Journal of Intelligent & Fuzzy Systems 13 (2002/2003) 111–122
IOS Press

Model selection via Genetic Algorithms for RBF networks

Estefane G.M. de Lacerda[a], André C.P.L.F. de Carvalho[b] and Teresa B. Ludermir[a]
[a]*Federal University of Rio Grande do Norte, Brazil*
E-mail: estefane@dca.ufrn.br
[b]*ICMC – University of São Paulo, São Carlos, SP, Brazil*
E-mail: andre@icmc.usp.br

Abstract. This work addresses the problem of finding the adjustable parameters of a learning algorithm using Genetic Algorithms. This problem is also known as the model selection problem. In this paper, some model selection techniques (e.g., crossvalidation and bootstrap) are used as objective functions of a Genetic Algorithm. The Genetic Algorithm is modified in order to allow the efficient use of these objective functions by means of occam's razor, growing, and other heuristics. Some modifications explore intrinsic features of Genetic Algorithms, such as their ability to handle multiple and noise objective functions. The proposed techniques are very general and may be applied to a large range of learning algorithms.

Keywords: Genetic Algorithms, model selection, crossvalidation, holdout, bootstrap, RBF networks

1. Introduction

Most Machine Learning techniques focus the problem of approximating a target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ by using the information from a sample of examples $(\mathbf{x}_i, y_i)$, for $i = 1, \ldots, p$, where $\mathbf{x} \in \mathcal{X}$ and $y_i = f(\mathbf{x}_i)$. In principle, a learner $\mathcal{L}$ (a learning algorithm) builds a hypothesis function $h \in \mathcal{H}$ that approximates the target function f by determining the parameter setting θ of the hypothesis. Most of the time, however, the hypothesis has a few parameters which the learner itself is unable to determine: the adjustable parameters.

Thus, for a particular learner $\mathcal{L}$, there are two kinds of parameters in a hypothesis: the parameters τ (training parameters) that are determined automatically by $\mathcal{L}$ and the parameters λ (adjustable parameters) that are not determined by $\mathcal{L}$. $\theta = \tau \cup \lambda$ and $\tau \cap \lambda = \emptyset$. The adjustable parameters are typically determined by human subjective judgment based on previous experience, rule of thumbs or heuristics provided by authors and practitioners of the learning algorithm.

In other words, the adjustable parameters are determined by minimizing an estimative of the *true prediction error* [2] (true error for short) over a set of adjustable parameters that is known to work well on the training dataset. A common example of this procedure is to train a neural network using the backpropagation learning algorithm [17]. The backpropagation algorithm sets the weight parameters of the network, but is unable to define the number of hidden units (which are the adjustable parameters of the problem). The search carried out by the user in order to find the best adjustable parameter is essentially a trial-and-test optimization method, which becomes inefficient if the search space is too large. Hence, the human subjective judgment and heuristics to make the trial-and-test method more efficient are needed. Others common examples of adjustable parameters are the amount of pruning of a decision tree, the degree of a polynomial fit to a set of points, and the ridge parameter of the ridge regression.

The problem of estimating the true error of hypothesis using different adjustable parameters in order to choose the (approximate) best one is known as model selection [8]. The purpose of this work is to present Genetic Algorithms (GAs) as an alternative to the trial-and-test optimization method for model selection.

1064-1246/02/03/$8.00 © 2002/2003 – IOS Press. All rights reserved

More formally, the optimization problem to be solved by GA can be described as follows:

A hypothesis $h : \mathcal{X} \rightarrow \mathcal{Y}$ (where $\mathcal{X} \times \mathcal{Y} = \mathcal{E}$ is the example space) is built by the learner $\mathcal{L}$ from a choice of the value of the adjustable parameter λ and a training dataset $\mathcal{D}$. It is assumed that the learner is deterministic. That is, for a particular choice of λ and $\mathcal{D}$, the learner always builds the same hypothesis. Thus, hypothesis h can be represented as $h(\lambda, \mathcal{D})$. The notation $h(\mathbf{x}; \lambda, \mathcal{D})$ represents the prediction of $h(\lambda, \mathcal{D})$ for the data point $\mathbf{x}$.

The true error measures how well a hypothesis $h(\lambda, \mathcal{D})$ predicts the response value of a future example. The true error may be defined as

$$e(\lambda) = \mathrm{E}\langle \delta(y, h(\mathbf{x}; \lambda, \mathcal{D})) \rangle \quad (1)$$

where $\delta(x, y)$ is a loss function. The expectation in Eq. (1) refers to repeated sampling of examples randomly drawn from the example space $\mathcal{E}$ with the same probability that was used to select the examples from training dataset $\mathcal{D}$. Regression problems usually employ the quadratic loss function $\delta(x, y) = (x - y)^2$. Classification problems usually make use of the $0/1$ loss function: $\delta(x, y) = 1$ if $x = y$, otherwise $\delta(x, y) = 0$. By using the $0/1$ loss function, the function $e(\lambda)$ represents the probability of correctly classified examples.

The problem of choosing the adjustable parameter (i.e., model selection problem) can be defined as:

$$\text{Find } \lambda \text{ that minimizes } e(\lambda) \quad (2)$$

The true error $e(\lambda)$ is unknown, once the only available information on the target function is contained into the dataset $\mathcal{D}$. Because of this, a number of methods for estimating the true error have been proposed [2]. The most commonly used methods for true error estimation are:

- Holdout;
- Crossvalidation;
- Bootstrap.

