Time in Connectionist Models*

Jean-Cédric Chappelier[1], Marco Gori[2] and Alain Grumbach[3]

[1] DI–LIA, EPFL, Lausanne (Switzerland)
[2] Dipartimento di Ingegneria dell'Informazione, Universitá di Siena, Siena (Italy)
[3] Département Informatique, ENST, Paris (France)

1 Introduction

The prototypical use of "classical" connectionist models (including the multi-layer perceptron (MLP), the Hopfield network and the Kohonen self-organizing map) concerns *static* data processing. These classical models are not well suited to working with data varying over time. In response to this, temporal connectionist models have appeared and constitute a continuously growing research field. The purpose of this chapter is to present the main aspects of this research area and to review the key connectionist architectures that have been designed for solving temporal problems.

The following section presents the fundamentals of temporal processing with neural networks. Several temporal connectionist models are then detailed in section 3. As a matter of illustration, important applications are reviewed in the third section. The chapter concludes with the presentation of a promising future issue: the extension of temporal processing to even more complex structured data.

2 Fundamentals

Before actually getting into the fundamentals of temporal connectionist models, we have to clarify some possible confusion between different kinds of time representation.

The issue we are concerned with is the study of models which are able to take the "*natural*" time of a problem into account. Some ambiguity may arise from the fact that some models also have an *internal* use of time. This internal time does not however correspond to any temporal dimension of the problem dealt with; the problem considered by such a model could even be static (e.g. image recognition). Internal use of time is only necessary for the model's own dynamics (e.g. for relaxation of inner states to some equilibrium as in the Hopfield's network).

To make the difference clear, we shall speak of *external* time for the time of the problem which is considered. Some sort of mixed computation could of course happen where the use of internal time is also extended to the processing of

* Published in "*Sequence Learning: Paradigms, Algorithms and Applications*", R. Sun and L. Giles editors, Lecture Notes in Artificial Intelligence Serie, 1828, chapter 5, pp 105-134, Springer, 2001.

the input sequence. To the best of the authors' knowledge however, no such link between internal and external time has ever been investigated in the literature.

The chapter focuses only on neural network architectures that handle the temporal problem, i.e. that deal with external time.

Several aspects of time may be involved in temporal problems. These aspects correspond to different properties of time such as:

- time as a simple *order relation* (where time only consists of an index used to order events, e.g. the time embedded in the reading of a sentence);
- time as *metrics* (e.g. the time involved in speech, where duration is meaningful);
- *discrete* time versus *continuous* time;
- time over a *finite* versus *infinite* interval (never ending problems).

2.1 A Short Historical Overview

The chronological ordering of temporal connectionist models is characterized by the increase of the integration of time in the architecture.

A first phase in the development of temporal neural networks is characterized by architectures based on classical models, but *locally* modified so as to take the time dimension into account. For instance, recurrent networks based on the MLP were introduced. They include backward links with a one time step delay that represents the ordering relation between two successive inputs. This kind of approach typically focuses on *temporal sequence learning*.

For example, Jordan (1986) designed such an architecture in which the output vector at time t–1 is concatenated to the input vector at time t.[1] This architecture was used for modelling the sequence of movements of a robot arm.

Elman (1990) also designed a similar architecture, in which the hidden vector rather than the output vector is concatenated to the input vector. This architecture was successfully used for sentence processing.

These modifications of the architecture also affect the learning process: the backward links have to be taken into account. The learning algorithm used in this case, called "back-propagation through time", is a generalisation of the standard back-propagation learning algorithm (see for instance the paper of Williams and Peng (1990)).

A second phase in the development of temporal neural networks still considered classical architectures but focused on configuring their parameters more *globally* so as to be able to accommodate enough temporal informations for solving the problem. A first important parameter which can take into account the time dimension is the input vector: this could be a temporal window over the input signal for example. This choice triggers other choices such as the dimension of the hidden layers which needs to be proportional to the dimension of the input temporal window. A good prototype from this second phase is the TDNN (Lang

[1] see section 3.2 for further details on this architecture.

et al. 1990)[2]. This kind of architecture typically focuses on *temporal pattern recognition* of the kind needed in speech processing.

After the use of adapted classical architectures, a third phase appeared in the development: architectures specially designed for time processing appeared. Most of them are inspired by our knowledge of information processing in "natural" neural networks. These try to mimic one or several of their characteristics such as:

1. propagation delays on connections;
2. neurons sending discrete pulses rather than continuous activity (i.e. "spiking neurons");
3. neuron activity being also a function of time; for instance introducing some refractory period;
4. synchronization between neuron populations.

For instance, RST (Chappelier and Grumbach 1998) which has been applied to finding the moving part of an image, uses properties 2 and 3 above[3].

Considering architectures using spiking neurons, Maass (1997b) studied their computational power and showed that they have at least the same computational power as a classical MLP, but need fewer neurons.

Although several other architectures have also been designed with this kind of approach, we are still only at the beginning of this third phase.

2.2 Time Integration in Connectionist Models

Let us now detail the different approaches of time integration in connectionist models such as sketched out in the previous section. The hierarchy of models detailed hereafter is summarized in figure 1 (see also the paper of Chappelier and Grumbach (1994)).

External Representation of Time. The first approach in the integration of time into connectionist models consists in not introducing it *directly* in the architecture, rather leaving the time representation outside the neural network. The idea is to preprocess the data so that classical static connectionist models can proceed with the temporal task. Time is preprocessed through a time to space transformation; the network accessing then only spatial information, a dimension of which has semantics related to time[4].

Researchers who have taken this approach include Simpson and Deich (1988), Gorman and Sejnowski (1988), Bengio et al. (1989) and Goldberg and Pearlmutter (1989). One typical model of this category is the well known TDNN (Lang et al. 1990), which is described in section 3.1.

[2] This model is described in section 3.1.

[3] A description of this architecture is given in section 3.4.

[4] For instance, one effective way of constructing a spatial representation of temporally-occurring information (which is not only limited to neural computing) is to create the power spectrum of the incoming information and use it as a static input image.

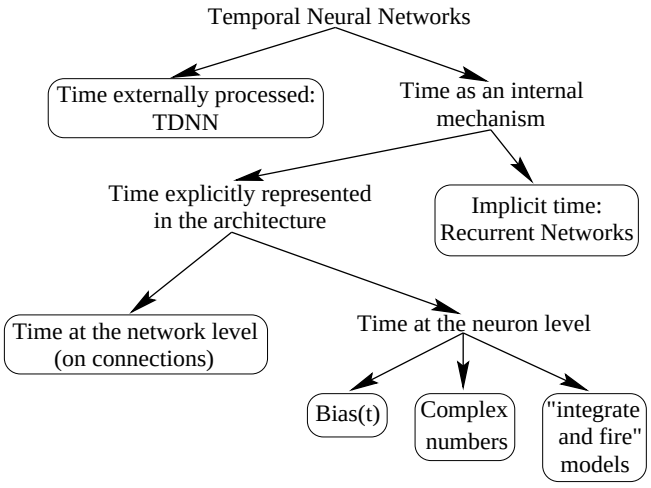


Fig. 1. A classification of connectionist models with respect to time integration.

Elman (1990) points out that there are several disadvantages with this kind of approach. First, it requires some buffering: how long should the buffer be and how often does the network look at it? This approach imposes therefore a rigid limit on the duration of patterns, which is not necessarily appropriate to the real temporal input. Secondly, no difference is made between relative and absolute temporal positions. Most of these architectures do not even have any representation of absolute temporal position at all. Finally, these methods also suffer from inadequate/inflexible time windowing and from over-training caused by an excessive number of weights, even if some clever strategies are used to share weights over time (as in TDNN).

Time as an Internal Index. Time itself can be introduced into connectionist models at several levels. First of all, time can be used as an index in a sequence of network states. There is no actual representation of time in the network strictly speaking but rather a use of time as an internal variable controlling the inner mechanism. We may say in this case that time is *implicitly* present in the model. This kind of network is typically illustrated by *recurrent networks* (RNNs, described further in section 3.2). As explained later, these models are nevertheless very powerful for temporal processing of sequences.

Time at the Connection Level. A step further in the introduction of time in a neural model is to represent it explicitly at the level of the network either on connections or at the neuron level (or both).

In the case where time is represented at the level of connections, it is usually done by some delays of propagation on the connections ("temporal weights"), or more generally by some convolution of the neuron input by a given tem-

poral kernel. The works of Béroule (1987), Jacquemin (1994) and Amit (1988) are, among others, three different but representative approaches of connections carrying temporal information.

Problems that are tackled with such architectures typically involve temporal matching between events.

Time at the Neuron Level. At the level of the neuron itself, there are several ways of introducing time, depending on how temporal information will be represented by the neural network activity. The two main approaches are:

- the information is contained in the time sequence of the neuron activities;
- the neuron activity consists of discrete events ("pulses" or "spikes"), the information being conveyed by the timings of these events.

These two points of view are summarized in figure 2.

