2

Advanced Methods for Time Series Prediction Using Recurrent Neural Networks

Romuald Boné[1,2] and Hubert Cardot[2]
[1]National Engineering School of Loire Valley
[2]University François Rabelais Tours
France

1. Introduction

Time series prediction has important applications in various domains such as medicine, ecology, meteorology, industrial control or finance. Generally the characteristics of the phenomenon which generates the series are unknown. The information available for the prediction is limited to the past values of the series. The relations which describe the evolution should be deduced from these values, in the form of functional relation approximations between the past and the future values.
The most usually adopted approach to consider the future values $\hat{x}(t+1)$ consists in using a function f which takes as input a time window of fixed size M representing the recent history of the time series.

$$\mathbf{x}(t) = [x(t), x(t-\tau), \dots, x(t-(M-1)\tau)] \tag{1}$$

$$\hat{x}(t+\tau) = f(\mathbf{x}(t)) \tag{2}$$

where $x(t)$, for $0 \le t \le l$, is the time series data that can be used for building a model.
Most of the current work on single-step-ahead prediction relies on a result released in (Takens, 1981) which shows that under several assumptions (among which the absence of noise), it is possible to obtain a perfect estimate of $x(t+\tau)$ according to (2) if $M \ge 2d+1$, where d is the dimension of the stationary attractor generating the time series. In this approach, the memory of the past is preserved in the sliding time window.
In multi-step-ahead prediction, given $\{x(t), x(t-\tau), \dots, x(t-n\tau), \dots\}$, one is looking for a good estimate $\hat{x}(t+h\tau)$ of $x(t+h\tau)$, h being the number of steps ahead.
Given their universal approximation properties, neural networks, such as multi-layer perceptrons (MLPs) or recurrent networks (RNs), are good candidate models for the global approaches. Among the many neural network architectures employed for time series prediction, one can mention MLPs with a time window in the input (Weigend et al., 1990), MLPs with finite impulse response (FIR) connections (equivalent to time windows) both from the input to the hidden layer and from the hidden layer to the output (Wan, 1994), recurrent networks obtained by providing MLPs with a feedback from the output (Czernichow, 1996), simple recurrent networks (Suykens & Vandewalle, 1995), recurrent

networks with FIR connections (El Hihi & Bengio, 1996), (Lin et al., 1996) and recurrent networks with both internal loops and feedback from the output (Parlos et al., 2000).
But the use of these architectures for time-series prediction has inherent limitations, since the size of the time window or the number of time delays of the FIR connections is difficult to choose.
An alternative solution is to keep a small length (usually $M = 1$) time window and enable the model to develop on its own a memory of the past. This memory is expected to represent the past information that is actually needed for performing the task more accurately. Time series prediction with RNNs usually corresponds to such a solution. Memory of the past – of variable length, see e.g. (Aussem, 2002; Hammer & Tino, 2003) – is maintained in the internal state of the model, $\mathbf{s}(t)$, of finite dimension d at time t, which evolves (for $M = 1$) according to:

$$\mathbf{s}(t+\tau) = \mathbf{g}(\mathbf{s}(t), x(t)) \tag{3}$$

where $\mathbf{g}$ is a mapping function assumed to be continuous and differentiable. The time variable t can either be continuous or discrete and $\mathbf{h}$ is the output function. Assuming that the system is noise free, the observed output is related to the internal dynamics of the system by:

$$\hat{x}(t+\tau) = \mathbf{h}(\mathbf{s}(t)) \tag{4}$$

where $\hat{x}(t+\tau)$ is the estimate of $x(t+\tau)$ and the function $\mathbf{h}$ is called the measurement function.
Globally feed-forward architectures, both very common and with a short calculation time, are widely used. They share the characteristic of having been initially elaborated for using the error gradient back-propagation of feed-forward neural networks (some of which have an adapted version today (Campolucci et al., 1999)). Hence the locally recurrent globally feed-forward networks (Tsoi & Back, 1994) introduce particular neurons, with local feedback loops. In the most general form, these neurons feature delays in inputs as well as in their loops. All these architectures remain limited: hidden neurons are mutually independent and therefore, cannot pick up some complex behaviors which require the collaboration of several neurons of the hidden layer. In order to overcome this problem, a certain number of recurrent architectures have been suggested (see (Lin et al., 1996) for a presentation). It has been shown that in practice the use of delay connections in these networks gives rise to a reduction in learning time (Guignot & Gallinari, 1994) as well as an improvement in the taking into account of long term dependencies (Lin et al., 1996; Boné et al., 2002). The resulting network is named Time Delay Recurrent Neural Networks (TDRNN). In this case, unless to apply an algorithm for selective addition of connections with time delays (Boné et al., 2002), which improve forecasting performance capacity but at the cost of increasing computations, the networks finally retained are often oversized and use meta-connections with consecutive delay connections, also named Finite Impulse Response (FIR) connections or, if they contain loops, Infinite Impulse Response (IIR) connections (Tsoi & Back, 1994).
Recurrent neural networks (RNNs) is a class of neural networks where connections between neurons form a directed cycle. They possess an internal memory owing to cycles in their connection graph and do no longer need a time window to take into account the past values

of the time series. They are able to model temporal dependencies of unspecified duration between the inputs and the associated desired outputs, by using internal memory. The passage of information from one neuron to the other through a connection is not instantaneous (one time step), unlike MLP, and thus the presence of the loops makes it possible to keep the influence of the information for a variable time period, theoretically infinite. The memory is coded by the recurrent connections and the outputs of the neurons themselves. Throughout the training, the network learns how to complete three complementary tasks: the selection of useful inputs, their retention in coded form and their use in the calculation of its outputs.
RNNs are computationally more powerful than feed-forward networks (Siegelmann et al, 1997), and valuable approximation results were obtained for dynamical systems (Seidl & Lorenz, 2001).

2. RNNs learning

During the last two decades, several methods for supervised training of RNNs have been explored. BackPropagation Through Time (BPTT) is probably the most widely used method. BPTT is an adaptation of the well-known backpropagation training method known from feedforward networks. It is therefore a gradient-based training method.

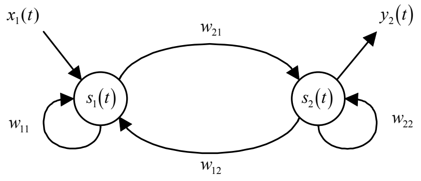

Fig. 1. A recurrent neural network

The feedforward backpropagation algorithm cannot be directly transferred to RNNs, because the backpropagation pass presupposes that the connections between the neurons induce a cycle-free ordering. Considering a time series of length l, the central idea of BPTT algorithm is to unfold the original recurrent networks (Fig. 1) in time so as to obtain a feedforward network with l layers (Fig. 2), which in turn makes it possible to apply the learning method by backpropagation of gradient of the error through time. BPTT unfolds the network in time by stacking identical copies of the RNN, and duplicating connections within the network to obtain connections between subsequent copies.
The weights between successive layers must remain identical in order to be able to show up in the original recurrent network. In practice, it amounts to cumulating the changes of the weights for all the copies of a particular connection and to adding the sum of the changes to all these copies after each learning iteration.
Let us consider the application of BPTT for the training of recurrent networks between time t_1 and t_l. f_i is the transfer function of neuron i, $s_i(t)$ its output at time t, and w_{ij} its connection from neuron j. A value, provided to the neuron at time t, coming from outside, is noted $x_i(t)$.
The algorithm supposes an evolution of neurons of recurrent networks given by the following equations:

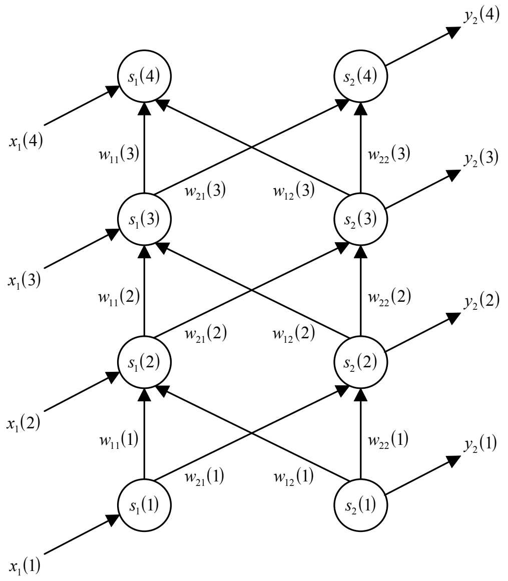

Fig. 2. RNN of Fig. 1 unfolded in time

$$s_i(t) = f(net_i(t-1)) + x_i(t)\,;\;\; i = 1, \ldots, N \tag{5}$$

$$net_i(t-1) = \sum_{j \in pred(i)} w_{ij}(t-1) s_j(t-1) \tag{6}$$

The set $\mathrm{Pred}(i)$ contains, for each neuron i, the index of the incoming neurons $\mathrm{Pred}(i) = \left\{ j \in N \,\middle|\, \exists \left(w_{ij}, \tau_{ij} \right) \right\}$. Likewise, we have defined the successors of a neuron i: $\mathrm{Succ}(i) = \left\{ j \in N \,\middle|\, \exists \left(w_{ji}, \tau_{ji} \right) \right\}$.

The variation of the weight for all the sequence is calculated by the sum of the variations of this weight on each element of the sequence. By noting $T(\tau)$ the set of neurons which have a desired output $d_p(\tau)$ at time τ, we define the mean quadratic error $E(t_1, t_l)$ of the recurrent neural networks between time t_1 and t_l as:

$$E(t_1, t_l) = \frac{1}{2} \sum_{t=t_1}^{t_l} \sum_{p \in T(t)} \left(d_p(t) - s_p(t) \right)^2 \tag{7}$$

To minimize total error, gradient descent is used to change each weight in proportion to its derivative with respect to the error, provided the non-linear activation functions are differentiable (η is the learning step):

$$\Delta w_{ij}(t_1, t_l - 1) = -\eta \frac{\partial E(t_1, t_l)}{\partial w_{ij}} = -\eta \sum_{\tau=t_1}^{t_l-1} \frac{\partial E(t_1, t_l)}{\partial w_{ij}(\tau)} \tag{8}$$

with

$$\frac{\partial E(t_1,t_l)}{\partial w_{ij}(\tau)}=\frac{\partial E(t_1,t_l)}{\partial net_i(\tau)}\frac{\partial net_i(\tau)}{\partial w_{ij}(\tau)} \tag{9}$$

where $w_{ij}(\tau)$ is the duplication of the weight w_{ij} of the original recurrent networks, for the time $t=\tau$. We expand $\partial E(t_1,t_l)/\partial net_i(\tau)$:

$$\frac{\partial E(t_1,t_l)}{\partial net_i(\tau)}=\frac{\partial E(t_1,t_l)}{\partial s_i(\tau+1)}\frac{\partial s_i(\tau+1)}{\partial net_i(\tau)} \text{ with } \frac{\partial s_i(\tau+1)}{\partial net_i(\tau)}=f'(net_i(\tau)) \tag{10}$$

If neuron i belongs to the last layer ($\tau=t_l-1$):

$$\frac{\partial E(t_1,t_l)}{\partial s_i(\tau+1)}=\frac{\partial e(t_l)}{\partial s_i(t_l)}=\underset{i\in T(\tau+1)}{\delta}\left(s_i(t_l)-d_i(t_l)\right) \tag{11}$$

where $\underset{i\in T(\tau+1)}{\delta}=1$ if $i\in T(\tau+1)$ and 0 otherwise. If neuron i belongs to the preceding layers:

$$\frac{\partial E(t_1,t_l)}{\partial s_i(\tau+1)}=\frac{\partial e(\tau+1)}{\partial s_i(\tau+1)}+\sum_{j\in Succ(i)}\left(\frac{\partial E(t_1,t_l)}{\partial net_j(\tau+1)}\frac{\partial net_j(\tau+1)}{\partial s_i(\tau+1)}\right) \tag{12}$$

As $\partial net_j(\tau+1)/\partial s_i(\tau+1)=w_{ji}(\tau+1)$, the equations of BPTT algorithm are finally obtained:

$$\Delta w_{ij}(t_1,t_l-1)=-\eta\sum_{\tau=t_1}^{t_l-1}\frac{\partial E(t_1,t_l)}{\partial net_i(\tau)}s_j(\tau) \tag{13}$$

- with, for the output layer

$$\frac{\partial E(t_1,t_l)}{\partial net_i(\tau)}=\begin{cases}\left[s_i(t_l)-d_i(t_l)\right]f_i'(net_i(\tau)) \text{ if } i\in T(\tau+1)\\ 0 \quad \text{else}\end{cases} \tag{14}$$

- and for the hidden layer

$$\frac{\partial E(t_1,t_l)}{\partial net_i(\tau)}=\begin{cases}\left[\begin{array}{l}\left[s_i(\tau+1)-d_i(\tau+1)\right]\\ +\sum\limits_{j\in Succ(i)}\dfrac{\partial E(t_1,t_l)}{\partial net_j(\tau+1)}w_{ji}(\tau+1)\end{array}\right].f_i'(net_i(\tau)) \text{ if } i\in T(\tau+1)\\ \sum\limits_{j\in Succ(i)}\dfrac{\partial E(t_1,t_l)}{\partial net_j(\tau+1)}w_{ji}(\tau+1)f_i'(net_i(\tau)) \quad \text{else}\end{cases} \tag{15}$$

Eq. (13) to (15) allow to apply error gradient backpropagation through time: after the forward pass, witch consists in updating the unfolded network, starting from the first copy of the recurrent network and working upwards through the layers, $\partial E(t_1,t_l)/\partial net_i(\tau)$ is computed, by proceeding backwards through the layers $t_l,..,t_1$.

One epoch requires O(lM) multiplications and additions, where M is the total number of network connections. Many speed-up techniques for gradient descent approach are

described in the literature, e.g. dynamic learning rate adaptation schemes. Another approach to achieve faster convergence is to use second-order gradient descent techniques.
Unfortunately, the gradient descent algorithms which are commonly used for training RNNs have several limitations, the most important one being the difficulty of dealing with long-term dependencies in the time series (Bengio et al, 1994; Hochreiter & Schmidhuber 1997) i.e. problems for which the desired output depends on the inputs presented at times far in the past.
Backpropagated error gradient information tends to "dilute" exponentially over time. This phenomenon is called "vanishing gradient" or "forgetting behavior" (Frasconi et al., 1992; Bengio et al, 1994). (Bengio et al, 1994) have demonstrated the existence of a condition on the eigenvalues of the RNN Jacobian to be able to store information for a long period of time in the presence of noise. But this implies that the portion of gradient due to information at times $\tau << t$ is insignificant compared to the portion of gradient at times near t.