Consider that $\widehat{e}(\lambda, \mathcal{D})$ is an estimation of $e(\lambda)$ using the information from the dataset $\mathcal{D}$ by means of those methods. The problem of optimizating the adjustable parameter can be reformulated as:

$$\text{Find } \lambda \text{ that minimizes } \widehat{e}(\lambda, \mathcal{D}) \quad (3)$$

The best estimations of the true error are usually obtained by the crossvalidation and bootstrap methods, which transform this optimization problem into a difficult problem. Crossvalidation and bootstrap are computer procedures, but may be interpreted as simple functions from the optimization view point. As objective functions, they have the following drawbacks:

- These functions are not easy to be handled algebraically, and hence derivatives are not easy to calculate.
- These functions are also multimodals (i.e., there are several values of λ that minimizes the function locally).
- For a given point λ, to obtain the response of the crossvalidation and bootstrap functions, a large amount of CPU time may be needed, making their minimization hard for any optimization technique (including GAs).

Because of the complex nature of this optimization problem, GAs are a natural candidate for solving it. Unlike other optimization techniques, GAs provide a suitable framework for the model selection problem, due to its peculiar search mechanism able to handle many hypotheses simultaneously. This search mechanism enables GAs to use (and to create) new heuristics (as shown later), to cope with noise estimation of the true error and to be used for multiobjective optimization.

This work addresses GAs search mechanism, instead of other more studied aspects, like encoding and genetic operators (mainly for Neural Networks [21]). This paper is organized as follows. A simple GA method with occam elitism for model selection is described in Section 2. Section 3 shows an example of a Machine Learning model (that will be used throughout the paper), its genetic representation and experimental studies. Crossvalidation, bootstrap and other objective functions are presented in Section 4, together with some experimental results. Section 5 evaluates a multiobjective GA employing two heuristics (growing and shuffling) that aim to improve the quality of the genetic hypothesis. Finally, Section 6 presents the main conclusions.

2. A simple GA method for model selection

The application of GAs for model selection involves the optimization of the adjustable parameters. It is worth noting that the adjustable parameters of a hypothesis do not only depend on the features of the hypothesis, but also on the learner. Different learners can produce different adjustable parameters for the same hypothesis.

To avoid proliferating notation, the symbol λ will represent both the chromosome and the adjustable parameters it encodes. Throughout the text, the dataset

$$\mathcal{D} = \{(\mathbf{x}_i, y_i); \quad i = 1, \ldots, p\} \tag{4}$$

represents the set of all the examples avaliable for the problem at hand.

A typical GA for model selection has the following characteristics:

1. The chromosome encodes only the adjustable parameters.
2. A learner $\mathcal{L}$ is embedded in the procedure that evaluate the chromosome.
3. A method for estimating the true error.

A simple GA for model selection is illustrated in Fig. 1. The procedure evaluate() in Fig. 1 computes the fitness function, that is, the estimation of the true error $\widehat{e}$ for each chromossome by using traditional methods for model selection, such as holdout, crossvalidation and bootstrap. In the next section, the holdout function is presented.

2.1. The holdout fitness function

One of the most used methods for model selection, the holdout method [10], as can be seen in Fig. 2, divides a dataset $\mathcal{D}$ into two parts: the training set $\mathcal{D}_t$, on which the hypothesis is built, and the holdout set $\mathcal{D}_h$ (also known as validation set), on which its performance is measured. Thus,

$$\mathcal{D}_h = \mathcal{D} \setminus \mathcal{D}_t \tag{5}$$

Let $h(\lambda, D_t)$ be the hypothesis built from the dataset $\mathcal{D}_t$ using the adjustable parameter λ. The fitness of the chromosome λ is given by:

$$f(\lambda) = \frac{1}{|\mathcal{D}_h|} \sum_{(\mathbf{x},y)\in\mathcal{D}_h} \delta(h(\mathbf{x}; \lambda, D_t), y) \tag{6}$$

In order to improve the simple GA from the Fig. 1, the use of the occam elitism is proposed.

2.2. The occam elitism

Occam's razor is a principle attributed to the 14th century philosopher William of Occam. The principle states that "*entities should not be multiplied unnecessarily*". With this "razor", Occam cut out all superfluous, redundant explanations. Scientists have reintepreted the Occam's razor. A useful statement of the principle for scientists is, "*when you have two competing theories which make exactly the same predictions, the one that is simpler is the better*." Machine learning scientists [13] have stated that principle as "*prefer the simplest hypothesis that fits the data*". For this work, the following statement is used:

main program

"*Let* $P(t) = \{\lambda_1, \ldots, \lambda_N\}$
be the population at generation t*.*"

$t \leftarrow 0$
initialize($P(t)$)
evaluate($P(t)$)
while the stopping criterion is not satisfied **do**
  $t \leftarrow t + 1$
  $P(t) \leftarrow$ select($P(t-1)$)
  $P(t) \leftarrow$ crossover($P(t)$)
  $P(t) \leftarrow$ mutation($P(t)$)
  evaluate ($P(t)$)
end while

procedure evaluate(P)

for each λ_i in the population P **do**
  Estimate the true error $\widehat{e}(\lambda_i, \mathcal{D})$
  Set fitness f_i of λ_i equal to $\widehat{e}(\lambda_i, \mathcal{D})$
end for

Fig. 1. A Genetic Algorithm for Model Selection.

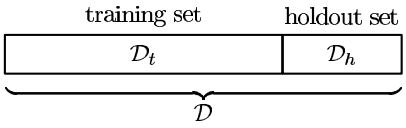

Fig. 2. The data set partition by the holdout method.

Occam's razor: given two hypothesis with the same estimation of true error, the one that is simpler is the better because it is likely to have lower true error.

The traditional elitism is a well known strategy in GAs which works by never replacing the best chromosome in a population with inferior solutions. That is,

Elitism: the best chromosome is kept from generation to generation.

