Time Series Prediction with the Self-Organizing Map: A Review

Guilherme A. Barreto

Department of Teleinformatics Engineering, Federal University of Ceará
Av. Mister Hull, S/N - C.P. 6005, CEP 60455-760, Center of Technology
Campus of Pici, Fortaleza, Ceará, Brazil
guilherme@deti.ufc.br

Summary. We provide a comprehensive and updated survey on applications of Kohonen's self-organizing map (SOM) to time series prediction (TSP). The main goal of the paper is to show that, despite being originally designed as an unsupervised learning algorithm, the SOM is flexible enough to give rise to a number of efficient supervised neural architectures devoted to TSP tasks. For each SOM-based architecture to be presented, we report its algorithm implementation in detail. Similarities and differences of such SOM-based TSP models with respect to standard linear and nonlinear TSP techniques are also highlighted. We conclude the paper with indications of possible directions for further research on this field.

Key words: Self-organizing map, time series prediction, local linear mappings, vector-quantized temporal associative memory, radial basis functions.

1 Introduction

Time series prediction (TSP) (or forecasting) is a function approximation task whose goal is to estimate future values of a sequence of observations based on current and past values of the sequence. This is a rather general definition, which makes no distinction between the nature of the time series data samples; that is, whether they are deterministic (chaotic) or stochastic, linear or not, stationary or not, continuous or discrete, among other characterizations. Anyway, TSP is a mature field, with well-established linear and nonlinear techniques and well-defined data transformation methods contributed from many scientific domains, such as statistics, economy, engineering, hydrology, physics, mathematics and computer science.

From the last fifteen years on, a growing interest in TSP methods has been observed, particularly within the field of neural networks [62, 44]. To some extent, this interest can be explained by the fact that, in its simplest form, the TSP problem can be easily cast as a supervised learning problem, in which the input vectors are formed by the current and past values of a certain

time series and the target values are the corresponding (one-step-ahead) future values. This fact has allowed users to explore the function approximation and generalization abilities of some well-known supervised neural architectures, such as the Multilayer Perceptron (MLP) and the Radial Basis Functions (RBF) networks. Not surprisingly, the vast majority of applications of neural network models to TSP tasks are based exactly on these two architectures and their variants.

The *Self-Organizing Map* (SOM) [31, 30, 29] is an important unsupervised neural architecture which, in contrast to the supervised ones, has been less applied to time series prediction. We believe that this occurs because TSP is usually understood as a function approximation problem, while the SOM is commonly viewed as a neural algorithm for data vector quantization (VQ), clustering and visualization [63, 17]. Despite this view, the use of the SOM as a stand-alone time series predictor is becoming more and more common in recent years, as we shall review in this paper.

In a global perspective, the main goal of the paper is then to show that, despite being originally designed as an unsupervised learning algorithm, the SOM is flexible enough to give rise to a number of efficient supervised neural architectures devoted to TSP tasks. To our knowledge, this paper is the first of its type, since former surveys on neural networks for time series prediction (see, for example, [44, 18, 64, 13]) only report just a few SOM-based applications (or even none at all!). In this regard, the contributions of this review are manifold, as we provide a comprehensive list of references, give detailed description of architectures, highlight relationships to standard linear and nonlinear TSP techniques and indicate possible directions for further research on this field. We do not follow a chronological order of presentation of the SOM-based models for TSP, but rather an order that favors understandability of the concepts involved[1].

The remainder of the paper is organized as follows. In Section 2 a brief description of the SOM algorithm is carried out. Then, in Section 3, we present the basic principles of time series prediction and discuss some reasons to using the SOM for this purpose. Several SOM-based models for TSP are presented from Section 4 to Section 8. In Section 9 possible directions for further research on the field are suggested. The paper is concluded in Section 10.

2 The Self-Organizing Map

The SOM is a well-known unsupervised neural learning algorithm. The SOM learns from examples a mapping from a high-dimensional continuous input space $\mathcal{X}$ onto a low-dimensional discrete space (lattice) $\mathcal{A}$ of q neurons which are arranged in fixed topological forms, e.g., as a rectangular

[1] Matlab codes of most SOM-based TSP models described in this paper can be downloaded from http://www.deti.ufc.br/∼guilherme/competitive.html.

2-dimensional array[2]. The map $i^*(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{A}$, defined by the weight matrix $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q), \mathbf{w}_i \in \mathbb{R}^p \subset \mathcal{X}$, assigns to the current input vector $\mathbf{x}(t) \in \mathbb{R}^p \subset \mathcal{X}$ a neuron index

$$i^*(t) = \arg\min_{\forall i} \|\mathbf{x}(t) - \mathbf{w}_i(t)\|, \tag{1}$$

where $\|\cdot\|$ denotes the euclidean distance and t is the discrete time step associated with the iterations of the algorithm.

The weight vectors, $\mathbf{w}_i(t)$, also called prototypes or codebook vectors, are trained according to a competitive-cooperative learning rule in which the weight vectors of a winning neuron and its neighbors in the output array are updated after the presentation of the input vector:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t)h(i^*, i; t)[\mathbf{x}(t) - \mathbf{w}_i(t)] \tag{2}$$

where $0 < \alpha(t) < 1$ is the learning rate and $h(i^*, i; t)$ is a weighting function which limits the neighborhood of the winning neuron. A usual choice for $h(i^*, i; t)$ is given by the Gaussian function:

$$h(i^*, i; t) = \exp\left(-\frac{\|\mathbf{r}_i(t) - \mathbf{r}_{i^*}(t)\|^2}{2\sigma^2(t)}\right) \tag{3}$$

where $\mathbf{r}_i(t)$ and $\mathbf{r}_{i^*}(t)$ are, respectively, the coordinates of the neurons i and i^* in the output array $\mathcal{A}$, and $\sigma(t) > 0$ defines the radius of the neighborhood function at time t. The variables $\alpha(t)$ and $\sigma(t)$ should both decay with time to guarantee convergence of the weight vectors to stable steady states. In this paper, we adopt for both an exponential decay, given by:

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_T}{\alpha_0}\right)^{(t/T)} \quad \text{and} \quad \sigma(t) = \sigma_0 \left(\frac{\sigma_T}{\sigma_0}\right)^{(t/T)} \tag{4}$$

where α_0 (σ_0) and α_T (σ_T) are the initial and final values of $\alpha(t)$ ($\sigma(t)$), respectively.

Weight adjustment is performed until a steady state of global ordering of the weight vectors has been achieved. In this case, we say that the map has converged. The resulting map also preserves the topology of the input samples in the sense that adjacent patterns are mapped into adjacent regions on the map. Due to this topology-preserving property, the SOM is able to cluster input information and spatial relationships of the data on the map. Despite its simplicity, the SOM algorithm has been applied to a variety of complex problems [32, 42] and has become one of the most important ANN architectures.

[2] The number of neurons (q) is defined after experimention with the dataset. However, a useful rule-of-thumb is to set it to the highest power of two that is less or equal to $\sqrt{N}$, where N is the number of input vectors available [46].

3 Time Series Prediction in a Nutshell

Simply put, a (univariate) time series is an ordered sequence $\{x(t)\}_{t=1}^{N}$ of N observations of a scalar variable x, $x \in \mathbb{R}$. Since the dynamics of the underlying process generating the sequence is generally unknown, the main goal of the time series analyst is testing a number of hypothesis concerning the model that best fits to the time series at hand. Once a time series model has been defined, the next step consists of estimating its parameters from the avaliable data. Finally, the last step involves the evaluation of the model, which is usually performed through an analysis of its predictive ability.

A general modeling hypothesis assumes that the dynamics of the time series can be described by the *nonlinear regressive model* of order p, NAR(p):

$$x(t+1) = f\left[x(t), x(t-1), \ldots, x(t-p+1)\right] + \varepsilon(t) \tag{5}$$

$$= f\left[\mathbf{x}^{-}(t)\right] + \varepsilon(t) \tag{6}$$

where $\mathbf{x}^{-}(t) \in \mathbb{R}^p$ is the input or regressor vector, $p \in \mathbb{Z}^+$ is the order of the regression, $\mathbf{f}(\cdot)$ is a nonlinear mapping of its arguments and ε is a random variable which is included to take into account modeling uncertainties and stochastic noise. For a stationary time series, it is commonly assumed that $\varepsilon(t)$ is drawn from a gaussian white noise process. We assume that the order of the regression p is optimally estimated by a suitable technique, such as AIC, BIC or Cao's method.