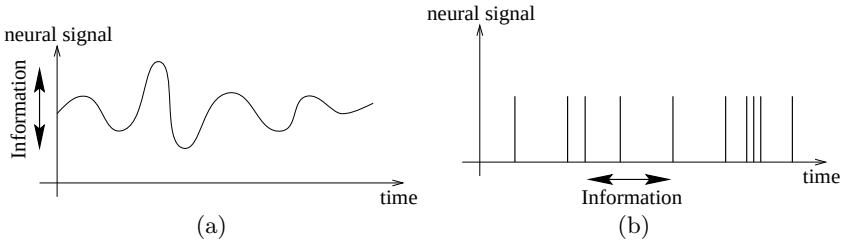


Fig. 2. An illustration of the two different points of view on temporal coding at the neuron level: a) variations (in time) of the amplitude of the neuron signal; b) different timings of neural events.

Static neural networks can clearly be emulated by the first type of temporal neural networks as they constitute a very special case (no time). It is furthermore interesting to notice that "static" neural networks can also be emulated with infinite precision in the second framework (Maass 1997a; Maass 1997b), the activity of a given static neural network being transcoded into a set of timings of a spiking neural network. Networks of spiking neurons therefore have at least the same computational power as classical neural networks.

The introduction of time at the neuron level can be done either by simulating biological properties or by building up neuron models from an engineering point of view, introducing time without specific biological inspiration.

The first approach usually leads to neuron models based on differential equations (Rinzel and Ermentrout 1989; Abbott and Kepler 1990). The model most often used in this context is the so-called "integrate and fire" model, or the "leaky-integrator". The principle underlying these models consists in summing the inputs of the neuron over a period of time. When this sum becomes greater than a given threshold (specific to each neuron), the neuron state changes. For a survey of this kind of model, we refer to the paper of Gerstner (1995).

On the other hand, the engineering approach is often purely algebraic:

- by changing the usual representation (scalar) to complex numbers (Vaucher 1996),
- or by introducing an artificial time varying bias (Horn and Usher 1991),
- or by considering that standard equation of neural networks $y_j = f\left(\sum w_{ij}\, x_i\right)$ concerns equilibrium and could be generalized to

$$\tau \frac{\mathrm{d}y_j}{\mathrm{d}t} = -y_j + f\left(\sum_i w_{ij}\, x_i\right)$$

  which is finally equivalent to an "integrate and fire" model.

The introduction of time at the neuron level often leads to dynamical properties and complex behaviours implying oscillations (Horn and Usher 1991) and synchronizations at the network level (Hirch 1991; Ramacher 1993; Lumer and Huberman 1992).

2.3 Temporal Components of Connectionist Models

Having detailed the different approaches to the time integration in connectionist models, we are now able to detail the different temporal components used in these models.

Any connectionist architecture that processes temporal patterns contains two (at least conceptually) distinct components: a short-term **memory** and a **predictor**[5].

The short term memory has to retain those aspects of the input sequence that are relevant for the problem. The predictor, on the other hand, uses the content of the short term memory to predict/classify and produce some output.

These two modules can either be embedded in each other or be as explicitly distinct as, for instance, in the model of Catfolis (1994), which consists of a RNN followed by a MLP. When explicit, the predictor will most generally be a classical static connectionist architecture.

Concerning the short term memory, several types are considered which can be classified along three axes: memory *form*, memory *content* and memory *plasticity*. These concepts were introduced rather informally by Mozer (1994). In order to formalize them a little further, we define a memory as some function f of time and previous inputs: $f(t, x(t-1), ..., x(t-k))$.

Memory form

The memory form deals with the function f itself. It can be as simple as a buffer containing the k most recent inputs[6]. This kind of memory is used within the spatial approach to time representation.

[5] Several other authors make this distinction including Mozer (1994) and de Vries and Principe (1992).

[6] In this case the function f returns a vector of size k consisting of $x(t-1), ..., x(t-k)$.

But the memory form does not have to necessarily consist of the raw input sequence itself. It could include some transformation of the representation of the input. Such types of memory form include decaying neuronal activity or delayed connections performing some convolution of the neuron input signal[7].

Memory content

The memory content is related to the number of arguments f actually takes into account. This can be related to the Markovian/non-Markovian aspect of the problem. "Markovian" means that the output at any time step can be determined uniquely from the input and target values for a given number of time steps in the recent past[8]. It is the purpose of the memory to keep these past values, either directly or combined with previous memory/output states.

Notice that Markovian problems of order higher than 1 can always be transformed into a Markovian problem of order 1. In the context of neural networks, this transformation leads to some "delay lines" where the input at a given time step is augmented with copies of k past inputs. However, if k is large, the network is likely to be overwhelmed by large amounts of redundant and irrelevant information. A standard technique for reducing the number of weights is so-called "weight sharing" or "weight tying", constraining a given set a weights to have the same (unspecified) numerical value[9]. Sharing of weights is linked to some known symmetries of the problems. However it should be emphazised that using RNNs rather than delay lines keeps down the number of parameters to be learned.

Non-Markovian problems, in which outputs are dependent on inputs that are an unbounded distance in the past, require the memory content to depend on some internal states[10].

Memory plasticity

Memory plasticity focuses on how the memory evolves through time[11], which can formally be defined as $\frac{\partial f}{\partial t}$.

The memory can either be *static*, when all its parameters are fixed in advance, or *adaptive*. Adaptive memory consisting in learning either delays (Bodenhausen and Waibel 1991; Unnikrishnan et al. 1991), decay rates (Mozer 1989; Frasconi et al. 1992) or some other parameters characterizing the memory (de Vries and Principe 1992).

Static memories are mentioned here since they still can be of some interest when there is adequate domain knowledge to constrain the type of information that should be represented in the memory. For instance, the memory may have a high resolution for recent events and decreasing resolution for more distant ones as illustrated by the work of Tank and Hopfield (1987).

[7] For instance $f = K(t) \otimes x(t)$ for some function K being the "kernel" of the memory.

[8] i.e. depends only on a *temporal neighbourhood* of the input.

[9] For a illustration of that point, see TDNN explained in the next section.

[10] which are not explicited in the chosen presentation of the memory form f but rather "hidden" in the t (first argument) dependency of f.

[11] i.e. "adapting" or "learning".

2.4 The Computational Power of Temporal Connectionist Models

On the basis of the classification made in the previous sections, this section addresses the question of the kind of temporal problems connectionist models can actually handle.

Standard (Static) Networks. It is widely known that classical neural networks[12], even with one single hidden layer, are universal function approximators (Hornik et al. 1989; Hornik 1991). This means that any continuous function with compact domain and compact range can be approximated with an arbitrarily degree of precision with regard to the norm of uniform convergence by a network of this type (provided that it has enough hidden neurons). This universality theorem provides a theoretical framework for the application of MLPs to various problem domains and explains their success.

Such feedforward networks, inherently static, can nevertheless have some applications in temporal domain, as already explained in the last two sections. The most simple temporal problems could be handled using feedforward networks. Indeed, MLPs are enough to learn such mapping where the input data at each time step contains enough information to determine the output at that time, (i.e. where $o(t) = F(i(t))$, with i is the input and o the output of the network).

It can furthermore be proved that such static models implement some nonlinear generalization of usual statistical AR-predictors; i.e. $o(t) = F(i(t), ..., i(t-k))$ (Lapedes and Farber 1987).

Recurrent Networks. Similarly to the universality of static neural-networks in function approximation, recurrent neural networks (RNNs) are universal approximators of dynamical systems (Funahashi and Nakamura 1993). It is also easy to show that RNNs can simulate any arbitrary finite state machine (FSA) (Cleeremans et al. 1989). Siegelmann and Sontag (1995) have even shown that a RNN can simulate any Turing Machine.

In the case where the previous output is needed as well as the current input to determine the output, i.e. if $o(t) = F(i(t), o(t-1))$, simple RNNs with feedback weighted connections from the ordinary target nodes (such as Jordan networks) or even feedforward networks with 'teacher forcing' techniques could be employed (Rohwer 1994).

In the most general case however, the approximation theorem mentioned before constitutes only a theoretical result as they do not say anything on how a given machine should be approximated. These theorems do unfortunately not imply that RNNs could easily be trained from examples to do complex temporal tasks.

This is the reason why relatively few cases have been reported showing the successful learning of *complex* dynamics by fully connected RNNs. One of the reasons lies in the cost of the computing of the error gradient for large scale

[12] more precisely MLP.

RNNs. High-order Markovian problems present severe difficulties to temporal neural networks. Although some preliminary solutions have been proposed (Bengio et al. 1993; Hochreiter and Schmidhuber 1997b), the computation of long time dependencies still remains a major problem for RNNs. For a good overview and critique on this point, we refer to the contribution of Hochreiter and Schmidhuber (1997a).

Spiking Neural Networks. It can be demonstrated (Maass 1994; Maass 1996) that a "spiking neural network"[13] is at least as powerful as a Turing machine. This means that such an architecture is, with a finite number of neurons and with a boolean input, able to simulate in real-time any Turing machine with a finite number of tapes.

Furthermore, spiking neural networks are strictly more powerful than Turing machine. Indeed they can also simulate some machine that a Turing machine can not, since spiking neural networks are able to handle computation with real numbers (differences of temporal events).

It is then easy to claim that spiking neural networks are unlimited with respect to temporal computing. However, the problem remains that, as with RNNs, such a powerful theorem does unfortunately not tell us *how* a given function can be learned by such networks. Still, the implementation of several fundamental functions (such as multiplication, addition, comparison) has been detailed by Maass (1996).