The occam elitism is based on the assumption that the κ best hypotheses in the population are equivalent

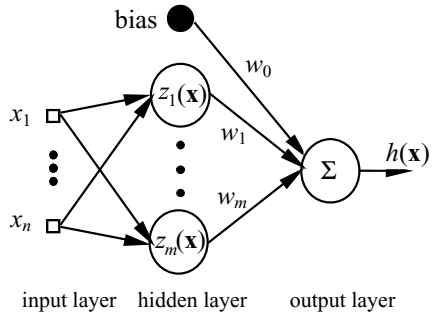

Fig. 3. RBF Network.

in terms of estimation of true error (at least under some statistical confident limit). Hence,

Occam elitism: the simplest hypothesis among the κ best chromosomes is kept from generation to generation.

In order to illustrate the performance achieved by the proposed GA for model selection, this method is employed to find the adjustable parameters of a machine learning model: Radial Basis Function (RBF) networks.

3. RBF networks design using Genetic Algorithms

RBF networks have their origin in the solution of the multivariate interpolation problem [16]. A hypothesis $h : \mathcal{R}^n \rightarrow \mathcal{R}$ is represented by a RBF network [14] (Fig. 3) as a linear combination of the basis function as follows:

$$h(\mathbf{x}) = w_0 + \sum_{i=1}^{m} w_i z_i(\|\mathbf{x} - \mathbf{c}_i\|) \tag{7}$$

where $\|\cdot\|$ is the Euclidean norm. The complete set of parameters of a typical RBF network is shown in Table 1.

A radial basis function is a nonlinear function that monotonically decreases (or increases) as $\mathbf{x}$ moves away from its vector center $\mathbf{c}_j$. Usual basis functions are:

1. The thin-plate-spline function:
$$z(v) = v^2 \log v \tag{8}$$
2. The Gaussian function:
$$z(v) = \exp(-v^2/\sigma^2) \tag{9}$$
3. The multiquadratic function:
$$z(v) = (v^2 + \sigma^2)^{1/2} \tag{10}$$
4. The inverse multiquadratic function:
$$z(v) = \frac{1}{(v^2 + \sigma^2)^{1/2}} \tag{11}$$

The parameter width σ is a scaling factor for the radius $\|\mathbf{x} - \mathbf{c}_i\|$.

The widths are usually defined by computationally inexpensive heuristics [18]. Moody and Darken, in [14], suggest that a single value σ for all basis functions gives good results. They used $\sigma = \langle\|\mathbf{c}_i - \mathbf{c}_j\|\rangle$, where $\mathbf{c}_j$ is the nearest center to $\mathbf{c}_i$ and $\langle\cdot\rangle$ indicates the average over all such pairs. Others methods use a different value σ_i for each basis function. In [18], each width σ_i is defined as:

$$\sigma_i = \alpha\|\mathbf{c}_i - \mathbf{c}_j\|, \tag{12}$$

where α is an overlap factor and $\mathbf{c}_i$ and $\mathbf{c}_j$ are defined as before.

In general, the weights are training parameters, whereas the other parameters are either adjustable or training parameter depending on the learner used. Once centers and widths have been fixed, the RBF network may be interpreted as a case of multivariate linear regression:

$$\mathbf{y} = \mathbf{Z}\mathbf{w} + \epsilon \tag{13}$$

where $\mathbf{y} = [y_1, y_2, \ldots, y_p]^\mathsf{T}$, $\mathbf{Z}$ denotes the design matrix with the j th column equal to:

$$[z_j(\|\mathbf{x}_1 - \mathbf{c}_j\|), z_j(\|\mathbf{x}_2 - \mathbf{c}_j\|), \ldots, z_j(\|\mathbf{x}_p - \mathbf{c}_j\|]^\mathsf{T},$$

$\mathbf{w} = [w_1, w_2, \ldots, w_m]^\mathsf{T}$ and ϵ is the vector of errors. The vector $\mathbf{w}$ is determined minimizing the sum of squared errors $\mathrm{SSE} = \epsilon^\mathsf{T}\epsilon$. The solution to this least squares problem can be obtained by solving the well-known linear system:

$$\left(\mathbf{Z}^\mathsf{T}\mathbf{Z}\right)\mathbf{w} = \mathbf{Z}^\mathsf{T}\mathbf{y} \tag{14}$$

3.1. The proposed Genetic Algorithm

This work optimizes the RBF network using the GA proposed by [11]. In this GA, the RBF network parameters are divided in adjustable (λ) parameters and training (τ) parameters, as follows:

$$\lambda = \{m, \mathbf{c}_1, \ldots, \mathbf{c}_m, \alpha\}$$

$$\tau = \{\mathbf{w}\}$$

where m is the number of basis functions, α is the overlap factor from Eq. (12) that define how large the

Table 1
Parameters set of a RBF network

Number of basis functions	m
Basis functions	$z_1(\cdot), \ldots, z_m(\cdot)$. Where $z_i(\cdot)$ is the function performed by the hidden layer node i.
Centers	$\mathbf{c}_j$ for $1, \ldots, m$. Where $\mathbf{c}_j = [c_{j1}, c_{j2}, \ldots, c_{jn}]^\mathsf{T}$ is the vector center of the basis function $z_j(\cdot)$.
Widths	$\sigma_1, \ldots, \sigma_m$. Where σ_j is the width of basis function z_j.
Weights	$\mathbf{w} = [w_1, \ldots, w_m]^\mathsf{T}$. Where w_j is the weight connecting the jth hidden unit to the output unit.

widths are, and $\mathbf{c}_j$'s are the centers. The weights $\mathbf{w}$ are training parameters, once they are obtained by the learner $\mathcal{L}$ (embedded in the procedure that evaluates the chromosome) using Eq. (14).