As mentioned before, the structure of the nonlinear mapping $\mathbf{f}(\cdot)$ is unknown, and the only thing that we have available is a set of observables $x(1), x(2), \ldots, x(N)$, where N is the total length of the time series. Our task is to construct a physical model for this time series, given only the data. This model is usually designed to provide an estimate of the next value of the time series, $\hat{x}(t+1)$, an approach called one-step-ahead prediction:

$$\hat{x}(t+1) = \hat{f}\left[\mathbf{x}^{-}(t)|\boldsymbol{\Theta}\right] \tag{7}$$

where $\boldsymbol{\theta}$ denotes the vector of adjustable parameters of the model. In nonlinear analysis of time series, Eq. (7) is commonly written in a slightly different way [27]:

$$x(t+1) = f\left[x(t), x(t-\tau), x(t-2\tau), \ldots, x(t-(p-1)\tau)|\boldsymbol{\Theta}\right] \tag{8}$$

where in this context the parameter p is usually called the embedding dimension and $\tau \in \mathbb{Z}^+$ is the embedding delay. Without loss of generality, in this paper we always assume $\tau = 1$. In this case, Eqs. (7) and (8) become equivalent.

The mapping $\hat{f}\left[\cdot|\boldsymbol{\Theta}\right]$ can be implemented through different techniques, such as linear models, polynomial models, neural network models, among others. For example, a linear version of Eq. (7), known as the autoregressive (AR) model [9], can be written as

$$\hat{x}(t+1) = \mathbf{a}^T \mathbf{x}^+(t) = a_0 + \sum_{l=1}^{p} a_l x(t-l+1), \tag{9}$$

where $\mathbf{a} = [a_0 \; a_1 \; \cdots \; a_p]^T$ is the coefficient vector of the model, with the superscript T denoting the transpose of a vector, and $\mathbf{x}^+(t) = [1 \;\; x(t) \;\; \cdots \;\; x(t-p+1)]^T$ is an augmented regressor vector at time step t. This is one of the simplest time series models, since it assumes that the next value of the time series is a linear combination of the past p values. More sophisticated models, composed of combinations of linear, polynomial and/or neural network models, are also very common in the literature.

Independently of the chosen model, its parameters need to be computed during an estimation (or training) stage using only a portion of the available time series, e.g. the first half of the sequence. Once trained, the adequacy of the hypothesized model to the training data is usually carried out through the analysis of the generated prediction errors (or *residuals*),

$$e(t+1) = x(t+1) - \hat{x}(t+1), \tag{10}$$

where $x(t+1)$ is the actually observed next value of the time series and $\hat{x}(t+1)$ is the predicted one. The model selection stage should be performed for different values of the parameter p. For each value of p, the model parameters should be estimated and the sequence of residuals computed. Usually, one choose the lowest value of p that results in an approximately gaussian and uncorrelated sequence of residuals.

Once the time series model has been selected, its predictive ability can be assessed through an *out-of-sample prediction* scheme, carried out using the sequence of residuals computed over the remaining portion of the time series. As reviewed by Hyndman and Koehler [26], there a number of measures of predictive accuracy of a model, but in this paper we use only the *Normalized Root Mean Squared Error* (NRMSE):

$$NRMSE = \sqrt{\frac{(1/M)\sum_{t=1}^{M}(x(t)-\hat{x}(t))^2}{(1/M)\sum_{t=1}^{M}(x(t)-\bar{x})^2}} = \sqrt{\frac{\sum_{t=1}^{M}(x(t)-\hat{x}(t))^2(t)}{M \cdot s_x^2}} \tag{11}$$

where $\bar{x}$ and s_x^2 are, respectively, the sample mean and sample variance of the testing time series, and M is the length of the sequence of residuals.

3.1 Why Should We Use the SOM for TSP?

This is the very first question we should try to answer before starting the description of SOM-based models for TSP. Since there are already available a huge number of models for this purpose, including neural network based ones, why do we need one more? Certainly, there is no definitive answers to this question, but among a multitude of reasons that could influence someone

to apply the SOM to TSP problems, including curiosity about or familiarity with the algorithm, one can surely take the following two for granted.

Local Nature of SOM-based Models - Roughly speaking, SOM-based models for time series prediction belong to the class of models performing local function approximation. By local we denote those set of models whose output is determined by mathematical operations acting on localized regions of the input space. A global model, such as MLP-based predictors, makes use of highly distributed representations of the input space that difficult the interpretability of the results. Thus, SOM-based TSP models allow the user to better understand the dynamics of the underlying process that generates the time series, at least within the localized region used to compute the current model's output. This localized nature of SOM-based model combined with its topology-preserving property are useful, for example, if one is interested in time series segmentation or time series visualization problems.

Simple Growing Architectures - As one could expect, clustering of the input space is a essential step in designing efficient SOM-based TSP models. A critical issue in the process of partitioning the input space for the purpose of time series prediction is to obtain an appropriate estimation (guess?) of the number of prototype vectors. Over- or under-estimation of this quantity can cause, respectively, the appearance of clusters with very few pattern vectors that are insufficient to building a local model for them and clusters containing patterns from regions with different dynamics.

One of the main advantages of the SOM and related architectures over standard supervised ones, such as MLP and RBF networks, is that there is no need to specify in advance the number of neurons (and hence, prototypes) in the model. This becomes possible thanks to a number of growing competitive learning architectures currently available (see [20] for a detailed review). Such growing architectures are much easier to implement than their supervised counterparts, such as the cascade-correlation architecture [15]. They are useful even if one has already initialized a given SOM with a fixed number of neurons. In this case, if necessary, new knowledge can be easily incorporated into the network in the form of a new prototype vector.

4 The VQTAM Model

A quite intuitive and straightforward way of using the SOM for time series prediction purposes is by performing vector quantization simultaneously on the regressor vectors $\mathbf{x}^-(t)$, $t = p, \ldots, N-1$, and their associated one-step-ahead observations $x(t+1)$. This is exactly the idea behind the *Vector-Quantized Temporal Associative Memory* (VQTAM) model [3, 5]. This model can be understood a generalization to the temporal domain of a SOM-based associative memory technique that has been widely to learn static (memoryless) input-output mappings, specially within the domain of robotics [4].

In a general formulation of the VQTAM model, the input vector at time step t, $\mathbf{x}(t)$, is composed of two parts. The first part, denoted $\mathbf{x}^{in}(t) \in \mathbb{R}^p$, carries data about the input of the dynamic mapping to be learned. The second part, denoted $\mathbf{x}^{out}(t) \in \mathbb{R}^m$, contains data concerning the desired output of this mapping. The weight vector of neuron i, $\mathbf{w}_i(t)$, has its dimension increased accordingly. These changes are formulated as follows:

$$\mathbf{x}(t) = \begin{pmatrix} \mathbf{x}^{out}(t) \\ \mathbf{x}^{in}(t) \end{pmatrix} \quad \text{and} \quad \mathbf{w}_i(t) = \begin{pmatrix} \mathbf{w}_i^{out}(t) \\ \mathbf{w}_i^{in}(t) \end{pmatrix} \tag{12}$$

where $\mathbf{w}_i^{in}(t) \in \mathbb{R}^p$ and $\mathbf{w}_i^{out}(t) \in \mathbb{R}^m$ are, respectively, the portions of the weight (prototype) vector which store information about the inputs and the outputs of the desired mapping.

For the univariate time series prediction tasks we are interested in, the following definitions apply:

$$\mathbf{x}^{out}(t) = x(t+1) \qquad \text{and} \qquad \mathbf{x}^{in}(t) = \mathbf{x}^{-}(t) \tag{13}$$

$$\mathbf{w}_i^{out}(t) = w_i^{out}(t) \qquad \text{and} \qquad \mathbf{w}_i^{in}(t) = \mathbf{w}_i^{-}(t) \tag{14}$$

where $\mathbf{w}_i^{-}(t) \in \mathbb{R}^p$ is the portion of the weight vector associated with the regressor vector $\mathbf{x}^{-}(t)$. Note that the vectors $\mathbf{x}^{out}(t)$ and $\mathbf{w}_i^{out}(t)$ in Eq. (12) are now reduced to the scalar quantities $x(t+1)$ and $w_i^{out}(t)$, respectively.