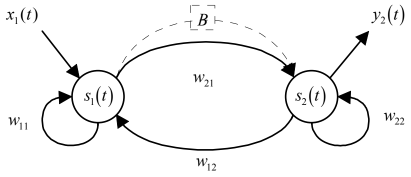

Fig. 3. Delayed connection added to a RNN (dotted line)

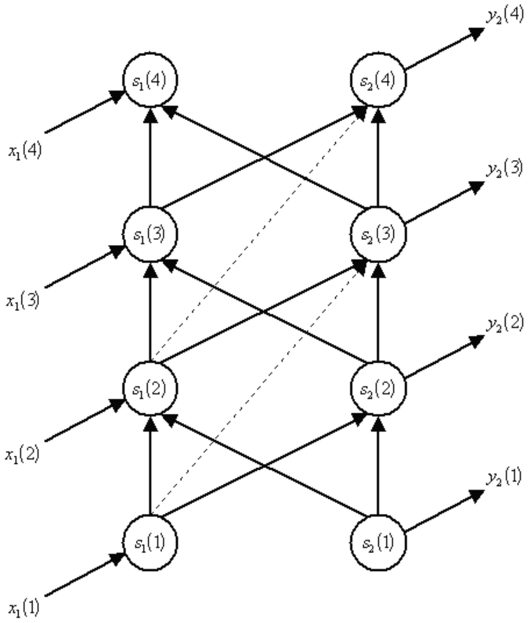

Fig. 4. The RNN of Fig. 3 unfolded in time. Duplicated dotted connections correspond to the added connection

We can give a more intuitive explanation for backpropagated gradient vanishing. Considering eq. (13) to (15), gradient calculation for each layer is done by a product with transfer function derivate. Most of the time, this last value is bounded between 0 and 1 (i.e. sigmoid function). Each time the signal is backpropagated through a layer, the gradient contribution of the forward layers is attenuated. Along the time-delayed connections the signal does no longer cross nonlinear activation functions between successive time steps (see Fig. 3 and Fig. 4).

Adding connections with time delays to the RNN (El Hihi & Bengio, 1996; Lin, T., et al., 1996) often allows gradient descent algorithms to find better solutions in these cases. Indeed, by acting as a linear link between two distant moments, such a connection has beneficial effects on the expression of the gradient. Adding a delayed connection to an RNN (Fig. 3) creates several connections in the unfolded network (Fig. 4) jumping as many layers as the delay. Gradient backpropagated by these connections avoids attenuation of intermediate layers.

But in the absence of prior knowledge concerning the problem to solve, how can one choose the locations and the delays associated to these new connections? By systematically adding meta-connections with consecutive delay connections, also named Finite Impulse Response (FIR) connections, one obtains oversized networks which are slow to train and have poor generalization abilities. Various regularization techniques can be employed in order to improve generalization and this further increases the computational cost.

Constructive approaches for adapting the architecture of a neural network are usually more economical. An algorithm for the addition of time-delayed connections to recurrent networks should start with a simple, ordinary RNN and progressively add new connections according to some heuristic. An alternative solution could be found in the learning of the connection delays themselves. We suggested, for an RNN that associates a delay to each connection, an algorithm based on the gradient which simultaneously adjusts weights and delays.

To improve the obtained results, we may also adapt general methods which authorize to improve the performances of various models. One such approach is to use a combination of models to obtain a more precise estimate than the one obtained by a single model. One such procedure is known under the name of boosting.

3. Constructive algorithms

Instead of systematically adding finite impulse response (FIR) connections to a recurrent network, each connection encompassing a whole range of delays, we opted for a constructive approach: starting with an RN having no time-delayed connections, then selectively adding a few such connections. The two algorithms we present in the following allow us to choose the location and the delay associated with a time-delayed connection which is added to an RN. The assumption we make is that significantly better results can be obtained by the addition of a small number of time-delayed connections to a recurrent network. The reader is invited to consult (Boné et al., 2000a; Boné et al, 2000b; Boné et al., 2002) for a more detailed discussion regarding the role of time-delayed connections in RNs. The iterative and constructive aspects diminish the effect of the vanishing gradient on the outcome of the algorithm. Indeed, by reinforcing the long-term dependencies in the network, the first time-delayed connections favor the subsequent learning steps. A high selectivity should allow us to avoid over-parameterized networks. For every iteration, we rank the candidate connections according to their relevance.

We retained two alternative methods for defining the relevance of a candidate connection. The first one is based on the amount by which the error diminishes after the addition of the connection. The second one relies on a more detailed study of various quantities computed inside the network during gradient descent.

3.1 Bounded exploration for the addition of time-delayed connections

The first heuristic is a breadth-first search (BFS). It explores the alternatives for the location and the delay associated with a new connection by adding that connection and performing a few iterations of the underlying learning algorithm. The connection that produces the largest increase in performance during these few iterations is then added, and the learning continues until error increases on the stop set. Another exploratory stage begins for the addition of a new connection. The algorithm eventually ends when the error on the stop set no longer decreases upon the addition of a new connection, or a (user-specified) bound on the number of new connections is reached. We employed BPTT as the underlying learning algorithm and we called this constructive algorithm Exploratory Back-Propagation Through Time (EBPTT). We must note that the breadth-first heuristic does not need any gradient information and can be applied in combination with learning algorithms which are not based on the gradient.

If the RNN we start with does not account well for the medium or long-term dependencies in the data, and these dependencies are not too complex, then by adding the appropriate connection the error is likely to diminish relatively fast.

Three new parameters are required for this constructive algorithm: the maximal value for the delay of a new connection, the maximal number of new connections and the number of BPTT steps performed for each candidate connection during the exploratory stage. In choosing the value of the first parameter one should ideally use prior knowledge related to the problem. If such information is not available one can rely on simple, linear measures such as auto or cross-correlations to find a bound for the long-term dependencies. Computational cost governs the choice of the two other parameters. However, the experiments we present in the following show that the contribution of the new connections diminishes quickly as their number increases. The complexity of the exploratory stage may seem quite high, $O(N^4)$, since after the addition of each candidate connection we carry out several steps of the BPTT algorithm on the entire network. The user is supposed to find a tradeoff between the quality of the results and the computation cost. When compared to the complete exploration of all the alternative architectures, this breadth-first search is only interesting if good results can be obtained with few learning steps during the exploratory stage. Fortunately, experimental evidence shows that this appears to be the case, so the global cost of the algorithm remains low.

3.2 Internal correlations

The second heuristic for defining the relevance of a candidate connection is closely dependent on BPTT-like underlying learning algorithms. Since this method makes use of quantities computed during gradient descent, its computation cost is significantly lower than for the breadth-first search.

When applying BPTT on the training set between t_1 and t_1, we obtain the following expression for the variation of one weight of delay k, $\Delta w_{ij}^{(k)}(t_1, t_1 - 1)$:

$$\Delta w_{ij}^{(k)}(t_1, t_l - 1) = \sum_{\tau=t_1+k}^{t_l-1} \Delta w_{ij}^{(k)}(\tau) = -\eta \sum_{\tau=t_1+k}^{t_l-1} \frac{\partial E(t_1, t_l)}{\partial w_{ij}^{(k)}(\tau)} \tag{16}$$

$w_{ij}^{(k)}(\tau)$ being the copy of $w_{ij}^{(k)}$ for $t = \tau$ in the unfolded network employed by BPTT. We may write

$$\frac{\partial E(t_1, t_l)}{\partial w_{ij}^{(k)}(\tau)} = \frac{\partial E(t_1, t_l)}{\partial net_i(\tau)} \cdot s_j(\tau - k) \qquad \tau \in [t_1 + k; t_l - 1] \tag{17}$$

We are looking for connections which are potentially useful in capturing (medium or long-term) dependencies in the data. A connection $w_{ij}^{(k)}$ is then useful only if it has a significant contribution to the computation of the gradient, i.e. $\partial E(t_1, t_l)/\partial w_{ij}^{(k)}(\tau)$ is significantly different from zero for many iterations of the learning algorithm. We select the output of a neuron $s_j(\tau - k)$ which best contributes to a reduction in error by means of $\partial E(t_1, t_l)/\partial net_i(\tau)$.
The resulting algorithm, called Constructive Back Propagation Through Time (CBPTT), computes during several BPTT steps the correlation between the values of $\partial E(t_1, t_l)/\partial net_i(\tau)$ and $s_j(\tau - k)$ for $\tau \in [t_1 + k; t_l - 1]$. The relevance of a candidate connection $w_{ij}^{(k)}$ is defined as the absolute value of this correlation. The connection with the highest relevance factor is then added to the RNN, its weight is initialized to 0, and learning continues. The process stops when a new connection has no further positive effect on the performance of the RNN, as evaluated on a stop set. The time complexity and the storage complexity of CBPTT is the same as for BPTT.
This constructive algorithm requires two new parameters: the maximal value for the delays of the new connections and the maximal number of new connections. The choice of these parameters is independent from the constructive heuristic, so the rules already mentioned for EBPTT should be applied. Experiments reported in (Boné et al., 2002) support the view that the precise value of this parameter does not have a high influence on the outcome, as long as it is higher than the significant linear dependencies in the data, which are given by the autocorrelation. The same experiments show that performance is not very sensitive to the bound on the number of new connections either, because the contribution of the new connections quickly diminishes as their number increases.
This definition for the relevance of a candidate connection is well adapted to time dependencies which are well represented in the available data. If this is not the case for the dependencies one is interested in, a more thorough study of the distribution of the product $s_j(\tau - k) \cdot \partial E(t_1, t_l)/\partial net_i(\tau)$ should suggest more adequate measures for the relevance.