The encoding is one of the key aspects to be considered when using GAs. The adjustable parameters are coded into a tuple given by:

$$\mathbf{P} = (\alpha, \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_K) \tag{15}$$

where α is the overlap factor, K is the number of regions (these are regions of the input space which will be discussed in Section 3.2), and $\mathbf{p}_j$ encodes a hidden unit, which is expressed by the following tuple:

$$\mathbf{p}_j = (b_j, c_{j1}, c_{j2}, \ldots, c_{jn}) \tag{16}$$

This tuple represents a basis function whose center $\mathbf{c}_j = [c_{j1}, \ldots, c_{jn}]^\mathsf{T}$ is inside the region R_j (i.e., $\mathbf{c}_j \in R_j$). The parameter b_j is a boolean flag: if b = TRUE then $\mathbf{p}_j$ is valid, otherwise $\mathbf{p}_j$ is discarded during the decoding. In spite of having fixed length (the number of coded hidden units is always equal to K), the chromosome can produce network topologies with variable sizes because some basis functions will be discarded (depending on whether b_j is TRUE or not).

3.2. Partitioning the input space

The partition of the input space creates a set of K rectangular regions $\{R_1, \ldots, R_K\}$, which are placed in the areas where there is a high density of training input examples, as showed in Fig. 4. The width of the region R_i along each coordinate direction is determined by the corresponding components of the vectors $\mathbf{l}_i = [l_{i1}, \ldots, l_{in}]^\mathsf{T}$ and $\mathbf{u}_i = [u_{i1}, \ldots, u_{in}]^\mathsf{T}$. That is,

$$\begin{aligned} R_i = \{\mathbf{x} \in \Re^n : l_{i1} \leqslant x_1 \leqslant u_{i1}, \ldots, \\ l_{in} \leqslant x_n \leqslant u_{in}\} \end{aligned} \tag{17}$$

In other words, the components l_{ik} and u_{ik} are the lower and upper limits, respectively, of the region R_i along a coordinate direction.

The vectors $\mathbf{l}_i$ and $\mathbf{u}_i$ are obtained from the clusters of training input examples generated by the K-means clustering algorithm. Consider S_i a cluster of input examples. The vectors $\mathbf{l}_i$ and $\mathbf{u}_i$ (which determine the region R_i) are obtained from the cluster S_i as follows:

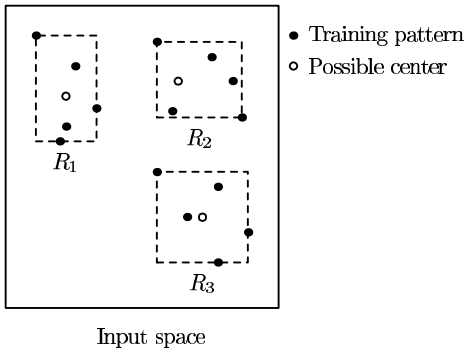

Fig. 4. Partioning the input space by means of clusters of examples.

$$l_{ik} = \min_{\mathbf{x} \in S_i} x_k \tag{18}$$

$$u_{ik} = \max_{\mathbf{x} \in S_i} x_k \tag{19}$$

for all $k = 1, \ldots, n$. By using this procedure, the region R_i embraces all examples of the cluster S_i.

3.3. Decoding the chromosomes

To evaluate the fitness of the chromosomes, they need to be decoded. Consider the following chromosome to be decoded: $\mathbf{P} = (\alpha, \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_K)$. The parameters $\alpha, c_{j1}, c_{j2}, \ldots, c_{jn}$ were encoded as floating-point values and normalized in the interval $[0, 1]$. The decoding is caried out as follows:

$$c'_{jk} = l_{r_j,k} + c_{jk}(u_{r_j,k} - l_{r_j,k}) \tag{20}$$

$$\sigma'_j = s\sigma_j \tag{21}$$

for $k = 1, \ldots, n$. The coefficient $s = 5$ is a scaling factor.

3.4. Genetic operators

The proposed GA uses a modified crossover operator, here named cluster crossover, and a set of mutation operators.

```
for j = 1 to K then
    q_j = p_1j or p_2j with equal probability.
endfor
```

Fig. 5. Cluster crossover.

```
for j = 1 to K then
    if (b_1j = 1) and (b_2j = 1) then
        q_j = BlendCrossover( p_1j , p_2j )
    else
        q_j = p_1j or p_2j with equal probability.
    endif
endfor
```

Fig. 6. Cluster crossover combined with blend crossover.

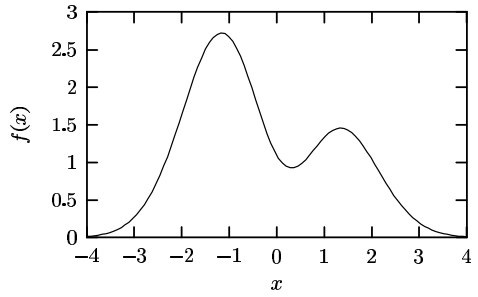

Fig. 7. Hermite polynomial.

The cluster crossover is described as follows. Let $\mathbf{P}_1 = (\mathbf{p}_{11}, \ldots, \mathbf{p}_{1K})$ and $\mathbf{P}_2 = (\mathbf{p}_{21}, \ldots, \mathbf{p}_{2K})$ be the parents, where $\mathbf{p}_{kj} = (b_{kj}, c_{kj1}, c_{kj2}, \ldots, c_{kjn})$. Let $\mathbf{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_K)$ be their child. The cluster crossover is shown in Fig. 5. Note that the cluster crossover itself does not change the value of the centers. To change these values, the cluster crossover is combined with the blend crossover [3] (a well-known crossover for real-coded GAs) as shown in Fig. 6, in which the subroutine BlendCrossover() refers to the blend crossover.