The winning neuron at time step t is determined based only on $\mathbf{x}^{in}(t)$:

$$i^*(t) = \arg\min_{i \in \mathcal{A}} \{\|\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t)\|\} = \arg\min_{i \in \mathcal{A}} \{\|\mathbf{x}^{-}(t) - \mathbf{w}_i^{-}(t)\|\} \tag{15}$$

However, for updating the weights, both $\mathbf{x}^{in}(t)$ and $x^{out}(t)$ are used:

$$\mathbf{w}_i^{in}(t+1) = \mathbf{w}_i^{in}(t) + \alpha(t)h(i^*, i; t)[\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t)] \tag{16}$$

$$w_i^{out}(t+1) = w_i^{out}(t) + \alpha(t)h(i^*, i; t)[x^{out}(t) - w_i^{out}(t)] \tag{17}$$

where $0 < \alpha(t) < 1$ is the learning rate and $h(i^*, i; t)$ is a time-varying gaussian neighborhood function as defined in Eq. (3). In words, Eq. (16) performs topology-preserving vector quantization on the input space and Eq. (17) acts similarly on the output space of the mapping being learned. As training proceeds, the SOM learns to associate the input prototype vectors $\mathbf{w}_i^{in}$ with the corresponding output prototype vectors w_i^{out} (see Figure 1).

During the out-of-sample prediction phase, the one-step-ahead estimate at time step t is then given by

$$\hat{x}(t+1) = w_{i^*}^{out}(t), \tag{18}$$

where the winning neuron at time step t is found according to Eq. (15), for each new incoming regressor vector $\mathbf{x}^{in}(t)$. Instead of using a single output weight $w_{i^*}^{out}(t)$, one can compute an improved one-step-ahead prediction as the average value of the output weights of the K ($K > 1$) closest prototypes.

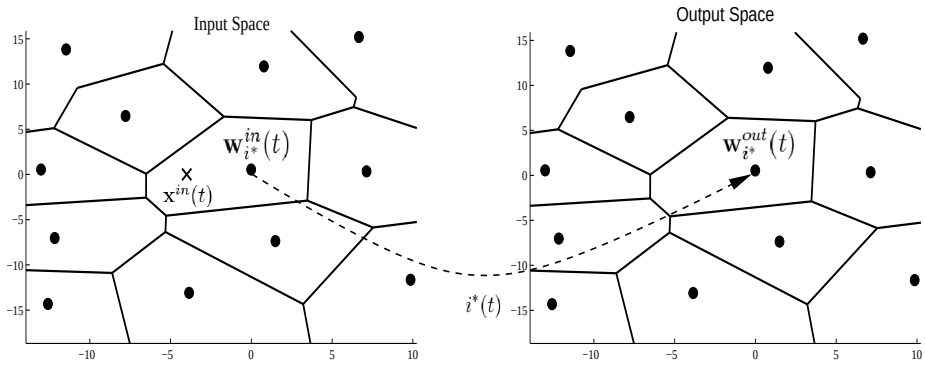


Fig. 1. Mapping between input and output Voronoi cells of a VQTAM model. The symbols '•' denote the prototype vectors of each input/output cell. The symbol '×' denotes an input regressor.

Related Architectures: VQTAM's philosophy is the same as that of the local model proposed in the seminal paper by Farmer and Sidorowich [16]. These authors used all the vectors $\mathbf{x}(t) = [x^{out}(t)\ \mathbf{x}^{in}(t)]$, $t = 1, \ldots, N-p$, to compute predictions for a given time series. The procedure is quite simple: find the closest vector to the current input vector on the basis of $\mathbf{x}^{in}(t)$ only and, then, uses the prediction associated to the closest vector as an approximation of the desired one. Roughly speaking, the VQTAM approach to TSP can then be understood as a VQ-based version of the Farmer-Sidorowich method.

Baumann and Germond [8] and Lendasse et al. [37] developed independently SOM-based architectures which are in essence equivalent to the VQTAM and successfully applied them to the prediction of electricity consumption. More recently, Lendasse et al. [36] used standard unsupervised competitive learning to implement a number of vector quantization strategies for time series prediction purposes. One of them consists in assigning different weights to the components of the regressor vector $\mathbf{x}^{in}(t)$ and to the target output value $x^{out}(t) = x(t+1)$. In this case, the vector $\mathbf{x}(t)$ is now written as

$$\mathbf{x}(t) = [x^{out}(t) \mid \mathbf{x}^{in}(t)]^T \tag{19}$$

$$= [kx(t+1) \mid a_1 x(t)\ a_2 x(t-1)\ \cdots\ a_p x(t-p+1)]^T, \tag{20}$$

where $k > 0$ is a constant weight determined through a cross-validation procedure, and a_j, $j = 1, \ldots, p$, are the coefficients of an AR(p) model (see Eq. (9)), previously fitted to the time series of interest. This simple strategy was able to improve the performance of VQ-based models in a benchmarking time series prediction task.

Despite its simplicity the VQTAM and variants can be used as a standalone time series predictor or as the building block for more sophisticated models as we see next.

4.1 Improving VQTAM Predictions Through Interpolation

Since the VQTAM is essentially a vector quantization algorithm, it may require too many neurons to provide small prediction errors when approximating continuous mappings. This limitation can be somewhat alleviated through the use of interpolation methods specially designed for the SOM architecture, such as geometric interpolation [22], topological interpolation [23] and the *Parameterized Self-Organizing Map* (PSOM) [60].

Smoother output values can also be obtained if we use the VQTAM model to design RBF-like networks. For instance, an RBF network with q gaussian basis functions and a single output neuron can be built directly from the learned prototypes $\mathbf{w}_i^{in}$ and w_i^{out}, without requiring additional training. In this case, the one-step-ahead prediction at time step t can be computed as follows:

$$\hat{x}(t+1) = \frac{\sum_{i=1}^{q} w_i^{out} G_i(\mathbf{x}^{in}(t))}{\sum_{i=1}^{q} G_i(\mathbf{x}^{in}(t))} \tag{21}$$

where $G_i(\mathbf{x}^{in}(t))$ is the response of this basis function to the current input vector $\mathbf{x}^{in}(t)$:

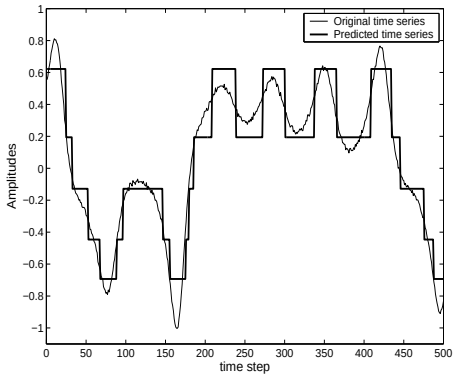
$$G_i(\mathbf{x}^{in}(t)) = \exp\left(-\frac{\|\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}\|^2}{2\gamma^2}\right) \tag{22}$$

where $\mathbf{w}_i^{in}$ plays the role of the center of the i-th basis function and $\gamma > 0$ defines its radius (or *spread*). This method has been successfully applied to time series prediction [36] and related tasks, such as system identification and adaptive filtering [7].

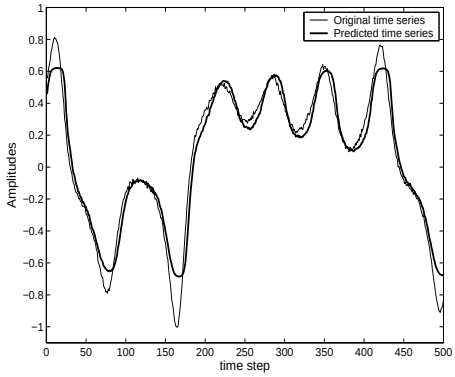
Results obtained by the application of the plain VQTAM and an VQTAM-based RBF model to an one-step-ahead prediction task are shown in Figure 2. Only the first 500 predictions are shown in this figure. Details about the generation of the time series data are given in Appendix 1. For this simulation, the plain VQTAM model has only 5 neurons. The VQTAM-based RBF model has also 5 neurons and a common value for the spread of the gaussian basis functions ($\gamma = 0.40$). As expected, the VQTAM has a poor prediction performance ($NRMSE = 0.288$), while the RBF-like interpolation method performs quite well for the same number of neurons ($NRMSE = 0.202$). Obviously, even for the VQTAM-based RBF model, the predictions are poor for high curvature points.

