On the Prediction of Solar Activity Using Different Neural Network Models

F. Fessant S. Bengio D. Collobert

France Telecom CNET,
LAB/RIO/TNT,
Technopole Anticipa,
2 avenue Pierre Marzin,
22307 Lannion Cedex, France.
email: bengio@iro.umontreal.ca

Abstract

Accurate prediction of ionospheric parameters is crucial for telecommunication companies. These parameters strongly rely on solar activity. In this paper, we analyze the use of neural networks for sunspots time series prediction. Three types of models are tested and experimental results are reported for a particular sunspots time series: the $IR5$ index.

1 Introduction

France Telecom's research center, the *Centre National d'Etudes des Telecommunications* (CNET) has a ionospheric prediction service which publishes each month prediction reports about ionospheric propagation of radio waves, addressed to users of long distance communications with mobiles (airplanes, ships, cars), radio services or more generally to users of the frequency band between 3 and 30 MHz.

Since ionosphere conditions have a high variability and since radio services use ionosphere, predictions are needed for these reports, which take into account some ionospheric characteristics.

Ionosphere is the ionized region of terrestrial atmosphere which extends from 50 to 2000 km above earth surface. It is generally divided into three parts in increasing altitude order: the D, E and F layers. The latter is the most ionized and acts as a reflector for decametric waves propagation (or HF waves), from 3 to 30 MHz, depending upon ionization rate, and allows transmissions between two far away terrestrial points.

To make accurate predictions we have to measure and analyze not only several ionospheric parameters but also solar parameters. In fact ionospheric state directly depends on solar activity, and in order to make a prediction of its future state we have to predict this solar activity.

Solar activity level is represented by the relative number of sunspots R_i (also named international sunspot number) which presents short term fluctuations. The monthly mean follows a cycle of around 11 years (but which varies in fact from 9 to 14 years).

Most organizations making ionospheric predictions use a monthly index $R12$: a twelve months running mean of R.

There exists a variety of methods that have been tested to predict this index. The method that yields the best performance comes from (McNish and Lincoln, 1949): they describe a multiple regression technique that predicts annual or monthly sunspot means based on computation of an average cycle which is then modified by taking into account the current cycle behavior.

The CNET ionospheric prediction service uses another index called $IR5$, a five months running mean of R, which seems more adapted to its users (Bourdila and Hanbaba, 1984). Its prediction is based on harmonic analysis, cycle repetition after some years and relations between maximum solar activities and duration of ascendant phases.

In fact, the CNET ionospheric prediction service measures and uses many solar or ionospheric informations:

- the $IR5$ index,
- the solar flux, which represents the solar radio noise,
- a long series of geomagnetic data, and
- ionospheric parameters such as the critical frequency of the ionospheric F2 layer (called fof2) which is measured by sounding and represents the maximal frequency that ensure a radio-electric link.

Among the above list, we decided to use this $IR5$ index for the experiments reported in this paper mainly because sunspots series is known to be difficult to predict, but the methods reported here could also be used for other ionospheric time series.

Neural network models are an alternative to classical methods of time series prediction. In this paper we use them to predict the $IR5$ index. They are currently successfully used in various fields: feature recognition such as handwritten characters (Le Cun et al., 1989) or speech (Lippmann, 1989), image compression (Cottrell et al., 1987), games (Tesauro, 1992), and many others.

The use of neural networks for time series prediction is relatively new: it was first proposed by (Lapedes and Farber, 1987) but it is now applied on real world time series like electric or water consumption (Canu et al., 1990; Park et al.,

1991) or financial and economical time series prediction (Varfis and Versino, 1990).

In the field of solar and terrestrial parameters prediction, (Koons and Gorney, 1990) were the first to predict the timing and R value of the 22^{nd} cycle maximum. Later, Lundstedt forecasted solar-terrestrial effects and geomagnetic storms from solar wind data (Lundstedt, 1992; Lundstedt and Wintoft, 1994) (see also the work of (Macpherson, 1993) on solar activity).