The cluster crossover operator works like the traditional uniform crossover. Its main difference is that it exchanges regions (which represents clusters) R_i instead of structural chunks of chromosomes (as performed by the standard uniform crossover). The cluster crossover is a method for the exchange of hypervolumes on the phenotype space, whereas the standard uniform crossover operators are performed on the genotype space.

The following mutation operators were used:

– Uniform mutation: replaces centers and the overlap factor with uniform random numbers;
– Addition and delete operators add and delete randomly chosen basis functions.

Next section shows experiments using this GA. It also compares the traditional elitism with the occam elitism operator.

3.5. Experimental studies

In the experiments performed, the proposed GA was applied to a benchmark dataset: a Hermite polynomial approximation [12,15] (see Fig. 7), which is defined by:

$$f(x) = 1.1(1 - x - 2x^2)\exp\left(-\frac{x^2}{2}\right) \qquad (22)$$

The datasets used in the experiment were generated under the same conditions adopted in [15]. Each dataset has 40 examples randomly chosen in the range $[-4, +4]$ with added gaussian noise. The following noise variances were used: 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, and 0.1. For each noise variance, the GA program was executed, at least, 50 times (where each execution used a different random dataset) and the results were averaged.[1] The GA program used the parameters shown in Table 2. The holdout method generated the holdout dataset with 50% of the original dataset. The performance of the hypotheses was measured using the Root-Mean-Square Error (RMSE = $p^{-1}\sqrt{\text{SSE}}$) over a test set with 200 uniformly spaced noiseless examples in the range $[-4, +4]$. The occam elitism was used with $\kappa = 5$. Thus, the smallest network among the five best chromosomes was kept from generation to generation.

According to the results presented in the Fig. 8, which shows the number of centers as a function of noise level, the occam elitism produced notably smaller networks than the traditional elitism. Moreover, GA with occam elitism also produced a lower true error, as shows the Fig. 9. These results suggest that the occam elitism may be an effective method. Because of this, all the following experiments in this work will use the occam elitism (with $\kappa = 5$).

[1] GA may produce outliers (spurious networks with large errors) due to premature convergence. The average value is strongly influenced (become biased towards) by the outliers. As a result a 5% trimmed mean was used instead of the mean. The 5% trimmed mean results in the elimination of both the top 5% and the bottom 5% of the ranked samples. The mean is calculated for the remaining samples.

Table 2
GA parameters

Population	500
Generations	500
Number of regions	15
Crossover rate	0.60
Mutation rate	0.05
Addition rate	0.3
Deletion rate	0.3
Basis function	Gaussian

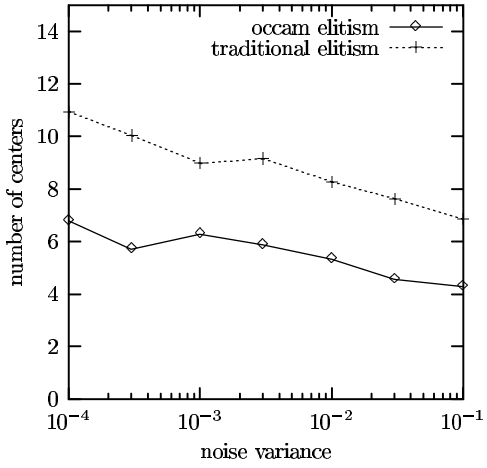

Fig. 8. Comparing the performance of occam and traditional elitism in terms of the number of basis functions.

4. Others fitness functions for model selection

This section presents methods widely used in machine learning for model selection (to select a hypothesis (a model) among several candidate hypotheses).

4.1. The k-fold-crossvalidation fitness function

This method [20,10] divides the dataset $\mathcal{D}$ (Eq. (4)) in k subsets (also named *folds*): $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$. The folds have equal size and are mutually exclusive. Thus, the method produces k hypothesis $h_1, \ldots, h_k$, where each hypothesis is built from the dataset $\mathcal{D} \setminus \mathcal{D}_j$ (see Fig. 10) using the adjustable parameter λ, as follows:

$$h_j = h(\lambda, \mathcal{D} \setminus \mathcal{D}_j), \quad \text{for} = \; j = 1, \ldots, k \qquad (23)$$

The performance obtained by each hypothesis h_j is measured on the dataset D_j. The fitness of λ is equal to the average of the performances of the k hypotheses. The fitness of the chromosome λ is given by:

$$f(\lambda) = \frac{1}{|\mathcal{D}|} \sum_{j=1}^{k} \sum_{(\mathbf{x},y)\in\mathcal{D}_j} \delta(h_j(\mathbf{x}), y) \qquad (24)$$

If $k = |\mathcal{D}|$, the k-fold-crossvalidation is known as leave-one-out.

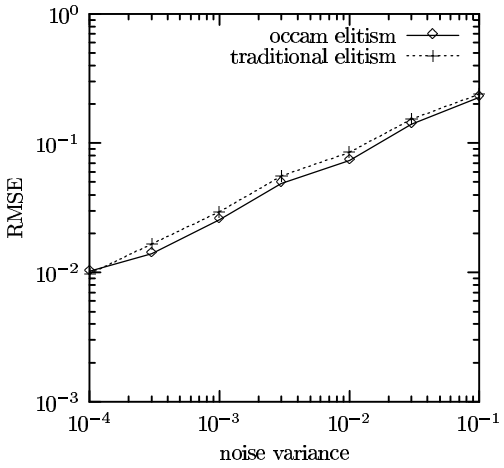

Fig. 9. Comparing the performance of occam and traditional elitism in terms of the error on the test set.

