Neurocomputing 71 (2008) 2529–2537

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Learning radial basis neural networks in a lazy way: A comparative study

José M. Valls *, Inés M. Galván, Pedro Isasi

Computer Science Department, Carlos III University, Avenida de la Universidad, 30, 28911 Leganés, Madrid, Spain

ARTICLE INFO

Available online 10 May 2008

Keywords:
Lazy learning
Local learning
Radial basis neural networks
Pattern selection

ABSTRACT

Lazy learning methods have been used to deal with problems in which the learning examples are not evenly distributed in the input space. They are based on the selection of a subset of training patterns when a new query is received. Usually, that selection is based on the k closest neighbors and it is a static selection, because the number of patterns selected does not depend on the input space region in which the new query is placed. In this paper, a lazy strategy is applied to train radial basis neural networks. That strategy incorporates a dynamic selection of patterns, and that selection is based on two different kernel functions, the Gaussian and the inverse function. This lazy learning method is compared with the classical lazy machine learning methods and with eagerly trained radial basis neural networks.

1. Introduction

Lazy learning methods [1,2,15] are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions. These learning algorithms defer the decision of how to generalize beyond the training data until a new query is encountered. When the query instance is received, a set of similar related patterns is retrieved from the available training patterns set and is used to approximate the new instance. Similar patterns are chosen using a distance measured with nearby points having high relevance.

The lazy methods, also called local learning methods, generally work by selecting the k least distant input patterns from the query points, often in terms of the Euclidean distance. Afterwards, a local approximation using the selected samples is carried out with the purpose of generalizing the new instance. That local approximation can be constructed using different strategies. The most basic form is the k-nearest neighbor (NN) method [4]. In this case, the approximation of the new sample is just the most common output value among the k selected examples. A refinement of this method, called weighted k-NN [4], can be also used, which consists of weighting the contribution of each of the k neighbors according to the distance to the new query, giving greater weight to closer neighbors. Other strategy to determine the approximation of the new sample is the locally weighted linear regression [2] that constructs an explicit and linear approximation of the target function over a region around the new query instance. The regression coefficients are based on the k nearest input patterns to the new query.

When lazy learning techniques are used, the target function is represented by a combination of many local approximations constructed in the neighborhood of the new query instances. On the other hand, eager learning methods construct global approximations and the generalization is carried out beyond the training data before observing the new instance. That global approximation over the training data representing the domain could lead to poor generalization properties, mainly if the target function is complex or when data are not evenly distributed in the input space. In these cases, lazy methods could be appropriate because the complex target function could be described by a collection of less complex local approximations.

Artificial neural networks can be considered as eager learning methods because they construct a global approximation that covers the entire input space and all future query instances. Although radial basis neural networks (RBNN) [7,10] use multiple local approximations, they are also eager learning methods because the network must commit to the hypothesis before the query point is known. The local approximations they create are not specially targeted to the query point to the same degree as in lazy learning methods: RBNN are built eagerly from local approximations centered around the training samples or around clusters of training samples, but not around the unknown future query point. That could contribute to poor generalization properties of RBNN.

Bottou and Vapnik [3] introduce a lazy approach in the context of artificial neural networks. The approach is based on the selection, for each query pattern, of the k closest examples from the training set. With these examples, a linear neural network classifier is trained in order to predict the test patterns. However, the idea of selecting the k nearest patterns might not be the most appropriate, mainly because the network will always be trained with the same number of training examples for each new instance.

* Corresponding author. Tel.: +34 916248845; fax: +34 916249129.
E-mail address: jvalls@inf.uc3m.es (J.M. Valls).

0925-2312/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2007.10.030

The main idea of the lazy strategy presented in this work is to recognize from the whole training data set, the most similar patterns to each new query to be predicted. Those most similar patterns are determined by using two different weighting kernel functions: the Gaussian and the inverse function. The number of retrieved patters will depend on the new query point location in the input space and on the kernel function, but not on the k parameter as in the classical lazy techniques. Some studies with the inverse and the Gaussian selection functions can be found in [11–13].

The goal of this paper is, on one hand, the study of the influence of the patterns selected—or the kernel function used for making the selection—in the success of the network, when it is trained using a lazy strategy. On the other hand, the aim is also to compare the behavior of the lazy strategy presented in this work with the most classical lazy machine learning methods.

The weighting kernel functions assign high weights to training patterns close—in terms of Euclidean distance—to the new query instance received. They reach the maximum value when the distance to the query is null, decreasing the value smoothly as this distance increases. The amount of training patterns selected by the first kernel function, the Gaussian, depends on its width. When the width is small, the function is tight and high and few patterns are selected. If the width increases, the function becomes wider and lower, selecting more training patterns. In order to avoid the dependence on the width parameter, a second weighting kernel function, the inverse function, is evaluated. In this case, the function does not depend on any parameter but since it is convenient to have some control on the amount of training data selected, an external parameter determining the extension of the neighborhood around the novel sample is used.

The proposed lazy learning strategy to train RBNN and the different kernel functions of selecting instances are validated in different domains and compared with other lazy methods, as k-NN, weighted k-NN and locally weighted linear regression. The proposed method is also compared with RBNN trained in an eager way, that is, the whole training data set is used to train the network, constructing a global approximation of the target function.

2. Lazy learning for RBNN using different weighting functions

The general idea consists of selecting those patterns close to the new instance, in terms of the Euclidean distance. In order to give more importance to the closest examples, some weighting measure must be taken into account. There are two alternative ways of doing it: weighting the data directly, which can be viewed as replicating relevant data and discarding irrelevant or distant examples, and weighting the error criterion used by the RBNN in such a way that the neural model must fit more tightly the closest patterns [2]. Being both ways equally valid, we have chosen the first one due to its simplicity and the fact that it uses standard RBNN. 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. Next, the selection of the training patterns and details about the training of the RBNN in a lazy way are explained.

2.1. Selection of the training patterns

Let us consider $\mathbf{q}$, an arbitrary query instance, described by an n-dimensional vector and let $X = \{\mathbf{x}_k, \mathbf{y}_k\}_{k=1,\ldots,N}$ be the whole available training data set. When a new instance $\mathbf{q}$ must be predicted, a subset of training patterns, named X_q, is selected from the whole training data X.