Summary. As far as estimation is concerned, a parallel could be made between usual statistical estimators, that are inherently linear, and non-linear estimators resulting from temporal neural networks. The non-linear equivalent of autoregressive (AR) estimator is the "standard" MLP, whereas non-linear equivalent of autoregressive with moving average (ARMA) estimators are represented by RNNs (Connor et al. 1992; Connor and Martin 1994). Temporal connectionist models therefore appear to be a richer and more powerful family of estimators than the usual linear ones.

A summary of all the temporal potentialities of connectionist models is given in table 1.

3 Most Relevant Temporal Connectionist Models

The aim of this section is to present in detail some temporal connectionist models in order to more concretely illustrate the theoretical issues presented up to here.

3.1 TDNN

The "Time-Delay Neural Network" (TDNN) model is a modification of the MLP architecture, the input of which consists of a "delay line", i.e. a whole set of

[13] i.e. a connectionist architecture built on spiking neurons and having some propagation delay on the connections;

connectionist model	able to emulate	non-linear generalization of estimator
standard static networks	continuous mapping	AR
RNNs	dynamical system/Turing Machine	ARMA
spiking neural networks	Turing machine	any?

Table 1. A summary of theoretical properties of temporal connectionist models.

"time-slices" as illustrated in figure 3. This set of time slices is shifted from left to right at each time step. The hidden layer is also divided into sub-slices (which are not shifted but computed from the previous layer). In order to keep the time consistency, the weights are set to be equal among time slices. In that sense, times slices "share" a unique weight set (one for each layer). For instance the weight of the connection from the first neuron of a time slice of the input layer to the first corresponding neuron of the hidden layer is always the same among all time slices of the input layer (see figure 3).

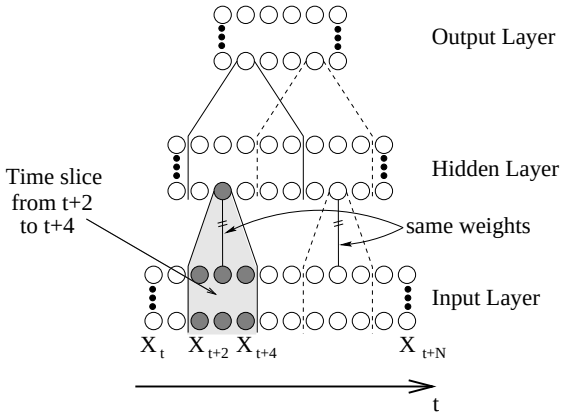


Fig. 3. TDNN architecture: a MLP architecture modified so that units take their inputs from only a part of the previous layer corresponding to a time slice. Different time slices are represented with different line styles. All time slices of a given layer share the same weights.

This model has been introduced by Lang et al. (1990) in a speech recognition context. The problem considered there was to recognize four kinds of phoneme. The input of the TDNN consisted in a sequence of spectrogram "slices".

3.2 Recurrent Networks

Recurrent neural networks (RNNs) constitute the major family of temporal connectionist models. A RNN can be defined in the most general way as a neural network containing at least one neuron the state of which depends either directly or indirectly on at least one of its anterior states. Formally, a RNN can be described by an equation like:

$$\begin{aligned} \mathbf{X}_t &= f\left(\mathbf{X}_{t-1}, \mathbf{U}_t, t, \Theta_t^f\right) \\ \mathbf{Y}_t &= g\left(\mathbf{X}_t, t, \Theta_t^g\right) \end{aligned} \tag{1}$$

where $\mathbf{U}$ stands for the input signal sequence, $\mathbf{Y}$ for the output sequence and Θs for the parameters of the network (typically the weights). $\mathbf{X}$ represents the set of recurrent variables.

There are a huge number of RNNs which have been proposed by various groups (Jordan 1986; Williams and Zipser 1989; Narendra and Parthasarathy 1990; Elman 1990; Back and Tsoi 1991; de Vries and Principe 1992). Some of these architectures do not bear much resemblance (at least superficially) to one another. There were therefore many attempts to find unifying themes in this variety of architectures (Narendra and Parthasarathy 1990; Nerrand et al. 1993; Tsoi and Back 1997; Tsoi 1998).

As a matter of illustration, we now detail three examples of RNNs.

Jordan and Elman Networks. The Jordan (1986) and Elman (1990) architectures are RNNs both based on the MLP architecture. They consist of adding recurrent links from one part of the network to the input layer, either from the output layer (Jordan (1986), figure 4a) or from the hidden layer (Elman (1990), figure 4b). At time $t-1$, the recurrent part is copied into the input layer as a complement to the actual input vector at time t (i.e. the signal to be processed). When computing a new output, the information goes downstream as in a classical MLP, from the input to the output layer. The complementary input vector, that is copied after each computation step, thus represents context information about the past computation.

Mathematically speaking, the Elman's network is described by the following simplification[14] of equation 1:

$$\begin{aligned} \mathbf{X}_t &= f\left(\mathbf{X}_{t-1}, \mathbf{U}_t, \Theta^f\right) \\ \mathbf{Y}_t &= g\left(\mathbf{X}_t, \Theta^g\right) \end{aligned} \tag{2}$$

where $\mathbf{U}$ is the actual input vector sequence, $\mathbf{X}$ is the vector sequence of neuron states from the hidden layer and $\mathbf{Y}$ is the vector sequence of states from the output layer (see figure 4b). Θ^f represents the set of weights from the input layer to the hidden layer, Θ^g the set of weights from the hidden layer to the output layer and the f and g functions represent a vectorial form of the usual

[14] $\mathbf{X}_t$ and $\mathbf{Y}_t$ do not depend directly on t, and Θs do not depend on t at all.

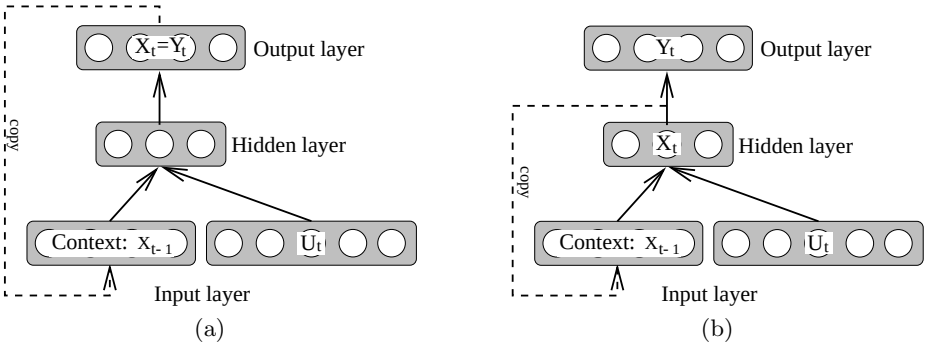


Fig. 4. Two typical recurrent neural networks: (a) Jordan's and (b) Elman's architectures.

sigmoidal function. Notice that the input layer of the network is made up of the concatenation of $\mathbf{U}_t$ and $\mathbf{X}_{t-1}$.

Similarly, Jordan's architecture can also be expressed in terms of another simplification of equation 1 as:

$$\begin{aligned} \mathbf{X}_t &= f\left(\mathbf{X}_{t-1}, \mathbf{U}_t, \Theta^f\right) \\ \mathbf{Y}_t &= \mathbf{X}_t \end{aligned} \tag{3}$$

where $\mathbf{X}$ is the vector sequence of states from the output layer (as well as $\mathbf{Y}$ for notation compatibility purposes). The function f needs however to be developed further. It results from the combination of the hidden and the output layer computations:

$$f\left(\mathbf{X}_{t-1}, \mathbf{U}_t, \Theta^f\right) = f_2\left(f_1\left(\mathbf{X}_{t-1}, \mathbf{U}_t, \Theta^{(1)}\right), \Theta^{(2)}\right)$$

with f_1 representing the computation of hidden layer states from input layer states, f_2 the computation of output layer states from hidden layer states and $\Theta^f = (\Theta^{(1)}, \Theta^{(2)})$.

Jordan and Elman networks are typically used for memorizing (and recalling) sequences such as poems (sequences of words), robot arm movements (sequences of positions), etc...

δ-NARMA. The δ-NARMA neural network model (Bonnet et al. 1997c; Bonnet et al. 1997b) was designed for signal prediction. Let us consider a temporal information sequence, for instance the daily number of railroad travellers from Paris to Lyon. Assume we know this number from three years ago up until yesterday. How can we forecast the number of travellers of today? Such problems can be tackled with usual statistical methods, such as ARMA models. But these methods have at least two important drawbacks: they cannot take into account non-stationary information and they are unable to deal with non-linear temporal relationships. These limitations are the major reasons why the neural

network approach has been investigated. The problem is now: how to design a connectionist architecture which is devoted to temporal forecasting?

D. Bonnet answered this question by the design of the δ-NARMA neural network. This network takes as inputs the values of the variable from date $t-p$ to date $t-1$ ($p \geq 2$). The output is the predicted value of the variable for date t. The architecture has two levels: the ε-NARMA neuron and the δ-NARMA network.