4. Time Delay Learning

An alternative to the adding of connections with time delays could be found in the learning of the connection delays themselves. (Duro & Santos Reyes, 1999) (see also (Pearlmutter 1990)) have suggested, for a feed-forward neural networks that associate a delay to each connection, an algorithm based on the gradient which simultaneously adjusts weights and delays. We adapted this technique to a recurrent architecture.

Considering an RNN in which two values are associated to each connection from a neuron j to a neuron i, these two values are of a usual weight w_{ij} of the signal and a delay τ_{ij} which is a real value indicating the needed time for the signal to propagate through the connection. Note that this parameter is not the same as the maximal order of a FIR connection: indeed, when we consider a connection of delay τ_{ij}, we do not have simultaneously $\tau_{ij}-1$ connections with integer delays between 1 and τ_{ij}. The neuron output $s_i(t)$ is given by:

$$s_i(t)=f_i(net_i(t-1)) \text{ with } net_i(t-1)=\sum_{j\in Pred(i)} w_{ij}s_j(t-\tau_{ij}-1) \tag{15}$$

The values $s_j(t-\tau_{ij}-1)$ are obtained by applying a linear interpolation between the two nearest whole numbers of the delay τ_{ij}.

We have adapted the BPTT algorithm to this architecture with a simultaneous learning of weights and delays of the connections, inspired from (Duro & Santos Reyes, 1999). The variation of a delay τ_{ij} can be computed as the sum of the variations of this parameter copies corresponding to the times from t_1 to t_l. Then we add this variation to all copies of τ_{ij}. We will only give here the demonstration of the learning of the delays as the learning of the weight can easily be deducted from it.

We note $\tau_{ij}(\tau)$ the copy of τ_{ij} for $t=\tau$ in the unfold in time neural net which is virtually constructed with BPTT. $\lceil . \rceil$ is the operator of upward roundness.

We apply a back-propagation of the gradient of the mean quadratic error $E(t_1,t_l)$ which is defined as the sum of the instantaneous errors $e(t)$ from t_1 to t_l:

$$E(t_1,t_l)=\sum_{t=t_1}^{t_l} e(t)=\sum_{t=t_1}^{t_l}\frac{1}{2}\sum_{p\in T(t)}(d_p(t)-s_p(t))^2 \tag{16}$$

$$\Delta\tau_{ij}(t_1,t_l-1)=-\lambda\frac{\partial E(t_1,t_l)}{\partial \tau_{ij}}=\sum_{\tau=t_1+\lceil\tau_{ij}\rceil}^{t_l-1}\Delta\tau_{ij}(\tau)=-\lambda\sum_{\tau=t_1+\lceil\tau_{ij}\rceil}^{t_l-1}\frac{\partial E(t_1,t_l)}{\partial\tau_{ij}(\tau)} \tag{17}$$

We can write $\partial E(t_1,t_l)/\partial\tau_{ij}(\tau)=\partial E(t_1,t_l)/\partial net_i(\tau)\bullet\partial net_i(\tau)/\partial\tau_{ij}(\tau)$. With a first order approximation, $\partial net_i(\tau)/\partial\tau_{ij}(\tau)\approx w_{ij}(s_j(\tau-\tau_{ij}-1)-s_j(\tau-\tau_{ij}))$. We expand $\partial E(t_1,t_l)/\partial net_i(\tau)$ following Eq. 10. If neuron i belongs to the last layer ($\tau=t_l-1$), we apply Eq. 11. If neuron i belongs to one of the preceding layers:

$$\frac{\partial E(t_1,t_l)}{\partial s_i(\tau+1)}=\frac{\partial e(\tau+1)}{\partial s_i(\tau+1)}+\sum_{j\in Succ(i)}\left(\frac{\partial E(t_1,t_l)}{\partial net_j(\tau+\tau_{ji}+1)}\frac{\partial net_j(\tau+\tau_{ji}+1)}{\partial s_i(\tau+1)}\right) \tag{18}$$

As $\partial net_j(\tau+\tau_{ji}+1)/\partial s_i(\tau+1)=w_{ji}(\tau+1)$, we obtain the final relations to learn the delay associated to each connection:

$$\Delta\tau_{ij}(t_1,t_l-1)=-\lambda\sum_{\tau=t_1+k}^{t_l-1}\frac{\partial E(t_1,t_l)}{\partial net_i(\tau)}w_{ij}(s_j(\tau-\tau_{ij}-1)-s_j(\tau-\tau_{ij})) \tag{19}$$

with for $\tau=t_l-1$

$$\frac{\partial E(t_1, t_l)}{\partial \mathrm{net}_i(\tau)} = \delta_{i \in T(\tau+1)} \left(s_i(t_l) - d_i(t_l)\right) f_i'\left(\mathrm{net}_i(\tau)\right) \tag{20}$$

and for $t_1 \le \tau < t_l - 1$

$$\frac{\partial E(t_1, t_l)}{\partial \mathrm{net}_i(\tau)} = \begin{bmatrix} \delta_{i \in T(\tau+1)} \left(s_i(\tau+1) - d_i(\tau+1)\right) \\ + \sum\limits_{j \in \mathrm{Succ}(i)} \dfrac{\partial E(t_1, t_l)}{\partial \mathrm{net}_j\left(\tau + \tau_{ji} + 1\right)} w_{ji}(\tau+1) \end{bmatrix} f_i'\left(\mathrm{net}_i(\tau)\right) \tag{21}$$

5. Boosting Recurrent Neural Networks

To improve the RNN forecasting results, we may use a combination of models to obtain a more precise estimate than the one obtained by a single model. In the boosting algorithm, the possible small gain a "weak" model can bring compared to random estimate is boosted by the sequential construction of several such models, which concentrate progressively on the difficult examples of the original training set. Boosting (Schapire, 1990; Freund & Schapire, 1997; Ridgeway et al., 1999) works by sequentially applying a classification algorithm to re-weighted versions of the training data, and then taking a weighted majority vote of the sequence of classifiers thus produced. Freund and Schapire (Freund & Schapire, 1997) presented the Adaboost. R algorithm that attacks the regression problem by reducing it to a classification problem.

A different approach to regressor boosting as residual-fitting was developed in (Duffy & Helmbold, 2002; Buhlmann & Yu 2003). Instead of being trained on a different sample of the same training set, as in previous boosting algorithms, a regressor is trained on a new training set having different target values (e.g. the residual error). Before presenting briefly our algorithm, studied in (Assaad et al, 2005), let us mention that in (Cook & Robinson, 1996) a boosting method is applied to the classification of phonemes, with RNNs as learners. The authors are the first ones to have noticed the implications of the internal memory of the RNNs on the boosting algorithm.

The boosting algorithm employed should comply with the restrictions imposed by the general context of the application. In our case, it must be able to work well when a limited amount of data is available and to accept RNNs as regressors. We followed (Assaad et al, 2008) the generic algorithm of (Freund, 1990). Our updates are based on the suggestion in (Drucker, 1999), but we apply a linear transformation to the weights before we employ them (see the definition of $D_{n+1}(q)$ in Table 1) in order to prevent the RNNs from simply ignoring the easier examples. Then, instead of sampling with replacement according to the updated distribution, we prefer to weight the error computed for each example (thus using all the data points) at the output of the RNN with the distribution value corresponding to the example.

