ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

A Method for Time Series Prediction using a Combination of Linear Models

David Martínez-Rego, Oscar Fontenla-Romero and Amparo Alonso-Betanzos [*]

Laboratory for Research and Development in Artificial Intelligence (LIDIA)
Department of Computer Science - University of A Coruña
Campus de Elviña s/n, 15071, A Coruña - Spain

Abstract. This paper presents a new approach for time series prediction using local dynamic modeling. The proposed method is composed of three blocks: a Time Delay Line that transforms the original time series into a set of $N-dimensional$ vectors, an Information-Theoretic based clustering method that segments the previous set into subspaces of similar vectors and a set of single layer neural networks that adjust a local model for each subspace created by the clustering stage. The results of this model are compared with those of another local modeling approach and of two representative global models in time series prediction: Tapped Delay Line Multilayer Perceptron (TDL-MLP) and Support Vector Regression (SVR).

1 Introduction

The *time series prediction* problem that will be treated in this work can be resumed in this expression:

$$x(t+p\gamma) = f(x(t), x(t-\gamma), x(t-2\gamma), ..., x(t-(N-1)\gamma)), \quad p \geq 1 \qquad (1)$$

where x represents the stochastic variable to be predicted, t represents time, $x(t), x(t-\gamma), x(t-2\gamma), ..., x(t-(N-1)\gamma)$ are past observations of variable x, γ represents the size of one delay step, p is the *prediction step* that is being predicted and f the goal function to estimate.
Global modeling based approaches have been the most used by machine learning methods in the past when applied to *time series analysis* [1]. Global modeling tries to estimate f adjusting only one model that explains all presented cases. Several methods based in this philosophy have been presented and commonly used in the past (Tapped Delay Line Multilayer Perceptron, Recurrent Networks, Support Vector Regression, Radial Basis Functions, etc).
When a local modeling approach is taken to resolve this problem, the overall predictive function f is estimated as the union of several local estimators

$$f(\mathbf{x}) = \bigcup_{i=0}^{M} \hat{f}_i(\mathbf{x}) \qquad (2)$$

where local estimators $\hat{f}_i(\mathbf{x})$ are defined in different regions of the input space. In recent years several topologies based on local models have been presented,

[*]We would like to acknowledge support for this project from INDRA Systems and Xunta de Galicia (project PGIDIT05DPI082E).

achieving good results [2, 3, 4].
In this work *time series prediction* problem is tackled using a local modeling approach, called *Distributed Local Experts based on Vector-Quantization using Information Theoretic Learning* (DLE-VQIT). The results of this model for three benchmark series are compared with those achieved by a previous local modeling approach proposed in [3] and by two global modeling approaches of neural networks and kernel methods commonly used: Tapped Delay Line Multilayer Perceptron (TDL-MLP)[1] and Support Vector Regression (SVR)[5].

2 Description of the model

The proposed method DLE-VQIT is composed of three blocks, as schematically depicted in Fig. 1:

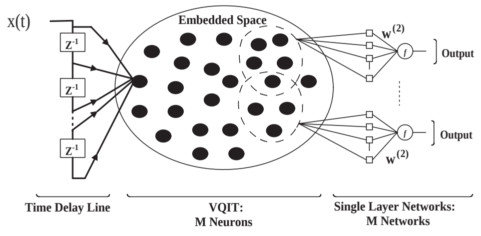

Fig. 1: Structure of the proposed VQIT-DLE

- An *embedding layer* implemented by a Time Delay Line. This layer embeds the original one-dimensional time series, $x(t)$, into a reconstruction space. The output of this layer is a set S of N-dimensional state vectors created from the input signal, $\mathbf{x}(t) = [x(t), x(t-\gamma), ..., x(t-(N-1)\gamma)]^T$, where γ represents the size of one delay step.
- A *Vector-Quantization using Information Theoretic Concepts* (VQIT) layer trained using the learning rule described in [6]. This model takes a physical approach to solve vector quantization and is less rigid than other classifiers chosen in previous works [3] [4]. VQIT considers two types of particles, input vectors and nodes of the network. These particles generate two types of interactions: attraction between input vectors and the nodes of the network and repulsion between the nodes of the network. These interactions are represented by a kernel function that decay with the distance, like the Gaussian kernel. VQIT achieves accurate representations of input spaces minimizing the divergence between the distribution of the nodes and the distribution of the data. The input of this network is the pair $(d(t), \mathbf{x}(t))$, where $d(t)$ is the desired response of the system. In a time series prediction scenario, $d(t) = x(t + p\gamma)$, where p is a desired *prediction step* in the future. To obtain an acceptable training time and a good distribution of the neurons across the *embedded space* (see figure 1), the original model presented in [6] was modified. The following two changes were made:

  - The original algorithm in [6] has an elevated computational cost for large sets of training data and big networks (each iteration has a complexity of $O(M^2N)$, where M is the number of nodes of the network and N the number of training patterns). In order to obtain an acceptable training time with a similar classification accuracy, instead of training with the whole training set S at each iteration of the algorithm, a random subset of D training data from the original set S (being D a parameter defined by the user) is taken. The variation of the algorithm chooses a different random sample of the original set for each iteration.
  - The *learning rate* α of the system takes variable values. It starts with high values at the first iterations to move all the nodes of the system near the input space. At each iteration, this *learning rate* is decremented multiplying its previous value by a constant $\beta \in (0, 1)$, in order to obtain a smooth adjustment of the system once the nodes are distributed across the input space.

- A set of *single layer neural networks*. The goal of this subsystem is to fit a local model for each subspace created by the previous layer. Once the VQIT is trained, the process below is followed:

  1. Calculate, for each vector of S, which is the closest VQIT node or, in VQIT terms, which is affected by the greatest attraction potential. For this calculus, the same kernel function (in this case a Gaussian kernel), as in VQIT training is used.
  2. Delete the VQIT nodes with less than η training vectors. These are useless nodes because they are redundant or lost[1] nodes.
  3. If there are deleted nodes from step 2, repeat step 1 for the remaining nodes. This action produces that those vectors belonging previously to deleted VQIT nodes change its owner; else, go to step 4.
  4. For each remaining node of the VQIT:
     (a) Construct a training set T with the vectors from S that are owned by this node and by the vectors of its L closest neighbors. This selection produces two effects:
       - It ensures that each node has enough data to train a local model with the algorithm selected for this purpose.
       - It produces an overlapping between the areas corresponding to each node. This ensures a smoothing continuity among neighbor models.

     (b) Train for this set, T, a single layer neural network using the powerful and fast training algorithm in [7]. This algorithm always obtains the global optimum in a direct (not iterative) manner. In

[1]A *lost* node is one that is very far from any vector of the training set.

[7] a new cost function, that measures the error committed by the network before the non linear activation function is applied:

$$Error = \sum_{i=1}^{S}(f^{'}(\bar{d}) * \bar{\epsilon_s})^2 \tag{3}$$

where $f^{'}$ is the derivative of the neural activation function, d the desired output, $\bar{d} = f^{-1}(d)$, $\bar{\epsilon_s} = \bar{d_s} - (\mathbf{w}^T\mathbf{x} + b)$, $\mathbf{w}$ is the weight vector and b the bias. Therefore, optimal weights can be obtained solving a system of $N+1$ linear equations with $N+1$ unknowns.

Once the training of VQIT-DLE model is completed, the system works as follows:

1. Given a new input vector $\mathbf{x}(t)$, supplied by the first block of the system, the closest node of the VQIT is selected. In this process only $\mathbf{x}(t)$ is used and not the desired output $d(t)$, which is unknown.
2. The single layer neural network associated with the closest node is activated and supplies the output of the system using vector $\mathbf{x}(t)$ as input.