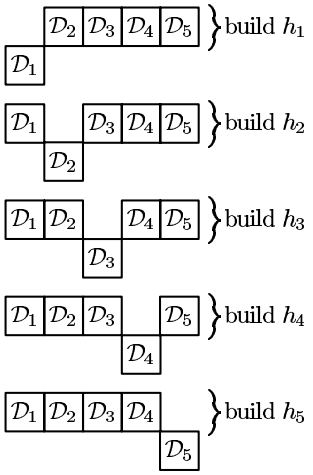

Fig. 10. The k-fold-crossvalidation method with $k = 5$.

4.2. The Generalized Cross-Validation fitness function

The Generalized Cross-Validation (GCV) is a formula derived from the leave-one-out crossvalidation under the assumption that the model is linear. Thus, GCV is often used for linear models. RBF networks are nonlinear models, but, in practice, GCV has been used in model selection for RBF networks [9,15]. Because GCV is computationally inexpensive, it becomes attractive to be used with GAs.

The chromosome is trained with the whole dataset $\mathcal{D}$. The fitness of the chromosome λ is given by GCV Equation:

$$f(\lambda) = \mathrm{GCV} = \frac{p\,\mathrm{SSE}}{(p-m)^2} \qquad (25)$$

where SSE denotes the sum of squared errors on the dataset $\mathcal{D}$, m is the number of weights (free parameters) and p is the number of examples in $\mathcal{D}$.

If the RBF network learning uses ridge regression (regularization) [9] to compute the weights, the GCV is modified in order to include the ridge parameter θ [7]:

$$f(\lambda) = \text{GCV} = \frac{(1/p)\|(I-A)\mathbf{y}\|^2}{[(1/p)\text{trace}(I-A)]^2} \tag{26}$$

where $A = \mathbf{Z}(\mathbf{Z}^\mathsf{T}\mathbf{Z} - \theta I)^{-1}\mathbf{Z}^\mathsf{T}\mathbf{y}$.

4.3. The .632 Bootstrap fitness function

The .632 Bootstrap method is a member of the bootstrap family introduced by Efron [2]. All bootstrap based estimates are computed by using a set of bootstrap datasets. A bootstrap dataset is created by sampling $p = |\mathcal{D}|$ examples (with replacement) from $\mathcal{D}$. This method creates B bootstrap datasets: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_B$. Note that the datasets are not mutually exclusive. This method produces B hypotheses $h_1(\lambda), \ldots, h_B(\lambda)$, where each one is built from the dataset $\mathcal{D}_j$ using the adjustable parameter λ. That is,

$$h_j = h(\lambda, \mathcal{D}_j), \qquad \text{for } j = 1, \ldots, B \tag{27}$$

The fitness of the chromosome λ, obtained by the .632 bootstrap function, is given by:

$$f(\lambda) = 0.368 \cdot \overline{\text{err}} + 0.632 \cdot \epsilon_0 \tag{28}$$

The terms of the Eq. (28) are described as follows: The term $\overline{\text{err}}$ is the performance of a hypothesis built from the dataset $\mathcal{D}$ measured on the same dataset $\mathcal{D}$. This measure refers to the training error of the hypothesis. Mathematically, it is given by:

$$\overline{\text{err}} = \frac{1}{p}\sum_{i=i}^{p} \delta(h(\mathbf{x}_i; \lambda, \mathcal{D}), y_i) \tag{29}$$

The term ϵ_0 denotes the average error obtained from bootstrap datasets not containing the example being predicted. In other words, ϵ_0 is computed by using a bootstrap dataset as training set and the remaining examples as test set. It is given by:

$$\epsilon_0 = \frac{1}{p}\sum_{i=1}^{p}\sum_{b \in C_i} \delta(h_b(\mathbf{x}_i), y_i)/B_i \tag{30}$$

where C_i is the set of indices of the bootstrap dataset not containing the ith sample, and B_i is a number of such bootstrap datasets.

Finally, the derivation of the coefficients of the Eq. (28) (namely 0.368 and 0.632) is complex, and thus is not described here (see [2] for details). In [8] there is an overview of the functions presented in this section. The following section shows the experiments performed to optimize these functions using the proposed GA.

4.4. Experimental studies

The experiments presented in this section compare four fitness functions: holdout, crossvalidation, bootstrap, and GCV. The holdout result was taken from Section 3.5. The crossvalidation function used ten folds (i.e., the 10-fold crossvalidation). Finally, the .632 bootstrap used 20 bootstrap datasets.

Note that in each chromosome evaluation using holdout only one training is carried out (namely the training over the holdout set). GCV also performs one training per evaluation. Whereas the 10-fold-crossvalidation performs 10 training sessions (owing to 10 folds) in each evaluation and bootstrap performs 20 training sessions (owing to 20 bootstrap datasets). Obviously, the crossvalidation and bootstrap consume much CPU-time. This is a drawback for GAs, once GAs need fast objective functions in order to reduce its processing time. A solution to this problem is shown in the next section.

Because holdout and GCV consume small CPU-time, they become attractive as fitness functions. The drawback is that holdout can lead the learning algorithm to overfit the holdout set, once it is always the same during the execution of the GA. Because GCV is not suitable for nonlinear models, it follows that GCV does not give a good estimate of the true error and overfits the dataset. Crossvalidation and bootstrap make a more efficient use of the dataset by re-sampling it several times. So, in the experiments performed, they presented, in general, better performance than holdout as it is confirmed in Fig. 12. However, in terms of complexity, holdout produced networks as small as those produced by 10-fold-crossvalidation (see Fig. 11).

