LRBNN: A Lazy Radial Basis Neural Network model

José M. Valls [*], Inés M. Galván and Pedro Isasi

Departamento de Informática, Universidad Carlos III de Madrid, Avenida de la Universidad, 30, 28911 Leganés, Madrid, Spain

Tel.: +34-916248845; Fax: +34-916249129; E-mail: jvalls@inf.uc3m.es

Abstract. In the domain of inductive learning from examples, usually, training data are not evenly distributed in the input space. This makes global and eager methods, like Neural Networks, not very accurate in those cases. On the other hand, lazy methods have the problem of how to select the best examples for each test pattern. A bad selection of the training patterns would lead to even worse results. In this work, we present a way of performing a trade-off between local and non-local methods using a lazy strategy. On one hand, a Radial Basis Neural Network is used as learning algorithm; on the other hand, a selection of training patterns is performed for each query in a local way. The selection of patterns is based on the analysis of the query neighborhood, to forecast the size and elements of the best training set for that query. Moreover, the RBNN initialization algorithm has been modifie in a deterministic way to eliminate any initial condition influence The method has been validated in three domains, one artificia and two time series problems, and compared with traditional lazy methods.

Keywords: Lazy learning, local learning, Radial Basis Neural Networks, pattern selection

1. Introduction

When the training data are not evenly distributed in the input space, the non-local learning methods could be affected by decreasing their generalization capabilities. One way of resolving such problem is by using local learning methods [2,17]. Local methods use only partially the set of examples for making the learning. They select, from the whole examples set, those that consider more appropriate for the learning task. The selection is made for each new test pattern presented to the system, by means of some kind of similarity measurement to that pattern. k-NN [4] is a typical example of these systems, in which the selected learning patterns are the k closest to the test pattern by some distance metric, usually the Euclidean distance.

Those local methods, usually known as lazy learning or instance-based learning algorithms [1] are based on the assumption that all the test patterns have the same structure and need the same selection procedure. This assumption is often invalid because the input space is neither isotropic nor homogeneous and has irrelevant and non-homogeneous features. Thus, these methods are highly dependent on the number of examples selected and on the metric used, being frequent the situations where an Euclidean metric might not be appropriate.

Bottou and Vapnik [3] introduces a dual, local/non-local, approach to give good generalization results in non-homogeneous domains. This approach is based on the following procedure:

- For each test pattern (local learning):
  - Select the k closest examples from the example set
  - Train a Neural Network using the above selected examples (non-local learning)
  - Apply the above trained Neural Network to predict the test pattern

This is a good combination between local and non-local learning. However, the neural network used is a linear classifie and the method assumes that Euclidean distance is an appropriate metric. Besides, it considers that all test patterns have the same structure but some domains would require different behaviors when being in different regions. This anisotropy of the space would need of some specifi selection procedures.

In this work we introduce some modification in the general procedure of Bottou and Vapnik, by con-

[*] Corresponding author

sidering the use of Radial Basis Neural Networks (RBNN) [5,8,12]. RBNN have some advantages when using dual techniques: they are non-linear, universal approximators [10] and their training is very fast, without increasing significat vely the computational cost of standard lazy approaches.

RBNN are eager or non-lazy learning methods, because they must estimate the target function before the test pattern is known. In this work, we use RBNN with a lazy learning approach, making the selection of training patterns based on a kernel function. This selection is not homogeneous, as happened in [3]; by opposite it is detected, for each testing pattern, how many training patterns would be needed, and what is the importance in the learning process of each one of them. This importance is taken into consideration, in the form of a weight, in the learning process of the network.

Regarding to RBNN, the initialization of the training algorithm is a critical factor that influence their performance. This algorithm has been modifie in a deterministic way to eliminate any initial condition influ ence. Regarding to the selection procedure by means of a kernel function, it may occur that no training pattern is selected for certain test patterns, due to the distribution of data in the input space. We propose two different approaches to treat this problem.

The fina method results to be a dual local/non-local method, where the initialization of the network is deterministic and the method is able to determine the degree of locality of each region of the space, by means of a kernel function that could be considered as a parameter, and modifie appropriately. In some cases a test pattern could be considered as non-local in the sense that it corresponds to more frequent situations. In this case almost the totality of the training patterns will be selected, and the method behaves like an non-local approach. This transition between local and non-local behavior is made automatically.

The rest of this paper is organized as follows. Section 2 describes the learning method. Section 3 reports the experiments carried out and, finall , Section 4 draws the conclusions of the paper.

2. Description of the method

The learning method proposed in this work consists of training RBNN with a lazy learning approach. This method has been called LRBNN (Lazy RBNN method) and is based on the selection, from the whole training data, of an appropriate subset of training patterns in order to improve the answer of the network for a novel pattern. For each new pattern received, a new subset of training examples is selected. The LRBNN method is based on the ideas found in [14]. The presented method deals with some problems found in the previous work when a lazy strategy is used to train a RBNN. LRBNN constitutes a complete and closed method.

The general idea consists of selecting those patterns close to the new query instance, in terms of the Euclidean distance. In order to give more importance to the closest examples, some weighting measure must be considered and there are two ways of doing it: weighting the data directly or weighting the error criterion used by the RBNN in such a way that the neural model must fi more tightly the closest patterns [2]. Both ways are equally valid and we have chosen the firs one. In order to use standard RBNN, we have chosen the replication of examples as a way of weighting the data, without using a weighting value associated to each example. Thus, selected patterns are included one or more times in the resulting training subset and the network is trained with the most useful information, discarding those patterns that not only do not provide any knowledge to the network, but might confuse the learning process.

The weighting measure assigns a weight to each training example; this is done by using a kernel function which depends on the Euclidean distance from the training pattern to the novel one. The maximum value of the kernel function must be given at zero distance and the function should decrease smoothly as distance increases [16]. In this work, the inverse function [Eq. (1)] is used:

$$K(d) = 1/d \tag{1}$$

where d is the distance from the training pattern to the new query.

To exploit this idea, a n-dimensional sphere centered at the test pattern is established, in order to select only those patterns placed into it. Its radius – named r – is a threshold distance, since all the training patterns whose distance to the novel sample is bigger than r will be discarded. Distances may have very different magnitudes depending on the problem domains, due to their different data values and number of attributes. It may happen that for some domains the maximum distance between patterns is many times the maximum distance between patterns for other domains. In order to make the method independent of this fact, both the sphere radius and the training patterns distances will be relative respect to the maximum distance to the test pattern. Thus, the relative threshold distance or relative radius, r_r, will be used to select the training patterns situated into the sphere centered at the test pattern, being r_r a parameter that must be established before the application of the learning algorithm.

