPTT learning, z_j is computed for each neuron and each time step since the starting. To prevent a memory explosion induced by the storage of every network states, a truncated BPTT algorithm (BPTT(h)) is proposed in [8]. This algorithm is keeping in memory only the past states included in a window following the current instant ($t_0 = t - h$ with h the window width). The z_j values are computed along this sliding window ($\tau \in [t\text{-}h;t]$). Hence, only a gradient approximation is computed. The previous (5) to (8) equations are constructed on the assumption that the parameters are constant along the time window. This assumption presented in [9] and [10] minimizes the process memory needs as the parameters history is not stored. In our study, these algorithms are experimented on the ROBIAN biped robot described in the next section.

3 ROBIAN identification description

3.1 Plant description and perturbation signals

To focus on the time window width influence, a model identification rather than a controller shaping will be carried out. Fig.1 shows how the identification of ROBIAN's torso influence on the ZMP (Zero Moment Point) will be performed. For more details concerning the ROBIAN biped and its torso mechanism see [1].

The ZMP is a key notion in the balance control of walking robot. It corresponds to the position of the center of pressure on the ground. The ZMP algorithm, introduced thirty five years ago [11], consists of controlling the equilibrium of the biped robot by keeping the ZMP inside the polygon defined by the contact points with the ground.

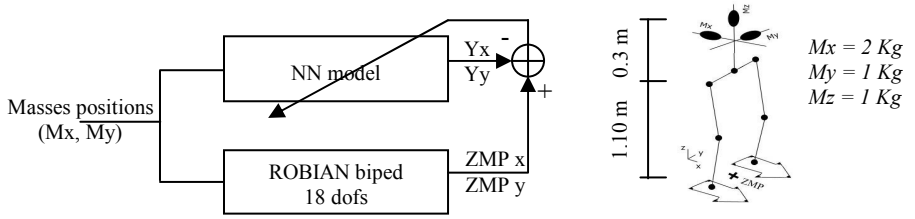

Fig. 1. Learning architecture for the identification (*left*) of ROBIAN robot (*right*)

The motion of the X and Y masses of the torso (Mx and My) are perturbing the robot balancing, leading to variations of the X and Y ZMP positions. During the learning process, the neural network (NN) computes the gap between the measured positions and its own outputs and modifies its parameters to vanish this error, by minimizing a cost function. This function is defined as the normalized squared errors sum on the two axes and the output errors ($e_j(t)$) can be expressed as following:

$$Cost = \left(\frac{ZMP_x - y_x}{maxZMP_x}\right)^2 + \left(\frac{ZMP_y - y_y}{maxZMP_y}\right)^2 \quad (9) \qquad e_{axis}(t) = \frac{\partial Cost}{\partial y_{axis}(t)} = -\frac{2\cdot\left(ZMP_{axis} - y_{axis}(t)\right)}{maxZMP_{axis}} \quad (10)$$

Where y_x, y_y are the output activities and $maxZMP_{axis}$ is the maximum amplitude on each axis (for normalization).

Representative situations should be given 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 for the X and Y axis. Fig 2 represents the learning pattern adopted.

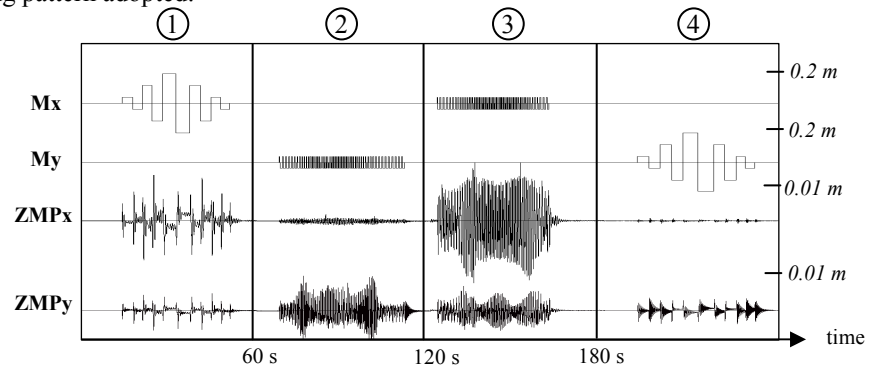

Fig. 2. Masses positions and ZMP positions on the two axis for the learning pattern (sampling rate = 45Hz). In periods 1 and 4, the X and then the Y mass, are submitted to successive steps with various amplitudes. In the 2 and 3 periods, steps with varying frequency are applied

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 axes. In the test pattern, a sine command with a varying frequency is applied on both axes.