4.2 Rule Extraction from a Trained VQTAM Model

As mentioned in the beginning of this section, better interpretability is a clear-cut advantage of local models over global ones. In particular, interpretation of the results can be carried out by means of rule extraction procedures and prototype-based self-organizing algorithms are well-suited to this task [25]. For



(a)



(b)

Fig. 2. One-step-ahead prediction results for the (a) VQTAM model and (b) for a VQTAM-based RBF model.

example, through a trained VQTAM approach one can easily build interval-based rules that helps explaining the predicted values. This procedure can be summarized in a few steps as follows.

Step 1 - Present the set of training data vectors once again to the VQTAM model and find the neuron each vector $\mathbf{x}^{in}(t)$ is mapped to.

Step 2 - Let $\mathbf{x}_j^i = [x_{j,0}^i \quad x_{j,1}^i \quad \cdots \quad x_{j,p}^i]^T$ denote the j-th training data vector that has been mapped to neuron i. Then, find the subset $\mathcal{X}_i = \left\{\mathbf{x}_1^i, \mathbf{x}_2^i, \ldots, \mathbf{x}_{n_i}^i\right\}$ of n_i $(0 < n_i \ll N - p)$ training data vectors which are mapped to neuron i. Note that $\sum_{i=1}^{q} n_i = N - p$.

Table 1. Interval-based rules obtained for a VQTAM model with $q = 5$ neurons.

Neuron	$x(t+1)$	$x(t)$	$x(t-1)$	$x(t-2)$	$x(t-3)$	$x(t-4)$
$i = 1$	[-0.94,-0.38]	[-0.94,-0.44]	[-0.94,-0.50]	[-0.94,-0.54]	[-0.95,-0.52]	[-0.93,-0.49]
$i = 2$	[-0.67,-0.12]	[-0.63,-0.17]	[-0.60,-0.21]	[-0.58,-0.26]	[-0.61,-0.24]	[-0.67,-0.22]
$i = 3$	[-0.36,0.10]	[-0.36,0.09]	[-0.32,0.07]	[-0.29,0.05]	[-0.32,0.09]	[-0.37,0.16]
$i = 4$	[-0.14,0.55]	[-0.08,0.49]	[-0.03,0.47]	[0.03,0.43]	[0.02,0.48]	[0.00,0.58]
$i = 5$	[0.15,1.00]	[0.24,1.00]	[0.34,1.00]	[0.41,1.00]	[0.38,1.00]	[0.33,1.00]

Step 3 - Find the range of variation of the l-th component $x^i_{j,l}$ within $\mathcal{X}_i$:

$$\left[x^i_l\right]^- \leq x^i_{j,l} \leq \left[x^i_l\right]^+, \qquad j = 1, \ldots, n_i \tag{23}$$

where $\left[x^i_l\right]^- = \min_{\forall j}\{x^i_{j,l}\}$ and $\left[x^i_l\right]^+ = \max_{\forall j}\{x^i_{j,l}\}$.

Step 4 - Then, associate the following interval-based prediction rule R_i to neuron i:

$$R_i : \begin{cases} \text{IF} & \left[x^i_1\right]^- \leq x(t) \leq \left[x^i_1\right]^+ \text{and } \left[x^i_2\right]^- \leq x(t-1) \leq \left[x^i_2\right]^+ \text{and} \cdots \\ & \cdots \text{ and } \left[x^i_p\right]^- \leq x(t-p+1) \leq \left[x^i_p\right]^+, \\ \text{THEN} & \left[x^i_0\right]^- \leq x(t+1) \leq \left[x^i_0\right]^+ \end{cases} \tag{24}$$

Once all the rules are determined, a certain rule R_i is activated whenever $i^*(t) = i$, for a new incoming vector $\mathbf{x}^{in}(t)$. As an example, using the nonlinear time series data described in Appendix 1, the resulting five interval-based prediction rules obtained by a VQTAM model with $q = 5$ neurons are shown in Table 1. Figure 3 shows the time steps in which the fifth rule in Table 1 is activated during the out-of-sample (testing) prediction task.

It is worth emphasizing that fuzzy linguistic labels, such as very high, high, medium, small and very small, can be assigned by an expert to each component of the rule R_i to facilitate comprehension of the different clusters in the data. Such an approach has been already successfully tested in a number of real-world complex tasks, such as analysis of telecommunication networks [34], data mining [40, 14] and classification of multispectral satellite images [43].

5 The Double Vector Quantization Method

Simon et al. [53, 52] proposed the *Double Vector Quantization* (DVQ) method, specifically designed for long-term time series prediction tasks. The DVQ method has been successfully evaluated in a number of real-world time series, such as the far-infrared chaotic laser and electrical consumption data sets.

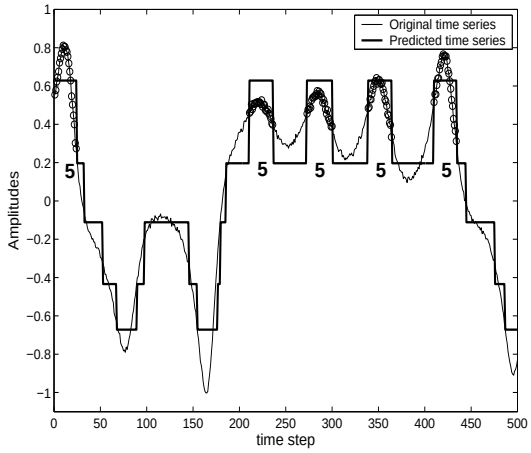


Fig. 3. Regions where the rule R_5 associated to neuron $i = 5$ is activated are highlighted with small open circles ('$\circ$').

Using VQTAM's notation, the DVQ method requires two SOM networks: one to cluster the regressors $\mathbf{x}^{in}(t)$, and a second one to cluster the associated *deformations* $\Delta\mathbf{x}^{in}(t) = \mathbf{x}^{in}(t+1) - \mathbf{x}^{in}(t)$. By definition, each deformation $\Delta\mathbf{x}^{in}(t)$ is associated to a single regressor $\mathbf{x}^{in}(t)$. The number of neurons in each SOM is not necessarily the same.

Double quantization of regressors and deformations only gives a static characterization of the past evolution of the time series. It is worth noting, however, that the associations between the deformations $\Delta\mathbf{x}^{in}(t)$ and their corresponding regressors $\mathbf{x}^{in}(t)$, can contain useful dynamical information about how the series has evolved between a regressor and the next one. These associations can be stochastically modelled by a transition probability matrix $P = [p_{ij}]$, $1 \leq i \leq q_1$ and $1 \leq j \leq q_2$, where q_1 and q_2 are the number of prototypes in the first and second SOMs, respectively.

Let $\mathbf{w}_i^{(1)}$, $i = 1, \ldots, q_1$, be the i-th prototype of the first SOM. Similarly, let $\mathbf{w}_j^{(2)}$, $j = 1, \ldots, q_2$, be the j-th prototype of the second SOM. Then, each element p_{ij} is estimated as

$$\hat{p}_{ij} = \frac{\#\{\mathbf{x}^{in}(t) \in \mathbf{w}_i^{(1)} \text{ and } \Delta\mathbf{x}^{in}(t) \in \mathbf{w}_j^{(2)}\}}{\#\{\mathbf{x}^{in}(t) \in \mathbf{w}_i^{(1)}\}}, \tag{25}$$

where the symbol # denotes the cardinality of a set. In fact, $\hat{p}_{ij}$ is an estimate of the conditional probability that the deformation $\Delta\mathbf{x}^{in}(t)$ is mapped to the prototype $\mathbf{w}_i^{(2)}$ given that the regressor $\mathbf{x}^{in}(t)$ is already mapped to $\mathbf{w}_i^{(1)}$.