Next, the sequential structure of LRBNN method is presented. Let us consider **q** an arbitrary novel pattern described by a n-dimensional vector, $\mathbf{q} = (q_1, \ldots, q_n)$, where q_i represents the attributes of the instance q. Let X be the whole available training data set:

$$X = \{(x_k, y_k) k = 1 \ldots N; \\ x_k = (x_{k1}, \ldots, x_{kn}); y_k = (y_{ki}, \ldots, y_{km})\} \quad (2)$$

For each new pattern **q**,

1. The standard Euclidean distances d_k from the pattern **q** to each input training pattern are calculated.
2. In order to make the method independent on the distances magnitude, relative distances must be used. Thus, a relative distance d_{rk} is calculated for each training pattern: $d_{rk} = d_k/d_{max}$, where d_{max} is the distance from the novel pattern to the furthest training pattern.
3. A weighting function or kernel function $K()$ is used to calculate a weight for each training pattern from its distance to the test pattern. This function is the inverse of the relative distance d_{rk}:

$$K(x_k) = \frac{1}{d_{rk}}; \quad k = 1 \ldots N \quad (3)$$

4. These values $K(x_k)$ are normalized in such a way that the sum of them equals the number of training patterns in X. These normalized values are called normalized frequencies, f_{nk}, and are calculated in the following way:

$$f_{nk} = V \cdot K(x_k) \quad (4)$$

where

$$V = \frac{N}{\sum_{k=1}^{N} K(x_k)} \quad (5)$$

5. Both the relative distance d_{rk} and the normalized frequency f_{nk} are used to decide whether the training pattern (x_k, y_k) is selected and – in that case – how many times is included in the training subset. They are used to generate a natural number, n_k, following the next rule:

$$\begin{array}{ll} \text{if} \quad d_{rk} < r_r & \text{then} \\ \quad n_k = int(f_{nk}) + 1 & \\ \text{else} & \\ \quad n_k = 0 & \end{array} \quad (6)$$

where $int(f_{nk})$ is the largest integer lower than f_{nk}. At this point, each training pattern in X has an associated natural number, n_k, which indicates how many times the pattern (x_k, y_k) will be used to train the RBNN when the new instance **q** is reached. If the pattern is selected, $n_k > 0$ otherwise $n_k = 0$.
6. A new training subset associated to the query instance **q**, X_q, is built up. Given a pattern (x_k, y_k) from the original training set X, that pattern is included in the new subset if the value n_k is higher than zero. In addition, the pattern (x_k, y_k) is placed n_k times randomly in the training set X_q.
7. The RBNN is trained using the new subset X_q. This training process implies the determination of the centers and dilations of the hidden neurons and the determination of the weights associated from those hidden neurons to the output neuron. Those parameters are calculated as follows:
   - The centers of neurons are calculated in an unsupervised way using the K-means algorithm in order to clusterize the input space formed by all the training patterns included in the subset X_q, which contains the replicated selected patterns.
   - The neurons dilations or widths are evaluated as the geometric mean of the distances from each neuron center to its two nearest centers.
   - The weights associated to the connections from the hidden neurons to the output neuron are obtained, in an iterative way, using the gradient descent method to minimize the mean square error measured in the output of the RBNN over the training subset X_q.

In order to apply the LRBNN method, two features must be taken into account. On one hand, the results would depend on the random initialization of K-means algorithm which is used to determine the locations of the RBNN centers and must be applied for each novel or test pattern. Hence, running the K-means algorithm with different random initialization for each test sample would not be very appropriate due to the high computational cost.

On the other hand, a problem arises when the test pattern belongs to regions of the input space with a low data density: it would be possible that the sphere centered at the test sample would not include any train example into it. In this case, the method should offer some alternative in order to provide an answer for the test sample. We present solutions to both problems, which are described in the following.

1. **K-means initialization.** Having the objective of achieving the best performance, a deterministic initialization, instead of the usual random ones, is proposed. The idea is to obtain a prediction of the network with a deterministic initialization of the centers whose accuracy is similar to the one obtained when several random initializations are done. The initial location of the centers will depend on the location of the closest training examples selected. The deterministic initialization is obtained as follows:

   - Let $(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_l)$ be the l selected training patterns, ordered by their normalized frequencies $(f_{n1}, f_{n2}, \ldots, f_{nl})$ calculated in Eq. (4).
   - Let m be the number of hidden neurons of the RBNN to be trained.
   - The center of the i_{th} neuron is initialized to the $\mathbf{x}_i$ position, for $i = 1, 2, \ldots, m$.

   It is necessary to avoid the situations where $m > l$. The number of hidden neurons must be fi ed to a number smaller than the selected patterns, since the opposite would not have any sense.

2. **Empty training set.** It has been observed that when the input space data is highly dimensional, in certain regions of it the data density can be so small that the sphere centered at the query instance does not include any train pattern into it if the relative radius is small. When this situation occurs, an alternative way to select the training patterns must be taken. In our work, we propose two different approaches which are experimentally evaluated.

   (a) If the subset X_q associated to a test sample $\mathbf{q}$ is empty, then we apply the method of selection to the closest training pattern, as if it was the test pattern. In more detail: let $\mathbf{x}_c$ be the closest training pattern to $\mathbf{q}$. Thus, we will consider $\mathbf{x}_c$ the new test pattern, being named $\mathbf{q'}$. We apply our lazy method to this pattern $\mathbf{q'}$, that is, the selection sphere center is placed at $\mathbf{q'}$, and the associated training set $X_{q'}$ is generated. Since $\mathbf{q'} \in X$, $X_{q'}$ will always have, at least, one element. At this point, the network is trained with the set $X_{q'}$ to answer to the test point $\mathbf{q}$.

   (b) If the subset X_q associated to a test sample $\mathbf{q}$ is empty, then the network is trained with X, the set formed by all the training patterns. In other words, the network is trained as usual, with all the available patterns.

   As it was previously mentioned, if no training examples are selected, the method must provide some answer to the test pattern. Maybe, the most intuitive solution consists of using the whole training data set X; this alternative does not have disadvantages since the networks will be trained, in a fully global way, with the whole training set only for a few test samples. However, in order to maintain the coherence of the idea of training the networks with some selection of patterns, we suggest the firs alternative. The experiments carried out show that this alternative behaves better.

3. Experimental validation

In this section, the LRBNN learning method has been applied to three different problems: two artificia problems and a real one. The firs one corresponds to an artificia regression problem (a piecewise-define function), the second one corresponds to a well known artificia time series prediction problem – the Mackey-Glass time series-and, finall , a real time series prediction problem define by means of a time-series describing the behavior of the water level at Venice Lagoon.

In the next subsections, the features of the different problems and the results obtained for each domain are presented and analyzed. A comparative study with other lazy techniques and with the traditional training of RBNN is also included.

3.1. Domains description

The three domains cited above are described in detail in the following paragraphs.