An ε-NARMA neuron is a recurrent neuron. But, instead of feeding the output value back into the input vector, it feeds the error from $t-q$ to $t-1$ (i.e. the difference between the predicted output value and the real output value) back into the input vector (see figure 5). In all other ways, it acts like a standard neuron.

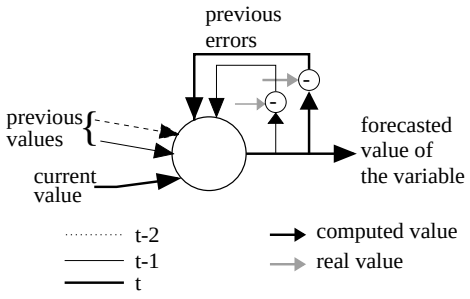


Fig. 5. An ε-NARMA neuron.

There are two main differences with the neuron model of Frasconi et al. (1992): **the recurring information**, which is the neuron output *error* in the ε-NARMA architecture whereas it is the neuron output itself in Frasconi's model, and *the number of values which are fed back*: only the last one in Frasconi's architecture and the last q values in an ε-NARMA neuron.

A δ-NARMA network consists of an MLP network with ε-NARMA neurons.

The learning algorithm is an adaptation of the classical stochastic back-propagation.

In such a forecasting problem, the time dimension is twofold: an order relation, and a phenomenon which is captured at periodical time points (day or month). These characteristics are grounded in the architecture through two features: the input vector which takes into account the previous values of the forecasted variable, and recurrent connections which mean that the forecasted value depends on the previous errors.

This architecture has been successfully applied to railroad traffic prediction where it gave better results than usual statistical methods (Bonnet et al. 1997a).

3.3 Temporal Extensions of Kohonen Maps

There have been several attempts at integrating temporal information into Self-Organizing Feature Maps[15].

As for other classical connectionist models, a first technique consists of adding temporal information externally, on the input of the map. For example, exponential averaging of inputs and delay lines were considered (Kangas 1990; Kohonen 1991).

Another common method is to use layered maps so that the second map tries to capture the spatial dynamics of the input moving on the first map (Kangas 1990; Morasso 1991).

A third approach that has been investigated, consists of integrating memory into the map, typically with some exponential decay of activities (e.g. by using leaky-integrator neurons) (Privitera and Morasso 1993; Chappell and Taylor 1993).

Another example of this kind of approach is given by the work of Euliano and Principe (1996). They add a spatio-temporal coupling to Kohonen maps so as to create temporally and spatially localized neighbourhoods. The spatio-temporal coupling is based on travelling waves of activity which attenuate over time. When these travelling waves reinforce one another, temporal activity wavefronts are created which are then used to enhance the possibility of a given neuron being active[16] in the next cycle.

Finally, more mathematically grounded approaches were developed by Kopecz (1995), Mozayyani et al. (1995) or Chappelier and Grumbach (1996).

Kopecz (1995) creates a Kohonen map with a lateral coupling structure which has symmetric and antisymmetric coupling (for temporal ordering). Once trained, the antisymmetric weights allow active regions of the map to trigger other regions in the map, thus reproducing the trained temporal pattern.

Mozayyani et al. (1995) use a coding with complex numbers where the time dimension is embedded into the phase of the complex representation.

Chappelier and Grumbach (1996) embed the map into a high dimensional space, classifying temporal inputs as functions of time (i.e. map inputs are no longer 2 or 3-D vectors but higher dimension vectors, each vector representing a function of time).

3.4 Networks of Spiking Neurons

Synfire Chains. Synfire chains were proposed by Abeles (1982) as a model of cortical function. A synfire chain consists of small layers of neurons connected together in a feedforward chain so that a wave of activity propagates from layer to layer in the chain. Interest in them has grown because they provide a possible explanation for otherwise mysterious measurements of precise neural activity (so-called "spike") timings. Many spatio-temporal patterns can be stored in a

[15] also called "Kohonen Maps".
[16] technically: "to win".

network where each neuron participates in several chains (chains are different but have non-empty intersections) although this introduces crosstalk noise which ultimately limits the network capacity. This capacity is however non zero allowing the effective use of such a model. It should furthermore be noted that since only a small fraction of neurons are active at a given time, many synfire chains can be simultaneously active, providing a possible mechanism for a higher level of organization.

RST. The approach of Chappelier and Grumbach (1998) consists of embedding spatial dimensions into the network and integrating time at both neuron and connection levels. The aim is to take both spatial relationships (e.g. as between neighbouring pixels in an image) and temporal relationships (e.g. as between consecutive images in a video sequence) into account at the architecture level.

Concerning the spatial aspect, the network is embedded into the actual space (2 or 3-D), the metrics of which directly influence its structure through a connection distribution function. A given number of neurons is randomly distributed in a portion of the space delimited by two planes, the input and output layers. Links are then created between the neurons according to their neighbourhood in the embedding space.

For the temporal aspect, they used a leaky-integrator neuron model with a refractory period and post-synaptic potentials[17]. The implemented model is mainly described by two variables: a membrane potential V and a threshold θ. V is the sum of a specific potential U and an external potential I which stands for the input of the neuron. Most of the time V is less than θ. Whenever V reaches θ, the neuron "fires". It sends a spike to the downstream neurons and changes its state as follows: θ is increased by some amount called adaptation or fatigue and the specific potential U is lowered down to a post-spike value. When the neuron does not fire, the variables U and θ decay exponentially to their resting values.

The input of a given neuron is the sum over space (all the input neurons of the considered neuron) and time (all the firing instants of its input neurons). In order to provide temporal robustness, the spikes sent by inputs are received as post-synaptic potentials described by a function of the kind $t \mapsto t \cdot \exp(1 - t/\tau)$.

The propagation of neuron spikes in the network as spatiotemporal synchronized waves enables RST to perform time and space correlation detection, e.g. motion detection in a video sequence (see figure 6). Spike synchronization plays the main role in RST for filtering static input patterns from moving ones.

4 Applications

A major reason of interest in connectionist models of intelligent processes is that they have been successfully applied to an impressive number of different

[17] for more details on "integrate and fire" neurons, we refer to the paper of Gerstner (1995) or the book of MacGregor and Lewis (1977).

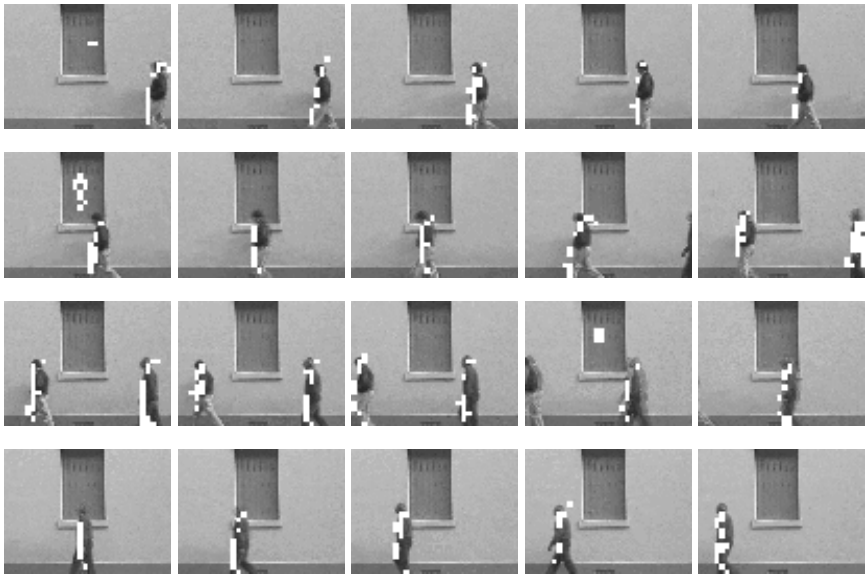


Fig. 6. Application of RST network to motion detection in video sequences. The response of the network (i.e. spiking neurons) is superimposed as white squares on the original input images.

application domains (e.g. see the book of Fogelman-Soulié and Gallinari (1998)). In many applications these models are required to deal with time and exhibit different dynamic behaviour.

4.1 Speech Processing

Most problems from automatic speech recognition are very difficult to address using traditional pattern recognition approaches designed for static data types. Speech has an inherent dynamic nature and, therefore, an effective model needs to be able to capture important temporal dependencies. The use of temporal connectionist models for speech processing is motivated by a number of different reasons:

- Speech recognition and speech understanding, due to the huge amount of variability in the signal, require high learning capabilities;
- large-scale speech processing projects (e.g. ARPA) have demonstrated the importance of high performance at the phonetic level;
- the statistical hypotheses of the best current models (hidden Markov models) are quite restrictive;
- learning and prior knowledge can be framed homogeneously in connectionist models;

– connectionist models are an intrinsically parallel computational scheme which turns out to be useful for very demanding applications.

Connectionist models have been proposed for both phoneme recognition or isolated word recognition with interesting results (e.g. see the book of Gori (1992)).