Once the two SOMs are trained and the transition matrix $\hat{P} = [\hat{p}_{ij}]$ is estimated, one-step-ahead predictions using the DVQ model can be computed as summarized in the steps.

1. **Step 1** - Find the closest prototype $\mathbf{w}_{i^*}^{(1)}(t)$ to the current regressor vector $\mathbf{x}^{in}(t)$ using Eq. (15).
2. **Step 2** - Choose at random a deformation prototype $\mathbf{w}_{j^*}^{(2)}(t)$ from the set $\{\mathbf{w}_j^{(2)}\}$, $1 \leq j \leq q_2$, according to the conditional probability distribution defined by row $i^*(t)$ of the transition matrix, i.e. $P[i^*(t), 1 \leq j \leq q_2]$.
3. **Step 3** - Compute an estimate of the next regressor vector as follows:

$$\hat{\mathbf{x}}^{in}(t+1) = \mathbf{x}^{in}(t) + \mathbf{w}_{j^*}^{(2)}(t). \tag{26}$$

4. **Step 4** - Extract the one-step-ahead prediction from $\hat{\mathbf{x}}^{in}(t+1)$, i.e.

$$\hat{x}(t+1) = \hat{x}_1^{in}(t+1), \tag{27}$$

where $\hat{x}_1^{in}(t+1)$ is the first component of $\hat{\mathbf{x}}^{in}(t+1)$.

To estimate $h > 1$ values into the future, the Steps 1-4 are then iterated, inserting each new one-step-ahead estimate $\hat{x}(t+1)$ into the regressor vector of the next iteration of the algorithm. Note that, for each time iteration t, the Steps 1-4 need to be repeated using a Monte-Carlo procedure, since the random choice of deformation according to the conditional probability distributions given by the rows of the transition matrix is stochastic.

6 Local AR Models from Clusters of Data Vectors

Lehtokangas et al. [35] and Vesanto [58] independently proposed a SOM-based model that associates a local linear AR(p) model to each neuron i. In this approach, there is a coefficient vector $\mathbf{a}_i \in \mathbb{R}^p$, defined as

$$\mathbf{a}_i = [a_{i,0} \;\; a_{i,1} \;\; \cdots \;\; a_{i,p}]^T, \tag{28}$$

associated to each prototype vector $\mathbf{w}_i$. Thus, the total number of adjustable parameters of this model is $2pq$ (i.e., q units of p-dimensional weight vectors plus q units of p-dimensional coefficient vectors).

The procedure for building local AR models requires basically four steps. The first two steps are equivalent to training the VQTAM model, being repeated here just for the sake of convenience.

1. **Data preparation** - From a training time series of length N, one can build $N - p$ input vectors $\mathbf{x}(t) = [x(t+1) \;\; \mathbf{x}^-(t)]^T$, $t = p, \ldots, N-1$, by sliding a time window of width $p+1$ over the time series.
2. **SOM training** - The training data vectors are presented to the SOM for a certain number of epochs. The order of presentation of the data vectors can be randomly changed at each training epoch. The winning neuron at time step t is found using $\mathbf{x}^-(t)$ only, as in Eq. (15). For weight updating, however, one should use the entire vector $\mathbf{x}(t)$.

3. **Data partitioning** - Once training is completed, the set of training data vectors is presented once again to the SOM in order to find the neuron each data vector is mapped to. Let $\mathcal{X}_i = \left\{\mathbf{x}_1^i, \mathbf{x}_2^i, \ldots, \mathbf{x}_{n_i}^i\right\}$ denote the subset of n_i ($0 < n_i \ll N - p$) training data vectors which are mapped to neuron i, with $\mathbf{x}_j^i = [x_{j,0}^i \;\; x_{j,1}^i \;\; \cdots \;\; x_{j,p}^i]^T$ representing the j-th training data vector that has been mapped to neuron i.
4. **Parameter estimation** - The coefficient vector $\mathbf{a}_i$ of the i-th AR local model is computed through the pseudoinverse (least squares) technique using the data vectors in the set $\mathcal{X}_i$:

$$\mathbf{a}_i = \left(\mathbf{R}_i^T\mathbf{R}_i + \lambda\mathbf{I}\right)^{-1}\mathbf{R}_i^T\mathbf{p}_i \tag{29}$$

where $0 < \lambda \ll 1$ is a small constant, $\mathbf{I}$ is an identity matrix of dimension $p+1$, and the vector $\mathbf{p}_i$ and the matrix $\mathbf{R}_i$ are defined, respectively, as

$$\mathbf{p}_i = \begin{pmatrix} x_{1,0}^i \\ x_{2,0}^i \\ \vdots \\ x_{n_i,0}^i \end{pmatrix} \quad \text{and} \quad \mathbf{R}_i = \begin{pmatrix} 1 & x_{1,1}^i & x_{1,2}^i & \cdots & x_{1,p}^i \\ 1 & x_{2,1}^i & x_{2,2}^i & \cdots & x_{2,p}^i \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n_i,1}^i & x_{n_i,2}^i(t) & \cdots & x_{n_i,p}^i \end{pmatrix}. \tag{30}$$

Numerical problems may arise when implementing Eq. (29), which can be caused by any of the following situations:

1. No data vector is assigned to neuron i at all, i.e. the cardinality of the corresponding set $\mathcal{X}_i$ is zero. In this case, the best thing to do is to eliminate this neuron from the set of neurons allowed to build the local models.
2. Some data vectors are indeed assigned to neuron i, but the cardinality of $\mathcal{X}_i$ is much less than p. In this case, it is recommended to join the subset $\mathcal{X}_i$ of this neuron with that of its closest neighbor.

Once all the (q) local AR models have been determined, one-step-ahead predictions can be computed for each new incoming vector $\mathbf{x}^+(t)$ as follows:

$$\hat{x}(t+1) = \mathbf{a}_{i^*}(t)\mathbf{x}^+(t) = a_{i^*,0}(t) + \sum_{l=1}^{p} a_{i^*,l}(t)x(t-l+1) \tag{31}$$

where $\mathbf{a}_{i^*}(t)$ is the coefficient vector of the current winning neuron. It is worth noting that if the number of neurons is set to one ($q = 1$), the network reduces to an ordinary AR(p) model.

Related Architectures: Koskela et al. [33] introduced a SOM-based approach that uses a temporal variant of the SOM, called the *Recursive SOM* (SOM), to cluster the training data vectors. They argue that this type of dynamic SOM algorithm can better capture the temporal dependencies of consecutive samples. The approach introduced by Sfetsos and Siriopoulos [51] uses the well-known K-means clustering algorithm to partition data vectors for building local AR models. Poliker and Geva [47] used a fuzzy clustering

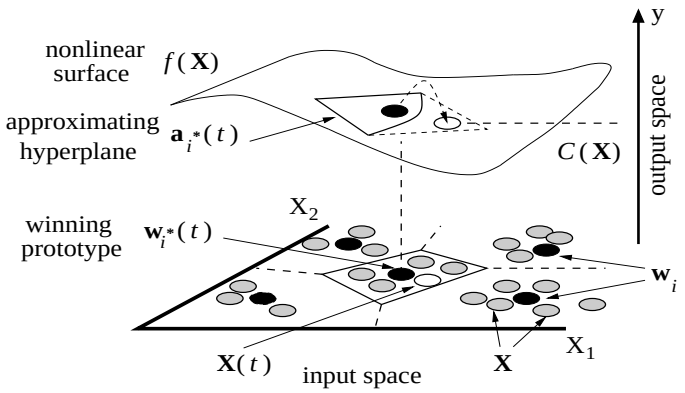


Fig. 4. Sketch of the local modeling approach implemented by the LLM architecture.

algorithm for the same purpose. In [50], data partitioning is performed with the SOM, while local TSP models are built using functional networks. More recently, Pavlidis et al. [45] introduced a local nonlinear approach and applied it successfully to financial time series prediction tasks. They used the *Growing Neural Gas* algorithm [20] for time series clustering purposes and a feedforward supervised neural network (e.g. MLP) to build local nonlinear TSP models.

It is worth noting that each of the methods above uses a different clustering algorithm to partition the data. How sensitive are the local linear/nonlinear AR models with respect to the choice of the clustering algorithm? This is an open question. We return to this issue in Section 9.