- **A piecewise-defined function**
  The piecewise-define function is a single variable function given by Eq. (7).

$$f(x) = \begin{cases} -2.186x - 12.864 & \text{if } -10 \leqslant x < -2 \\ 4.246x & \text{if } -2 \leqslant x < 0 \\ 10e^{-0.05x-0.5} & \text{if } 0 \leqslant x \leqslant 10 \\ \quad \times \sin((0.03x + 0.7)x) & \end{cases} \tag{7}$$

  The training set is composed of 120 input-output points randomly generated by an uniform distribution in the interval $[-10, 10]$. 80 input-output points generated in the same way are used as test patterns. Data are normalized in the interval [0, 1].
- **The Mackey-Glass time series prediction**
  This time series is widely used in the literature about RBNN, being some of the most important works the next: [8,11,18,19]. This domain represents a chaotic time series created by the Mackey-Glass delay-difference equation [6]:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = -bx(t) + a\frac{x(t-\tau)}{1 + x(t-\tau)^{10}} \tag{8}$$

  In the same way as in the studies mentioned above, the series has been generated using the next values for the parameters: $a = 0.2, b = 0.1$, and $\tau = 17$. The task for the RBNN is to predict the value of the time series at point $x[t + P]$ from the earlier points $(x[t], x[t-6], x[t-12], x[t-18])$. The number of sample steps P has been set to 50. Thus, the function (whose dimension is 4) to be learned by the network is:

$$x(t) = f(x(t-50), x(t-50-6), \\ x(t-50-12), x(t-50-18)) \tag{9}$$

  5000 values of the time series are generated using the Eq. (9) and fixin $x(0) = 0$. The initial 3500 samples are discarded in order to avoid the initialization transients. 1000 data points, corresponding to the sample time between 3500 and 4499, have been chosen for the training set. The test set is composed by the points corresponding to the time interval $[4500, 5000]$. All data points are normalized in the interval [0, 1].
- **Prediction of water level at Venice Lagoon**
  This real world time series represents the behavior of the water level at Venice Lagoon. Unusual high tides result from a combination of chaotic climatic elements with the more normal, periodic, tidal systems associated with a particular area. The prediction of high tides has always been the subject of intense interest, not only from a human point of view, but also from an economic one, and the water level of Venice Lagoon is a clear example of these events [7,9]. The most famous example of floodin in the Venice Lagoon occurred in November 1966 when the Venice Lagoon rose by nearly 2 meters above the normal water level. That phenomenon is known as "high water" and many efforts have been made in Italy to develop systems for predicting sea level in Venice, mainly for the prediction of the high water phenomenon [13]. Different approaches have been developed for the purpose of predicting the behavior of sea level at the Venice Lagoon [13,15]. Multilayer feedforward neural networks have also been used to predict the water level [20] obtaining same advantages over linear and traditional models.
  In this work, LRBNN method has been used with the purpose of predicting the next sampling time from some samples measured at previous times. There is a great amount of data representing the behavior of the Venice Lagoon time series. However, corresponding data associated to the stable behavior of the water are very abundant opposed to those data associated to high water phenomena. This situation leads to the fact that RBNN trained with a complete data set is not very accurate in predictions of high water phenomena. Hence, the aim in this context is to observe whether a selection of training patterns may help to obtain better predictions of high water phenomena.
  Since the goal in this work is to predict only the next sampling time, a nonlinear model using the six previous sampling times, i.e., data of the six previous hours, may be appropriate. Thus, the function to be learned, whose dimension is 6, is:

$$x(t) = f(x(t-1), x(t-2), x(t-3), \\ x(t-4), x(t-5), x(t-6)) \tag{10}$$

  A training data set of 3000 points corresponding to the water level measured each hour has been extracted from available data (water level of Venice Lagoon between 1980 and 1994 sampled every hour). This set has been chosen in such a way that both stable situations and high

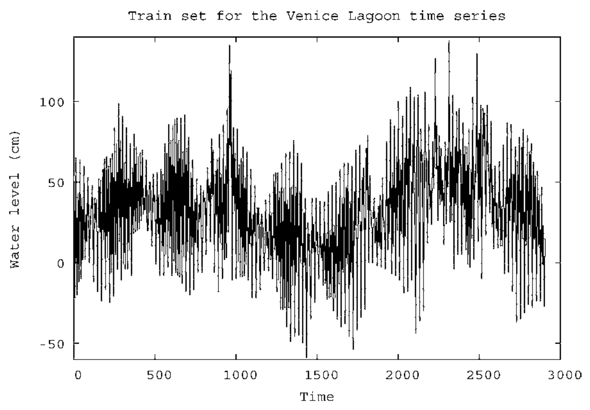

Fig. 1. Water level at Venice Lagoon during four months. Training set.

water situations appear represented in the set (see Fig. 1). High-water situations are considered when the level of water is not lower than 110 cm. 1010 test patterns have also been extracted from the available data containing both high water and periodic situations.

3.2. *Experimental results using LRBNN*

We have applied our local learning method to all the domains described above. As it was remarked in Section 2, the relative radius r_r must be given as an external parameter of the method in order to study its influenc on the performance of the model. This parameter indicates the radius of the n-dimensional sphere centered at the test pattern to be predicted. Besides, RBNN with different architectures, i.e., different number of hidden neurons – must be trained so that the influenc of the network architecture can also be studied. The performance of the method has been measured in terms of the RBNN's errors over the whole test set. The error, e, for the test set is evaluated as:

$$e = \frac{1}{N}\sum_{k=1}^{N} e_k \qquad (11)$$

where N is the number of patterns in the test set and e_k represents the error for the kth test pattern, calculated as $e_k = |\tilde{y}_k - y_k|$, being $\tilde{y}_k$ the output of the network and y_k the desired output for that pattern. In all the studied domains the output is a real number.

In order to validate the proposed deterministic initialization, two groups of experiments for each domain have been done. In the firs group, ten different random initializations of RBNN centers have been carried out and the mean and standard deviation of the errors committed for each run have been calculated. In the second set of experiments, only one run has been done using the deterministic initialization. In this case, the error, given by Eq. (11), is calculated.

Next, the experimental results are presented.

3.2.1. *An artificia approximation problem: A piecewise-define function*

The learning method described in Section 2 has been applied to this problem for RBNN with different architectures, from 7 to 27 neurons, varying the relative radius from 0.02 to 0.3 taking a step of 0.04. As it was previously commented, the aim of these experiments consists of studying the influenc of the relative radius on the generalization ability of the networks. It is convenient to recall that the performance of RBNN depends on the location of the RBF centers, being this location the key issue of the learning process of these networks.

Usually, standard K-means algorithm (which is a non-linear optimization algorithm whose result depends on the initial values of the centers) is used to fin these locations. Thus, the performance of RBNN trained in the usual way depends on the initial random values of the centers, and several experiments with different initial center locations must be done in order to obtain representative results. As it was explained in Section 2, our approach provides a way to fix deterministically, the initial positions of the centers. These positions only depend on the location of the most weighted selected patterns and, thus, no additional experiments must be done.

In order to show that this deterministic initialization lead to an appropriate performance of RBNN when they are trained following the lazy learning approach, experiments with the lazy approach where the neurons centers are randomly initialized are also made. Table 1 shows the mean performance of the method for ten random initializations. Each value of the error for a specifi number of neurons and radius corresponds to the mean value of ten different errors [given by Eq. (11)]. In Table 2, the standard deviations for these ten error values corresponding to each network architecture and radius are shown.