In order to select X_q, which contains the most similar patterns to the query instance $\mathbf{q}$, Euclidean distances (d_k) from all the training samples $\mathbf{x}_k$ to $\mathbf{q}$ must be evaluated. To make the method independent on the distances magnitude, relative values must be used. Thus, a relative distance, d_{rk} is calculated for each training pattern. Let $d_{\max}$ be the maximum distance to the query instance, this is: $d_{\max} = \max(d_1, d_2, \ldots, d_N)$. Then, the relative distance is given by

$$d_{rk} = \frac{d_k}{d_{\max}}, \quad k = 1 \ldots N \tag{1}$$

The selection of patterns is carried out by establishing a weight for each training pattern, depending on its distance to the query instance. That weight is calculated using a kernel function which reaches its maximum value when the distance to the query point is null and decreases smoothly as this distance increases. In this work we have considered and compared two different kernel functions that fulfill the above conditions: the Gaussian and the inverse function.

- *Weighted selection of patterns using the Gaussian function*: The Gaussian function assigns to each training pattern $\mathbf{x}_k$ a real value or weight according to

$$K(\mathbf{x}_k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-d_{rk}^2/2\sigma^2}, \quad k = 1 \ldots N \tag{2}$$

  where σ is a parameter which indicates the width of the Gaussian function, and d_{rk} is the relative Euclidean distance from the query to the training input pattern $\mathbf{x}_k$.
  The weight values $K(\mathbf{x}_k)$, calculated in (2), are used to indicate how many times the training pattern $(\mathbf{x}_k, \mathbf{y}_k)$, will be included into the training subset associated to the new instance $\mathbf{q}$. Hence, those real values must be transformed into natural numbers. The most intuitive way consists in taking the integer part of $K(\mathbf{x}_k)$. Thus, each training pattern will have an associated natural number, $n_k = int(K(\mathbf{x}_k))$, which indicates how many times the pattern $(\mathbf{x}_k, \mathbf{y}_k)$ is included in the subset X_q. If $n_k = 0$ then the k-th pattern is not selected, and not included in the set X_q.
- *Weighted selection of patterns using the inverse function*: One problem that arises with the Gaussian function is its high dependence on the parameter σ. For this reason, another weighting function, the inverse function, given by Eq. (3), is used in order to compare their performance:

$$K(\mathbf{x}_k) = \frac{1}{d_{rk}}, \quad k = 1 \ldots N \tag{3}$$

  This function does not depend on any parameter, but it is important to have some control on the number of training patterns selected. For this reason, an n-dimensional sphere, centered at the test pattern, is established in order to select only those patterns placed into the sphere. 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. To make it domain independent, the sphere radius will be relative with 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.
  As it happens with the Gaussian function, the function values $K(\mathbf{x}_k)$ calculated in (3) are used to weight the selected patterns that will be used to train the RBNN. The main difference is that,

now, all the patterns placed into the sphere and only those will be selected. Thus, both the relative distance d_{rk} calculated previously and the weight value $K(\mathbf{x}_k)$ are used to decide whether the training pattern $(\mathbf{x}_k, \mathbf{y}_k)$ is selected and, in that case, how many times it will be included in the training subset X_q. Hence, they are used to generate a natural number, n_k, following the next rule:

$$\begin{aligned} &\text{if} \quad d_{rk} < r_r \quad \text{then} \\ &\qquad n_k = int(K(\mathbf{x}_k)) \\ &\text{else} \\ &\qquad n_k = 0 \end{aligned} \tag{4}$$

At this point, each training pattern in X has an associated natural number n_k (see Eq. (4)), which indicates how many times the pattern $(\mathbf{x}_k, \mathbf{y}_k)$ will be used to train the RBNN for the new instance $\mathbf{q}$. If the pattern is selected, $n_k > 0$ otherwise $n_k = 0$.

Examples of these kernel functions—Gaussian and inverse—are presented in Figs. 1 and 2. The x-axis represents the relative distance from each training example to the query, and the y-axis represents the value of the kernel function.

When the Gaussian kernel function is used (see Fig. 1), the values of the function for patterns close to $\mathbf{q}$ are high and decrease quickly when patterns are moved away. Moreover, as the width parameter decreases, tighter and higher is the Gaussian function. Hence, if the width parameter is small, only patterns which are situated very close to the new sample $\mathbf{q}$ are selected, being repeated a lot of times. However, when the width parameter is big, more training patterns are selected but they are replicated less times. It is observed that the value of σ affects the number of training patterns selected as well as the number of times they will be replicated.

If the selection is made with the inverse kernel function (see Fig. 2), the number of selected patterns only depends on the relative radius (r_r). Patterns close to the query instance $\mathbf{q}$, will be selected and repeated a lot of times. As the distance to the query $\mathbf{q}$ increases, the number of times that training patterns are replicated decreases, as long as this distance is lower than r_r (in Fig. 2, r_r has a value of 0.5). If the distance is greater than r_r they will not be selected. Fig. 2 shows that the closest patterns will always be selected and replicated the same number of times, regardless of the radius value. This behavior does not happen with the Gaussian function as it can be seen in Fig. 1.

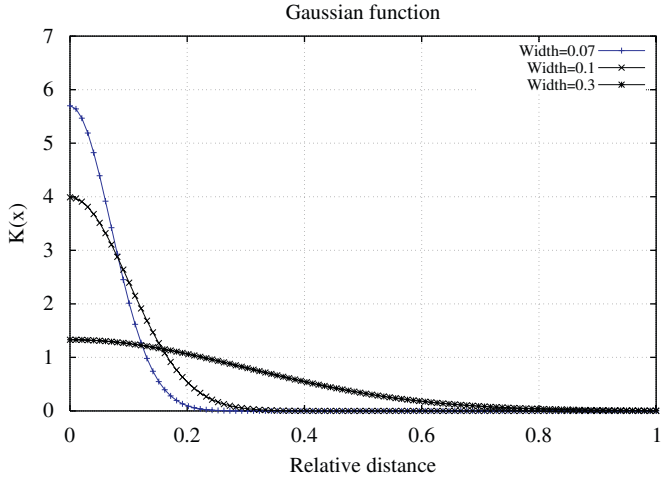


Fig. 1. Gaussian function with different widths.