For stage (2a), BPTT equations (14) and (15) become for the output layer:

$$\frac{\partial E(t_1, t_l)}{\partial \mathrm{net}_i(\tau)} = \begin{cases} [s_i(t_l)\text{-}d_i(t_l)] D_n(\tau+1) f_i'(\mathrm{net}_i(\tau)) & \text{if} \quad i \in T(\tau+1) \\ 0 \quad \text{else} \end{cases} \tag{22}$$

and for the hidden layer:

```
1. Initialize the weights for the examples: $D_1(q) = 1/Q$, and $Q$, the number of training examples. Put the iteration counter at 0: $n = 0$
2. Iterate
    (a) increment $n$. Learn with BPTT an RNN $h_n$ by using the entire training set and by weighting the squared error computed for example $q$ with $D_n(q)$, the weight of example $q$ for the iteration $n$;
    (b) update the weights of the examples:
        (i) compute $L_n(q)$ for every $q = 1,\cdots,Q$ according to the loss function:
            $L_n^{linear}(q) = \left|y_q^{(n)}(x_q) - y_q\right| / S_n$, $L_n^{quadratic}(q) = \left|y_q^{(n)}(x_q) - y_q\right|^2 / S_n^2$
            $L_n^{exponential}(q) = 1 - \exp\left(-\left|y_q^{(n)}(x_q) - y_q\right| / S_n\right)$, with
            $S_n = \sup_q \left|y_q^{(n)}(x_q) - y_q\right|$;
        (ii) compute $\varepsilon_n = \sum_{q=1}^{Q} D_n(q) L_n(q)$ and $\alpha_n = (1 - \varepsilon_n)/\varepsilon_n$;
        (iii) the weights of the examples become ($Z_n$ is a normalizing constant)
            $D_{n+1}(q) = \frac{1 + k \cdot p_{n+1}(q)}{Q + k}$ with $p_{n+1}(q) = \frac{D_n(q)\alpha_n^{(L_n(q)-1)}}{Z_n}$ until $\varepsilon_n < 0.5$.
3. Combine RNNs by using the weighted median.
```

Table 1. The boosting algorithm proposed for regression with recurrent neural networks

$$\frac{\partial E(t_1,t_l)}{\partial net_i(\tau)} = \begin{cases} \left[\begin{array}{l} [s_i(\tau+1) - d_i(\tau+1)]D_n(\tau+1) \\ + \sum_{j \in Succ(i)} \frac{\partial E(t_1,t_l)}{\partial net_j(\tau+1)} w_{ji}(\tau+1) \end{array}\right] f_i'(net_i(\tau)) & \text{if} \quad i \in T(\tau+1) \\ \sum_{j \in Succ(i)} \frac{\partial E(t_1,t_l)}{\partial net_j(\tau+1)} w_{ji}(\tau+1) f_i'(net_i(\tau)) & \text{else} \end{cases} \tag{23}$$

6. Single step ahead prediction results

The results we present here concern univariate regression only, but our algorithms are obviously not limited to such problems. We employed a natural dataset (sunspots) and two synthetic datasets (Mackey-Glass), which allow us to perform comparisons since many related results are published in the literature.

We applied our algorithms to RNNs having an input neuron, a linear output neuron, a bias unit and a recurrent hidden layer composed of neurons with the symmetric sigmoid (tanh) as activation function. We randomly initialized the weights in [-0.3, 0.3]. For the sunspots dataset we tested RNNs having 2 to 15 neurons in the hidden layer and for the Mackey-Glass RNNs having dataset 2 to 8 neurons. Except for boosting, we performed 20 experiments for each architecture. For boosting, we limited the experiments to 5 trial runs for each configuration: (linear, squared or exponential loss functions; value of parameter k), due to heavy calculation time, using the best architecture found by BPTT (12 neurons in the

hidden layer for sunspots, 7 neurons for the Mackey-Glass series). We set the maximal number n of RNNs at 50 for each experiment.
In the following we employ the normalized mean square error (NMSE) which is the ratio between the mean square error and the variance of the time series. It is defined, for a time series $x(t)_{t=t_1,\ldots,t_l}$, by

$$\frac{\sum_{t=t_1}^{t_l}\left(x(t)-\hat{x}(t)\right)^2}{\sum_{t=t_1}^{t_l}\left(x(t)-\overline{x}(t)\right)^2}=\frac{\sum_{t=t_1}^{t_l}\left(x(t)-\hat{x}(t)\right)^2}{l\sigma^2} \tag{24}$$

where $\hat{x}(t)$ is the prediction given by the RNN and $\overline{x}(t)$, σ^2 are the mean value and variance estimated from the available data. A value of NMSE=1 is achieved by predicting the unconditional mean of a time series. The normalized root mean squared error (NRMSE) used for some of the results in the literature is the square root of the NMSE.
We compared the results obtained using our algorithms to other results in the literature.

6.1 Sunspots

The sunspots dataset (Fig. 5) is a natural dataset that contains the yearly number of dark spots on the sun from 1700 to 1979. The time series has a pseudo-period of 10 to 11 years. It is common practice to use as the training set the data from 1700 to 1920 and to evaluate the performance of the model on two sets, composed respectively of the data from 1921 to1955 (test1) and of the date from 1956 to 1979 (test2). Test2 is considered to be more difficult.

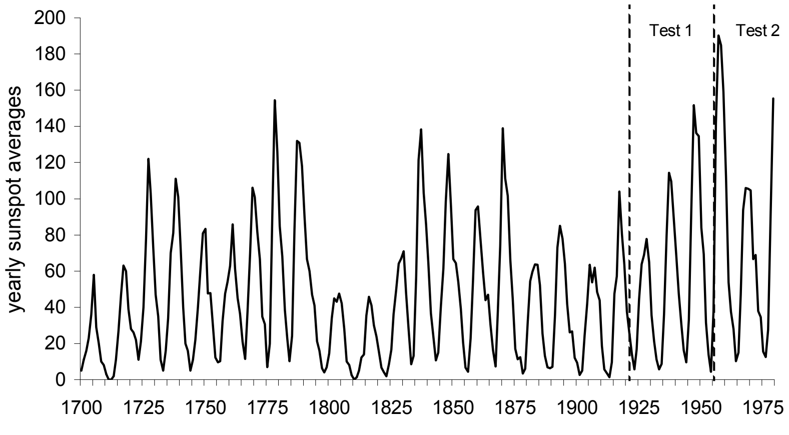

Fig. 5. The sunspots time series

For both CBPTT and EBPTT we set to 20 the upper bound for the delays of the new connections, to 4 the maximal number of new connections and to 20 the number of BPTT iterations performed for each candidate connection during the exploratory stage of EBPTT. Tables 2 and 3 show the NMSE obtained by various models on the two test sets of this benchmark, and the total number of parameters.