5. The multiobjective Genetic Algorithm

In the case of optimizing multiple objectives, there is, in general, no best solution because the solution may be the best with respect to an objective and the worst with respect to another. Thus, there is usually a set of solutions, being not possible to say what is the best overall solution. Such set of solution is called the Pareto optimal solution. Formally:

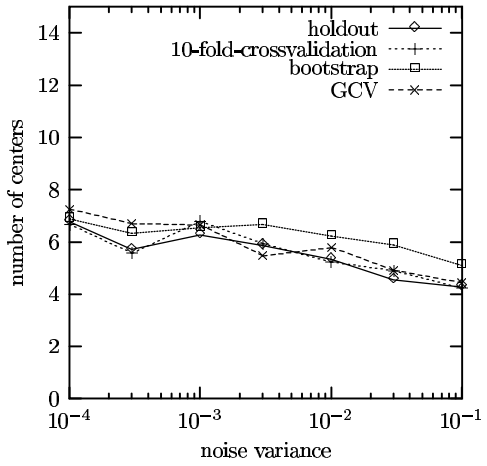

Fig. 11. Comparing the performance of several kinds of fitness in terms of the number of centers.

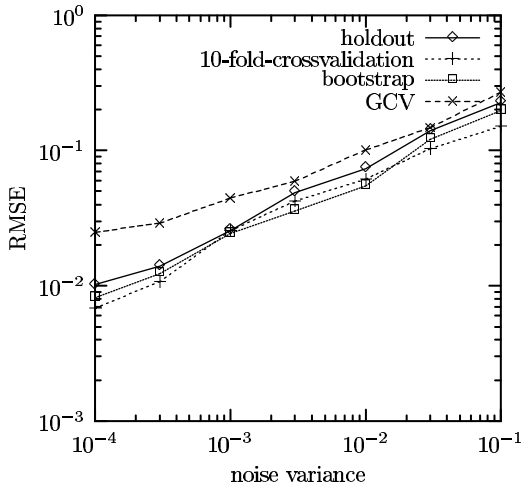

Fig. 12. Comparing the performance of several kinds of fitness in terms of the error on the test set.

Let $f_1, f_2, \ldots, f_q$ be the set of objective functions to be minimized. A chromosome fitness is defined by a vector where each component is the value of an objective function. In order to compare chromosomes through these vectors, the following definitions are used [4]:

Definition 1 *Let* **a** *and* **b** *be vectors of objective function values. A vector* **b** *is said to be dominated by* (*or inferior to*) *a vector* **a** *iff* **a** *is partially-less-than* **b** (*in symbols* $\mathbf{a} <_p \mathbf{b}$), *where:*

$$\mathbf{a} <_p \mathbf{b} \iff \forall i (a_i \leqslant b_i) \wedge \exists i (a_i < b_i) \qquad (31)$$

Definition 2 *A vector* **a** *is said to be non-dominated* (*or non-inferior*) *if there is not any other vector* (*in the population*) *that dominates* **a**.

procedure EvaluateWithShuffling(P)

for each λ in the population P **do**
  Set $\mathcal{D}^*$ equal to shuffle($\mathcal{D}$)
  Estimate the true error $\widehat{e}(\lambda, \mathcal{D}^*)$
  Set the fitness f of λ equal to $\widehat{e}(\lambda, \mathcal{D}^*)$
end for

Fig. 13. Evaluation procedure with shuffling.

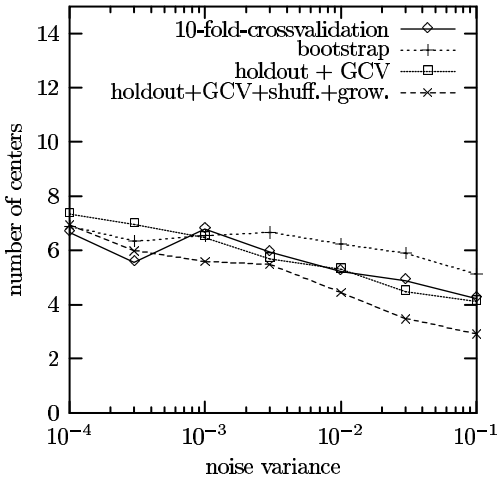

Fig. 14. Comparing the performance of the multiobjective GA in terms of the number of centers.

The Pareto-optimal set is defined as the set of all non-dominated vectors of the population. The goal of this multiobjective optimization is to find the Pareto-optimal set. Several GAs variants have been proposed in order to find the Pareto set (see [4,5] for more information).

5.1. Model selection

In this work, two inexpensive objective functions are optimized: holdout and GCV.

As before, the dataset $\mathcal{D}$ is divided into two parts: the training set $\mathcal{D}_t$, on which the hypothesis is built, and the holdout set $\mathcal{D}_h$. The first fitness of the chromosome λ is given by holdout function:

$$f_1(\lambda) = \frac{1}{|\mathcal{D}_h|} \sum_{(\mathbf{x},y)\in\mathcal{D}_h} \delta(h(\mathbf{x}; \lambda, D_t), y) \qquad (32)$$

The second fitness of the chromosome λ is given by GCV function over the dataset $\mathcal{D}_t$:

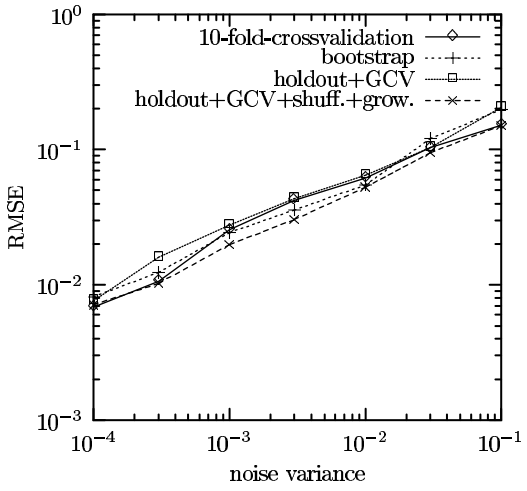

Fig. 15. Comparing the performance of the multiobjective GA in terms of the error on the test set.

$$f_2(\lambda) = \frac{p_t \, \mathrm{SSE}_t}{(p_t - m)^2} \tag{33}$$

where SSE_t denotes the sum of squared errors on the dataset $\mathcal{D}_t$, m is the number of weights (free parameters) and p_t is the number of examples in $\mathcal{D}_t$.