Phoneme Recognition. The first problem consists of coding a given speech utterance by means of the corresponding phonemes. The speech signal is typically pre-processed so as to produce a sequence of frames, each composed of a vector of discriminative features (e.g. spectral parameters). In practice, the frames are produced at a rate[18] which is related to the speed of the commands that the brain uses to control the articulatory system. A possible approach to predicting phonemes is to simply rely on a fixed speech window composed of a predefined number of frames. Unfortunately, the information required to predict different phonemes is spread over a significantly varying number of frames (Bourlard and Morgan 1994; Bourlard and Morgan 1998). RNNS are much better suited for dealing with such a problem. The basic problem of choosing a suitable speech window is in fact overcome by the inherent dynamical nature of the model. The input can simply be taken at frame level and the network is expected to capture the temporal dependencies which turn out to be useful for an effective phoneme classification. A possible recurrent architecture for this problem can consist of a simple one-layer network which takes a single speech frame as input and in which only self-loop connections are adopted. The speech signal is processed frame by frame along time and for each speech frame the neural network outputs a prediction of the corresponding phoneme. It has been pointed out that this architecture turns out to be suitable to incorporate the *forgetting behaviour* that a phoneme classifier is expected to exhibit (Bengio et al. 1992). Basically, the phoneme classification is supposed to depend on the speech frame being processed and on the close frames, but it is supposed not to depend on remote information. Very successful results for the problem of phoneme recognition on the DARPA-TIMIT speech data base have been found by Robinson (1994), where, in addition to the adoption of recurrent architectures, proper integration schemes with hidden Markov models are proposed.

Isolated Word Recognition. In principle, RNNs can also be used for isolated word recognition. Isolated words are in fact sequences of speech frames that can properly be labelled at the end by a target for specifying the sequence membership. Unfortunately, in spite of the experimental efforts of many research groups, this direct approach has not produced very successful results yet. There are at least two major reasons for this experimental lack of result. First, isolated words are sequences composed of hundreds of frames and it is now well-known that long-term dependencies are difficult to capture by a gradient-based learning algorithm (Bengio et al. 1994). Hence, RNN-based classifiers develop the trend to perform the prediction on the basis of the last part of the word, which is in

[18] in the order of magnitude of 10 ms.

fact a strong limitation especially when dealing with large dictionaries. Second, regardless of the architecture and weights, RNNs are difficult to use when the number of classes becomes very large. Basically, RNNs inherit the property of MLPs in exhibiting very strong discrimination capabilities but also the poor scaling with respect to the number of classes.

4.2 Language Processing

Theoretical foundations of natural language have been swinging, pendulum-like, between fully symbolic-based models, to approaches more or less based on statistics. Since the renewal of interest in neural networks, the connectionist approach has been immediately recognized as a neat way of dealing with the inherent uncertainty of natural languages. Languages, however, have also an inherently sequential nature and, therefore, only connectionist models that incorporate time[19] are good candidates for language processing. Typical tasks in language processing propose time as an external variable acting at different levels. For instance, lexical analyses require processing letters of a given alphabet in a sequential way, whereas in syntactical analyses, time is used to scan different words composing a sentence. In the last few years, RNNs appeared to be very well-suited for performing interesting language processing tasks.

Prediction of Linguistic Elements. Let us consider the problem of predicting linguistic elements. A preliminary investigation was carried out by Servan-Schreiber et al. (1991) concerning the prediction of terminal items for Reber's grammar (Reber 1976) (see figure 7).

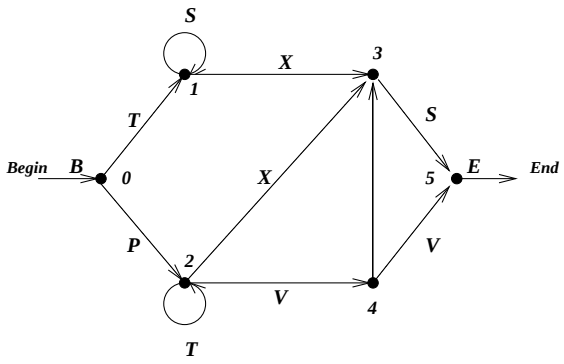


Fig. 7. An automaton representation of Reber's finite-state grammar.

Elman's recurrent network was used for the experiments. The network had an input aimed at coding the symbols and some context units representing the

[19] at least as an order relation.

state. The output layer had the same number of units as the input layer and one-hot coding[20] was adopted for the symbols of both layers. For each time step, a symbol from the alphabet of a Reber string was provided at the input and the recurrent network was asked to predict the next one. The training was carried out with a sample of Reber strings and the network subsequently exhibited very good generalization capabilities. Interestingly enough, the network developed automata-like internal representations and was even capable of performing correctly on arbitrarily long sequences.

Early experiments shown in the article of Elman (1990) were aimed at predicting the next word of a given part of a small sentence. The lexical items (inputs and outputs) were presented in a localist form using basis vectors. Hence, lexical items were orthogonal to one another and there was no encoding of the item's category membership.

Other interesting language processing tasks were presented by Elman (1991). His simple RNN was trained to learn the correctness of a given sentence on the basis of the presentation of positive and negative examples. For instance, Elman studied the problem of detecting the *agreement* of nouns with their verbs. Thus, for example, `John feeds dogs` and `Girls sees Mary` are grammatical and un-grammatical, respectively. No information concerning the grammatical role (subject/object, etc.) is provided to the network. The grammar of the language used in the experiment is given in Table 2

`S` → `NP VP` ‘`.`’
`NP` → `PropN` | `N` | `N RC`
`VP` → `V (NP)`
`RC` → who `NP VP` | who `VP (NP)`
`N` → boy | girl | cat | dog | boys | girls | cats | dogs
`PropN` → John | Mary
`V` → chase | feed | see | hear | walk | live | chases | feeds | sees | hears | walks | lives
Additional restrictions:
– number agreement between `N` and `V` within clause, and (where appropriate) between head `N` and subordinate `V`.
– verb arguments: chase, feed: require a direct object see, hear: optionally allow a direct object walk, live: preclude a direct object (observed also for head/verb relations in relative clauses)

Table 2. The grammar used by Elman (1990) for different language tasks, like noun-verb agreement, verb argument structure, and interactions with relative clauses.

[20] an exclusive coding in which one and only one output neuron is high.

The network is expected to learn that there are items which function as what we would call nouns, verbs, etc. and then must learn which items are examples of singular or plural, and which nouns are subjects and objects. Related successful experiments have been carried out concerning the verb argument structure, and the interactions with relative clauses.

Grammatical Inference. The theoretical result stating that a RNN can behave as an automaton is fully illustrated by some applications of RNNs to natural language processing. Unlike automata, however, the neural activations are continuous-valued variables and, therefore, an understanding of the network's internal representation developed during the training, is non trivial. RNNs are basically adaptive parsers the behaviour of which depends upon the parameters developed during the training. Formally, an *adaptive neural parser* can be regarded as a 4-tuple $\{U, X, \Phi, Z\}$, where $U \in R^M$ is the *alphabet* of *symbols*, $X \in R^N$ is the *state*, $\Phi(W) : R^N \times R^M \rightarrow R^N$ is the *state transition function* which depends on a vector of parameters $W \in R^P$, and $Z : R^N \rightarrow \{0, 1\}$ is the *decision function* that decides whether a given state is accepted or not. Given a set of labelled examples, one could try to relate the learning of the network and the developed internal representation with the grammar which generates the language. In the literature, the grammatical inference of the hidden rule is stated as the search for a parser capable of classifying the strings. The inference process adapts the neural parser to the given learning set by means of a search in the parameter space that defines the state transition function $\Phi(W)$.

In order to process symbolic strings by neural networks, each symbol of the input alphabet Σ has to be encoded. Basically, Σ is mapped to a set of vectors $\mathcal{U} = \{U_1, \ldots, U_S\}$ ($U_k \in \mathcal{R}^M$) and each string of Σ^* corresponds to a sequence of vectors that is used as input to the RNN[21]. The classification of each string is decided looking at the output of neuron N at the end of the input sequence.

The training set is composed of a set of L pairs (s, d), where $s \in \mathcal{U}^*$ and $d \in \{d^+, d^-\}$, being $d^+, d^- \in \mathcal{R}$. The learning algorithm adapts the neural network parameters by using a back-propagation through time (e.g. see the paper of Williams and Peng (1990)).

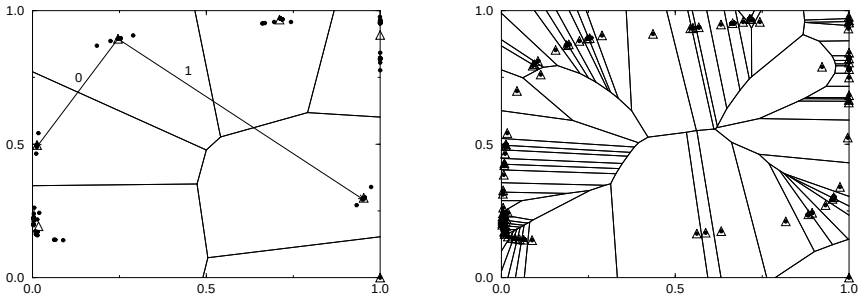
When processing symbolic strings by RNNs, the state vector X describes complex trajectories. As proposed by Kolen (1994), these trajectories can be studied in the framework of Iterated Function Systems (IFSs).

Basically, the symbolic interpretation emerges from partitioning the state space into a set of regions that are associated to the states of a finite machine. The volume of these regions defines the *resolution* of the extraction process.