Model	Parameters	Test1	Test2
Carbon copy	0	0.427	0.966
TAR	18	**0.097**	**0.280**
MLP Weigend	43	**0.086**	**0.350**
MLP Kouam	43	**0.082**	**0.320**
RNR/BPTT	181	0.102	0.371
RNR/EBPTT 20 it.	15	0.096	0.320
RNR/EBPTT 100 it.	15	0.092	0.308
RNR/CBPTT	15	**0.094**	**0.281**
RNR/Boosting (quad., 20)		0.090	0.296
RNR/Boosting (lin., 10)		**0.082**	**0.314**

Table 2. Mean NMSE obtained by various models on the sunspots time series

Model	Parameters	Test1	Test2
MLP Czernichow	30	**0.078**	**0.283**
TAR	18	0.097	0.280
DRNN1	30	0.091	0.273
DRNN2	45	0.093	0.246
RNN/BPTT	155	0.084	0.300
RNN/EBPTT	23	**0.078**	**0.227**
RNN/CBPTT	15	0.092	0.251
RNN/Delay learning	34	0.081	0.261
RNN/Boosting (quad., 5)		**0.078**	**0.250**
RNN/Boosting (lin., 10)		0.080	0.270

Table 3. Best NMSE obtained by various models

The threshold autoregressive (TAR) model in (Tong & Lim, 1980) employs a threshold to switch between two AR models. The MLP in (Weigend et al., 1991) has a time window of size 12 in the input layer; Table 2 gives the results obtained with weight decay and pruning, which start with 8 hidden neurons and reduce their number to 3. The Dynamical RNNs (DRNNs) are RNNs having FIR connections. We show here the best results obtained in (Aussem, 1999) on each of the two test sets; mean values were not available. DRNN1 has 2 hidden neurons, fully connected by FIR connections of order 5. DRNN2 has 5 hidden neurons, fully connected by FIR connections of order 2. The author found the order of these connections after several trials.

The best result is obtained by EBPTT with 100 iterations, for an RNR with 3 hidden neurons. Constructive algorithms added most of the time 4 connections. For the delay learning algorithm, the experiments show an occasionally unstable behaviour, some learning attempts being soon blocked with high values of error. The internal state of the network (the set of neuron outputs belonging to the hidden layer) happens to be very sensitive to delay variation. The choice of the two learning steps, either for the weights or for connection delays, requires a very precise tuning. The boosting algorithm develops 9 networks with linear and quadratic functions and 36 networks with exponential function.

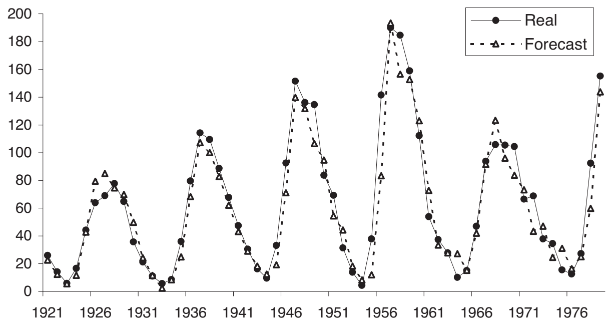

Fig. 6. The predictions obtained with EBPTT on the sunspots test sets

6.2 Mackey-Glass series

The Mackey-Glass benchmarks (Mackey and Glass, 1977) are well-known for the evaluation of SS and MS prediction methods. The time series are generated by the following nonlinear differential equation:

$$\frac{dx}{dt} = -0.1 \cdot x(t) + \frac{0.2 \cdot x(t-\theta)}{1 + x^{10}(t-\theta)} \tag{25}$$

The behavior is chaotic for τ > 16,8. The results in the literature usually concern τ = 17 (known as MG17, see Fig. 7) and τ = 30 (MG30). The data is generated and then sampled with a period of 6, according to the common practice, see e.g. (Wan 1993). We use the first 500 values as our learning set and the next 100 values as our test set.

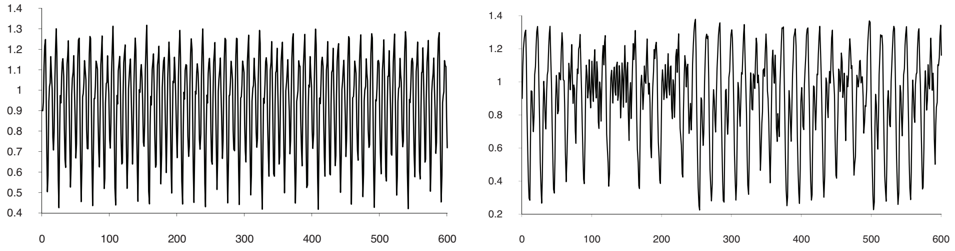

Fig. 7. The Mackey-Glass time series for θ = 17 (left) and θ = 30 (right)

The linear, polynomial, local approaches, RBF and MLP models are mentioned in (Casdagli, 1989). The FIR MLP put forward in (Wan, 1993) has 15 neurons in the hidden layer. FIR connections of order 8 are employed between the inputs and the hidden neurons, while the order of the connections between the hidden neurons and the output is 2. The resulting networks have 196 parameters. The feed-forward network employed in (Duro & Santos Reyes, 1999) consists of a single input neuron, 20 hidden neurons and one output neuron. A delay is associated to every connection in the network, and the value of the delay is modified by a learning algorithm inspired by back-propagation. In (McDonnell & Waagen, 1994) an evolutionary algorithm produces an RNN having 2 hidden neurons with sinusoidal transfer functions and several time-delayed connections.

Model	MG(17)	MG(30)
Linear	269	324
Polynomial	11.2	39.8
Local approach 1	33.1	57.5
Local approach 2	12.9	380.2
RBF	10.7	25.1
FFN	10	31.6
RNN/BPTT	0.99	13.1
RNN/EBPTT	0.62	1.8
RNN/CBPTT	1.66	2.51
Boosting (quad., 100)	**0.16**	**0.45**
Boosting (quad., 200)	**0.18**	**0.45**

Table 4. Mean EQMN (*10^3) obtained by various models on the MG time series

The DRNNs have FIR connections of order 4 between the input and the hidden layer, FIR connections of order 2 between the 4 to 7 hidden neurons, and simple connections to the output neuron (for a total of 197 parameters).
Throughout our experiments, for both EBPTT and CBPTT we set to 34 the maximal value for the delays of the new connections and to 10 the maximal number of new connections. The number of BPTT steps performed for each candidate connection during the exploratory stage of EBPTT was always set to 20; a higher value has a negligible effect here. The mean results reported in Table 4 were obtained with RNNs having 6 hidden neurons, so with 10 time-delayed connections we have a maximum of 65 parameters. The best results were obtained with RNNs having up to 7 hidden neurons, for a maximum of 81 parameters.
Our constructive algorithms significantly improve the results reported in the literature for the two datasets, with regard both to the mean NMSE and to the lowest NMSE. There is also an improvement upon BPTT without the constructive stage.
During our experiments we noticed that the mean value of the delays associated with the new connections was significantly lower for MG(17) than for MG(30). Also, CBPTT added on the average fewer connections than EBPTT. Again, only the first new connections produce a significant reduction in the NMSE.

Model	MG(17)	MG(30)
FIR MLP	4.9	16.2
TDFFN	0.8	
RNN evolutionary algorithm		2.5
DRNN	4.7	7.6
RNRN/BPTT	0.23	0.89
RNN/EBPTT	**0.13**	**0.05**
RNN/CBPTT	0.14	0.73
RNN/delay learning	0.15	
Boosting (lin.,150)	**0.13**	**0.45**
Boosting (quad., 100)	**0.15**	**0.41**

Table 5. Best EQMN (*10^3) obtained by various models on the MG time series

7. Multi step ahead prediction results

While reliable multi-step-ahead (MS) prediction has important applications ranging from system identification to ecological modeling, most of the published literature considers single-step-ahead (SS) time series prediction. The main reason for this is the inherent difficulty of the problems requiring MS prediction and the fact that the results obtained by simple extensions of algorithms developed for SS prediction are often disappointing. Moreover, if many different techniques perform rather similarly on SS prediction problems, significant differences show up when extensions of these techniques are employed on MS problems.

There are several methods for dealing with a MS prediction problem after finding a satisfactory solution to the associated SS problem.

The first and most common method consists in building a predictor for the SS problem and using it recursively for the corresponding MS problem. The estimates provided by the model for the next time step are fed back to the input of the model until the desired prediction horizon is reached. This method is usually called iterated prediction. This simple method is plagued by the accumulation of errors on the difficult data points encountered; the model can quickly diverge from the desired behavior.