In Table 1, the column named "NP" (Null Patterns) displays the number of test patterns for which the number of selected training patterns is zero. As we can see, in all cases, even when the relative radius is very small, the number of selected training patterns is bigger than zero (the associated training set for all the test patterns is not empty). This hap-

Table 1
Mean errors with random initialization of centers. Piecewise-define function

r_r	Hidden neurons						NP	% PP
	7	11	15	19	23	27		
0.02	0.00975	0.01178	0.01643	0.03824	0.03424	0.05668	0	100
0.06	0.00774	0.00315	0.00289	0.00273	0.00389	0.00469	0	100
0.1	0.01063	0.00401	0.00332	0.00285	0.00348	0.00315	0	100
0.14	0.01079	0.00563	0.00440	0.00374	0.00373	0.00316	0	100
0.18	0.01441	0.00861	0.00567	0.00474	0.00405	0.00366	0	100
0.22	0.01865	0.01262	0.00848	0.00565	0.00461	0.00420	0	100
0.26	0.02492	0.01506	0.01130	0.00675	0.00545	0.00477	0	100
0.3	0.02787	0.02028	0.01340	0.00855	0.00607	0.00508	0	100

Table 2
Standard deviations of errors corresponding to ten random initialization of centers. Piecewise-define function

r_r	Hidden neurons					
	7	11	15	19	23	27
0.02	0.001722	0.001097	0.003988	0.005335	0.015298	0.014543
0.06	0.001060	0.000661	0.000361	0.000533	0.000380	0.000561
0.1	0.001006	0.000740	0.000329	0.000292	0.000536	0.000666
0.14	0.001914	0.000874	0.000520	0.000414	0.000452	0.000543
0.18	0.001967	0.000697	0.000614	0.000558	0.000714	0.000504
0.22	0.001915	0.001350	0.001169	0.000622	0.000433	0.000735
0.26	0.002775	0.001461	0.001206	0.000987	0.000690	0.000798
0.3	0.002690	0.001113	0.001555	0.001215	0.000618	0.000846

pens because the piecewise-define function is an one-dimensional domain and the data are uniformly distributed in the input space; thus, the test patterns always have some training patterns in its neighborhood for the radius taken in these experiments. The column named % PP shows the percentage of predicted patterns. Obviously, when the number of "null patterns" is zero, all the test patterns (100%) can be predicted using the method.

In Table 3, errors over the whole test set [see Eq. (11)], for each architecture and for each relative radius, are shown when the deterministic initialization is made. As it is possible to observe, the results are similar, or slightly better, than the ones showed in Table 1 obtained when different random initializations are made. In order to compare both tables more easily, values in Table 3 are written in boldface when they are lower than the corresponding values in Table 1.

Figure 2 shows graphically the results corresponding to Table 3. We can see that when the number of neurons is big enough (more than 11 neurons), the error decreases and when r_r is 0.1 or bigger the error does not change significat vely as the radius increases. This behavior is explained as follows: as the radius grows up, more patterns are selected allowing the network to perform better with the test set. If the number of neurons is very small, the network can not generalize properly when the number of training patterns is high, and that is why the error increases when the radius gets bigger. If the number of neurons is higher, the network can fi the training set, even if the number of patterns is high, keeping the value of the error relatively constant.

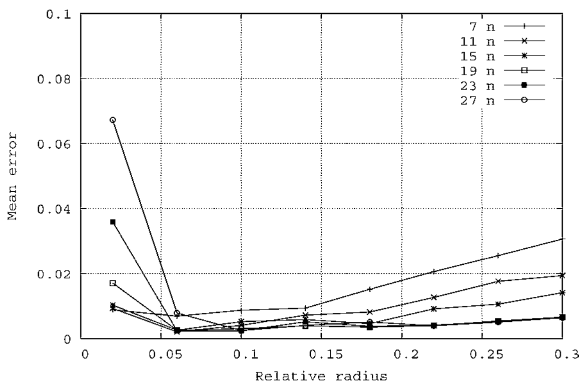

Fig. 2. Errors with deterministic initialization. Piecewise-define function.

Table 3
Errors with deterministic initialization of centers. Piecewise-define function

r_r	Hidden neurons						NP	% PP
	7	11	15	19	23	27		
0.02	**0.00883**	**0.00927**	**0.01027**	0.01707	0.03594	0.06726	0	100
0.06	**0.00690**	**0.00208**	**0.00257**	**0.00255**	**0.00237**	0.00785	0	100
0.1	**0.00872**	0.00407	0.00519	0.00302	**0.00237**	**0.00246**	0	100
0.14	**0.00938**	0.00719	0.00583	0.00391	0.00516	0.00404	0	100
0.18	0.01515	**0.00816**	**0.00459**	**0.00351**	**0.00375**	0.00507	0	100
0.22	0.02060	0.01266	0.00914	**0.00398**	**0.00415**	**0.00393**	0	100
0.26	0.02550	0.01760	**0.01059**	**0.00541**	**0.00518**	0.00505	0	100
0.3	0.03061	**0.01937**	0.01411	**0.00660**	0.00659	0.00640	0	100

3.2.2. An artificia time series prediction problem: The Mackey-Glass time series

The proposed LRBNN method has been applied to this artificia time series, where – in the same way as in the previous domain – RBNN of different architectures are used, varying the relative radius from 0.04 to 0.24. As in the previous problem, the aim of these experiments consists of studying the influenc of the relative radius on the generalization ability of the networks, when the centers of the neurons are deterministically initialized. In order to evaluate this deterministic initialization and compare it with the usual (random) one, we have also made experiments with the lazy learning approach when the neurons centers are randomly initialized. Ten runs have been made and the mean values of the corresponding errors [see Eq. (11)] are shown in Table 4. Table 5, shows the corresponding standard deviations for these values.

On the other hand, when the proposed deterministic initialization is applied, the obtained results are shown in Table 6. As in the previous domain, values lower than the corresponding errors in Table 4 are written in boldface. We can observe that the error values are slightly better than the ones obtained when the neurons centers were randomly located.

As in the piecewise-define function, the column named “NP” displays the number of “null patterns”, that is, test patterns for which the number of selected training patterns is zero. Opposite to the previous problem, when the relative radius is small (0.04) there are some test patterns with empty selected training sets. This situation arises because of the dimensionality of the problem and the non-uniform distribution of data. This can be explained as follows: as the dimensionality of the problem grows up, the data gets more and more scattered. Besides, since data density is not uniform in the input space, it is very likely to fin these anomalous situations into the low-density regions. The “PP” column displays the “Predicted Patterns” percentage, that is the percentage of test patterns that are correctly answered. As it is shown, when $r_r = 0.04$, there are 45 test patterns for which the networks can not make a prediction because the associated training sets are empty. Thus, these test patterns are discarded, corresponding the error values to the rest of patterns, that is, to the 91% of the whole test set.

As it was explained in Section 2, two alternative ways of treating these anomalous patterns are presented. *Method (a)* keeps the local approach, by find ing the closest training pattern to the novel one and selecting the training examples belonging to its neighborhood. On the contrary, *Method (b)* renounce to the local approach and follows a global one, assuming that the whole training set must be taken into account since no training patterns are situated in the novel pattern neighborhood.

Our learning method must give a prediction for every novel pattern, that is, we must guarantee a 100% of predicted patterns. With the aim of studying the performance of both approaches, the global and the local one, RBNN of different architectures are trained when a relative radius of 0.04 is taken. Both *Method (a)* and *Method (b)* have been applied and the obtained error values are shown in Table 7. Of course, a deterministic initialization of the K-means algorithm has been done.