The number of such regions provides interesting information concerning the rule extraction process. The more regions approximate the network trajectories, the more detailed description and, consequently, the more likely is the extracted machine to have a larger number of states (see figure 8). Equivalent states can of

[21] The notation Σ^* denotes the set of all the possible sequences created using the vectors contained in Σ.

course be removed by using a state minimization algorithm for finite state machines. There is experimental evidence that beyond a certain number of clusters, the extraction of the finite machine with the corresponding state minimization does not produce an increasing number of states but, instead, a maximum value is reached.



(a) (b)

Fig. 8. Finite State Automaton (FSA) extraction algorithm using a neural parser with a two-dimensional state space. (a) The number of regions is 7. The extracted FSA approximates exactly the network behaviour on all the strings with length up to 6. The transition rules from state 1 are shown. (b) The number of regions is 77. This number of clusters is necessary in order to have the same behaviour of the network on all the strings with length up to 11. The corresponding minimization yields an equivalent 31 states machine.

These basic steps for performing grammatical inference in the case of finite state machines are summarized in figure 9.

Most problems of language processing have been tackled by using Elman's recurrent network. However, second-order RNNs are more suitable for extracting the internal representation and, consequently for grammatical inference (e.g. see the paper of Miller. and Giles (1993) or the one of Omlin and Giles (1996)).

Parsing with Simple-Synchrony Networks. Simple Synchrony Networks (SSNs) (Lane and Henderson 1998; Lane and Henderson 2000) are an extension of RNNs, adding another usage of internal time in order to represent structural constituents[22]. This extension does however not change the way external time[23] is dealt with.

More precisely, SSNs can be seen as RNNs of pulsing units, which enables them to represent structures and to generalize across structural constituents. The SSN approach consists in representing structural constituents directly, rather

[22] non-terminals in the case of Natural Language parsing

[23] word sequence in the case of Natural Language sentences.

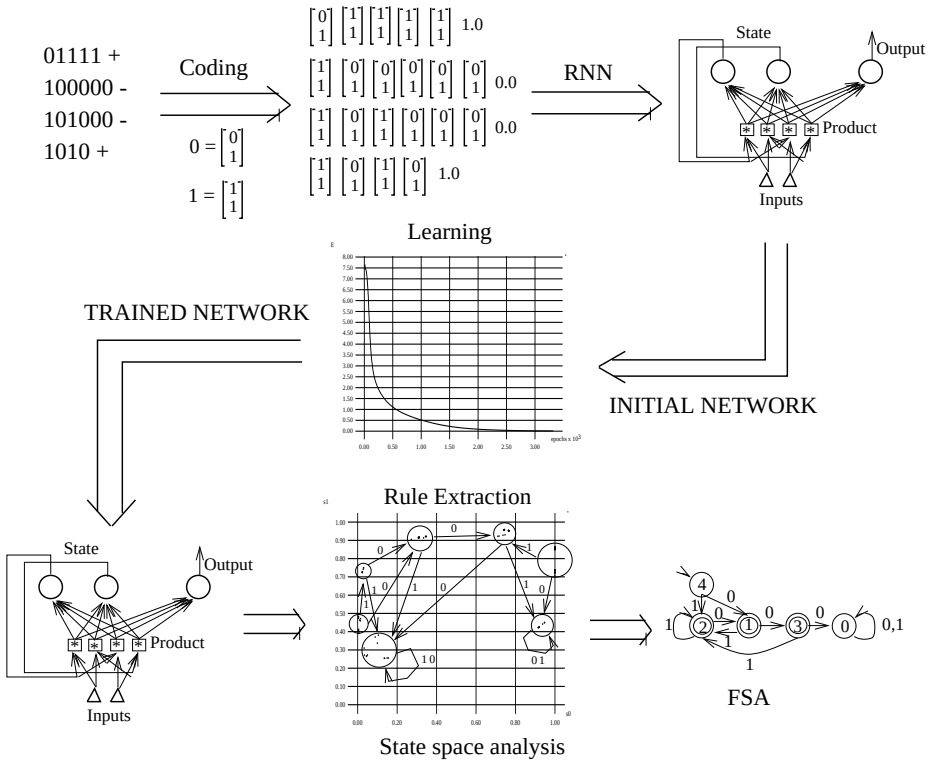


Fig. 9. Grammatical inference using neural networks: The learned configuration is subsequently used for the extraction of symbolic rules.

than using usual RNN indirect encoding. The structural relationships are represented by synchrony of neuron activation pulses, leading to an incremental representation of the structure over the constituents. Indeed, SNNs extend the incremental outputs of RNNs with as many output neurons as required by Temporal Synchrony Variable Binding (Shastri and Ajjanagadde 1993). The central idea is to divide each time period into several phases, each phase being associated with a unique constituent. For an input sentence of n words, the representation of the syntactic structure in the output is achieved by the unfolding of that structure in a temporal sequence of n phases in which unit synchrony represents some relationship between constituents in the structure (for instance the father-son relationship).

SSNs have been successfully applied on standard Natural Language parsing problems: taking English sentences drawn from a corpus of naturally occurring text, the model incrementally outputs a hierarchical structure representing how the words fit together to form constituents (i.e. a parse tree of the input sentence).

5 Extension: From Temporal to Structured Data Types

The learning of sequential information is the first step toward the adaptive computation of dynamic data types. RNNs, as presented in section 3.2, were conceived so as to exhibit a dynamic behaviour for incorporating time, i.e. dealing with temporal sequences.

From the structure point of view, any discrete sequence of real-valued variables defined over a time interval can be regarded as a list, which is in fact the simplest conceivable dynamic data type. RNNs can therefore be seen as good candidates for list processing and even more structured data types.

Early research in this direction was carried out by Pollack (1990) who introduced the RAAM model, which is capable of dealing with trees with labels in the leaves.

Processing Lists. If we represent an input signal sequence $\mathbf{U}$ of a RNN by a list in which each node, indexed by v, contains a real-valued vector $\mathbf{U}_v$, the general computational scheme described by equations 1 (page 115), and aimed at producing the output list $\mathbf{Y}$, can be written:

$$\begin{aligned} \mathbf{X}_v &= f\left(q^{-1}\mathbf{X}_v, \mathbf{U}_v, v, \Theta_v^f\right) \\ \mathbf{Y}_v &= g\left(\mathbf{X}_v, v, \Theta_v^g\right), \end{aligned} \tag{4}$$

where q^{-1} is the operator that, when applied to state $\mathbf{X}_a$, returns the state $\mathbf{X}_b$ of the next node of a.[24]

[24] The operator q^{-1} is introduced here in order to make the extension to more complex data structures easier. In the context of lists representing temporal sequences, it corresponds to the former time step, that is $q^{-1}\mathbf{X}_t \doteq \mathbf{X}_{t-1}$.

The model defined by equation 4 can itself be structured in the sense that the generic variable $X_{i,v}$ might be independent of $q^{-1}X_{j,v}$. Likewise other statements of independence might involve input-state variables and/or state-output variables. An explicit statement of independence is a sort of prior knowledge on the mapping that the machine is expected to learn. In general these statements can also be different for different nodes and can be conveniently expressed by a graphical structure that is referred to as a *recursive network*.

Lists can in fact be processed by means of an *encoding network* which is constructed by unfolding the input through the list. The corresponding network is created by associating each node of the list with input, state, and output variables, respectively. The state variables are connected graphically following the reverse direction of the list traversal, and the input and the output variables are connected with the associated state variables.

Generalization to Directed Ordered Acyclic Graphs. A nice extension of time sequences can be gained in the framework of dynamic data structures. Basically, giving a list corresponds with assigning a set of tokens where an order relation is defined. When paying attention to "temporal relations", a directed graph seems to be the most natural extension of list, in the sense that any directed graph is a way of defining a partial order over a set of homogeneous tokens. In particular, let us consider a directed ordered acyclic graph (DOAG) so that for any node v one can identify a set, potentially empty, of ordered children $ch[v]$. For each node, one can extend the next-state equation (4) as follows

$$\begin{aligned} \mathbf{X}_v &= f\left(\mathbf{X}_{ch[v]}, \mathbf{U}_v, v, \Theta_v^f\right) \\ \mathbf{Y}_v &= g\left(\mathbf{X}_v, v, \Theta_v^g\right). \end{aligned} \tag{5}$$

In the case of binary trees, the state associated with each node is calculated as a function of the attached label and of the states associated with the left and right children, respectively. The operators q_L^{-1} and q_R^{-1} make it possible to address the information associated with the left and right children of a given node and, therefore, straightforwardly generalize the temporal delay operator q^{-1}. Of course, for any node, the children must be ordered so as to be able to produce different outputs for binary trees $\{r, L, R\}$ and $\{r, R, L\}$. A list is just a special case of a binary tree in which one of the children is null.

The construction holds for any DOAG provided that a special node s of the graph, referred to as the *supersource*, is given together with the graph.

The computation of $\mathbf{Y}_v$ in the case of graphs is also more involved than the one associated with the simple recursive model of equations (4). The DOAG somehow represents the state update scheme, i.e. a graphical representation of the computation taking place in the recursive neural network (see figure 10). This graph plays its own role in the computation process either because of the information attached to its nodes or for its topology. This state update graphical representation emphasizes the structure of independence of some variables in the state-based model of equation 5. For instance, a classic structure of independence arises when the connections of any two state variables $\mathbf{X}_v$ and $\mathbf{X}_w$ only take place

between components $X_{i,v}$ and $X_{i,w}$ with the same index i. In the case of lists, this assumption means that only *local-feedback* connections are permitted for the state variables. Basically, the knowledge of a recursive network yields topological constraints which often make it possible to cut the number of learning parameters significantly.