7 Online Learning of Local Linear Models

One of the first applications of the SOM to time series prediction was proposed by Walter et al. [61] using the *Local Linear Map* (LLM) model. The underlying idea is the same of that presented in Section 6, i.e. to associate to each neuron a linear AR model. Thus, the total number of parameters of the LLM model is also $2pq$. Once trained, the one-step-ahead prediction is provided by the local AR model associated with the current winning neuron as defined by Eq. (31) (see Figure 4).

Unlike the model described in Section 6, the LLM model updates the coefficient vectors of the local models simultaneously with the clustering of the data vectors, thus reducing the time devoted to the model building process. Another important advantage is a drastic reduction of the computational efforts, since there is no need to compute the (pseudo)inverses of q matrices $\mathbf{R}_i$, $i = 1, \ldots, q$, as shown in Eq. (29).

For the LLM model, the input vector at time step t is defined as $\mathbf{x}^{in}(t) = \mathbf{x}^{-}(t)$. Hence, the weight vector of neuron i, $i = 1, \ldots, q$, is defined as $\mathbf{w}_i^{in}(t) = \mathbf{w}_i^{-}(t)$. Then, weight updating rule follows exactly the one given in Eq. (16). The learning rule of the coefficient vectors $\mathbf{a}_i(t)$ is an extension of the well-known LMS rule that takes into account the influence of the neighborhood function $h(i^*, i; t)$:

$$\begin{aligned} \mathbf{a}_i(t+1) &= \mathbf{a}_i(t) + \alpha h(i^*, i; t) e_i(t) \mathbf{x}^{+}(t) \\ &= \mathbf{a}_i(t) + \alpha h(i^*, i; t)[x(t+1) - \mathbf{a}_i^T(t)\mathbf{x}^{+}(t)]\mathbf{x}^{+}(t) \end{aligned} \tag{32}$$

where $e_i(t)$ is the prediction error of the i-th local model and $0 < \alpha < 1$ is the learning rate of the coefficient vectors. Once trained, one-step-ahead predictions are computed according to Eq. (31).

Related Architectures: Stokbro et al. [54], much the same way as implemented by the LLM architecture, associated a linear AR filter to each hidden unit of an RBF network, applying it to prediction of chaotic time series. The main difference between this approach and the LLM architecture is that the former allows the activation of more than one hidden unit, while the latter allows only the winning neuron to be activated. In principle, by combining the output of several local linear models instead of a single one, one can improve generalization performance. Martinetz et al. [41] introduced a variant of the LLM that is based on the Neural Gas algorithm and applied it successfully to TSP. It is important to point out the existence of a growing LLM algorithm introduced by Fritzke [19], which can be easily applied to TSP using the formulation just described.

It is worth pointing out the LLM model bears strong resemblance to the widely-known *Threshold AR* (TAR) model [55]. This is a nonlinear time-series model comprised of a number of linear AR sub-models (local linear AR models, in our jargon). Each sub-model is constructed for a specific amplitude range, which is classified by amplitude thresholds and a time-delay amplitude. In this basic formulation, the switching betwen sub-AR models is governed by a *scalar quantization* process built over the amplitudes of the observations. In this sense, the LLM can be understood as a VQ-based implementation of the TAR model.

8 Time-Varying Local AR Models from Prototypes

The last SOM-based model for time series prediction we describe was introduced by Barreto et al. [6], being called the $KSOM$ model. The KSOM model combines the vector quantization approach of the VQTAM model with that of building local linear AR models.

However, instead of q time-invariant local linear models, the KSOM works with a single time-variant linear AR model whose coefficient vector is recomputed at every time step t, directly from a subset of K ($K \ll q$)

weight vectors extracted from a trained VQTAM model. This subset contains the weight vectors of the current K first winning neurons, denoted by $\{i_1^*(t), i_2^*(t), \ldots, i_K^*(t)\}$:

$$\begin{aligned} i_1^*(t) &= \arg\min_{\forall i} \left\{ \|\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t)\| \right\} \\ i_2^*(t) &= \arg\min_{\forall i \neq i_1^*} \left\{ \|\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t)\| \right\} \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ i_K^*(t) &= \arg\min_{\forall i \neq \{i_1^*, \ldots, i_{K-1}^*\}} \left\{ \|\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t)\| \right\} \end{aligned} \tag{33}$$

The out-of-sample prediction at time step t is then computed as

$$\hat{x}(t+1) = \mathbf{a}^T(t)\mathbf{x}^+(t), \tag{34}$$

where the current coefficient vector $\mathbf{a}(t)$ is computed by means of the pseudoinverse method as:

$$\mathbf{a}(t) = \left(\mathbf{R}^T(t)\mathbf{R}(t) + \lambda \mathbf{I}\right)^{-1} \mathbf{R}^T(t)\mathbf{p}(t), \tag{35}$$

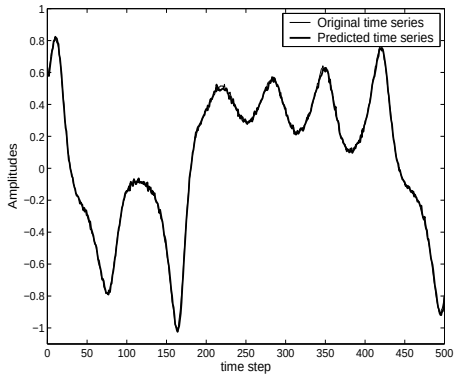
where $\mathbf{I}$ is a identity matrix of order p and $\lambda > 0$ (e.g. $\lambda = 0.01$) is a small constant added to the diagonal of $\mathbf{R}^T(t)\mathbf{R}(t)$ to make sure that this matrix is full rank. The prediction vector $\mathbf{p}(t)$ and the regression matrix $\mathbf{R}(t)$ at time t are defined as

$$\mathbf{p}(t) = [w_{i_1^*,1}^{out}(t) \;\; w_{i_2^*,1}^{out}(t) \;\; \cdots \;\; w_{i_K^*,1}^{out}(t)]^T, \tag{36}$$

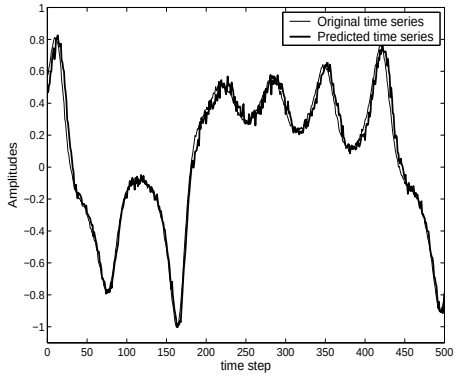
$$\mathbf{R}(t) = \begin{pmatrix} w_{i_1^*,1}^{in}(t) & w_{i_1^*,2}^{in}(t) & \cdots & w_{i_1^*,p}^{in}(t) \\ w_{i_2^*,1}^{in}(t) & w_{i_2^*,2}^{in}(t) & \cdots & w_{i_2^*,p}^{in}(t) \\ \vdots & \vdots & \vdots & \vdots \\ w_{i_K^*,1}^{in}(t) & w_{i_K^*,2}^{in}(t) & \cdots & w_{i_K^*,p}^{in}(t) \end{pmatrix}. \tag{37}$$

It is worth emphasizing that the KSOM model should be used when the number of neurons is much larger than K (e.g. $q > 2K$). If this is not the case, the choice of another SOM-based model, such as the one described in Section 6 or the LLM model, is highly recommended. As an example, we compare the performance of the LLM and KSOM models using only $q = 5$ neurons. The results are shown in Figure 5. As expected, the one-step-ahead prediction performance of the LLM model ($NRMSE = 0.039$) is much better than that of the KSOM model ($NRMSE = 0.143$, $K = 4$), since the number of neurons is too small. It is easy to see that the results of the LLM model present also less variability.

Related Architecture: Principe and Wang [49] proposed a neural architecture for nonlinear time series prediction which is equivalent to KSOM in the sense that a time-varying coefficient vector $\mathbf{a}(t)$ is computed from K prototype vectors of a trained SOM using the pseudoinverse method. However,



(a)



(b)

Fig. 5. One-step-ahead prediction results for the (a) LLM model and (b) for the KSOM model.

the required prototype vectors are not selected as the K nearest prototypes to the current input vector, but rather automatically selected as the winning prototype at time t and its $K-1$ topological neighbors. If a perfect topology preservation is achieved during SOM training, the neurons in the topological neighborhood of the winner are also the closest ones to the current input vector. However, if an exact topology preserving map is not achieved, as usually occurs for multidimensional data, the KSOM provides more accurate results.