In this paper we propose a six months ahead $IR5$ prediction, given the past values of this index (this six months delay is needed in order to edit and distribute a report to users) with a particular class of learning machine : the multi-layer perceptron trained by backpropagation. In the next section we briefly introduce neural networks for function approximation and specially prediction. Then we present the particular problem we are trying to solve and the exact neural network models we used.

Experiments and results are then given in details and compared to those previously used by the CNET ionospheric prediction service. Conclusions and remarks are given in the final section.

2 Neural Networks for Prediction

Suppose we have a given one-variable time series represented by the N values $\{x_1, x_2, \cdots, x_N\}$. Prediction then consists to find the future values $\{x_{N+1}, x_{N+2}, \cdots\}$. (Takens, 1981) showed that if the series is deterministic, there exists a scalar d (which is called the *embedding dimension*) and a function f such that for every $t > d$:

$$x_t = f(x_{t-1}, x_{t-2}, \cdots, x_{t-d}) \tag{1}$$

Then the prediction problem consists, given the first N values of a time series, to find the appropriate d and f. Of course one usually cannot be sure a given series is deterministic. Actually, statistical methods do exist to verify if a series is deterministic and to evaluate d but they require the size of the series to be on the order of 10^d which is rarely the case in real world problems. For the moment, let's assume we know d and want to find f. This is where *neural networks* comes in: it is a well known fact that they can be used as universal function approximators (Cybenko, 1989; Funahashi, 1989; Hornik et al., 1989).

2.1 Introduction to Neural Networks

Neural networks[1] consist of a large number of highly connected nonlinear simple units. In the models used in predictions, we can differentiate three types of units:

[1]For a more complete introduction to the field of neural networks, see for instance (Hertz et al., 1991).

- *input units* which are set to the previous values of the time series: x_{t-1}, x_{t-2}, $\cdots$, x_{t-d} where d is the *embedding dimension*,
- *output units* which give the results of the neural network. In the simplest case, we have only one output unit which should return x_t,
- and finally *hidden units* which are neither input nor output units, but are used to keep an internal representation of the problem.

Each connection between two units is directed and is given a weight. In fact, the *knowledge* of the network is kept in these weights. Each hidden and output units computes its value as the weighted sum of its inputs, passed through a nonlinear function such as $tanh(\cdot)$.

Simple networks usually consists of layers of units where all units in one layer are connected to all units in the next layer. See for instance Figure 1.

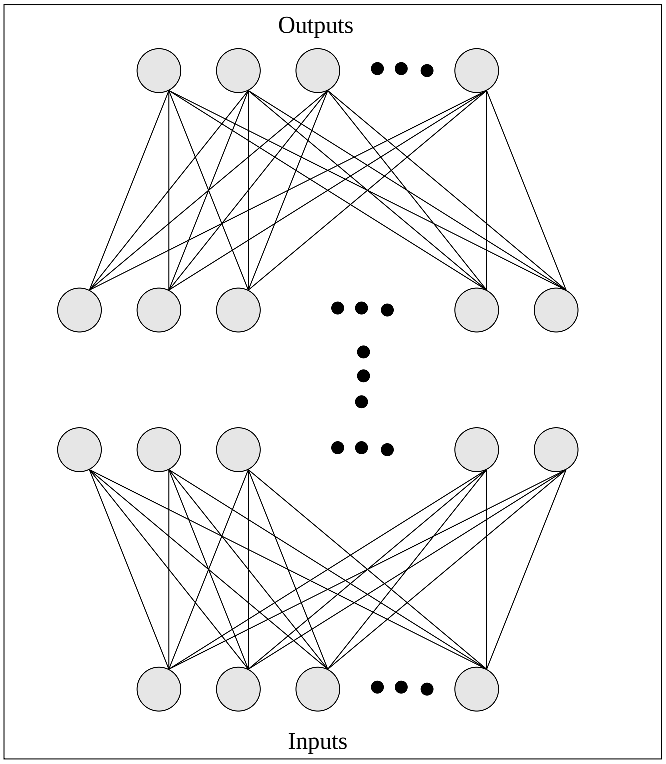

Figure 1: Illustration of a simple neural network architecture.