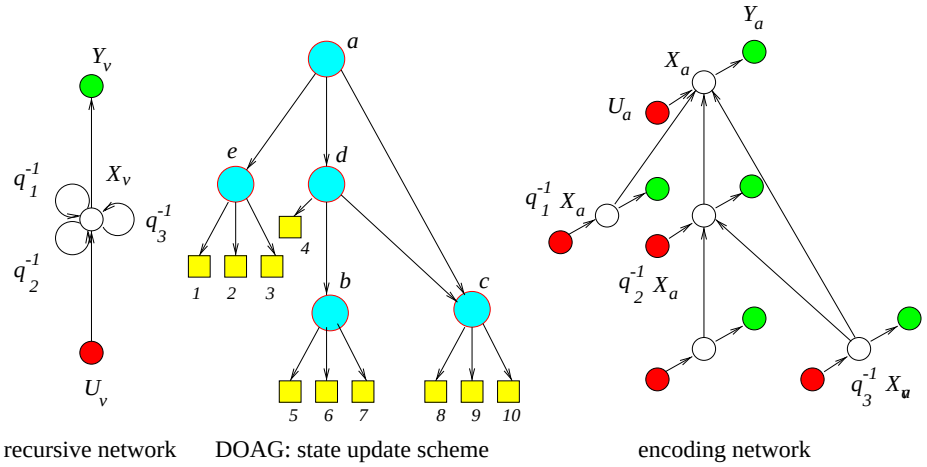


Fig. 10. Construction of the encoding network corresponding to a recursive network and a directed ordered acyclic graph. Proper frontier (initial) states are represented by squares. The encoding network inherits the structure of the input graph. When making the functional dependence explicit, the encoding network becames a neural network, which is used to calculate the corresponding output.

Furthermore, the information attached to the recursive network needs to be integrated with a specific choice of functions f and g which must be suitable for learning the parameters. A connectionist assumption for functions f and g turns out to be adequate especially to fulfil computational complexity requirements. The first-order recursive neural network is one of the simplest architectural choices. In this case, equation 5 becomes:

$$\begin{aligned} \mathbf{X}_v &= \sigma\left(\mathbf{A}_v \cdot q^{-1}\mathbf{X}_v + \mathbf{B}_v \cdot \mathbf{U}_v\right) \\ \mathbf{Y}_v &= \sigma\left(\mathbf{C}_v \cdot \mathbf{X}_v\right). \end{aligned} \tag{6}$$

Matrix $\mathbf{A}_v \in \mathcal{R}^{n,n}$ contains the weights associated with the feedback connections, whereas matrix $\mathbf{B}_v \in \mathcal{R}^{n,m}$ contains the weights associated with the input-neuron connections. Finally, $\mathbf{C}_v \in \mathcal{R}^{p,n}$ is the parameter for the definition of the state-output map. These equations produce the next-state and the output values by relying on a first-order equation, in which the outputs are bounded by using a squashing function[25]. An in-depth analysis of this models can be found in (Frasconi, Gori, and Sperduti 1998; Frasconi 1998).

[25] In equation 6, $\sigma(\cdot)$ denotes a vector of squashing functions operating on n of neurons.

Concerning the learning, since the structure of the encoding neural network is inherited by both the DOAG and the recursive network, the encoding neural network is essentially a feedforward network. Hence the back-propagation algorithm for feedforward neural networks can be conveniently extended to data structures. The learning algorithm is in this case referred to as *back-propagation through structure* (Sperduti 1998).

6 Conclusion

We gave in this chapter an overview of what is presently going on in the field of temporal connectionist models. The aim was not to be as exhaustive as possible but to exhibit the main concepts, ideas and applications in this area.

Temporal connectionist models already have the power to do arbitrary computations with time-varying data with the advantage of learning from examples. Fundamental theorems about their potential capabilities already exist. They however still need to find efficient ways to be used in practice: existing training methods still suffer from their inability to deal with very long time dependencies. Although the success so far of capturing and classifying temporal information with neural networks is still limited, the approach looks very promising and benefits from a rapid growth.

Acknowledgements

The authors would like to thank James B. Henderson, the reviewers and editors for their helpful comments on this chapter, and Peter Weyer-Brown for his careful proofreading of the manuscript.

References

Abbott, L. F. and T. B. Kepler (1990). Model neurons : from Hodgkin-Huxley to Hopfield. In L. Garrido (Ed.), *Statistical Mechanics of Neural Networks*, pp. 5–18. Springer.

Abeles, M. (1982). *Local cortical circuits: an electrophysiological study (Studies of brain functions, Vol. 6)*. Springer Verlag.

Amit, D. J. (1988). Neural network counting chimes. *Proc. Nat. Acad. Sci. USA 85*, 2141–2145.

Back, A. and A. Tsoi (1991). Fir and iir synapses: A new neural network architecture for time series modeling. *Neural Computation 3*(3), 375–385.

Bengio, Y., R. Cardin, R. de Mori, and E. Merlo (1989). Programmable execution of multi-layered networks for automatic speech recognition. *Communications of the ACM 32*, 195–199.

Bengio, Y., P. Frasconi, and P. Simard (1993). The problem of learning long-term dependencies in recurrent networks. In *IEEE Transactions on Neural Networks*, San Francisco, pp. 1183–1195. IEEE Press. (invited paper).

Bengio, Y., R. D. Mori, and M. Gori (1992, May). Learning the dynamic nature of speech with back-propagation for sequences. *Pattern Recognition Letters 13*(5), 375–386.

Bengio, Y., P. Simard, and P. Frasconi (1994, March). Learning long-term dependencies is difficult. *IEEE Trans. on Neural Networks 5*(2), 157–166.

Béroule, D. (1987). Guided propagation inside a topographic memory. In *1st int. conf. on neural networks*, San Diego, pp. 469–476. IEEE.

Bodenhausen, U. and A. Waibel (1991). The Tempo2 algorithm: adjusting time delays by supervised learning. In R. P. Lippmann, J. Moody, and D. S. Tourestzky (Eds.), *Advances in Neural Information Processing Systems*, Volume 3, San Mateo (CA), pp. 155–161. Morgan Kaufmann.

Bonnet, D., V. Perrault, and A. Grumbach (1997a). Daily passenger traffic forecasting using δ-NARMA neural networks. In *Proceedings of the World Congress on Railroad Research (WCRR'97)*, pp. CD–ROM.

Bonnet, D., V. Perrault, and A. Grumbach (1997b). δ-NARMA neural network: a new approach to signal prediction. *IEEE Transaction on Signal Processing 45*(11), 2799–2810.

Bonnet, D., V. Perrault, and A. Grumbach (1997c). δ-NARMA neural networks: a connectionist extension of ARARMA models. In M. Verleysen (Ed.), *Proceedings of the European Symposium on Artificial Neural Networks*, Brussels (Belgium), pp. 127–132. D Facto.

Bourlard, H. and N. Morgan (1994). *Connectionist Speech Recognition – A Hybrid Approach*. Kluwer Academic Publishers.

Bourlard, H. and N. Morgan (1998). Hybrid HMM/ANN systems for speech recognition: Overview and new research directions. In C. L. Giles and M. Gori (Eds.), *Adaptive Processing of Sequences and Data Structures*, Volume 1387 of *Lecture Notes in Artificial Intelligence*, pp. 389–417. Springer.

Catfolis, T. (1994, July). Mapping a complex temporal problem into a combination of static and dynamic neural networks. *Sigart Bulletin 5*(3), 23–28.

Chappelier, J.-C. and A. Grumbach (1994, July). Time in neural networks. *Sigart Bulletin 5*(3), 3–10.

Chappelier, J.-C. and A. Grumbach (1996, March). A Kohonen map for temporal sequences. In *NEURAP'95*, Marseille.

Chappelier, J.-C. and A. Grumbach (1998, May). RST: a connectionist architecture to deal with spatiotemporal relationships. *Neural Computation 10*(4), 883–902.

Chappell, G. J. and J. G. Taylor (1993). The temporal Kohonen map. *NN 6*, 441–445.

Cleeremans, A., D. Servan-Schreiber, and J. McClelland (1989). Finite state automata and simple recurrent networks. *Neural Computation 1*, 372–381.

Connor, J., L. E. Atlas, and D. R. Martin (1992). Recurrent network and NARMA modelling. In S. J. Hanson, R. P. Lippmann, J. E. Moody, and D. S. Touretzky (Eds.), *Advances in Neural Information Processing Systems*, Volume 4, pp. 301–308. San Mateo (CA): Morgan Kaufmann.

Connor, J. and D. R. Martin (1994). Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks 5*(2), 240–253.

de Vries, B. and J. C. Principe (1992). The gamma model. A new neural model for temporal processing. *Neural Networks 5*, 565–576.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science 14*(2), 179–211.

Elman, J. L. (1991, September). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning 7*(2), 195–226.