3.2 Example of learning

The cost function evolution is depicted on Fig. 3. 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 (learning rates: η_1 = 0.025 for weights and biases, η_2 = 1.25 for time constants) and a time window width h=20.

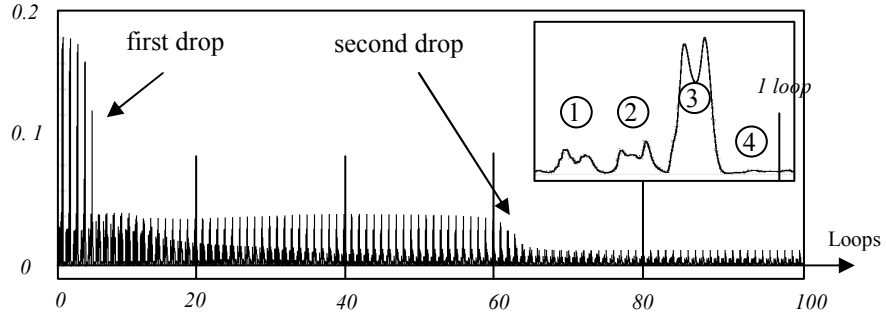

Fig. 3. Cost evolution, and a zoom on the first loop (*top right corner*), 1 loop =10200 iterations

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.3 gives the evolution of the cost during the learning. In the zoom area, the four peaks are due to the different kinds of command (amplitude changes: peaks 1 and 4, or frequency variation: 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 direction behavior) and peak 2 (learning on Y direction). The instant when the second drop happened (**SDI** : Second Drop Instant) is a good factor to quantify the convergence speed. It happens when the network identifies correctly the behavior of ROBIAN torso in the both two directions. Here it takes about 64 loops to arise. The SDI value can be interpreted as a physical expression of the convergence for our experiments. It can be related to the choice of a threshold below which the cost is considered as satisfactory.

After the learning, a relevant value for estimating the generalization ability of the learned network is the costs sum along the whole 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 is equal to zero. For the considered example, before the learning, TES = 363, and at the end of the learning process (after the 100th loop), the remaining error decreases to TES = 40.

4 Time window width influence

4.1 Stability limit with the "constant parameters" assumption

In the usual gradient calculus, the parameters are considered as constant along the time window [9], [10]. This approximation can be seen in the gradient equations (5) to (8) where the parameters (w_{ij}, b_j or S_j) are not indexed by time. The time window width reduces the use of this assumption by destabilizing the learning.
The identification of the ROBIAN's torso influence on the ZMP positions is carried out for different time window width values (TW) from 5 to 60 states stored. For each TW, ten learning courses are carried out for a statistical analysis of the results. Each network is evaluated on the test pattern after learning. The learning and test results are depicted on the following figure:

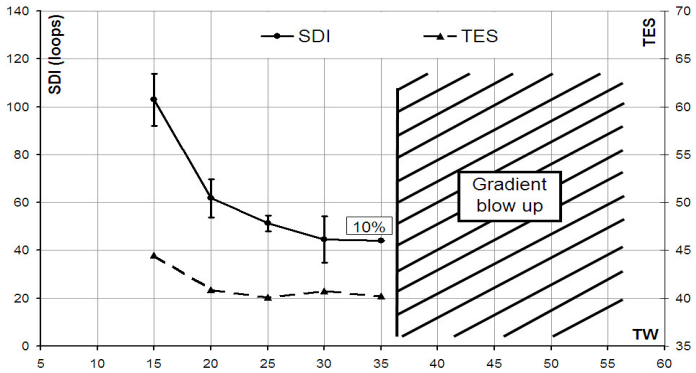

Fig. 4. Learning and test results following TW with η_1=0.025 for weights and biases, η_2=1.25 for time constants. The average value of SDI and TES are plotted. The error bars correspond to the standard deviation measured. They express the iteration range of convergence. If not all the 10 trials lead to a convergence, a mark is added on the graph giving the percentage of success. Here, it happens for TW=35 where there was only 10% of success