The idea then is to find, for a given network architecture and a given time series, the weights that minimize a cost which is function of the difference between the resulting values of the network and the desired values.

The time series is generally divided into two parts: a *training set* and a *test set*. The training set (for instance the first values of the time series) is used to find the weights by minimizing a cost function, whereas the test set (for instance the last values of the time series) is used to verify the real prediction performance of the network (that is, an estimated prediction error on future time series values).

2.1.1 Cost Function.

There exists a variety of cost functions that can be used to optimize a neural network performance. The most used cost function is the *Least Mean Square* (or LMS) criteria which can be written as:

$$J_{LMS} = \sum_{i \in P} \sum_{j \in O} (y_{i,j} - \hat{y}_{i,j})^2 \tag{2}$$

where J is the cost to minimize, P is the set of all patterns in the training set, O is the set of all output units, $\hat{y}_{i,j}$ is the value of output unit j after presentation of pattern i, and $y_{i,j}$ is the desired value of corresponding output unit.

In time series context, the patterns $i \in P$ are the set of vectors $\{x_{t-1}, x_{t-2}, \cdots, x_{t-d}\}$ where t varies from $d+1$ to the time series length, and there is usually only one output unit which corresponds to x_t.

One problem with the use of LMS criteria for time series prediction is the difficulty to compare adequately prediction results from two different time series. This can be done using a normalized version of the LMS, namely the *Average Relative Variance* (or ARV) criteria (Weigend et al., 1992), which is the same as the LMS criteria but normalized by the number of training examples N and the estimated variance of the data $\hat{\sigma}^2$:

$$J_{ARV} = \frac{1}{\hat{\sigma}^2} \frac{1}{N} J_{LMS} \tag{3}$$

2.1.2 Learning mechanisms.

Given a cost function, a network architecture and some data (the time series), the next step is to find the appropriate weights which minimize the cost function. This is usually done using an iterative procedure which consists of the following steps: first initialize the network weights randomly, then for each examples in the training set, compute the network output while feeding the example as input, compare the resulting output to the target and apply a correction to all weights which minimize the error. One iteration is the presentation of all examples. The procedure could last many iterations.

The most known learning mechanism for neural networks is the *backpropagation* rule (Rumelhart et al., 1986) which dates back to 1986. It is a simple gradient descent technique which minimizes the cost function in weight space. The idea is to modify the weights in the opposite direction of the gradient of the error with respect to the weights:

$$w_i = w_i - \mu \frac{\partial J}{\partial w_i} \tag{4}$$

where w_i is a weight, J is the cost function, and μ is a small real value which is usually called the *learning rate*. $\frac{\partial J}{\partial w_i}$ is then calculated using the standard *chain rule* by *backpropagating* error derivative information through all the connections.

Since 1986, a variety of improvements has been proposed (introduction of a *momentum* term, use of conjugate gradient techniques, use of second order information, etc)[2].

2.1.3 Generalization and Model Selection.

One of the most important features of learning systems is their ability to *generalize* to new situations. As we have seen, a learning machine such as a neural network is usually trained to minimize a cost function over a finite set of examples (the training set), but what we are really interested in is to find a function which minimizes our cost function over all the input domain. Particularly, in prediction problems, we train the network with *past examples* (thus, we minimize a *training error*) but we really want our network to perform well on *future examples* (thus, have a minimal *generalization error*). We usually use a test set (data not used to minimize the cost) to estimate generalization error. If the training set and the test set are drawn from the same (but unknown) probability distribution (and hence have the same statistical properties) we expect both costs (on training set and test set) to be minimized while effectively minimizing only the first.

This is only true if we have enough training examples and if the network architecture is adequate for the problem. If there are too many parameters (weights) in the network, it will learn a function which is specific to the training set and not to the whole real function. On the other hand, if there are not enough parameters, the network will not be able to find an appropriate function. Theoretical results such as (Vapnik, 1982) show that the smallest generalization error we can reach is function of the training set size, the network capacity (which is roughly a measure of the number of free parameters), and the training error.