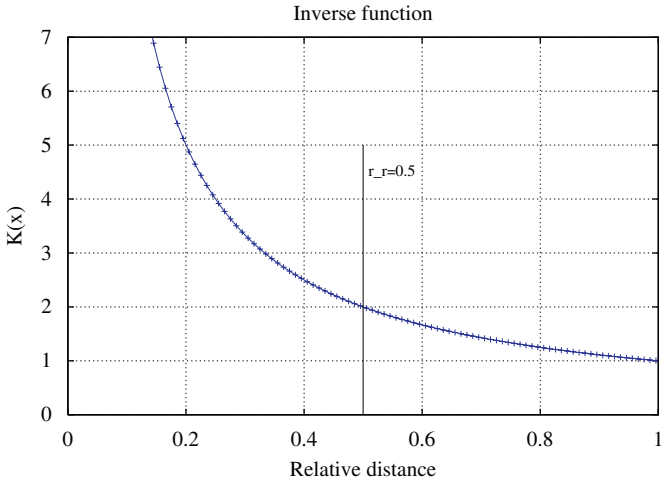


Fig. 2. Inverse function and relative radius.

2.2. Training RBNN in a lazy way

For each query instance $\mathbf{q}$, a RBNN is trained using X_q: the neuron centers are calculated in an unsupervised way using K-means algorithm in order to cluster the input training patterns included in the subset X_q. The neuron widths are evaluated as the geometric mean of the distances from each neuron center to its two nearest centers, and the RBNN weights are estimated in a supervised way in order to minimize the mean square error measured in the training subset X_q.

In order to apply the learning method to train RBNN, two features must be taken into account: On one hand, the results would depend on the random initialization of the K-means algorithm which is used to determine the locations of the RBNN centers and must be applied for each query. In the proposed method, having the objective of achieving the best performance, a deterministic initialization of the K-means algorithm, instead of the usual random one, 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 coincide with the location of the closest selected training examples. It is necessary to avoid the situations where the number of neurons is bigger than the number of selected patterns, situations that would lead to overfitting.

On the other hand, when the test pattern is located in a region of the input space where the examples are scarce, it could happen that no training examples are selected. When this situation occurs, an alternative way to select the training patterns must be taken. In our work, if the subset X_q associated to a query $\mathbf{q}$ is empty, then we apply the method of selection to the closest training pattern, as if it was the test pattern. Thus, the selected set will have, at least, one element.

3. Experimental framework

The lazy strategy described in Section 2, with either the Gaussian or the inverse kernel function, has been applied to RBNN of different architectures and the generalization capability of the networks has been measured in terms of the mean absolute error over the test data set. Four domains have been used with that purpose. The results obtained with that lazy strategy have been compared, on the one hand, with the results provided by the networks when a global approximation over the whole training data set is constructed, that is, when the network is trained as

usual. On the other hand, the results have also been compared with classic lazy methods, as k-NN, weighted k-NN and weighted local regression.

In this section, the features of the different domains and the conditions of the experiments carried out are described. Finally, the results obtained with the different lazy learning methods are presented and compared.

3.1. Domains description

Different domains have been used to compare the different lazy strategies: The approximation of a piecewise-defined function, the approximation of the Hermite polynomial, the prediction of the Mackey-Glass time series and the prediction of the water level at Venice Lagoon time series. Next, the characteristics of all of them are presented.

- *Theoretical problem*: *a piecewise-defined function approximation*. The function is given by the equation

$$f(x)=\begin{cases}-2.186x-12.864 & \text{if } -10\leqslant x<-2\\ 4.246x & \text{if } -2\leqslant x<0\\ 10e^{-0.05x-0.5}\sin((0.03x+0.7)x) & \text{if } 0\leqslant x\leqslant 10\end{cases} \tag{5}$$

  The original training set is composed of 120 input–output points randomly generated by an uniform distribution in the interval $[-10,10]$. The test set is composed of 80 input–output points generated in the same way as the points in the training set. Both sets have been normalized in the interval $[0,1]$.
- *Theoretical problem*: *the Hermite polynomial*. The Hermite polynomial is given by

$$f(x)=1.1(1-x+2x^2)e^{-(1/2)x^2} \tag{6}$$

  This domain has been widely used in RBNN literature [5,8,17]. A random sampling with an uniform distribution over the interval $[-4,4]$ is used in order to obtain 40 input–output points for the training data. The test set is composed of 200 input–output points that are generated in the same way as the points in the training set. Both sets have been normalized in the interval $[0,1]$.
- *The Mackey-Glass time series prediction*. This time series is widely regarded as a benchmark for comparing the generalization ability of RBNN [9,17,7,16]. It is a chaotic time series created by the Mackey-Glass delay-difference equation [6]:

$$\frac{dx(t)}{dt}=-bx(t)+a\frac{x(t-\tau)}{1+x(t-\tau)^{10}} \tag{7}$$

  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 $x[t+1]$ from the earlier points $(x[t],x[t-6],x[t-12],x[t-18])$. Fixing $x(0)=0$, 5000 values of the time series are generated using Eq. (7). The initial 3500 samples are discarded in order to avoid the initialization transients. One thousand data points, corresponding to the sample time between 3500 and 4499, have been chosen for the training set. The test set is composed of the points corresponding to the time interval $[4500,5000]$. All data points are normalized in the interval $[0,1]$.
- *Real life problem*: *prediction of water level at Venice Lagoon*. Unusually high tides result from a combination of chaotic climatic elements in conjunction with the more normal, periodic, tidal systems associated with a particular area. The prediction of such events have always been subjects of high interest. The water level of Venice Lagoon is a clear example of these events. That phenomenon is known as "high water". Different approaches have been developed for the purpose of predicting the behavior of sea level at Venice Lagoon [14,18]. In this work, a training data set of 3000 points, corresponding to the level of water measured each hour has been extracted from available data in such a way that both stable situations and high water situations appear represented in the set. The test set has also been extracted from the available data and it is formed by 50 samples including the high water phenomenon. A nonlinear model using the six previous sampling times seems appropriate because the goal is to predict only the next sampling time.

3.2. Experimental conditions

As it has been previously mentioned, different lazy learning strategies have been used to deal with different problems. Now, the conditions of the experiments are described. k-NN, weighted k-NN and weighted local regression methods [2] have been run for different values of k parameter (number of patterns selected), depending on the domains; the k variation ranges depend on the number of patterns of each training data set. In Table 1 the applied values of k are showed.

When RBNN are trained using a lazy strategy based on either the Gaussian or the inverse function to select the most appropriate training patterns, some conditions must be also defined. Regarding to the Gaussian kernel function, experiments varying the value of the width parameter from 0.05 to 0.3 has been carried out for all the domains. That parameter determines the shape of the Gaussian and therefore, the number of patterns selected to train the RBNN for each sample test. Those maximum and minimum values have been chosen in such a way that the shape of the Gaussian allows the selection of same training patterns, although in some specific cases no training patterns might be selected. With respect to the inverse selective learning method, and for similar reasons, different values of the relative radius have been set. It varies from 0.04 to 0.24 for all the domains. In addition, experiments varying the number of hidden neurons have also been carried out, in order to study the influence of that parameter in the performance of the method. In the rare cases where no training patterns are selected, due to the specific characteristics of the data space and the value of the selection parameter, the lazy RBNN are able to predict the test pattern, as we have explained in Section 2.2.