9 Directions for Further Work

The SOM-based TSP models described in this paper have shown, among other things, that the SOM is a highly flexible neural architecture, in the sense that it can be successfully applied to unsupervised learning tasks (e.g. data clustering and visualization), as well as to typical supervised learning tasks (e.g. function approximation).

Despite the relatively large number of TSP models developed based on the SOM, this field is still in its first infancy if compared to those based on the MLP or the RBF networks. In the remaining part of this paper we list a number of issues that still need to addressed carefully in the future to allow SOM-based applications to time series prediction to establish themselves as a rich and mature field of research.

Time Series Clustering - Clustering of the input vectors is an important step in designing SOM-based models for time series prediction. A recent survey by Liao [38] only reports a number of methods to cluster time series data, but fails in providing hints about which one should be preferred for handling a certain time series of interest (I wonder if it is really possible!).

Currently, most of SOM-based TSP models use conventional clustering methods for static (i.e. non-temporal) data (see [63] for a review of these methods). However, it is still not clear how useful temporal information present in time series data can be extracted using such methods[3]. More recently, several temporal vector quantization algorithms have been proposed [10, 33, 2, 48, 59, 1], specially within the field of neural networks, that aim at performing better than static clustering methods when dealing with time series data.

Independently of the type of clustering methods, if static of temporal ones, much work still needs to be developed in this regard, since there is no comprehensive empirical or theoretical study evaluating how sensitive is a certain local time series model to the vector quantization algorithm used to cluster the data.

Multi-Step-Ahead Time Series Prediction - The vast majority of the works described in this paper were concerned in computing one-step-ahead predictions. However, the computation of predictions for larger time horizons is much more challenging. Just a few papers have explored this application using SOM-based TSP models [49, 52, 3, 53], revealing that there is still room for applications and models.

Two approaches have been used for multi-step-ahead TSP using SOM-based models. In the first one, called *recursive prediction* [49], every new one-step-ahead estimate $\hat{x}(t+1)$ is fed back to the input regressor until the desired prediction horizon is reached. In the second one, called *vector forecasting* [52, 3, 53], a whole bunch of future values within a desired horizon is predicted at once. In this case, the one-step-ahead prediction model in Eq. (5) is generalized

[3] There is even a recent controversy on this issue. See [28] and [11] for more detail.

and written as

$$[x(t+1), x(t+2), \ldots, x(t+h)] = f[\mathbf{x}^{-}(t)] + \varepsilon(t), \tag{38}$$

where $h > 1$ is the desired prediction horizon. To use the VQTAM model for vector forecasting purposes is straightforward as long as we apply the following definitions:

$$\mathbf{x}^{out}(t) = [x(t+1), x(t+2), \ldots, x(t+h)] \tag{39}$$

$$\mathbf{x}^{in}(t) = [x(t), x(t-1), \ldots, x(t-p+1)] \tag{40}$$

Time Series Visualization and Knowledge Discovery - The SOM is particularly well-known in the field of data mining as a nonlinear projection and visualization tool. Thus, an interesting field of application would be temporal knowledge discovery, where the SOM can be used to capture regular behaviors within time series data. There are already some papers on this topic [57, 21, 12, 56, 39], but much more still can be done, specially for the analysis of multivariate time series [24].

Prediction of Multivariate Time Series - A challenging task arises when one is interested in predicting two or more variables at the same time. While gathering the bibliography for this survey, we were unable to find applications of SOM-based models for multivariate time series prediction. Hence, this seems to be an entirely open research field, starving for new models and applications.

10 Conclusion

In this paper we reviewed several applications of Kohonen's SOM-based models to time series prediction. Our main goal was to show that the SOM can perform efficiently in this task and can compete equally with well-known neural architectures, such as MLP and RBF networks, which are more commonly used. In this sense, the main advantages of SOM-based models over MLP- or RBF-based models are the inherent local modeling property, which favors the interpretability of the results, and the facility in developing growing architectures, which alleviates the burden of specifying an adequate number of neurons (prototype vectors).

Acknowledgment

The authors would like to thank CNPq (grant #506979/2004-0) and CAPES (PRODOC grant) for their financial support.

APPENDIX

The time series data set $\{x(t)\}_{t=1}^{N}$ used in this paper is generated by numerical integration of the Lorenz dynamical system of equations:

$$\begin{aligned}\frac{dx}{dt} &= a[y(t) - x(t)] \\ \frac{dy}{dt} &= bx(t) - y(t) - x(t)z(t) \\ \frac{dz}{dt} &= x(t)y(t) - cz(t)\end{aligned} \tag{41}$$

where a, b and c are constants. Using the Euler forward method, a discrete time version of the Lorenz system is given by

$$\begin{aligned}x(t+1) &= x(t) + ha[y(t) - x(t)] \\ y(t+1) &= y(t) + h[bx(t) - y(t) - x(t)z(t)] \\ z(t+1) &= z(t) + h[x(t)y(t) - cz(t)]\end{aligned} \tag{42}$$

where $h > 0$ is the step size.

A chaotic time series with $N = 5000$ sample points was generated using $h = 0.01$, $a = 10$, $b = 28$ and $c = 8/3$. The time series was then rescaled to the range $[-1, +1]$. Finally, white gaussian noise, with zero mean and standard-deviation $\sigma_\varepsilon = 0.01$, is added to each sample point, i.e.

$$\tilde{x}(t) = x(t) + \varepsilon(t), \qquad \varepsilon(t) \sim N(0, \sigma_\varepsilon^2). \tag{43}$$

The first 4000 sample points $\{\tilde{x}(t)\}_{t=1}^{4000}$ are used for training the SOM-based TSP models, while the remaining 1000 points are used for performance evaluation purposes.

For all the simulations performed in this paper we use a unidimensional SOM architecture. The order of regression is always $p = 5$. For a given number of neurons (q), we train a given SOM-based model for $10 \times q$ epochs. Initial and final neighborhood widths are set to $\sigma_0 = q/2$ and $\sigma_T = 10^{-3}$, respectively. Initial and final values of the learning rate are set to $\alpha_0 = 0.5$ and $\sigma_T = 10^{-3}$, respectively. Weights are randomly initialized.

References

1. V. Baier. Motion perception with recurrent self-organizing maps based models. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN'05)*, pages 1182–1186, 2005.
2. G. A. Barreto and A. F. R. Araújo. Time in self-organizing maps: An overview of models. *International Journal of Computer Research*, 10(2):139–179, 2001.
3. G. A. Barreto and A. F. R. Araújo. Identification and control of dynamical systems using the self-organizing map. *IEEE Transactions on Neural Networks*, 15(5):1244–1259, 2004.