The results showed in Tables 6 and 7 are merged and graphically represented in Fig. 3 in the following way: the figur on the left shows the errors obtained when *Method (a)* is applied if null test patterns are found. The figur on the right corresponds to *Method (b)*. In all cases, the 100% of the test patterns are predicted. Both figure are very similar because they only differ when the relative radius is 0.04 and, in this case, only 45 null patterns are found; that is, methods (a)

Table 4
Mean errors with random initialization of centers. Mackey-Glass time series

r_r	Hidden neurons						NP	% PP
	7	11	15	19	23	27		
0.04	0.02527	0.02641	0.02683	0.02743	0.02691	0.02722	45	91
0.08	0.02005	0.01891	0.01705	0.01571	0.01716	0.01585	0	100
0.12	0.02379	0.01954	0.01792	0.01935	0.01896	0.01940	0	100
0.16	0.02752	0.02223	0.01901	0.02106	0.02228	0.02263	0	100
0.2	0.03031	0.02427	0.02432	0.02287	0.02281	0.02244	0	100
0.24	0.03422	0.02668	0.02627	0.02482	0.02635	0.02798	0	100

Table 5
Standard deviations of errors corresponding to ten random initialization of centers. Mackey-Glass time series

r_r	Hidden neurons					
	7	11	15	19	23	27
0.04	0.001272	0.001653	0.002001	0.001148	0.001284	0.001280
0.08	0.001660	0.000503	0.001205	0.000900	0.002154	0.001906
0.12	0.001334	0.001387	0.001583	0.001703	0.000986	0.001324
0.16	0.001495	0.001847	0.002146	0.000844	0.000969	0.001327
0.2	0.001423	0.001160	0.001655	0.001712	0.001336	0.001218
0.24	0.001978	0.001322	0.001213	0.001775	0.001453	0.001992

Table 6
Errors with deterministic initialization of centers. Mackey-Glass time series

r_r	Hidden neurons						NP	% PP
	7	11	15	19	23	27		
0.04	0.02904	0.03086	0.03096	0.03109	0.03231	0.03295	45	91
0.08	**0.01944**	**0.01860**	**0.01666**	**0.01565**	**0.01551**	0.01585	0	100
0.12	**0.02131**	**0.01742**	**0.01644**	**0.01607**	**0.01628**	**0.01602**	0	100
0.16	**0.02424**	**0.02029**	**0.01812**	**0.01729**	**0.01783**	**0.01809**	0	100
0.2	**0.02837**	**0.02083**	**0.01927**	**0.01874**	**0.02006**	**0.02111**	0	100
0.24	**0.03082**	**0.02439**	**0.02256**	**0.02199**	**0.02205**	**0.02293**	0	100

Table 7
Errors with deterministic initialization of centers and null patterns processing ($r_r = 0.04$). Mackey-Glass time series

	Hidden neurons						NP	% PP
	7	11	15	19	23	27		
Method (a)	0.02974	0.03043	0.03132	0.03114	0.03309	0.03373	45	100
Method (b)	0.03385	0.03641	0.03545	0.03464	0.03568	0.03408	45	100

and (b) are only applied to 9% of the patterns when $r_r = 0.04$.

It is possible to observe that the performance of the networks is scarcely influence by the value of the relative radius when it is bigger than a certain value and the number of neurons is big enough. The error decreases with the radius until $r_r = 0.08$, and then it maintains its value nearly constant as the radius increases if the number of neurons is bigger than 7. Thus, the relative radius is not a critical parameter if the number of neurons is bigger than 7 and the relative radius is bigger than 0.08. When the number of neurons is small, the performance of the networks gets worse as the radius increases. This is explained because the number of training patterns selected is very big and the number of neurons of the RBNN are insufficien to fi such training set.

With respect to the null patterns treatment, in Table 7 we can see that method (b) behaves slightly worse than method (a) in all the cases. Thus, when a local

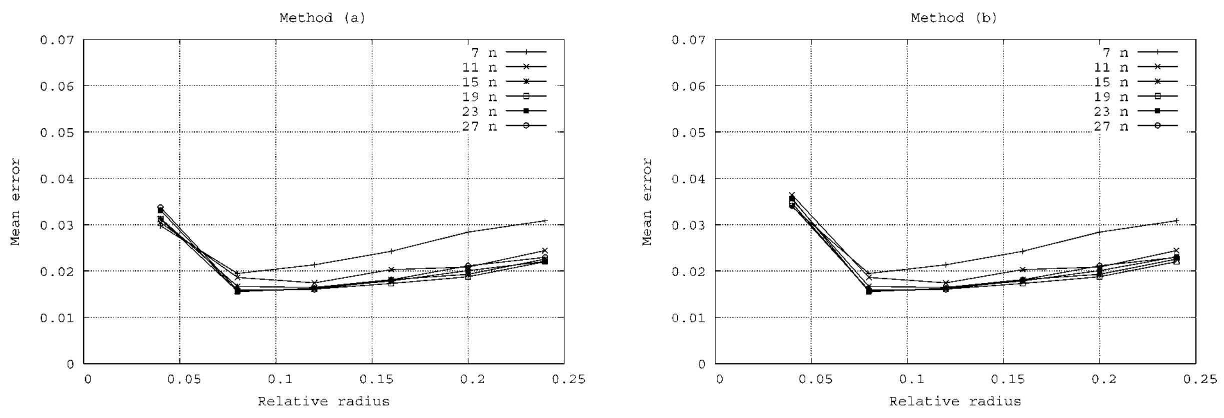

Fig. 3. Errors with deterministic initialization and null patterns processing. Left: Method (a), right: Method (b). Mackey-glass time series.

approach is taken, the method gets better results than when all the available patterns are used to train the networks. Besides, it is important to recall that no random initializations are necessary to obtain the centers positions of the neurons because the method provides a way to fi their initial positions.

3.2.3. *A real time series prediction problem: The Venice Lagoon time series*

The local learning method described in Section 2 has also been applied to the Venice Lagoon time series domain; RBNN with different architectures are trained and the relative radius has been fi ed to different values from 0.04 to 0.2. As in the previous domains, two sets of experiments have been done: the firs one corresponds to the usual random K-means initialization; in order to obtain representative results, ten runs of the method have been carried out and the mean values and standard deviations of the results are showed in Tables 8 and 9.

The second set of experiments, carried out only once, corresponds to the deterministic initialization of the neurons centers. The results are displayed on Table 10. As it happened on the previous domains, when the deterministic initialization of the centers is done, the results are similar or slightly better than when the centers are randomly located (the values written in boldface correspond to errors lower than the corresponding ones in Table 8).

We can observe that there are null patterns even when the relative radius grows to 0.08. When $r_r = 0.04$, 34 test patterns can not be predicted. Thus, only a 96.63% of the test set can be properly predicted. And still for $r_r = 0.08$, 3 patterns are not predicted.

Table 11 shows the errors obtained when both methods (a) and (b) are applied if null patterns are found. It is important to realize that, although it seems that the results are worse than those seen on Table 10, a 100% of the test patterns are properly predicted.