Finally, for comparative purposes, different architectures of RBNN have also been trained as usual, that is, the network is trained using the whole training data set. After the training, they are used to calculate their answer to the test instances, being evaluated as eager approximators. In the next section, results with the best topology are shown.

3.3. Experimental results

3.3.1. Lazy training of RBNN

Figs. 3–6 show, for the four application domains, the behavior of the lazy strategy when the Gaussian and inverse Kernel

Table 1
Classic lazy methods

Domain	k
Piecewise-defined function	1–25
Hermite polynomial	1–15
Mackey-Glass time series	1–50
Level at Venice Lagoon	1–75

Variation of k.

functions are used to select the training patterns. In those cases, the mean error over the test data is evaluated for every value of the width for the Gaussian case, and for every value of the of relative radius for the inverse one.

In these figures it is possible to observe that the performance of the lazy learning method proposed to train RBNN does not depend significantly on the parameters that determine the number of patterns selected, when the selection is based on the

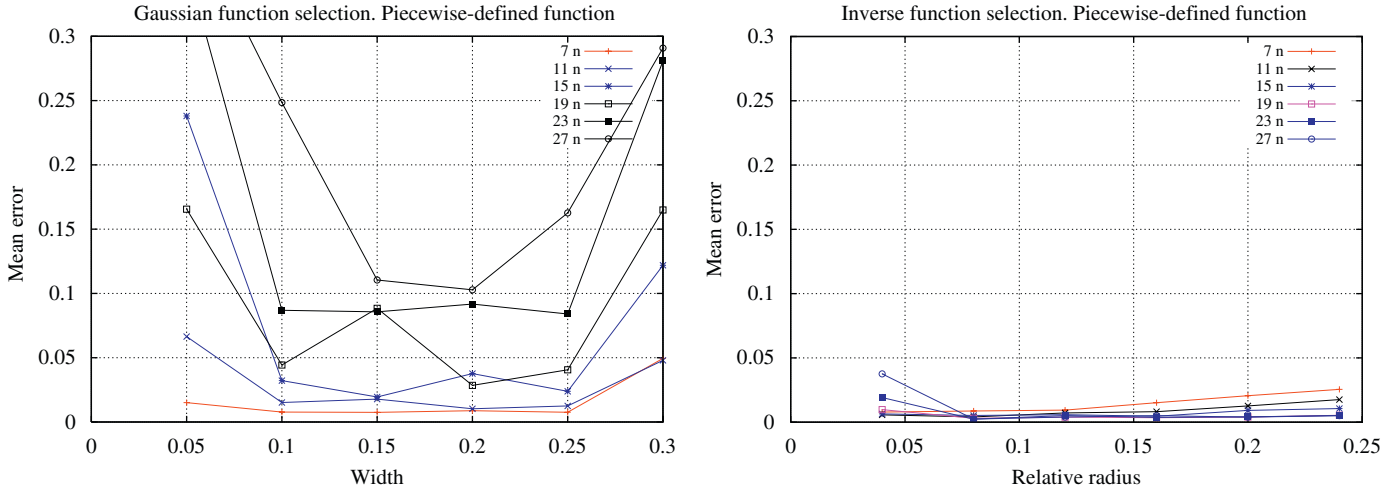


Fig. 3. Mean errors with Gaussian and inverse selection for the piecewise-defined function.

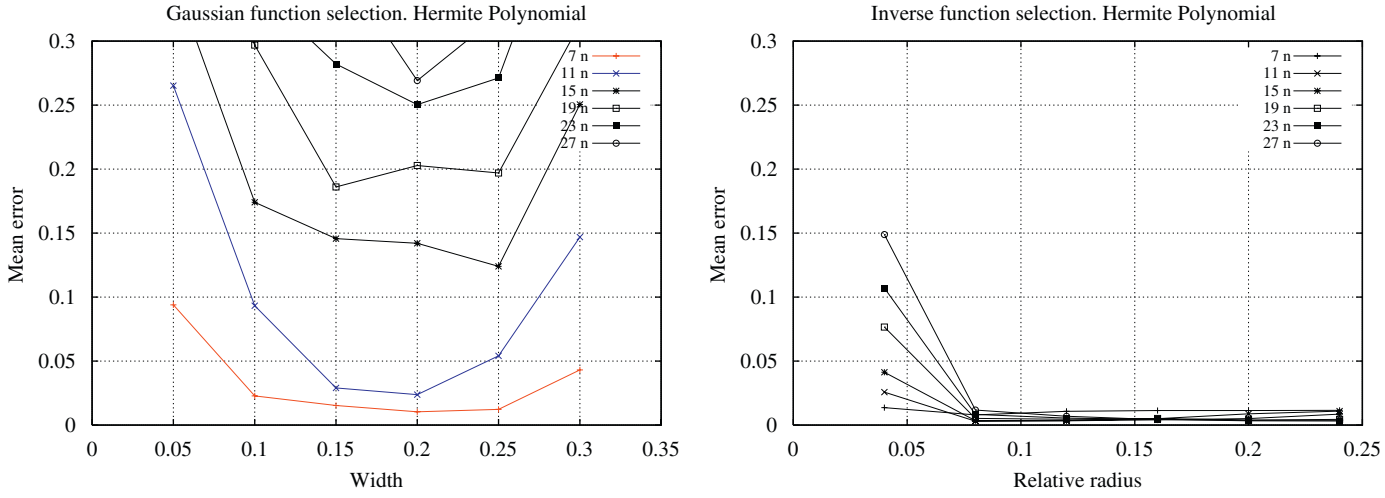


Fig. 4. Mean errors with Gaussian and inverse selection for the Hermite polynomial.