This means we have to find the best network architecture for a given problem and a given training set size. Many heuristics exist (such as described in (Fahlman and Lebiere, 1990; Le Cun et al., 1990) but this is still a hard problem.

[2] Again, refer to (Hertz et al., 1991) for an overview.

2.2 Finding the Embedding Dimension d

We have described a method to find the function f that gives us the next value of a given time series. But this method requires that we already know the embedding dimension d, which is used as the input size of our network. There exists many statistical methods to estimates d, but they all suffer the same problem: they need a huge amount of data[3].

An alternative to these classical techniques is once again to use neural networks to find the best embedding dimension by simple cross-validation: cut training set into two parts: the new training set and a validation set; then compare the validation error (cost on the validation set) obtained after training different network models (and more specifically different input sized networks) with the new training set and keep the best model (with respect to validation error). The input dimension of this model is then a good estimate of the embedding dimension d.

2.3 Horizon of Prediction

In most time series problems, one does not want to predict x_t, the next value of the series but rather x_{t+l} with $l > 0$, a future value in a mid or long range horizon. To solve this kind of problem, there is (at least) two solutions: the *iterated prediction* and the *direct prediction*.

The iterated prediction system consists in feeding the previously predicted values $\hat{x}_t, \hat{x}_{t+1}, \cdots, \hat{x}_{t+l-1}$ as inputs of the network to predict x_{t+l} (remember we do not know the exact values of $x_t, x_{t+1}, \cdots, x_{t+l-1}$, so we use the ones estimated by the network given the known past values).

The direct prediction system consists to train a network to learn directly x_{t+l} using $x_{t-1}, x_{t-2}, \cdots, x_{t-d}$ as inputs.

Obviously the direct prediction system can only be used when horizon l is small. In fact, experiments have shown that for short horizons ($l < 10$) it yields better performance than the iterated prediction system, but as soon as l gets bigger, one should use the latter.

3 Problem Description

The CNET ionospheric prediction service has given us the $IR5$ time series, which can be written as:

$$IR5_t = \frac{1}{5}(R_{t-3} + R_{t-2} + R_{t-1} + R_t + R_{t+1}) \tag{5}$$

where $IR5_t$ is the index for month t and R_t the mean sunspot number for the same month.

[3] See (Abarbanel et al., 1993) for an extended review of these methods.

The time series begins in January 1849 and ends in December 1991, so we have around 1700 values (in fact the series continues until now but for our experiments and for correct comparisons, we had to use the same set of values as the CNET prediction service used). As R, the $IR5$ index follows a mean cycle of 11 years, ranging from 9 to 14 years. A cycle rise is around 4.5 years long and it takes around 6.5 years to descend. Figure 2 shows the $IR5$ time series.

As explained before, we are interested in a six months ahead prediction of $IR5$:

$$IR5_{t+5} = f(IR5_{t-1}, IR5_{t-2}, \cdots, IR5_{t-d}) \tag{6}$$

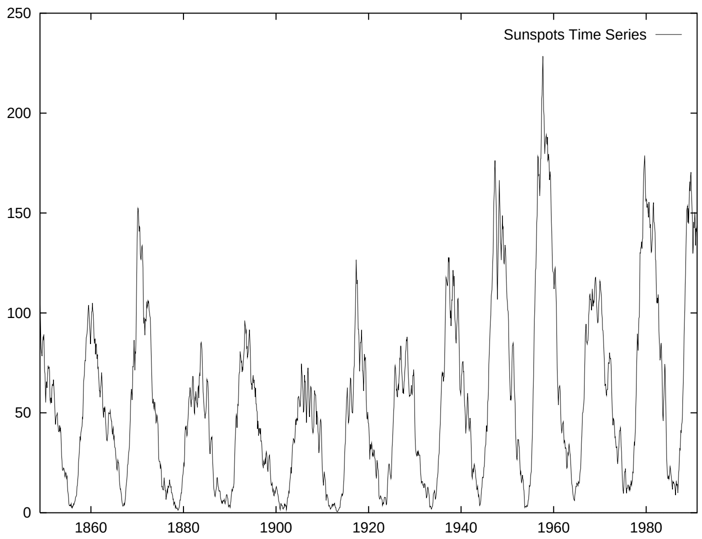

Figure 2: The $IR5$ sunspots series.