4. G. A. Barreto, A. F. R. Araújo, and H. J. Ritter. Self-organizing feature maps for modeling and control of robotic manipulators. *Journal of Intelligent and Robotic Systems*, 36(4):407–450, 2003.
5. G.A. Barreto and A.F.R. Araújo. A self-organizing NARX network and its application to prediction of chaotic time series. In *Proceedings of the IEEE-INNS Int. Joint Conf. on Neural Networks, (IJCNN'01)*, volume 3, pages 2144–2149, Washington, D.C., 2001.
6. G.A. Barreto, J.C.M. Mota, L.G.M. Souza, and R.A. Frota. Nonstationary time series prediction using local models based on competitive neural networks. *Lecture Notes in Computer Science*, 3029:1146–1155, 2004.
7. G.A. Barreto and L.G.M. Souza. Adaptive filtering with the self-organizing map: a performance comparison. *Neural Networks*, 19(6-7):785–798, 2006.
8. T. Baumann and A. J. Germond. Application of the Kohonen network to short-term load forecasting. In *Proceedings of the Second International Forum on Applications of Neural Networks to Power Systems (ANNPS'93)*, pages 407–412, 1993.
9. G. Box, G. M. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting & Control*. Prentice-Hall, 3rd edition, 1994.
10. G. J. Chappell and J. G. Taylor. The temporal Kohonen map. *Neural Networks*, 6(3):441–445, 1993.
11. J.R. Chen. Making subsequence time series clustering meaningful. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 1–8, 2005.
12. G. Chicco, R. Napoli, and F. Piglione. Load pattern clustering for short-term load forecasting of anomalous days. In *Proceedings of the 2001 IEEE Porto Power Tech Conference (PPT'01)*, pages 1–6, 2001.
13. G. Dorffner. Neural networks for time series processing. *Neural Network World*, 6(4):447–468, 1996.
14. M. Drobics, U. Bodenhofer, and W. Winiwarter. Mining clusters and corresponding interpretable descriptions - a three-stage approach. *Expert Systems*, 19(4):224–234, 2002.
15. S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, 1991.
16. J.D. Farmer and J.J. Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59(8):845–848, 1987.
17. A. Flexer. On the use of self-organizing maps for clustering and visualization. *Intelligent Data Analysis*, 5(5):373–384, 2001.
18. R.J. Frank, N. Davey, and S.P. Hunt. Time series prediction and neural networks. *Journal of Intelligent and Robotic Systems*, 31(1-3):91–103, 2001.
19. B. Fritzke. Incremental learning of local linear mappings. In *Proceedings of the International Conference On Artificial Neural Networks (ICANN'95)*, pages 217–222, 1995.
20. B. Fritzke. Unsupervised ontogenetic networks. In R. Beale and E. Fiesler, editors, *Handbook of Neural Computation*, pages 1–16. IOP Publishing/Oxford University Press, 1996.
21. T. C. Fu, F.L. Chung, V. Ng, and R. Luk. Pattern discovery from stock time series using self-organizing maps. In *Workshop Notes of KDD'2001 Workshop on Temporal Data*, pages 27–37, 2001.

22. J. Göppert and W. Rosenstiel. Topology preserving interpolation in self-organizing maps. In *Proceedings of the NeuroNIMES'93*, pages 425–434, 1993.
23. J. Göppert and W. Rosenstiel. Topological interpolation in SOM by affine transformations. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'95)*, pages 15–20, 1995.
24. G. Guimarães. Temporal knowledge discovery for multivariate time series with enhanced self-organizing maps. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, pages 165–170, 2000.
25. B. Hammer, A. Rechtien, M. Strickert, and T. Villmann. Rule extraction from self-organizing networks. *Lecture Notes in Computer Science*, 2415:877–883, 2002.
26. R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.
27. H. Kantz and T. Schreiber. *Nonlinear Time Series Analysis*. Cambridge University Press, 1999.
28. E. Keogh, J. Lin, and W. Truppel. Clustering of time series subsequences is meaningless: implications for previous and future research. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 115–122, 2003.
29. T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.
30. T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, Heidelberg, 2nd extended edition, 1997.
31. T. Kohonen. The self-organizing map. *Neurocomputing*, 21:1–6, 1998.
32. T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas. Engineering applications of the self-organizing map. *Proceedings of the IEEE*, 84(10):1358–1384, 1996.
33. T. Koskela, M. Varsta, J. Heikkonen, and S. Kaski. Time series prediction using recurrent SOM with local linear models. *International Journal of Knowledge-based Intelligent Engineering Systems*, 2(1):60–68, 1998.
34. P. Lehtimäki, K. Raivio, and O. Simula. Self-organizing operator maps in complex system analysis. *Lecture Notes in Computer Science*, 2714:622–629, 2003.
35. M. Lehtokangas, J. Saarinen, K. Kaski, and P. Huuhtanen. A network of autoregressive processing units for time series modeling. *Applied Mathematics and Computation*, 75(2-3):151–165, 1996.
36. A. Lendasse, D. Francois, V. Wertz, and M. Verleysen. Vector quantization: a weighted version for time-series forecasting. *Future Generation Computer Systems*, 21(7):1056–1067, 2005.
37. A. Lendasse, J. Lee, V. Wertz, and M. Verleysen. Forecasting electricity consumption using nonlinear projection and self-organizing maps. *Neurocomputing*, 48(1-4):299–311, 2002.
38. T. W. Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.
39. J. Lin, E. Keogh, and S. Lonardi. Visualizing and discovering non-trivial patterns in large time series databases. *Information Visualization*, 4(3):61–82, 2005.
40. J. Malone, K. McGarry, S. Wermter, and C. Bowerman. Data mining using rule extraction from Kohonen self-organising maps. *Neural Computing & Applications*, 15(1):9–17, 2005.

41. T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. Neural-gas network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993.
42. M. Oja, S. Kaski, and T. Kohonen. Bibliography of self-organizing map (SOM) papers: 1998-2001 addendum. *Neural Computing Surveys*, 3:1–156, 2003.
43. N. R. Pal, A. Laha, and J. Das. Designing fuzzy rule based classifier using self-organizing feature map for analysis of multispectral satellite images. *International Journal of Remote Sensing*, 26(10):2219–2240, 2005.
44. A.K. Palit and D. Popovic. *Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications*. Springer, 1st edition, 2005.
45. N. G. Pavlidis, D.K. Tasoulis, V.P. Plagianakos, and M.N. Vrahatis. Computational intelligence methods for financial time series modeling. *International Journal of Bifurcation and Chaos*, 16(7):2053–2062, 2006.
46. C. E. Pedreira and R.T. Peres. Preliminary results on noise detection and data selection for vector quantization. In *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI'06)*, pages 3617–3621, 2006.
47. S. Poliker and A.B. Geva. Hierarchical-fuzzy clustering of temporal-patterns and its application for time-series prediction. *Pattern Recognition Letters*, 20(14):1519–1532, 1999.
48. J. Principe, N. Euliano, and S. Garani. Principles and networks for self-organization in space-time. *Neural Networks*, 15(8-9):1069–1083, 2002.
49. J. C. Principe and L. Wang. Non-linear time series modeling with self-organizing feature maps. In *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing (NNSP'95)*, pages 11–20, 1995.
50. N. Sanchez-Maroño, O. Fontela-Romero, A. Alonso-Betanzos, and B. Guijarro-Berdiñas. Self-organizing maps and functional networks for local dynamic modeling. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'95)*, pages 39–44, 2003.
51. A. Sfetsos and C. Siriopoulos. Time series forecasting with a hybrid clustering scheme and pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, A-34(3):399–405, 2004.
52. G. Simon, A. Lendasse, M. Cottrell, J.-C. Fort, and M. Verleysen. Double quantization of the regressor space for long-term time series prediction: method and proof of stability. *Neural Networks*, 17(8-9):1169–1181, 2004.
53. G. Simon, A. Lendasse, M. Cottrell, J.-C. Fort, and M. Verleysen. Time series forecasting: obtaining long term trends with self-organizing maps. *Pattern Recognition Letters*, 26(12):1795–1808, 2005.
54. K. Stokbro, D. K. Umberger, and J. A. Hertz. Exploiting neurons with localized receptive fields to learn chaos. *Complex Systems*, 4(3):603–622, 1990.
55. H. Tong. *Non-linear Time Series: A Dynamic System Approach*. Oxford University Press, 1993.
56. C.-Y. Tsao and S.-H. Chen. Self-organizing maps as a foundation for charting or geometric pattern recognition in financial time series. In *Proceedings of the IEEE International Conference on Computational Intelligence for Financial Engineering*, pages 387–394, 2003.
57. J.J. Van Wijk and E.R. Van Selow. Cluster and calendar based visualization of time series data. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'99)*, pages 4–9, 1999.

58. J. Vesanto. Using the SOM and local models in time-series prediction. In *Proceedings of the Workshop on Self-Organizing Maps (WSOM'97)*, pages 209–214, Espoo, Finland, 1997.
59. T. Voegtlin. Recursive self-organizing maps. *Neural Networks*, 15(8-9):979–991, 2002.
60. J. Walter and H. Ritter. Rapid learning with parametrized self-organizing maps. *Neurocomputing*, 12:131–153, 1996.
61. J. Walter, H. Ritter, and K. Schulten. Non-linear prediction with self-organizing map. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'90)*, volume 1, pages 587–592, 1990.
62. A. Weigend and N. Gershefeld. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, 1993.
63. R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.
64. G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14(1):35–62, 1998.