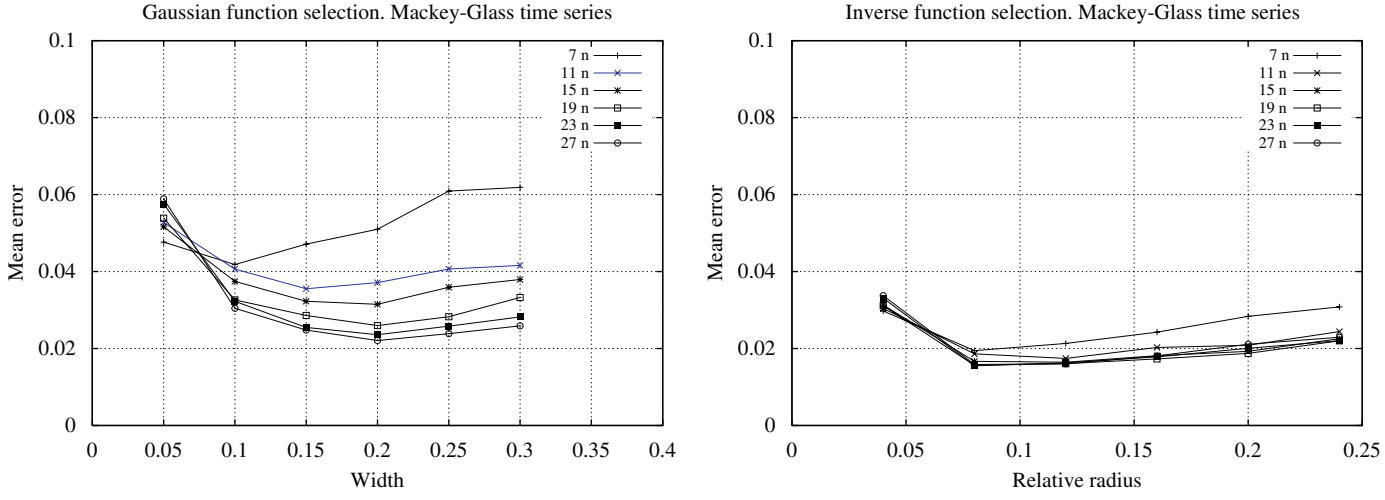


Fig. 5. Mean errors with Gaussian and inverse selection for the Mackey-Glass time series.

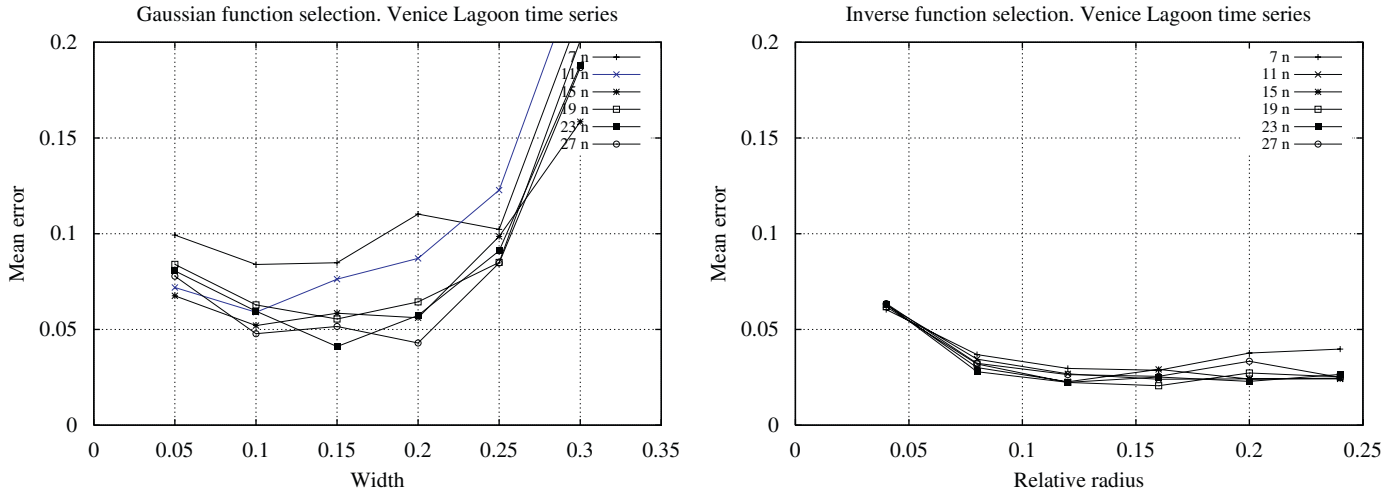


Fig. 6. Mean errors with Gaussian and inverse selection for the Lagoon level prediction.

inverse kernel function. With the inverse function and for all application domains, the general tendency is that there exists a wide interval of relative radius values, r_r, in which the errors are very similar for all architectures. Only when r_r parameter is fixed to small values, the generalization of the method is poor in some domains. This is due to the small number of selected patterns, insufficient to construct an approximation. However, as the relative radius increases, the mean error decreases and then does not change significatively. It is also observed that the number of neurons is not a critical parameter in the method.

When the lazy strategy is based on the Gaussian selection, the performance of the method presents some differences with respect to the inverse selection. Firstly, although there is also an interval of width values in which the errors are similar, if the width is also fixed to high values, the error increases. For those values, the Gaussian is more flattened and could also happen that an insufficient number of patterns are selected. Secondly, in this case, the architecture of the RBNN is a more critical factor in the behavior of the method.

3.3.2. *Traditional lazy techniques*

Figs. 7–10 show the behavior of the classical lazy strategies, *k*-NN, weighted *k*-NN and weighted local regression, in the studied domains. The mean absolute error over the test data sets for each value of *k* is represented.

Figs. 7–9 show that for the piecewise-defined function, the Hermite polynomial and the Mackey-Glass time series, the performance of classical lazy strategies is in general influenced by the value of *k*, increasing the error as the *k* value increases. This is more for the one-dimensional domains (piecewise-defined function and the Hermite polynomial). However, when weighted *k*-NN is used, the influence of the parameter *k* is lower and it obtains the best performance over the rest of traditional lazy methods.