In Fig. 4 (left and right), errors are separately showed, depending on which method is used, when null patterns are found. The figur on the left shows the errors obtained when the method (a) is applied if null test patterns are found and the figur on the right shows the results corresponding to method (b). In this domain, the differences between both figure are significant specially when the relative radius is 0.04. In this case, 34 null patterns are found, that is, 3.36% of the whole test set. We can appreciate that method (a) achieves lower errors that method (b). Thus, when a lazy learning approach is applied the result is better than when the RBNN are trained with all the available training patterns.

As in previous cases, it is possible to observe that when the relative radius is small errors are high due to the shortage of selected training patterns; besides, as the relative radius increases, the error decreases and then it does not change significat vely. Thus, as it happened with the previous domains, the relative radius is not a critical parameter if the number of neurons and the relative radius are big enough.

3.3. *Comparison of LRBNN with global and lazy methods*

The proposed method has been compared in two different ways. First, LRBNN has been compared with RBNN trained in a traditional (global) way. And, second, since our method is based on a lazy strategy, it has also been compared with traditional lazy learning methods.

Table 8
Mean errors with random initialization of centers. Venice Lagoon time series

r_r	Hidden neurons						NP	% PP
	7	11	15	19	23	27		
0.04	0.01840	0.01881	0.01888	0.01951	0.02053	0.02080	34	96.63
0.08	0.02000	0.01950	0.01859	0.01841	0.01919	0.01985	3	99.70
0.12	0.02043	0.01702	0.01621	0.01627	0.01681	0.01778	0	100
0.16	0.01891	0.01569	0.01587	0.01618	0.01684	0.01770	0	100
0.2	0.02067	0.01597	0.01573	0.01617	0.01724	0.01799	0	100
0.24	0.02468	0.01632	0.01612	0.01667	0.01733	0.01771	0	100

Table 9
Standard deviations of errors corresponding to ten random initialization of centers. Venice Lagoon time series

r_r	Hidden neurons					
	7	11	15	19	23	27
0.04	0.000379	0.001171	0.001591	0.002235	0.001515	0.001074
0.08	0.001284	0.001573	0.001360	0.001667	0.000905	0.000940
0.12	0.001350	0.001505	0.002025	0.001658	0.001191	0.002127
0.16	0.000976	0.001553	0.001376	0.000971	0.001706	0.001512
0.2	0.001652	0.001137	0.001052	0.000871	0.001388	0.000783
0.24	0.001142	0.000894	0.000819	0.001970	0.001793	0.001952

Table 10
Errors with deterministic initialization of centers. Venice Lagoon time series

r_r	Hidden neurons						NP	% PP
	7	11	15	19	23	27		
0.04	0.01896	0.01905	0.01917	0.02016	0.02010	0.02093	34	96.63
0.08	0.02063	**0.01933**	**0.01800**	**0.01802**	**0.01754**	0.01910	3	99.70
0.12	**0.02019**	**0.01683**	**0.01563**	**0.01539**	**0.01555**	**0.01562**	0	100
0.16	0.01893	0.01602	**0.01418**	**0.01441**	**0.01477**	**0.01568**	0	100
0.2	0.02123	**0.01483**	**0.01423**	**0.01416**	**0.01449**	**0.01597**	0	100
0.24	**0.02435**	**0.01607**	**0.01466**	**0.01426**	**0.01514**	**0.01511**	0	100

Table 11
Errors with deterministic initialization of centers and null patterns processing. Venice Lagoon time series

r_r	Meth	Hidden neurons						NP	% PP
		7	11	15	19	23	27		
0.04	(a)	0.02139	0.02186	0.02291	0.02347	0.02285	0.02407	34	100.00
0.04	(b)	0.02596	0.02468	0.02453	0.02481	0.02610	0.02762	3	100.00
0.08	(a)	0.02063	0.02061	0.01981	0.01890	0.01959	0.01900	0	100.00
0.08	(b)	0.02100	0.02108	0.02034	0.01953	0.02071	0.02003	0	100.00

3.3.1. LRBNN versus global RBNN

In order to compare the lazy learning strategy (LRBNN) with the traditional one, RBNN with different number of hidden neurons have been trained, in a global way, using the whole training data set in order to build a global approximation. When RBNN are trained as usual, the standard K-means algorithm is used and several experiments with different initial centers locations are made. In Table 12, the test errors [Eq. (11)] obtained for different application domains are shown. Results corresponding to RBNN with less than 10 neurons are not shown because they are worse than the ones showed on the table.

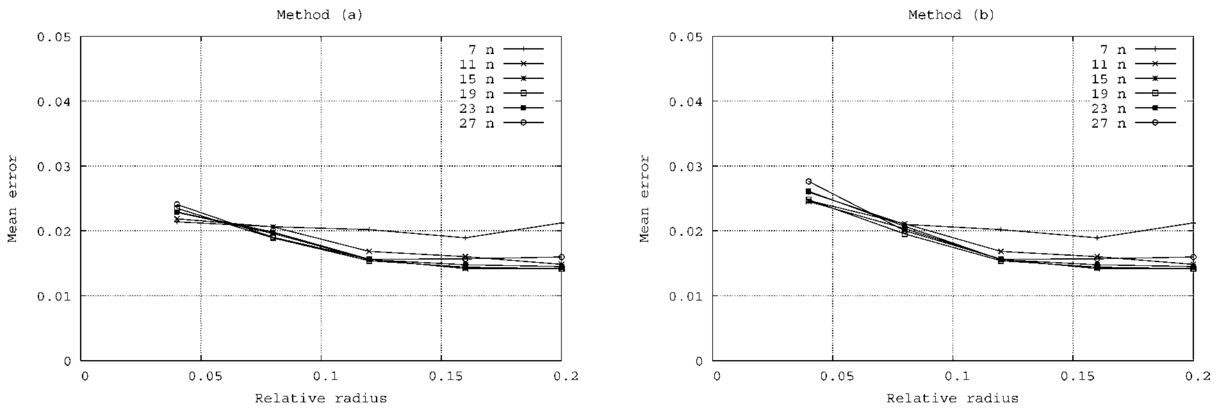

Fig. 4. Errors with deterministic initialization and null patterns processing. Left: Method (a), right: Method (b). Venice Lagoon time series.

Table 12
Errors with traditional learning of RBNN

Hidden neurons	Piecewise-define function	Mackey-Glass time series	Venice Lagoon time series
10	0.15291	0.13296	0.14397
20	0.06766	0.13556	0.09021
30	0.05510	0.12714	0.06257
40	0.04666	0.12768	0.06048
50	0.05287	0.11229	0.05086
60	0.04803	0.10520	0.05036
70	0.04312	0.12740	0.05894
80	0.04418	0.11154	0.06833
90	0.04405	0.11771	0.08715
100	0.04156	0.11628	0.09843
110	0.04496	0.10273	0.08937
120	0.04361	0.11144	0.10303
130	0.04281	0.12768	0.10969

In Table 13, the best results obtained in the different domains for both methods, lazy and traditional ones, are shown. As it is possible to observe, in all application domains the performance of RBNN is significa tively improved when a weighted selection of training patterns is made. In all cases, as we have already commented, the value of the parameters (number of hidden neurons and relative radius) in LRBNN are not critical because for all the domains, if the relative radius and the number of neurons is big enough, the performance of the local method is significat vely better than the performance of the traditional learning approach.