This graph can be decomposed in three parts. When TW is less than 15, the learning is so slow that it doesn't manage to converge (no results on the graph). For TW$\in$[15;35], the SDI, and TES seem to decrease proportionally to TW, thus the convergence speed and the generalization ability are better for larger TW. Finally, if TW is greater than 35 states stored, the algorithm is diverging with a gradient blow up. For a stronger learning rate (η=0.1) the gradient blow up occurs since TW=15.

4.2 Gradient blow up divergence and parameters evolution storage

We called this divergence problem gradient blow up since it's caused by a sudden explosion of the neurons gradient values. In Fig. 5, the evolution in time of the gradi-

ent values for each neuron are depicted. In this experiment, the time window width is TW=100 states stored (with a sampling rate=45Hz, it corresponds to an interval of 2.22s). The three graphs represent the initiation of the gradient blow up. Just after the last graph, all the gradient values are exploded.

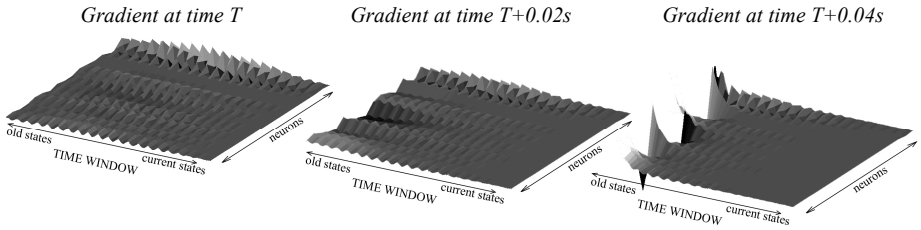

Fig. 5. Gradient blow up initiation. The little waves correspond to the "normal" backpropagation of the error in the time window on the neurons. The strong drops and jumps in the third graph correspond to the "abnormal" divergence called gradient blow up

The blow up begins with a divergence of the oldest past values. Through the recurrent links raised in the network, the explosion is spread in the entire network. Finally, the divergence is so important that the parameters take infinite values and the algorithm is stopped. Our guess is that the divergence of the farer gradients is due to the "constant parameter" assumption. This assumption means that the parameters are the same along the entire time window. However, if the window width is large compared to the parameters variation, it's obviously false. Actually, if the width is important, the parameter values that contribute to create the farer past states stored could be strongly different from the current ones. This happens when either the first or the second drop is initiated, i.e. when the parameters are strongly modified. That's why the divergence is initiated in the oldest part of the time window. Hence, the assumption could only be used with short window or small learning rates.

A way to avoid this blow up problem is to store the parameters history along the time window. So, the gradient equations will take into account their evolutions as in (5'), (6') and (7'). The weights, biases and time scale parameters become a function of time τ. The storage of the parameter history along the sliding time window implies an increase of memory needs. The number of stored values is multiplied by TW.

$$\frac{\partial E}{\partial w_{jk}} = \int_{t0}^{t} \left[S_k(\tau-1) \cdot z_k(\tau) \cdot f'\left(x_k(\tau-\Delta t)\right) \cdot y_j(\tau-\Delta t) \right] \cdot d\tau \qquad (5')$$

$$\frac{\partial E}{\partial b_j} = \int_{t0}^{t} \left[S_j(\tau-1) \cdot z_j(\tau) \cdot f'\left(x_k(\tau-\Delta t)\right) \right] \cdot d\tau \qquad (6')$$

$$z_j(\tau) = \sum_k \left[z_k(\tau+\Delta t) \cdot S_k(\tau) \cdot f'\left(x_k(\tau)\right) \cdot w_{jk}(\tau) \right] + z_j(\tau+\Delta t) \cdot \left(1 - S_j(\tau)\right) + e_j(\tau) \cdot \Delta t \qquad (8')$$

Equation (7) is not changed since no parameter is involved. This modification of the classical BPTT learning equations will be used for the analysis of TW influence on the speed and accuracy of the learning in next section.