For the prediction of the water level at Venice Lagoon problem, the behavior of the traditional lazy methods is different (see Fig. 10). On one hand, there is a stabilization of the error after a certain value of *k* when the regression method is used. On the other hand, *k*-NN algorithms have a worse behavior: the error increases as *k* increases, and even the best results are not very good. This is due to the local influence of data in the Venice Lagoon domain. In consequence, it seems that the performance of the traditional lazy approaches depends on the domain of application. Weighted *k*-NN would be appropriate for some

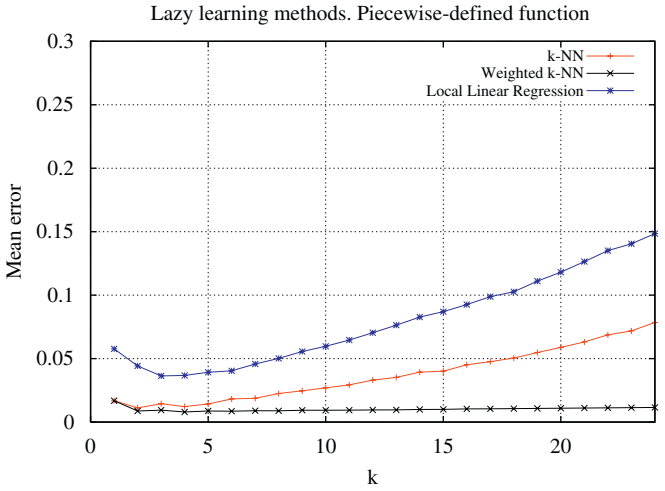


Fig. 7. Mean errors with *k*-NN, weighted *k*-NN and linear regression for the Piecewise-defined function.

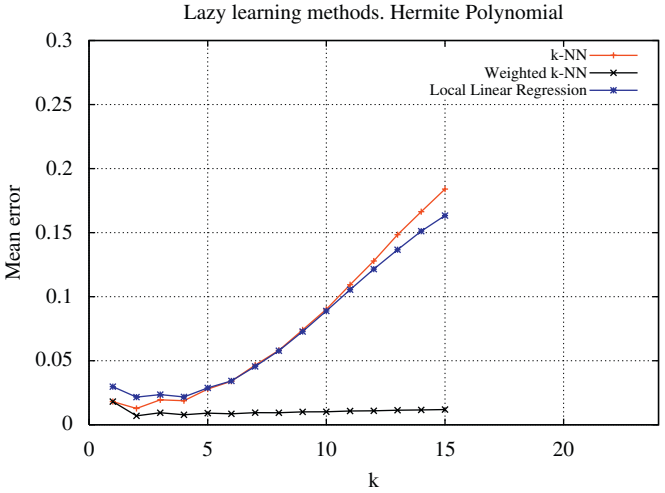


Fig. 8. Mean errors with *k*-NN, weighted *k*-NN and linear regression for the Hermite polynomial.

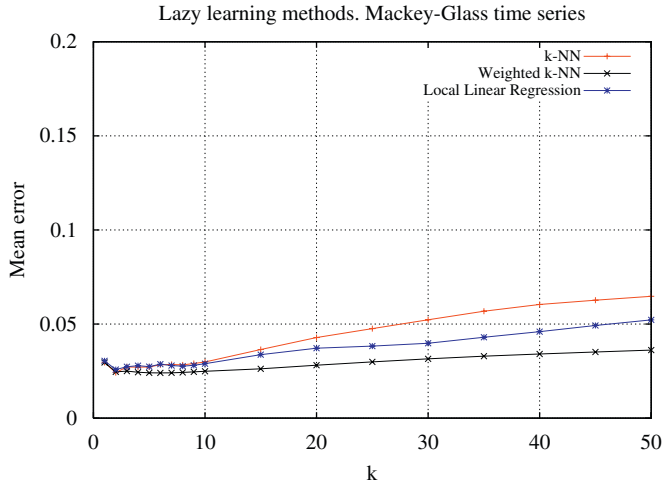


Fig. 9. Mean errors with *k*-NN, weighted *k*-NN and linear regression for the Mackey-Glass time series.

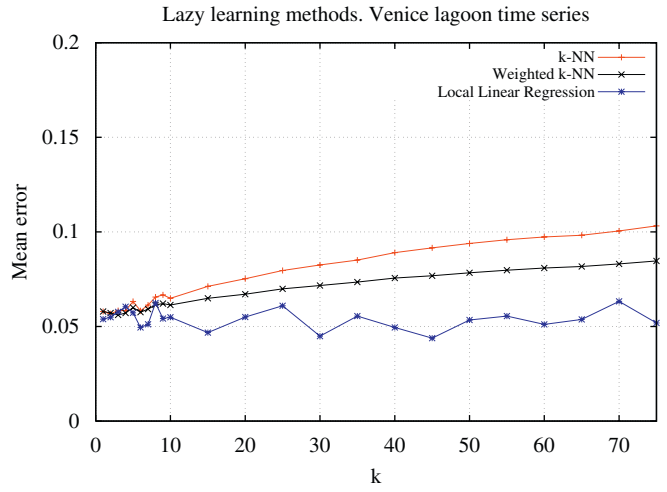


Fig. 10. Mean errors with *k*-NN, weighted *k*-NN and linear regression for the Venice Lagoon level prediction.

domains, whereas local linear regression may provide better results in other situations, in which input patterns have local dependencies.

3.3.3. *Comparative analysis*

Comparing the results obtained by the classical lazy techniques (see Figs. 7–10) with those obtained by the lazy strategy presented in this work (see Figs. 3–6), it is possible to observe, from the point of view of the general tendency of the methods, that the proposed method, with the inverse kernel function, seems to be less domain dependent and less sensitive to the parameters. For the inverse selection, the parameter that controls the number of selected patterns is not a critical factor in the behavior of the method, being more crucial the value of *k* for the classical strategies and the width value for the Gaussian selection.

Different RBNN architectures, from 10 to 130 neurons, have been trained for all the domains in an eager way, using the whole training data set. The best mean errors over the test set for all the domains obtained by the different methods (lazy RBNN, eager RBNN, traditional lazy methods) are shown in Tables 2 and 3. The training process of these RBNN is the following: the neuron centers are calculated in an unsupervised way using K-means algorithm, the neuron widths are evaluated as the geometric mean of the distances from each neuron center to its two nearest centers, and the RBNN weights are estimated in a supervised way in order to minimize the mean square error measured in the training set.