3 Experimental results

The proposed system was compared with a local modeling approach proposed in [3] which in this work is called *Distributed Local Models* (DLM), and also with two commonly used global modeling approaches: *Tapped Delay Line Multilayer Perceptron* (TDL-MLP) trained with the Scaled Conjugate Gradient Algorithm [8] and *ϵ-Support Vector Regression* [5].
All methods were run in Matlab 7.0(R14). Specifically, for ϵ-SVR the Spider Toolbox 1.71 [9], for TDL-MLP the implementation of the Matlab Neural Networks Toolbox and for DLM and DLE an own implementation, were used. In order to make the comparisons three benchmark time series were employed: Henon [10], Lorenz [10] and the Dow-Jones index from 52/02/07 to 91/10/31 [11]. We used two different sizes, medium and large, for the datasets in order to analyze its influence on the performance of the methods. The number of input data were 1500 (medium) and 15000 (large) for Henon, 3000 and 30000 for Lorenz and 1000 and 10000 for Dow-Jones. These series were normalized in the interval $[0.05, 0.95]$. For all of them the goal is to predict the next value using five previous values. The following configurations were tested:

- For TDL-MLP, topologies with a hidden layer and with logistic sigmoidal activation functions. The number of neurons of the hidden layer was varied between 5 and 50.
- For ϵ-SVR, the method described in [12] was used to select the variance σ of the RBF kernel and the soft margin C. The ϵ value of the ϵ-insensitive function was varied in the set $\{0.1, 0.05, 0.025, 0.0125\}$.

- For DLM, a *Self Organizing Map* (SOM) with a 20×20 topology trained using 2000 epochs was employed. The neighborhood size was varied between 7 and 27.
- For DLE-VQIT, a VQIT with 400 nodes, $\alpha_{init} = 0.9$, $\beta = 0.99$ and $D = 500$ was used. The value of η was set to 2. The neighborhood size L was varied between 7 and 27.

To estimate the final configuration for each method, ten 10-fold crossvalidation were run. The performances of the methods were calculated using the mean Normalized Mean Squared Error (NMSE) over the 100 trials. The optimal topologies are shown in table 1.

		Henon	**Lorenz**	**Dow-Jones**
medium	**SVR**	$\epsilon = 0.0125, \sigma = 0.45,$ $C = 1.35$	$\epsilon = 0.05, \sigma = 0.17,$ $C = 0.81$	$\epsilon = 0.025, \sigma = 0.24,$ $C = 1.07$
	TDL-MLP	$5-15-1$	$5-15-1$	$5-20-1$
	DLM	$L = 7$	$L = 9$	$L = 15$
	DLE-VQIT	$L = 7$	$L = 7$	$L = 7$
large	**SVR**	$\epsilon = 0.0125, \sigma = 0.45,$ $C = 1.35$	$\epsilon = 0.0125, \sigma = 0.18,$ $C = 1.04$	$\epsilon = 0.0125, \sigma = 0.38,$ $C = 0.77$
	TDL-MLP	$5-25-1$	$5-10-1$	$5-20-1$
	DLM	$L = 7$	$L = 10$	$L = 20$
	DLE-VQIT	$L = 7$	$L = 7$	$L = 21$

Table 1: Optimal topologies and training parameters for each method.

Finally, and in order to evaluate the efficiency of the methods, the CPU time was measured using a computer with a 2,13 GHz processor. Table 2 and 3 show the performance and CPU times obtained. Results in bold face are the best ones for each data set.