A better method consists in training the predictor on the SS problem and, at the same time, in making use of the propagation of penalties across time steps in order to punish the predictor for accumulating errors in MS prediction. This method is called corrected iterated prediction. When the models are MLPs or RNNs, such a procedure is directly inspired from the BPTT algorithm performing gradient descent on the cumulated error. The model is thus simultaneously trained on both the SS and the associated MS prediction problem. Unfortunately, the gradient of the error usually "vanishes" when moving away from the time step during which the penalty was received (Bengio, 1994).

According to the direct method, the predictor is no longer concerned with an SS problem and is directly trained on the MS problem. By a formal analysis of the expected error, it is shown in (Atiya et al., 1999) that the direct method always performs better than the iterated method and at least as well as the corrected iterated method. However, this result relies on several assumptions, among which the ability of the model to perfectly learn the different target functions (the one for SS prediction and the one for direct MS prediction). The results of the learning algorithm may been improved, e.g. when it suffers from the vanishing gradient phenomenon. For instance, improved results were obtained by using recurrent networks and training them with progressively increasing prediction horizons (Suykens & Vandewalle, 1995) or including time-delayed connections from the output of the network to its input (Parlos et al., 2000).

We decided to test on MS prediction problems the previous algorithms that were originally developed for learning long-term dependencies in time series (Boné & al, 2000) or for improving general performance. Constructive algorithms provide a selective addition of time-delayed connections to recurrent networks and were shown to produce parsimonious models (few parameters, linear prior on the longer-range dependencies) with good results on SS prediction problems. These results, together with the fact that a longer-range memory embodied in the time delays should allow a network to better retain the past information when predicting at a long horizon, let us anticipate improved results on MS prediction problems. Some further support for this claim is provided by the experimental evidence in (Parlos et al., 2000) concerning the successful use of time delays in recurrent networks for

MS prediction. We expected the constructive algorithms to identify the most useful delays for a given problem and network architecture, instead of using an entire range of delays.

7.1 Sunspots

All the tested algorithms perform better than standard BPTT and exhibit a fast degradation while simultaneously increasing prediction horizon (Table 6, Fig. 8).

Steps ahead h	Model						
	BPTT	CBPTT	EBPTT	lin. 10	quad. 20	quad. 5	exp. 20
1	0.24	**0.17**	0.19	0.18	**0.17**	0.18	0.18
2	0.88	0.69	0.53	0.43	**0.40**	0.43	0.42
3	1.14	0.99	0.79	**0.54**	**0.54**	0.56	0.67
4	1.22	1.17	0.80	0.67	0.73	**0.64**	0.76
5	1.01	0.99	0.88	0.74	**0.69**	0.73	0.77
6	1.02	1.01	0.84	0.73	0.68	**0.65**	0.74
10	-	-	-	**0.64**	0.69	0.67	0.75
12	-	-	-	0.86	0.97	**0.77**	1.09

Table 6. Best mean NMSE on the sunspots cumulated set (test1+test2) as a function of the prediction horizon

Boosted architectures give the best results. The boosting algorithm develops around 9 weak learners with the linear and quadratic loss functions, and 30 weak learners with the exponential function, as for the SS problem. The mean number of networks remains practically constant while the horizon increases.

If we distinguish between the results on test1 and test2 (not shown here) we can see that the deterioration is mainly due to test2. It is commonly accepted that the behavior on test2 can not be explained (by some longer-range phenomenon) given the available history. Short-range information available in SS prediction lets the network evaluate the rate of change in the number of sunspots. Such information is missing in MS prediction.

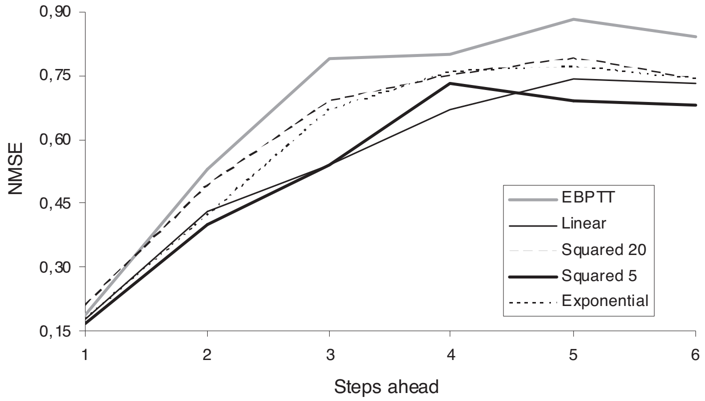

Fig. 8. Sunspots time series: mean NMSE on the cumulated test set as a function of the prediction horizon. Due to their poor results, BPTT and CBPTT algorithms are not represented

7.2 Mackey-Glass series

For the two MG series, we obtained (Tables 7 and 8) around 26 networks with the linear function, 37 with the squared function and between 46 and 50 for the exponential function. The maximal number of networks is set to 50. Tables 7 and 8 show the strong improvements obtained on the original BPTT and on the constructive algorithms.

Steps ahead h	Model					
	BPTT	CBPTT	EBPTT	Lin. 150	quad. 100	exp. 100
1	22	13	1	0.17	**0.16**	0.17
2	179	124	101	**0.24**	0.28	0.25
3	145	124	16	0.57	0.57	**0.52**
4	8	7	4	0.57	0.54	**0.52**
5	266	253	181	**0.98**	1.26	1.27
6	321	321	232	**2.11**	15.2	4.66
10	336	331	219	14.1	**12.2**	15.0
11	289	218	252	**9.80**	12.0	16.8
12	167	156	158	**6.72**	8.66	7.57

Table 7. Best mean results (NMSE*10^3) on MG17 as a function of the prediction horizon

Comparisons with other published results concerning MG17 MS prediction can only be performed for a horizon of 14; the results presented here are inferior to those of the local methods put forward in (Chudy & Farkas 1998; McNames 2000), but for the RNNs trained by our algorithm, significantly fewer data points were employed for training (500 compared to 3000 or 10000), which is the usual benchmark (Casdagli 1989; Wan, 1994). However, the use of a huge number of points for learning the MG17 artificial time series, generated without noise, can lead to models with poor generalization to noisy data.

Steps ahead h	Model					
	BPTT	CBPTT	EBPTT	lin. 300	quad. 200	exp. 150
1	11.7	2.5	1.8	**0.45**	**0.45**	0.47
2	19.9	9.7	3.3	0.49	**0.48**	0.59
3	4	2.2	1.6	0.56	**0.55**	0.64
4	2.2	2.1	1.6	0.47	**0.43**	0.48
5	2.6	2.3	0.9	0.85	**0.67**	0.72
6	8.9	8.3	6.4	**1.75**	1.92	1.80
7	70.1	65.6	64.3	2.98	4.56	**2.72**
8	336	203	112	**5.08**	109	57.0
9	801	379	257	84.0	276	**3.71**
10	892	383	73.7	2.79	204	**2.63**
11	411	230	285	**6.34**	21.3	8.05

Table 8. Best mean results (NMSE*10^3) on MG30 as a function of the prediction horizon

8. Conclusion

Adding time-delayed connections to recurrent neural networks helps gradient descent algorithms in learning medium or long-term dependencies. However, by systematically adding finite impulse response connections, one obtains oversized networks which are slow to train and need regularization techniques in order to improve generalization. We apply here two constructive approaches, which starts with a RNN having no time-delayed connections and progressively adds some, an approach based on a particular type of neuron whose connections have a real value and adapted to recurrent networks and a boosting algorithm. The experimental results we obtained on three benchmark problems show that by adding only a few time-delayed connections we are able to produce networks having comparatively few parameters and good performance for SS problems.
The results show also that boosting recurrent neural networks improve strongly MS forecasting. The boosting effect proved to be less effective for sunspots MS forecasts because some short-term dependencies are essential for the prediction of some parts of the data. The fact that for the Mackey-Glass datasets the results are better on the most difficult of the two sets (MG30) can be explained by noticing that long-range dependencies play a more important role for MG30 than for MG17.