4 Experimental Results

In this section, we describe the experiments we did, how data was used, what kind of network models were tested, and finally we give some comparative results.

4.1 Data Preprocessing

An important step when one wants to use a learning machine is data preprocessing. For instance, if the time series has a tendency (a non constant moving mean), it should be removed (see for instance (Box and Jenkins, 1970) for an introduction to time series analysis). Unfortunately, we don't know actually if the sunspots time series has a tendency because of the lack of data.

An other important preprocessing is the normalization stage. Data should have a zero mean and values should range from $[-1, 1]$ for neural networks to perform well[4]. Each $IR5$ value had thus been normalized as follows:

$$x_t = \frac{IR5_t - \mu}{\max_{i<N} |IR5_i - \mu|} \tag{7}$$

where N is the training set size, μ is the training set mean, and x_t is the new series value.

4.2 Network Models

We tried three types of neural network models: a *simple model*, a *modular model* and an *Elman network*. For all three models, we used cross-validation to select the number of units in each layer. The number of input units (which is the embedding dimension d) was empirically determined to be 40. The modular model had one output unit (x_{t+5}), while the simple and Elman models had 6 output units which correspond to $\{x_t, x_{t+1}, \cdots, x_{t+5}\}$, but generalization error was only computed on the 6^{th} output which is the six months ahead prediction we need.

4.2.1 The Simple Model.

The simple model is just the plain multi-layer perceptron with one hidden layer of units. All input units are connected to all hidden units and all hidden units are connected to all output units. There is no connection between input and output units. Figure 3 shows the network.

4.2.2 The Modular Model.

The idea of the modular model comes from one important concept: we have seen we shouldn't use as many hidden units (and connections) as we want because this will result in generalization problems. We thus have to restrict the number of free parameters (the connections). Modular systems tends to divide problems into smaller and hopefully simpler ones. Thus, one can use two small independant networks trying to find the solution by theirselves and then combining their outputs using a third small network. This is exactly what we did. The first small network tries to find x_{t+5} while the second tries to find also intermediate values $x_t, \cdots, x_{t+4}$.

The output of the first network and the outputs of the second network are then used as input to a third network, as can be seen in Figure 4. The three

[4]In fact, data should range between the two nonlinear parts of the function used in hidden units in order to help initial learning. In our case, we used the $tanh(\cdot)$ function.

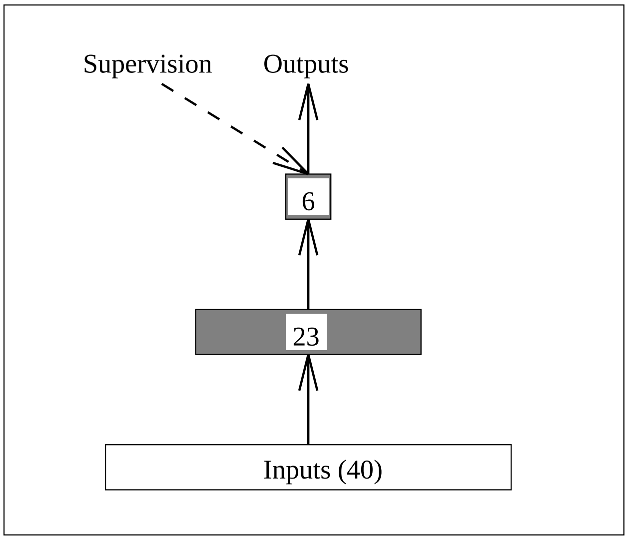

Figure 3: The simple neural network architecture. A full line between two layers means that all units in one layer are fully connected to all units in the other layer. The number of units in a layer is given in its center. The dashed line means external supervision is provided and the cost is minimized at that layer. In that case the outputs are $\{x_t, \cdots, x_{t+5}\}$ while the inputs are $\{x_{t-1}, \cdots, x_{t-40}\}$.

networks are trained simultaneously and supervision is provided to all output units. This means that connections of the first two networks are influenced both by supervision provided by their respective network but also by the third network, using the usual chain rule to find the exact error derivatives.