		Henon	**Lorenz**	**Dow-Jones**
medium	**SVR**	$2.01E-2 \pm 2.40E-3$	$1.46E-2 \pm 3.10E-3$	$6.70E-3 \pm 2.20E-3$
	TDL-MLP	$7.59E-2 \pm 5.09E-1$	**1.56E-4 ± 8.82E-5**	$2.53E-2 \pm 2.42E-1$
	DLM	$1.01E-1 \pm 5.72E-2$	$3.34E-4 \pm 1.54E-4$	$7.60E-3 \pm 1.17E-2$
	DLE-VQIT	**1.92E-2 ± 7.20E-3**	$2.58E-4 \pm 9.95E-5$	**1.10E-3 ± 5.92E-4**
large	**SVR**	$1.90E-2 \pm 6.90E-4$	$1.13E-2 \pm 7.08E-4$	$1.35E-2 \pm 3.90E-3$
	TDL-MLP	$7.18E-2 \pm 5.00E-1$	$9.27E-5 \pm 5.67E-5$	$3.39E-2 \pm 3.35E-1$
	DLM	$5.43E-4 \pm 5.65E-4$	$9.52E-6 \pm 4.11E-6$	$1.00E-3 \pm 6.61E-4$
	DLE-VQIT	**2.36E-5 ± 7.60E-6**	**5.59E-6 ± 1.79E-6**	**4.68E-4 ± 2.13E-4**

Table 2: Mean Normalized Mean Squared Error (NMSE) ± standard deviation for the three datasets and for the four methods compared.

4 Conclusions

In this work a new local modeling method using a three stage procedure is proposed. As it is supported by the experimental results, the local modelling

		Henon	**Lorenz**	**Dow-Jones**
large	**SVR**	0.50 ± 0.02	0.23 ± 0.03	0.06 ± 0.01
	TDL-MLP	231.10 ± 33.70	197.45 ± 20.48	111.91 ± 21.17
	DLM	26.13 ± 0.25	27.59 ± 0.44	25.49 ± 0.24
	DLE-VQIT	88.50 ± 0.46	115.69 ± 0.81	82.13 ± 0.44

Table 3: Mean training times, in seconds, and standard deviations for each method and for the three large size datasets. Corresponding training times are proportional for medium size datasets.

methods exhibit, for the data sets employed, a better performance and a more stable behaviour than the global approaches indicated by its small standard deviation of the NMSE. The difference between the errors obtained by global and local methods are smaller for medium size datasets although, in general, the NMSE obtained by local methods is still smaller in this case. Regarding both local methods, DLE-VQIT achieves considerably better results than the previous DLM approach. Finally, concerning the execution time, DLE-VQIT presents acceptable values for real applications.

References

[1] M. van Veelen, J. Nijhuis, and B. Spaanenburg. Neural network approaches to capture temporal information. In *Computing Anticipatory Systems - Third International Conference. AIP Conference Proceedings*, volume 517 of *American Institute of Physics Conference Series*, pages 361–371, May 2000.

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

[3] O. Fontenla-Romero, A. Alonso-Betanzos, E. Castillo, J.C. Principe, and B. Guijarro-Berdiñas. Local modeling using self-organizing maps and single layer neural networks. In *Lecture Notes In Computer Science*, volume 2415, pages 945–950, 2002.

[4] J. Vesanto. Using the SOM and local models in time-series prediction. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland*, pages 209–214. 1997.

[5] A.J. Smola and B. Schölkopf. A tutorial on support vector regression. Technical report, NeuroCOLT, 1998.

[6] J.C. Principe, T. Lehn-Schioler, A. Hedge, and D. Erdogmus. Vector-quantization using information theoretic concepts. *Natural Computing*, 4:39 – 51, 2005.

[7] O. Fontenla-Romero, A. Alonso-Betanzos, E. Castillo, and B. Guijarro-Berdiñas. A global optimum approach for one-layer neural networks. In *Lecture Notes In Computer Science*, volume 2415, pages 1429–1449, 2002.

[8] M. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.

[9] J. Weston, A. Elisseeff, G. BakIr, and F. Sinz. Spider SVM toolbox, 2006. http://www.kyb.tuebingen.mpg.de/bs/people/spider/, Last access: 2-11-2007.

[10] E.A. Wan. Time series data, 2005. http://www.cse.ogi.edu/∼ericwan/data.html, Last access: 2-11-2007.

[11] P. Vlachos. Statlib-datasets archive, 2005. http://lib.stat.cmu.edu/datasets/, Last access: 2-11-2007.

[12] V. Cherkassky and Y. Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Computation*, 17:113–126, 2002.