Table 2
Best mean error for the lazy strategy to train RBNN compared with eager training

Mean error (parameters)	Gaussian lazy method	Inverse lazy method	Traditional method
Piecewise-defined function	0.00753 *width* = 0.1, 7 neurons	0.00208 $r_r = 0.06$, 11 neurons	0.04156 100 neurons
Hermite polynomial	0.01040 *width* = 0.2, 7 neurons	0.002994 $r_r = 0.1$, 11 neurons	0.01904 40 neurons
Mackey-Glass time series	0.02207 *width* = 0.2, 27 neurons	0.01565 $r_r = 0.8$, 19 neurons	0.10273 110 neurons
Level at Venice Lagoon	0.04107 *width* = 0.15, 23 neurons	0.02059 $r_r = 0.16$, 19 neurons	0.09605 50 neurons

Table 3
Best mean error for *k*-NN, weighted *k*-NN, linear and nonlinear local regression methods

Mean error (*k* value)	*k*-Nearest neighbor	Weighted *k*-nearest neighbor	Local linear regression
Piecewise-defined function	0.01114 $k = 2$	0.00794 $k = 4$	0.02514 $k = 3$
Hermite polynomial	0.01274 $k = 2$	0.00697 $k = 2$	0.02156 $k = 2$
Mackey-Glass time series	0.02419 $k = 2$	0.02404 $k = 6$	0.02579 $k = 2$
Level at Venice Lagoon	0.05671 $k = 2$	0.05611 $k = 3$	0.04385 $k = 45$

In Table 2 it is possible to observe that the generalization capability of RBNN increases when they are trained using a lazy strategy, instead of the eager or traditional training that is usually used in the context of neural networks. The results improve significantly when the selection is based on the input information contained in the test pattern and when this selection is carried out with the inverse function.

Tables 2 and 3 show that the lazy training of RBNN using the inverse selection function improves significantly the performance of the traditional lazy methods. This does not happen when the Gaussian selection function is used: the errors are slightly better than the obtained by the traditional lazy methods, but not in all the cases. Therefore, the selection function becomes a key issue in the performance of the method.

3.4. *Other domains*

RBNN do not work well with all domains, due to their specific characteristics. This is why we have chosen the domains described above, some of them widely used in RBNN literature.

However, in order to show the behavior of the proposed method with other domains, we have applied it to two more regression domains: the Boston housing database obtained from

Table 4
Best mean error for the new domains

Mean error (parameters)	Lazy RBNN (Inverse function)	Traditional RBNN	k-NN	Weighted k-NN
Boston housing database	0.049947 $r_r = 0.24$, 11 neurons	0.068576 80 neurons	0.065429 $k = 2$	0.060714 $k = 4$
Bank8FM	0.06266 $r_r = 0.2$, 27 neurons	0.077857 90 neurons	0.106694 $k = 4$	0.103443 $k = 4$

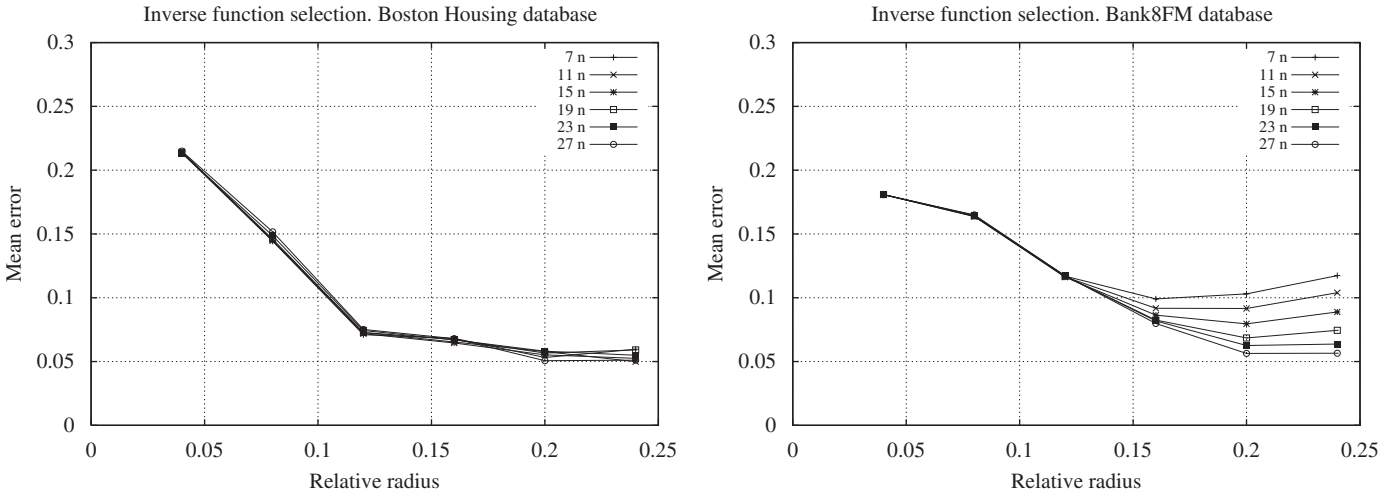


Fig. 11. Mean errors with inverse selection for the Boston housing and Bank8FM databases.

the UCI Machine Learning Repository,[1] and the Bank8FM data set, obtained from the Torgo group.[2] They are well-known domains although they have not been used with RBNN as much as the previous ones.

Some of their characteristics are the following:

- *Boston housing database*: The set has 506 instances characterized by 13 attributes. Twelve of them are continuous and one is a binary value. Three fifty examples have been randomly chosen for the training set and 156 for the testing set. Both sets have been normalized in the interval $[0, 1]$.
- *Bank8FM*: Four thousand five hundred training instances and 3692 testing instances, with eight continuous attributes. Both sets have been normalized in the interval $[0, 1]$.

The learning techniques that have been applied to the new domains are: RBNN trained in a lazy way using the inverse function, RBNN trained in an eager way, k-NN and weighted k-NN. When RBNN are trained in a lazy way, the relative radius varies between 0.04 and 0.28 and the number of neurons of the networks also varies between 7 and 27. Regarding to RBNN trained in the usual way, we have used networks of different architectures, from 5 to 100 neurons. As for k-NN methods, the parameter k varies between 1 and 50.

In Table 4 we summarize the obtained results showing the best absolute mean errors obtained by each method. It is possible to observe that the lazy RBNN method with the inverse selective function outperforms the rest of techniques.

Moreover, Fig. 11 show that the performance of the lazy RBNN method with the inverse function does not depend significantly on the relative radius and the number of neurons when these values are bigger enough, as it happened with the former domains.

4. Conclusions

The machine learning methods based on examples try, usually, to learn some implicit function or its approximation, by the set of examples provided by the domain. There are two ways of approaching this problem. On one hand, global approximators try to approximate the function in a global way. This means that they try to build up some kind of approximation that allow to forecast all the test examples, no matter the characteristics those examples have. However, there are many domains in which global approximations do not make good predictions. Those domains have non-homogeneous behavior and the examples are not evenly distributed in the input space. For this domains, lazy machine learning methods would be needed.