Euliano, N. R. and J. C. Principe (1996, June). Spatio-temporal self-organizing feature maps. In *IJCNN'96*, Volume 4, pp. 1900–1905.

Fogelman-Soulié, F. and P. Gallinari (Eds.) (1998). *Industrial Applications of Neural Networks*. World Scientific Publishing Co.

Frasconi, P. (1998). An introduction to learning structured information. In C. L. Giles and M. Gori (Eds.), *Adaptive Processing of Sequences and Data Structures*, Volume 1387 of *Lecture Notes in Artificial Intelligence*, pp. 99–120. Springer.

Frasconi, P., M. Gori, and G. Soda (1992). Local feedback multi-layered networks. *Neural Computation 4*(2), 120–130.

Frasconi, P., M. Gori, and A. Sperduti (1998, September). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks 9*, 768–786.

Funahashi, K. and Y. Nakamura (1993). Approximations of dynamical systems by continuous time recurrent neural networks. *Neural Networks 6*(6), 801–806.

Gerstner, W. (1995). Time structure of the activity in neural network models. *Physical Review E 51*, 738–758.

Goldberg, K. Y. and B. A. Pearlmutter (1989). Using backpropagation with temporal windows to learn the dynamics of the cmu direct-drive arm ii. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems*, Volume 1. Morgan-Kaufmann.

Gori, M. (Ed.) (1992). *Neural Networks for Speech Processing*. Lint.

Gorman, R. P. and T. J. Sejnowski (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *NN 1*, 75–89.

Hirch, M. W. (1991). Network dynamics : Principles and problems. In F. Paseman and H. Doebner (Eds.), *Neurodynamics, Series on neural networks*, pp. 3–29. World Scientific.

Hochreiter, S. and J. Schmidhuber (1997a). Bridging long time lags by weight guessing and "lond short term memory". In F. L. Silva, J. C. Principe, and L. B.

Almeida (Eds.), *Spatiotemporal Models in Biological and Artificial Systems*, pp. 65–72. IOS Press.

Hochreiter, S. and J. Schmidhuber (1997b). Long short-term memory. *NC 9*(8), 1735–1780.

Horn, D. and M. Usher (1991). Parallel activation of memories in an oscillatory neural network. *Neural Computation 3*(1), 31–43.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *NN 4*, 251–257.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward neural networks are universal approximators. *NN 2*(5), 359–366.

Jacquemin, C. (1994, July). A temporal connectionnist approach to natural language. *Sigart Bulletin 5*(3), 12–22.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proc. of the 8th annual conference on Cognitive Science*. Erlbaum.

Kangas, J. (1990). Time-delayed self-organizing maps. In *Proceedings of IJCNN'90*, Volume II, pp. 331–336.

Kohonen, T. (1991, dec). The hypermap architecture. In O. S. T. Kohonen, K. Makisara and J. Kangas (Eds.), *Artificial Neural Networks*, pp. 1357–1360. North-Holland.

Kolen, J. F. (1994). Recurrent networks: State machines or iterated function systems? In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend (Eds.), *Proceedings of the 1993 Connectionist Models Summer School*, Hillsdale NJ, pp. 203–210. Erlbaum.

Kopecz, K. (1995). Unsupervised learning of sequences on maos with lateral connectivity. In *Proceedings of ICANN'95*, Volume 2, pp. 431–436.

Lane, P. C. R. and J. B. Henderson (1998). Simple synchrony networks: Learning to parse natural language with temporal synchrony variable binding. In L. Noklasson, M. Boden, and T. Ziemke (Eds.), *Proc. of 8th Int. Conf. on Artificial Neural Networks (ICANN'98)*, Skövde (Sweden), pp. 615–620.

Lane, P. C. R. and J. B. Henderson (2000). Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Transactions on Knowledge and Data Engineering to appear*. Special Issue on Commenctionist Models for Learning in Structured Domains.

Lang, K. J., A. H. Waibel, and G. E. Hinton (1990). A time-delay neural-network architecture for isolated word recognition. *Neural Networks 3*(1), 23–44.

Lapedes, A. S. and R. Farber (1987). Nonlinear signal processing using neural networks: prediction and system modelling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos (CA).

Lumer, E. D. and B. A. Huberman (1992). Binding hierarchies : a basis for dynamic perceptual grouping. *NC* **4**, 341–355.

Maass, W. (1994). On the computationnal complexity of networks of spiking neurons. In *NIPS'94 Proc.*, Volume 7. MIT-Press.

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation 8*(1), 1–40.

Maass, W. (1997a). Analog computing with temporal coding in networks of spiking neurons. In F. L. Silva, J. C. Principe, and L. B. Almeida (Eds.), *Spatiotemporal Models in Biological and Artificial Systems*, pp. 97–104. IOS Press.

Maass, W. (1997b). Networks of spiking neurons: the third generation of neural network models. *Neural Networks 10*(9), 1659–1671.

MacGregor, R. J. and E. R. Lewis (1977). *Neural Modeling, Electric signal processing in the neurons systems*. Plenum Press.

Miller., C. B. and C. L. Giles (1993). Experimental comparison of the effect of order in recurrent neural networks. *Int. Journal of Pattern Recognition and Artificial Intelligence 7*(4), 849–872.

Morasso, P. (1991, dec). Self-organizing feature maps for cursive script recognition. In O. S. T. Kohonen, K. Makisara and J. Kangas (Eds.), *Artificial Neural Networks*, pp. 1323–1326. North-Holland.

Mozayyani, N., V. Alanou, J. Dreyfus, and G. Vaucher (1995). A spatio-temporal data-coding applied to Kohonen maps. In *International Conference on Artificial Neural Networks*, Volume 2, pp. 75–79.

Mozer, M. (1989). A focused back-propagation algorithm for temporal pattern recognition. *Complex Systems 3*, 349–381.

Mozer, M. C. (1994). Neural net architectures for temporal sequence processing. In A. Weigend and N. Gershenfeld (Eds.), *Time Series Prediction*, pp. 243–264. Addison–Wesley.

Narendra, K. P. and K. Parthasarathy (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks 1*, 4–27.

Nerrand, O., P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos (1993). Neural networks and nonlinear adaptive filtering: unifiying concepts and new algorithms. *Neural Computation 5*, 165–197.

Omlin, C. and C. Giles (1996). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM 43*(6), 937–972.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence 46*(1–2), 77–106.

Privitera, C. M. and P. Morasso (1993). A new approach to storing temporal sequences. In *Proc. IJCNN'93*, pp. 2745–2748.

Ramacher, U. (1993). Hamiltonian dynamics of neural networks. *Neural Networks 6*(4), 547–557.

Reber, A. S. (1976). Implicit learing of synthetic languages: The role of the instructional set. *Journal of Experimental Psycology: Human Learning and Memory 2*, 88–94.

Rinzel, J. and G. B. Ermentrout (1989). Analysis of neural excitability and oscillations. In C. Koch and I. Segev (Eds.), *Methods in Neural Modeling – From Synapses to Networks*, pp. 135–169. MIT Press.

Robinson, T. (1994, March). An application of recurrentn nets to phone probability estimation. *IEEE Trans. on Neural Networks 5*(2), 298–305.

Rohwer, R. (1994, July). The time dimension of neural network models. *Sigart Bulletin 5*(3), 36–44.

Servan-Schreiber, A. Cleeremans, and J. McClelland (1991, September). Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning 7*(2), 161–194.

Shastri, L. and V. Ajjanagadde (1993). From simple associations to systematic reasoning: a connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences 16*, 417–494.

Siegelmann, H. T. and E. D. Sontag (1995). On the computational power of neural nets. *Journal of Computers and System Sciences 50*, 132–150.

Simpson, P. K. and R. O. Deich (1988, sept). Neural networks, fuzzy logic and acoustic pattern generation. In *Proceedings of AAAIC'88*.

Sperduti, A. (1998). Neural network for processing data structures. In C. L. Giles and M. Gori (Eds.), *Adaptive Processing of Sequences and Data Structures*, Volume 1387 of *Lecture Notes in Artificial Intelligence*, pp. 121–144. Springer.

Tank, D. W. and J. J. Hopfield (1987). Neural computation by concentring information in time. *Proc. Nat. Acad. Sci. USA 84*, 1896–1900.

Tsoi, A. C. (1998). Recurent neural network architectures: An overview. In C. L. Giles and M. Gori (Eds.), *Adaptive Processing of Sequences and Data Structures*, Volume 1387 of *Lecture Notes in Artificial Intelligence*, pp. 1–26. Springer.

Tsoi, A. C. and A. Back (1997, June). Discrete time recurent neural network architectures: A unifying review. *NeuroComputing 15*(3 & 4), 183–223.

Unnikrishnan, K. P., J. J. Hopfield, and D. W. Tank (1991). Connected-digit speaker-dependent speech recognition using a neural network with time delay connections. *IEEE Transaction on Signal Processing 39*(3), 698–713.

Vaucher, G. (1996). Neuro-biological bases for spario-temporal data coding in artificial neural networks. *Lecture Notes in Computer Science 1112*, 703ff.

Williams, R. and D. Zipser (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation 1*(3), 270–280.

Williams, R. J. and J. Peng (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation 2*(4), 490–501.