9. References

Assaad, M.; Boné, R. & Cardot, H. (2005). Study of the Behavior of a New Boosting Algorithm for Recurrent Neural Networks, *Proceeding of Int. Conference on Artificial Neural Networks*, LNCS, Vol. 3697, 169-174, Warsaw, Poland, Springer-Verlag

Assaad, M.; Boné, R. & Cardot, H. (2008). A New Boosting Algorithm for Improved Time-Series Forecasting with Recurrent Neural Networks, *Information Fusion*, Vol. 9, No. 1, January 2008, 41-55

Atiya, A. F.; El-Shoura, S. M.; Shaheen, S. I. & El-Sherif, M. S. (1999). A Comparison Between Neural Network Forecasting Techniques - Case Study: River Flow Forecasting. *IEEE Transactions on Neural Networks*, Vol. 10, No. 2, 402-409.

Aussem, A. (1999). Dynamical Recurrent Neural Networks: Towards Prediction and Modelling of Dynamical Systems, *Neurocomputing*, Vol. 28, 207-232

Aussem, A. (2002). Sufficient Conditions for Error Backflow Convergence in Dynamical Recurrent Neural Networks, *Neural Computation*, Vol. 14, No. 8, 1907-1927.

Bengio, Y.; Simard, P. & Frasconi, P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult, *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, 157-166

Boné, R.; Crucianu, M. & Asselin de Beauville, J.-P. (2000a). An Algorithm for the Addition of Time-Delayed Connections to Recurrent Neural Networks, *Proceedings of European Symposium on Artificial Neural Networks*, pp. 293-298, Bruges, Belgium

Boné, R.; Crucianu, M.; Verley, G. & Asselin de Beauville, J.-P. (2000b). A Bounded Exploration Approach to Constructive Algorithms for Recurrent Neural Networks, *Proceedings of International Joint Conference on Neural Networks*, Vol. 3, pp. 27-32, Como, Italy

Boné, R.; Crucianu, M. & Asselin de Beauville, J.-P. (2002). Learning Long Term Dependencies by the Selective Addition of Time-Delayed Connections to Recurrent Neural Networks, *NeuroComputing*, Vol 48, 251-266

Bühlmann, P. & Yu, B. (2003). Boosting with L2-loss: Regression and Classification, *Journal of the American Statistical Association*, Vol. 98 , 324-340

Campolucci, P.; Uncini, A.; Piazza, F. & Rao, B. D. (1999). On-line learning algorithms for locally recurrent neural networks. *IEEE Transaction on Neural Networks*, Vol. 10, No. 2, 253-271

Casdagli, M. (1989). Nonlinear Prediction of Chaotic Time Series, *Physica*, Vol. 35D , 335-356.

Chudy, L. & Farkas, I. (1998). Prediction of Chaotic Time-Series Using Dynamic Cell Structures and Local Linear Models. *Neural Network World*, Vol. 8, 481-489

Cook, G.D. & Robinson, A.J. (1996). Boosting the Performance of Connectionist Large Vocabulary Speech Recognition, *Proceedings of International Conference in Spoken Language Processing*, pp. 1305-1308, Philadelphia, PA

Czernichow, T.; Piras, A.; Imhof, K.; Caire, P.; Jaccard, Y.; Dorizzi, B. & Germont, A. (1996) Short Term Electrical Load Forecasting with Artificial Neural Networks, *Engineering Intelligent Systems*, Vol. 4, No. 2, 85-99

Drucker, H. (1999). Boosting Using Neural Nets, In *Combining Artificial Neural Nets: Ensemble and Modular Learning*, A. Sharkey, (Ed.), 51-77, Springer

Duffy, N. & Helmbold, D. (2002). Boosting Methods for Regression, *Machine Learning*, Vol. 47, 153-200

Duro, R.J. & Santos Reyes, J. (1999). Discrete-Time Backpropagation for Training Synaptic Delay-Based Artificial Neural Networks. *IEEE Transactions on Neural Networks*, Vol. 10, No. 4, 779-789.

El Hihi, S. & Bengio, Y. (1996). Hierarchical Recurrent Neural Networks for Long-Term Dependencies, *Advances in Neural Information Processing Systems VIII*, M. Mozer, D. S. Touretzky and M. Perrone, (Eds), 493-499, MIT Press, Cambridge, MA

Frasconi, P. ; Gori, M. & Soda, G. (1992). Local Feedback Mulitlayered Networks. *Neural Computation*, Vol. 4, No. 1, 120-130

Freund, Y. (1990,). Boosting a Weak Learning Algorithm by Majority, *Proceedings of Workshop on Computational Learning Theory*, pp. 202-216

Freund, Y. 1 Schapire, R.E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, Vol. 55, 119-139

Guignot, J. & Gallinari, P. (1994). Recurrent Neural Networks with Delays, *Proceedings of International Conference on Artificial Neural Networks*, pp. 389-392, Sorrento

Hammer, B. & Tino, P. (2003). Recurrent Neural Networks with Small Weights Implement Definite Memory Machines, *Neural Computation*, Vol. 15, No. 8, 1897-1926

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, Vol. 9, No. 8, 1735-1780

Lin, T.; Horne, B.G.; Tino, P. & Giles, C.L. (1996). Learning Long-Term Dependencies in NARX Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, Vol.7, No. 6, 1329-1338

McNames, J. (2000). Local Modeling Optimization for Time Series Prediction. *Proceedings of 8th European Symposium on Artificial Neural Networks*, pp. 305-310, Bruges, Belgium

Mackey, M. & Glass, L. (1977). Oscillations and Chaos in Physiological Control Systems, *Science*, 197-287

Parlos, A.G.; Rais, O.T. & Atiya, A.F (2000). Multi-Step-Ahead Prediction Using Dynamic Recurrent Neural Networks. *Neural Networks*, Vol. 13, 765-786

Pearlmutter, B.A. (1990). Dynamic Recurrent Neural Networks. *Research Report CMU-CS-90-196*. Carnegie Mellon University.

Schapire, R.E. (1990). The Strength of Weak Learnability, *Machine Learning*, Vol. 5, 197-227.

Seidl, D.R. & Lorenz, R.D. (1991). A Structure by which a Recurrent Neural Network Can Approximate a Nonlinear Dynamic System, Proceedings of *International Joint Conference on Neural Networks*, pp. 709-714.

Siegelmann, H.T.; Horne, B.G. & Giles C.L. (1997). Computational Capabilities of Recurrent NARX Neural Networks. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 27, No. 2, 209-214

Suykens, J.A.K. & Vandewalle, J. (1995). Learning a Simple Recurrent Neural State Space Model to Behave Like Chua's Double Scroll. *IEEE Transactions on Circuits and Systems-I*, Vol. 42, 499-502

Takens, F. (1981). Detecting Strange Attractors in Turbulence. *Dynamical Systems and Turbulence*. Dynamical Systems and Turbulence, Lecture Notes in Mathematics, Vol. 898, 366–381, Springer-Verlag, Berlin

Tong, H. & Lim, K.S. (1980). Threshold Autoregression, Limit Cycles and Cyclical Data, *Journal of the Royal Statistical Society*, Vol. B42 , 245-292

Tsoi, A.C. & Back, A.D. (1994). Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures. *IEEE Transactions on Neural Networks*, Vol. 5, No 2, 229-239

Wan, E. A. (1993). Finite Impulse Response Neural Networks with Applications in Time Series Prediction, PhD Thesis, Stanford University, 140p

Wan, E.A. (1994). Time Series Prediction by Using a Connection Network with Internal Delay Lines, In: *Time Series Prediction: Forecasting the Future and Understanding the Past*, A. S. Weigend and N. A. Gershenfeld, (Eds), 195-217, Addison-Wesley

Weigend, A.S.; Rumelhart, D.E. & Huberman, B.A. (1991). Generalization by Weight-Elimination applied to Currency Exchange Rate Prediction, Proceedings of *International Joint Conference on Neural Networks*, pp. 837-841, Seattle, USA