Lazy machine learning methods have to deal with some extra difficulties. We need to know how many relevant different regions exist in the input space. The method must build up different approximations for each of the above mentioned regions. It must also know when a point belongs to a specific region. Both characteristics are very difficult to include in a method in an efficient way.

In this work a lazy learning method is described. The method incorporates a way to determine the relevant input space regions, and decides whether an example belongs to any specific region. Afterwards, a RBNN is used for making predictions.

[1] http://mlearn.ics.uci.edu/MLRepository.html
[2] http://www.liaad.up.pt/%7Eltorgo/Regression/DataSets.html

Local methods have been used for this purpose for very long. They show good results, specially in domains with a mostly linear behavior in the regions. When regions show a more complex behavior, those techniques have worse results.

We try to complement the good characteristics of each approach by using lazy learning for selecting the training set, but using RBNN, that show good behavior for non-linear predictions. In this way, we produce a method that can get the locality of the input space, and then using a non-linear method to approximate each region of the input space.

In this article we have presented two ways of doing the patterns selection, by means of a kernel function: using the Gaussian and the inverse functions. We have compared our approach with different lazy methods, like *k*-NN, weighted *k*-NN and local linear regression. We have also compared our results with eager global approximations built up with RBNN.

One conclusion of the results is that the kernel function used for selecting training patterns is relevant for the success of the network. Both functions produce different results. This means that the selection of the patterns is a relevant task, that could lead to better results. However, not all the kernel functions are good for selecting patterns. The selection of patterns is a crucial step in the success of the method: it is important to decide not only what patterns are going to be used in the training phase, but also how those patterns are going to be used and the importance that each pattern will have in the learning. In this work we see how the Gaussian function not always produces good results. In the validation domains, the results are rather poor when the selection is made using the Gaussian function, although they are better than the results obtained with eager methods. However, they are always worse than when using the inverse function to select the training patterns. This function have good results in all the domains.

We can also conclude that the use of lazy learning for prediction can improve the results, in some cases meaningfully. In most of the experimental domains, the lazy methods improve the results of the RBNN trained in a classical way.

Naturally, the domains have been chosen with this purpose, they have non-local behavior, and there are different regions with different problem expressions. This is easy to see in the piecewise function, where there are two clearly different regions, by the self-definition of the function.

Finally, the results also show that the combination of lazy learning and RBNN can produce significant improvements in some domains. Two different aspects must be taken into consideration:

- A good method for selecting training patterns is needed. This method must be compatible with lazy approaches. The patterns must be selected specifically for each new test pattern.
- A non-linear method must be used for prediction. The regions must have any kind of structure, so a general method will be needed.

This combination of lazy learning for RBNN and some local way of selecting patterns produces good results, improves substantially eager RBNN learning, and could reach to better results than other lazy techniques.

References

[1] D.W. Aha, D. Kibler, M. Albert, Instanced-based learning algorithms, Mach. Learn. 6 (1991) 37–66.

[2] C.G. Atkeson, A.W. Moore, S. Schaal, Locally weighted learning, Artif. Intell. Rev. 11 (1997) 11–73.

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

[4] B.V. Dasarathy (Ed.), Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, Los Alamitos, CA, 1991.

[5] A. Leonardis, H. Bischof, An efficient MDL-based construction of RBF networks, Neural Networks 11 (1998) 963–973.

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

[7] J.E. Moody, C.J. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 1 (1989) 281–294.

[8] M.J.L. Orr, Introduction to radial basis neural networks, Technical Report, Centre for Cognitive Science, University of Edinburgh, 1996.

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

[10] T. Poggio, F. Girosi, Networks for approximation and learning, in: Proceedings of the IEEE, vol. 78, 1990, pp. 1481–1497.

[11] J.M. Valls, I.M. Galván, P. Isasi, Lazy learning in radial basis neural networks: a way of achieving more accurate models, Neural Process. Lett. 20 (2) (2004) 105–124.

[12] J.M. Valls, I.M. Galván, P. Isasi, Improving the generalization ability of RBNN using a selective strategy based on the Gaussian kernel function, Comput. Inf. 25 (2006) 1–15.

[13] J.M. Valls, I.M. Galván, P. Isasi, Lazy training of radial basis neural networks, in: Proceedings of the International Conference on Artificial Neural Networks, ICANN 2006, Lecture Notes in Computer Science, vol. 4131, Springer, Berlin, 2006, pp. 198–207.

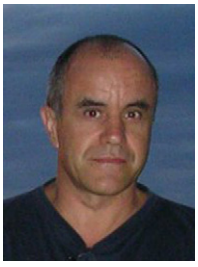
[14] G. Vittori, On the Chaotic features of tide elevation in the Lagoon Venice, in: Proceedings of the of the ICCE-92, 23rd International Conference on Coastal Engineering, Venice, 4-9, 1992, pp. 361–362.

[15] D. Wettschereck, D.W. Aha, T. Mohri, A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, Artif. Intell. Rev. 11 (1997) 273–314.

[16] B.A. Whitehead, T.D. Choate, Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction, IEEE Trans. Neural Networks 5 (1995) 15–23.

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

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



José M. Valls received his Ph.D. in Computer Science at Universidad Carlos III of Madrid (Spain) in 2004. He joined the Computer Science Department at the University Carlos III of Madrid in 1998, being associate professor since 2004. He is enrolled in the Neural Networks and Evolutionary Computation Laboratory of this University. His current research focuses on the application of Neural Networks, Evolutionary Computation and other soft computing techniques to engineering problems.

Inés M. Galván received a doctorate-fellowship, as research scientist, in the European Commission, Joint Research Centre Ispra (Italy) from 1992 to 1995. She received her Ph.D. in Computer Science at Universidad Politecnica de Madrid (Spain), in 1998. She has joined the Computer Science Department at the University Carlos III of Madrid in 1995 and she is associate professor of that department from 2000. Her current research focuses on Artificial Neural Networks and other soft computing techniques, as Evolutionary Computation and Multiagent Systems. Her research interests cover also application fields, as time series prediction and control of dynamic process.

Pedro Isasi received his Ph.D. in Computer Science from Politecnica de Madrid University in 1994. He joined the Computer Science Department of Carlos III de Madrid University in 1991, he was associate professor from 1997 until 2001, and he is now full professor of that department. He is the founder and director of the Neural Network and Evolutionary Computation Laboratory. His current research focuses in the application of soft computing techniques (NN, Evolutionary Computation, Fuzzy Logic and Multiagent Systems) to engineering problems as Power Plant control, robot control, cryptography or finances. Mainly in domains of prediction, classification, optimization and times series.