Although the best results for both approaches are shown in Table 13, they are the mean values for all the patterns of the corresponding test set. It is interesting to show the errors committed by both methods for each test pattern. Figures 5, 6 and 7 show these results, corresponding to the situations indicated in Table 13.

Figure 5 displays the errors for each test pattern of the piecewise-define function for both learning methods.

It is possible to observe that, for the majority of patterns, the error is smaller when the LRBNN method is used. Most of the test patterns of the piecewise-define function can be more accurately approximated when the RBNN is trained with an appropriate selection of patterns – the most relevant examples – instead of the whole training set.

In Fig. 6 the errors per test pattern corresponding to the Mackey-Glass time series are shown. As in the previous case, although the mean error comparison shows that the LRBNN method behaves better than the usual one, it is interesting to verify that this better behavior occurs for the majority of the test patterns.

Figure 7 shows the errors committed by the different learning strategies for each test pattern of the Venice-lagoon time series domain. As in the previous cases, most of the test patterns are better approximated when LRBNN is used. The error, when the network is trained in the traditional way is significantl higher, for the majority of patterns, than the corresponding to the lazy learning method, when an appropriate selection of patterns is made.

3.3.2. LRBNN versus traditional lazy learning methods

We have also compared LRBNN with some of the most well-known lazy methods: K-nearest neighbor, weighted k-nearest neighbor and local linear regression methods [2]. The local linear regression method is specially interesting because is similar to the method proposed in [3], where a linear model is trained with the k nearest examples to the query.

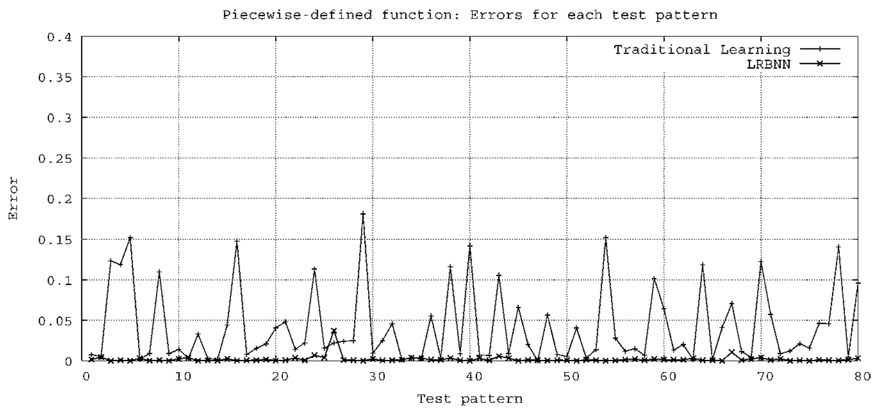

Fig. 5. Piecewise-define function: Errors for each test pattern.

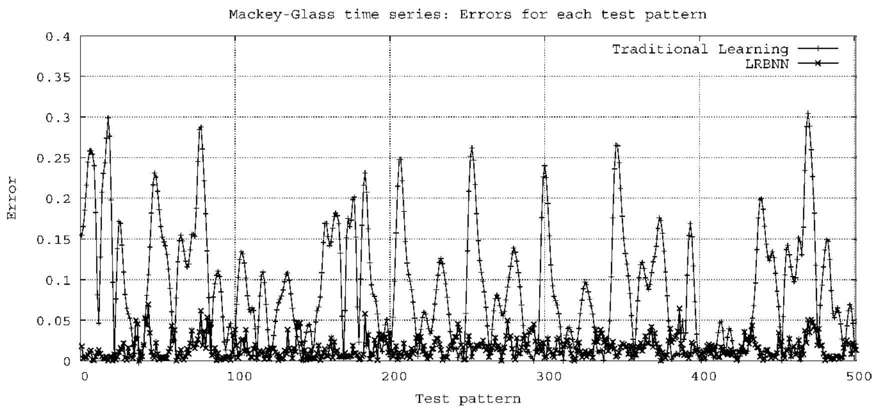

Fig. 6. Mackey-Glass time series: Errors for each test pattern.

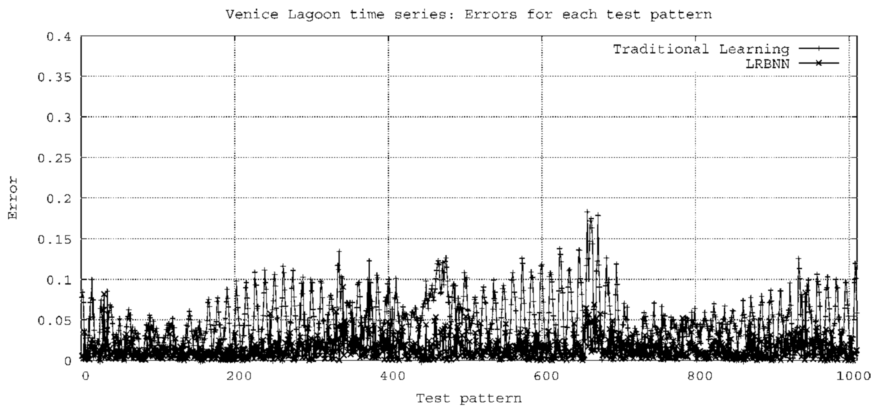

Fig. 7. Venice Lagoon time series: Errors for each test pattern.

Table 13
LRBNN versus traditional learning of RBNN

	Piecewise-define function	Mackey-Glass time series	Venice Lagoon time series
LRBNN	0.00208 $r_r = 0.06$, 11 neurons	0.01551 $r_r = 0.08$, 15 neurons	0.01416 $r_r = 0.2$, 19 neurons
Traditional method	0.04156 100 neurons	0.10273 110 neurons	0.05036 60 neurons

Table 14
Best results for LRBNN, k-NN, Weighted k-NN and Local linear regression methods

	Piecewise-define function	Mackey-Glass time series	Venice Lagoon time series
LRBNN	0.00208 $r_r = 0.06$, 11 neurons	0.01551 $r_r = 0.08$, 15 neurons	0.01416 $r_r = 0.2$, 19 neurons
kNN	0.01113 $k = 2$	0.02731 $k = 3$	0.02072 $k = 8$
Weighted kNN	0.00793 $k = 4$	0.02403 $k = 6$	0.01915 $k = 6$
Local linear regression	0.02513 $k = 3$	0.02747 $k = 3$	0.02124 $k = 4$

The different lazy methods have been run for different values of k parameter (number of patterns selected). For the piecewise-define function, k is varied from 1 to 25; for the Mackey-Glass time series k is varied from 1 to 50 and for the Venice Lagoon time series k is varied form 1 to 75, because more data are available. In Table 14, the best errors obtained by these methods together with those obtained by LRBNN are shown.

It can be observed that the LRBNN method obtains better results than the classic lazy techniques for all the domains. This is due to two main reasons. Firstly, the lazy strategy proposed in this work selects a different number of training examples depending on the location of the query in the input space, whereas the other ones select k patterns for all the queries. Secondly, a non-linear approximation is used to fi the training examples selected, which can be an advantage in same cases. Moreover, the classic lazy techniques depend highly on the k parameter, whereas the LRBNN method does not depend significantl on the radius, as it has been shown in previous sections.