4.3 Influence on the convergence speed and on the accuracy of the learned nets

The same identification process is carried out with the modified learning algorithm, for time window width values from 5 to 60 states stored, and with 10 initial random nets for each TW value. The learning and test results are depicted on the fig 6:

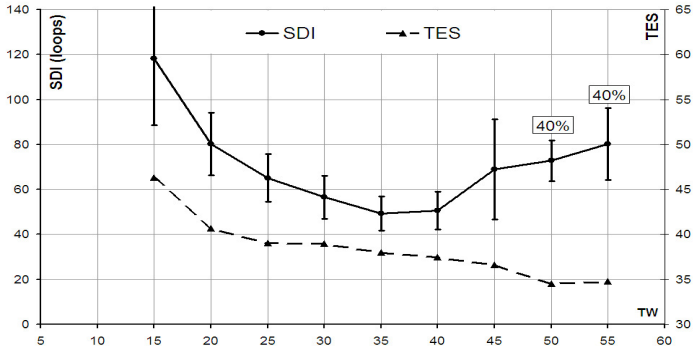

Fig. 6. Learning and test results following the TW values with η_1=0.025 and η_2=1.25. All the learning succeed for TW≤45, for TW= 50 and 55, only 40% of the nets reach a convergence

The previous graph can be decomposed in three parts depending on the TW value. If TW<15 states stored, the algorithm is not able to converge to a correct solution. Next, for TW varying between from 15 to 35, the convergence speed average is increased while the average TES is decreased. Then, with TW>35, the convergence process is more oscillating. The standard deviations are thus larger, and not all the 10 initial random networks lead to a correct solution. But, for these TW values, in case of convergence, the algorithm finds better solutions (i.e. the TES values are smaller). This is due to the amount of data taken into account. The modified algorithm allows also increasing the learning rates. Figure 7 represents the results obtained with η_1=0.1:

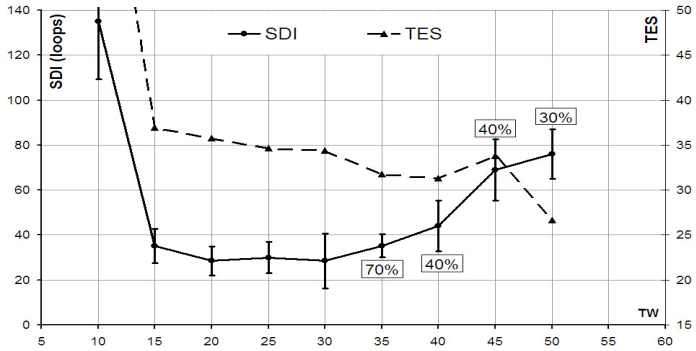

Fig. 7. Learning and test results with η_1=0.1 for weights and biases and η_2=1.25 for time constants. All the learning succeed for TW≤35, for TW= 40, 45, 50 and 55, the percentages of success are respectively 70%, 40%, 40%, 30% and 0% for 55

Compared to the experiments done with η_1=0.025, the convergence speed gets faster since a TW = 15 states stored. Nevertheless, the learning is more unstable with this

higher learning rate, and the algorithm meets convergence difficulties earlier. For larger TW values, the number of success is less important as the learning is oscillating, due to a too important learning rate. But, when they converge, the networks learned are more accurate. Here again, the TES obtained are the best for the biggest TW. As far as our experiment is concerned, the best configuration ensuring maximum speed, best quality and high percentage of success, is TW=30 and η_1=0.1.