The total number of free parameters in the modular model is still half the number of free parameters in the simple model.

4.2.3 Elman's Recurrent Network.

(Elman, 1990) has proposed a partially recurrent neural network: a model intended to deal with the structural aspects of language as it varies in time. This model has been used by other authors and promises to be useful in various time related domains.

This variant of the multi-layer perceptron consists in the addition of partially recurrent links into the hidden layer. They serve to copy activations of hidden layer units at a given time to an additional "context layer", which in turn feeds the activation back to the hidden layer at the next time step. The hidden layer activation is thus a function both of current inputs and past hidden values. We say it is a *partially* recurrent network because the recurrent weights are not learned but rather fixed to 1. See Figure 5 for an illustration.

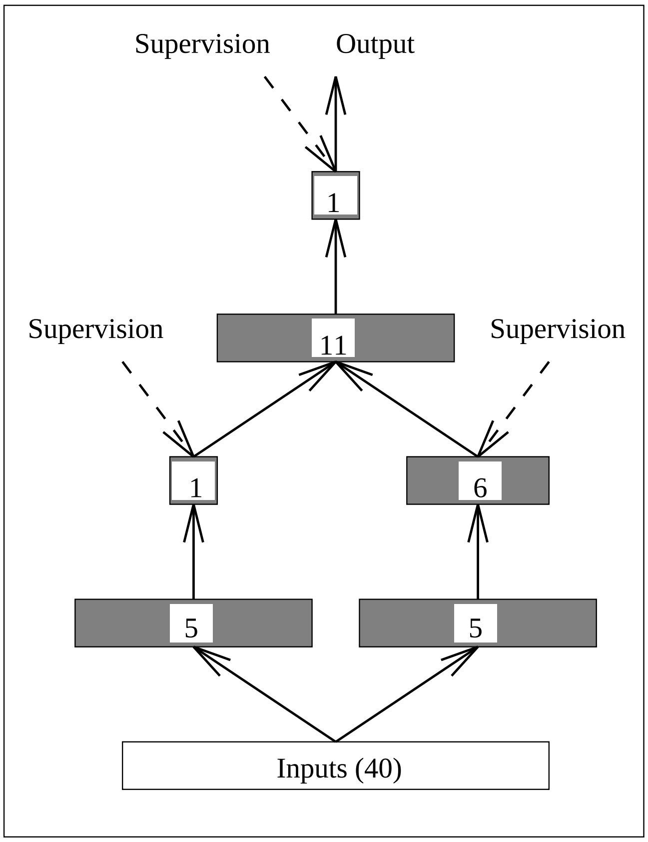

Figure 4: The modular neural network architecture. A full line between two layers/modules means that all units in one layer are fully connected to all units in the other layer. The number of units in a layer is given in its center. The dashed line means external supervision is provided and the cost is minimized at that layer. In that case the output is either $\{x_{t+5}\}$ and $\{x_t, \cdots, x_{t+5}\}$ for the intermediate output modules, or $\{x_{t+5}\}$ for the final output module. The inputs are $\{x_{t-1}, \cdots, x_{t-40}\}$.

4.3 Results

For all three models, data was used as follows: we kept the last 238 examples for the test set and used the other 1428 examples for the training set. Each network was trained starting from initial random weights and the training was stopped using a cross-validation method. We used the stochastic version of backpropagation with a small weight decay of 10^{-6} to prevent overfitting and a momentum term of 0.9.