4. Conclusions

Global learning methods estimate the target function once for the whole instance space. They build a general and explicit approximation that allows to forecast all the test patterns, no matter the characteristics those examples have. On the other hand, local methods, instead of estimating the target function for the entire instance space, estimate it locally and differently for each new query instance.

Usually, the input space has a non-homogeneous structure, being data points unevenly distributed in the input space. In these cases, where the target function is very complex, the accuracy of global methods could be affected and local methods might be more appropriate, allowing to describe the complex target function as a collection of less complex local approximations.

However, local learning methods have to deal with some drawbacks. They are usually based on the assumption that all the test patterns have the same structure and need the same selection procedure. They assume some kind of linear behavior at a local scale leading to a high dependency on the number of examples selected and on the metric used, being frequent the situations where an Euclidean metric might not be appropriate.

Although some methods that combine local and non-local strategies have produced good results in some domains, they still assume that Euclidean distance is an appropriate metric and consider that all the test patterns have the same structure and need the same selection procedure. Other domains would require specifi and non-linear behaviors for different regions of the input space.

We try to complement the good characteristics of local and non-local approaches by using a lazy learning method for selecting the training set, but using RBNN for making predictions. RBNN have some advantages: they are universal approximators and therefore the assumption of local linear behavior is no longer needed; besides, their training is very fast, without increasing significat vely the computational cost of standard local learning approaches. We have presented a method (LRBNN) that can get the locality of the input space, and then uses a non-linear method to approximate each region of the input space. In addition, the selection of patterns is made using a kernel function, taking into account the distribution of data.

When a lazy learning strategy is used, two important aspects related to RBNN training and patterns selection have been taken into account. In the firs place, the initialization of the neurons centers is an important factor that influence RBNN performance. Usually, the initial location of centers are randomly established, but in a lazy strategy, in which a network must be trained for each new query, random initialization must be avoided. For this reason, in this work the algorithm has been modifie in a deterministic way to eliminate any initial condition influenc with the objective of achieving the best performance; we propose a way to determine the initial location of the neurons centers, depending on the location of the closest training examples selected. Regarding to the selection procedure, in which the Inverse kernel function is used, it may occur that no training pattern is selected for certain test patterns, due to the distribution of data in the input space. We have proposed and validated two different approaches to treat this problem.

LRBNN has been applied to three different domains: an artificia regression problem, and two time series prediction problems, an artificia one (the well known Mackey-Glass time series) and a real one (representing the Venice Lagoon water level). For all domains, we present the results obtained by LRBNN when a deterministic centers initialization is made. Besides, with the aim of showing the advantages of this deterministic initialization, the same method is applied but the RBNN are trained with a random initialization of their centers. We show the mean results of several random initializations. As we said before, LRBNN provides two alternative ways of guarantying the selection of training examples for all the query instances. When the use of these alternative methods is necessary, the obtained results are also showed.

The results obtained by LRBNN improves signifi cantly the ones obtained by RBNN trained in a global way and those obtained by the classic lazy techniques. Besides, the proposed deterministic initialization of the neurons centers produces similar or slightly better results than the usual random initialization, being thus preferable because only one run is necessary. Moreover, the method is able to predict 100% of the test patterns, even in those extreme cases when no train examples would be selected using the normal selection method. The experiments show that the relative radius, parameter of the method, is not a critical factor because if it reaches a minimum value and the network has a sufficien number of neurons, the error on the test set keeps its low value relatively constant.

Thus, we can conclude that the combination of lazy learning and RBNN, can produce significan improvements in some domains.

Acknowledgements

This article has been finance by the Spanish Ministry of Education. Research project OP:LINK, Ref:-TIN2005-08818-C04-02.

References

[1] D.W. Aha, D. Kibler and M.K. Albert, Instance-based learning algorithms, *Machine Learning* **6**(1991), 37–66.

[2] C.G. Atkenson, A.W. Moore and S. Schaal, Locally weighted learning, *Artificia Intelligence Review* **11** (1997), 11–73.

[3] L. Bottou and V. Vapnik, Local learning algorithms, *Neural Computation* **4**(6) (1992), 888–900.

[4] B.V. Dasarathy, *Nearest Neighbour(NN) Norms: NN Pattern Classificatio Techniques*, IEEE Computer Society Press, 1991.

[5] J. Ghosh and A. Nag, *An Overview of Radial Basis Function Networks*, R.J. Howlett and L.C. Jain, eds, Physica Verlag, 2000.

[6] M.C. Mackey and L. Glass, Oscillation and chaos in physiological control systems, *Science* **197** (1977), 287–289.

[7] A. Michelato, R. Mosetti and D. Viezzoli, statistical forecasting of strong surges and application to the lagoon of Venice, *Boll. Ocean. Teor. Appl.* **1** (1983), 67–83.

[8] J.E. Moody and C. Darken, Fast learning in networks of locally tuned processing units, *Neural Computation* **1** (1989), 281–294.

[9] E. Moretti and A. Tomasin, Un contributo matematico all-elaborazione previsionale dei dati di marea a Venecia, *Boll. Ocean. Teor. Appl.* **1** (1984), 45–61.

[10] J. Park and I. W. Sandberg, Universal approximation and radial-basis-function networks, *Neural Computation* **5** (1993), 305–316.

[11] J. Platt, A resource-allocating network for function interpolation, *Neural Computation* **3** (1991), 213–225.

[12] T. Poggio and F. Girosi, Networks for approximation and learning, *Proc. IEEE* **78** (1990), 1481–1497.

[13] A. Tomasin, A computer simulation of the Adriatic Sea for the study of its dynamics and for the forecasting of flood in the town of Venice, *Comp. Phys. Comm.* **5** (1973), 51.

[14] J.M. Valls, I.M. Galván and P. Isasi, Lazy learning in radial basis neural networks: a way of achieving more accurate models, *Neural Processing Letters* **20** (2004), 105–124.

[15] G. Vittori, On the chaotic features of tide elevation in the lagoon Venice, in: *Proc. of the ICCE-92, 23rd International Conference on Coastal Engineering*, 1992, pp 4–9.

[16] D. Wettschereck, D.W. Aha and T. Mohri, A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, *Artificia Intelligence Review* **11** (1997), 273–314.

[17] D. Wettschereck and T. Dietterich, Improving the performance of radial basis function networks by learning center locations, *Advances in Neural Information Processing Systems* **4** (1992), 1133–1140.

[18] B.A. Whitehead and T.D. Choate, Cooperative – competitive genetic evolution of radial basis function centeres and widths for time series prediction, *IEEE Transactions on Neural Networks* **5** (1995), 15–23.

[19] L. Yingwei, N. Sundararajan and P. Saratchandran, A sequential learning scheme for function approximation using minimal radial basis function neural networks, *Neural Computation* **9** (1997), 461–478.

[20] J.M. Zaldívar, E. Gutiérrez, I.M. Galván, F. Strozzi and A. Tomasin, Forecasting high waters at Venice Lagoon using chaotic time series analysis and nonlinear neural networks, *Journal of Hydroinformatics* **2** (2000), 61–84.