5 Discussion and conclusion

5.1 Storing the parameters evolution?

The comparison between the results of the "constant parameters" assumption and those presented for the modified algorithm, shows that the second one is better for our identification experiment. First, the classical one is only valid for small TW values whereas the second one allows larger ones. Next, for the same TW range, the classical method finds networks with worst generalization abilities. The only advantage of the classical one seems to be a faster convergence speed. Nevertheless, as a stronger learning rate is not prohibited with the modified method, this speed advantage can be overcome. In that case, the modified algorithm results are comparable to the ones achieved with a smaller learning rate and a larger TW value. With the modified algorithm, for the largest TW values, the convergence is not always met. The algorithm fails to find a correct solution. But no gradient blow up occurs.
The TW limit that leads to a gradient blow up for the classical method is probably linked to the dynamic of the studied system and the convergence speed defined by the learning rates values. If the learning rates are strong, the parameters modification will be fast and they couldn't be approximated as constant along the time window. In the same way, if the system is fast, the learning will tend to bring the time constants and the network to a faster dynamic. Thus, the errors will be quickly backpropagated in the time window, and the algorithm will behave as if the time window was bigger and will diverge for smaller TW. So, the choice between using or not the "constant parameters" assumption will depend on the dynamic of the system.

5.2 Convergence speed VS generalization ability

The comparison of the convergence speeds and the quality of the learned networks demonstrates that the TW value choice must be a trade-off between these two results. Choosing the largest one, i.e. the best quality, the speed could be strongly decreased or the convergence not guaranteed. Choosing the fastest convergence, with a shorter TW, could lead to non suitable networks. There is no general rule for optimizing the learning. The choice must be done following the needs and criteria fixed for the learning process. Here again, the dynamic of the studied system will act upon the TW that

leads to the fastest convergence. For the same criteria, the chosen TW could be different for a slow or fast system. But, allowing the use of larger TW is obviously useful.

5.3 Conclusions

In this paper, we studied the influence of the time window width parameter upon the stability, the learning speed and the quality of the networks obtained. Based on statistical experiments, aiming to identify the links between the balancing of a biped robot and its torso motion, we discussed the net parameters evolution storage and the choice of an optimal TW value. We found that it could be useful to enlarge the window to reach faster and more accurate learning. As the "constant parameter" assumption classically adopted for BPPTT(h) is not adapted for larger window, a modified algorithm taking into account the parameters history can be advantageous.
In the future, we will first carry out similar experiments on faster or slower systems to find out the influence of the system dynamic on the learning. Next, we will perform a mathematical analysis of the modified algorithm. The learning algorithm will be also applied to carry out inverse model identification and control.

References

1. Mohamed, B., Gravez, F., Ouezdou, F.B.: Emulation of the dynamic effects of human torso during walking gait. In Journal of Mechanical Design, vol. 126, p830-841, Sept 2004.
2. Tsung, F-S.: Modeling Dynamical Systems with Recurrent Neural Networks. PhD thesis, Depart-ment of Computer Science. University of California, San Diego, 1994.
3. Nguyen, M.H., Cottrell, G.W.: Tau Net: A neural network for modeling temporal variability. In Neurocomputing 15 pp. 249-271, 1997.
4. Hochreiter, S., Younger, A.S., Conwell, P.R.: 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.
5. Pearlmutter, B.A.:Gradient calculation for dynamic recurrent neural networks: a survey. In Transactions on Neural Networks, 6(5):1212-1228, 1995.
6. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, vol. 78, no. 10, pp. 1550 1560, 1990.
7. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: 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.
8. Williams, R. J., Zipser, D.: 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.
9. Williams, R. J., Peng, J.: An efficient gradient--based algorithm for on--line training of recurrent network trajectories. In Neural Computation, vol 2 p 490-501 (MIT Press). 1990
10. Campolucci, P., Uncini, A., Piazza, F., Rao, B. D.: On-Line Learning Algorithms for Locally Recurrent Neural Networks. In IEEE-NN vol 10 p253. March 1999.
11. Vukobratovic, M. and Borovac, B.:Zero-moment point – thirty five years of its life. In International Journal of Humanoid Robotics. 1(1) p 157-173. 2004.