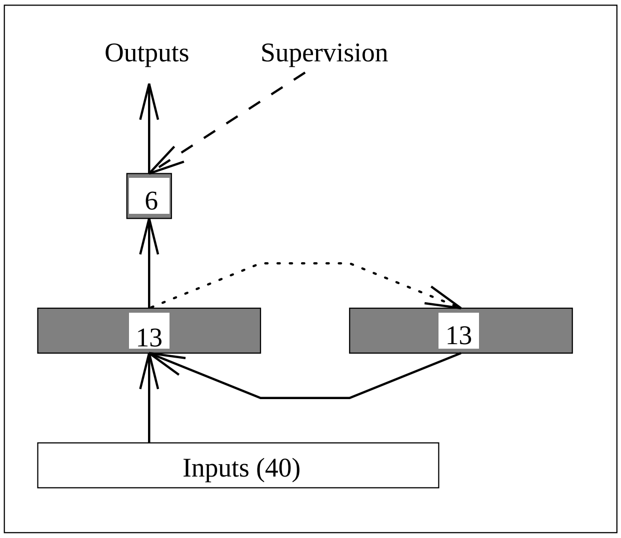

Figure 5: The Elman neural network architecture. A full line between two layers means that all units in one layer are fully connected to all units in the other layer. The dotted line means that the connection is fixed to 1 so there is no real recurrence in this network. Furthermore this is a one-to-one type of connection (which means this are only 13 connections and not 13 times 13 connections). The dashed line means external supervision is provided and the cost is minimized at that layer. The number of units in a layer is given in its center. In that case the outputs are $\{x_t, \cdots, x_{t+5}\}$ while the inputs are $\{x_{t-1}, \cdots, x_{t-40}\}$.

Table 1: Comparison of the different predictors. ARV means the Average Relative Variance and is computed as indicated previously in the paper, while SEP stands for Strong Error Percentage. It represents the percentage of test set examples for which the distance between expected value and obtained value is more than 30.

	CNET heuristic	Simple Net	Modular Net	Elman Net
ARV	0.1130	0.0884	0.0748	0.0737
SEP	5.0420	5.0420	1.6807	1.6807

We can see in table 1 the generalization results for all three networks, compared to the results given by the CNET prediction service. As we can see, all three networks were better than the heuristic, but both modular and Elman networks performed significantly better than the simple network. Fig. 6 shows the best performance we obtained compared to the actual 238 test set values.

The distribution of the errors should also be verified. It would be better

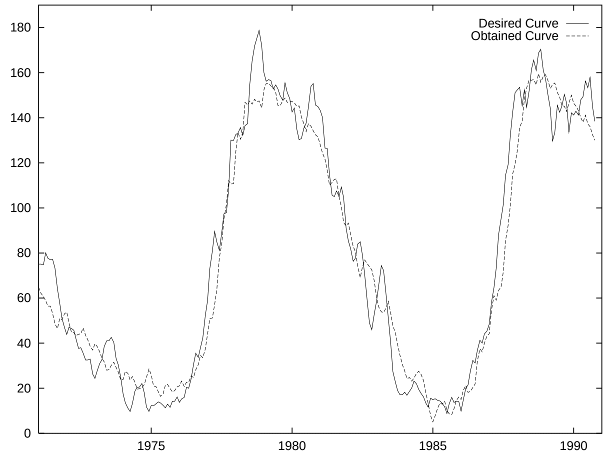

Figure 6: The desired curve corresponds to the 238 actual $IR5$ values used in the test set, while the obtained curve corresponds to the best performance obtained by a neural network.

to have a system that does many small errors than a system that does one big prediction error. This is why we also show in table 1 the percentage of strong errors obtained by the various predictors. We can see that once again, modular and Elman models performed better than the simple model and the heuristic. Fig. 6 shows that the network results strongly follows actual series.

5 Conclusion

We have seen in this paper the importance of ionospheric parameters accurate prediction systems for telecommunications. Neural networks can be used for this task and usually yield better results than classical methods but the particular neural network models for a given time series has to be carefully chosen in order to reach a good performance. Neural networks cannot be used as black boxes and a good understanding of underlying mechanisms is necessary to use them as prediction machines.

In this paper, we proposed two particular models, namely a modular model and a partially recurrent model, specially chosen for the $IR5$ time series. Many other models still need to be experimented, such as fully recurrent neural networks (Williams and Zipser, 1989), time-delay neural networks (Lang and Hinton, 1988), radial basis functions (Moody and Darken, 1989; Poggio and Girosi, 1990) and adaptive mixtures of experts (Jacobs et al., 1991). We will also look at these models in a perspective where we mix different input windows (embedding dimension).

References

Abarbanel, H., R. Brown, J. Sidorowich, and L. Tsimring. The analysis of observed chaotic data in physical systems. *Reviews of Modern Physics*, **65**:4, 1331–1392, 1993.

Bourdila, A. and R. Hanbaba. Long term ionospheric radiopropagation predictions choices of indices. In *Solar Terrestrial Predictions Proceedings of a Workshop at Meudon, France*, 491–499, 1984.

Box, G. and G. Jenkins. *Time Series Analysis, Forecasting and Control.* Holden-Day, San Francisco, 1970.

Canu, S., R. Sobral, and R. Lengelle. Formal neural network as an adaptative model for water demand. In *Proceedings of the International Neural Network Conference (INNC)*, 131–135, Paris, France, 1990.

Cottrell, G., P. Munro, and D. Zipser. Learning internal representations from gray-scale images: An example of extensional programming. In *Ninth Annual Conference of the Cognitive Science Society*, 462–473, Seattle. Lawrence Erlbaum, Hillsdale, 1987.

Cybenko, G. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signal and Systems*, **2**, 303–314, 1989.

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

Fahlman, S. and C. Lebiere. The Cascade-Correlation learning architecture. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems: Proceedings of the 1989 Conference*, 524–532. Morgan Kauffmann, 1990.

Funahashi, K. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, **2**, 183–192, 1989.

Hertz, J., A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation.* Santa fe Institute studies in the Sciences of Complexity. Addison Wesley, 1991.

Hornik, K., M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, 359–366, 1989.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, **3**, 79–87, 1991.

Koons, H. and D. Gorney. A sunspot maximum prediction using a neural network. *Transactions of American Geophysical Union*, **71**:18, 677–688, 1990.

Lang, K. J. and G. E. Hinton. The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University, Department of Computer Science, 1988.

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

Le Cun, Y., B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1**, 541–551, 1989.

Le Cun, Y., J. Denker, and S. Solla. Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems: Proceedings of the 1989 Conference*, 598–605. Morgan Kauffmann, 1990.

Lippmann, R. P. Review of neural networks for speech recognition. *Neural Computation*, **1**, 1–38, 1989.

Lundstedt, H. Neural networks and predictions of solar-terrestrial effects. *Planet Space Science*, **40**:4, 457–464, 1992.

Lundstedt, H. and P. Wintoft. Prediction of geomagnetic storms from solar wind data with the use of a neural network. *Annales Geophysicae*, **12**, 19–24, 1994.

Macpherson, K. Neural network computation techniques applied to solar activity prediction. *Advanced Space Research*, **13**:9, 447–450, 1993.

McNish, A. and J. Lincoln. Prediction of sunspot numbers. *Transactions of American Geophysical Union*, **30**:5, 673–685, 1949.

Moody, J. and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, **1**, 281–294, 1989.

Park, D. C., M. A. El-Sharkawi, and R. J. Marks II. Electric load forecasting using an artificial neural network. *IEEE Transaction on Power Systems*, **6**:2, 442–449, 1991.

Poggio, T. and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, **247**, 978–982, 1990.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Rumelhart, D. E. and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

Takens, F. Detecting strange attractors in turbulence. In Rand, D. A. and L.-S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, 366–381, Warwick 1980. Springer-Verlag, Berlin, 1981.

Tesauro, G. Practical issues in temporal difference learning. *Machine Learning*, **8**:34, 1992.

Vapnik, V. N. *Estimation of Dependencies Based on Empirical Data.* Springer-Verlag, New-York, NY, USA, 1982.

Varfis, A. and C. Versino. Univariate economic time series forecasting by connectionist methods. In *Proceedings of the International Neural Network Conference (INNC)*, 342–345, Paris, France, 1990.

Weigend, A. S., B. A. Huberman, and D. E. Rumelhart. Predicting sunspots and exchange rates with connectionist networks. In Casdagli, M. and S. Eubank, editors, *Nonlinear modeling and forecasting*, 395–431. Addison Wesley, 1992.

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