The occam elitism was adapted for the multiobjective GA as follows: all the simplest hypotheses of the Pareto set are kept from generation to generation. In the last generation, a criterion is needed to select an unique hypothesis. The following criterion was adopted: pick up the simplest hypothesis of the Pareto set. If there is more than one such hypothesis, then use the median of them.

Note that the holdout set is kept constant during all generations. Hence, the holdout function still may overfit the holdout set. In order to avoid the holdout to be set kept constant, the shuffling schema is proposed.

5.2. Shuffling

In order to improve the quality of the genetic hypotheses, the population evaluation procedure is modified by shuffling the dataset just before the fitness evaluation. Consider that shuffle() is a computer procedure that shuffles the dataset $\mathcal{D}$ producing the shuffled dataset $\mathcal{D}^*$. The modified evaluation procedure is shown in Fig. 13. It replaces the corresponding procedure from Fig. 1.

The shuffling's underling principle is to avoid the holdout set to be kept constant, and therefore avoid overfitting. Nevertheless, the shuffling transforms the holdout into a noise function. GAs, however, are robust to deal with noise functions [6]. In case of elitism, the chromosome that is kept from generation to generation should be revaluated in each generation in order to verify whether its good performance is not due to stochastic errors. Other heuristic investigated, growing, is presented in next section.

5.3. Growing

Stanley and Miikkulainen [19] observed that complexity in nature is developed over time, rather than introduced in the beginning. Based on this observation, they proposed evolving hypotheses starting with minimal hypotheses. This means to set the initial population with minimal networks (e.g., one basis function) instead of random (and probably large) hypotheses. New hypotheses are introduced incrementally as mutations occur. Only those hypotheses that are found to be useful survive through generations.

An experimental justification for the use of growing is as follows: In all experiments performed so far, the number of basis function was limited to 15 basis functions (see the number of regions parameter from Table 1, which also indicates the maximum number of basis function). This low limit was used because if GAs start with a higher limit (e.g., 35 basis functions), they may arrive to poor results and large hypothesis. This occurs due to the large hypothesis of the initial population that tends to dominate the population, causing a phenomena known as premature convergence. Such large hypotheses are suboptimal solutions, named superindividuals. The use of the growing approach seems to solve this problem, once it avoids the production of large hypotheses (superindividuals) in early stages of the evolution. Growing was successfully used for genetic design of RBF networks in [1]. Experiments involving the multiobjective GA and the heuristics discussed are presented in the next section.

5.4. Experimental studies

In the experiments carried out, the holdout function did not perform well because large hypotheses tend to overfit the holdout set. The same happened with the GCV method. However, the optimization of both holdout and GCV seems to be a promising idea, as suggest the results shown in Figs 14 and 15. The shuffling and growing heuristics incorporated into a GA improved its performance in both error on test set and complexity. The results obtained suggest that Multiobjective GA

with shuffling and growing is comparable to crossvalidation and bootstrap.

A possible explanation for this good result is that GCV penalizes large hypotheses (once its denominator diminishes in large hypotheses). As a consequence, GCV contributes to avoid the overfitting of the holdout. In spite of making the holdout function a noise function, the use of shuffling also contributes to avoid holdout overfitting (once the holdout set is not fixed). However, in some experiments involving optimization with an unique objective function, shuffling did not lead to good results (these results are not reported here). Growing adds basis functions incrementally, making the selection of centers more rigorous (once complexity is only added if necessary) and natural. The combined effect of all theses methods seems be useful for model selection.

6. Conclusion

Experiments showed that the use of the holdout method as objective function is not an effective method and that crossvalidation and boostrap are, in general, the best objective functions for model selection using GAs. Nevertheless, they consume much CPU-time. GAs need fast objective functions to have a reasonable processing time, so the use of crossvalidation and boostrap may not be suitable for GAs.

Experiments also showed that by means of modifications in the traditional GA, it is possible to make a GA (with holdout) an efficient model selection algorithm without the need to use crossvalidation and boostrap. Four modifications in the traditional GA were carried out which generated better hypotheses than those hypotheses generated by holdout using the traditional GA. The modifications are:

- Use of the occam elitism to keep the simplest hypothesis (among the best ones) from generation to generation. Experiments showed that the occam elitism diminishes the complexity of the hypotheses and their true error.
- Optimize two inexpensive objective functions (namely holdout and GCV) simultaneously seems to be better than optimize each one individually.
- Shuffle the dataset just before computing the fitness function may avoid the overfitting of the holdout method.
- Use of the growing mechanism to start the initial population with minimal hypotheses and add complexity incrementally. Growing avoids the bias towards large hypotheses in the initial population. Growing makes the evolving of complexity a more rigorous and natural process.

All of theses modifications are computationally inexpensive and, if combined, they may produce results equivalent to both crossvalidation and bootstrap, which require a large amount of computer processing. Thus, GAs may provide a suitable framework for the model selection problem, but this potential still needs be further explored.

Acknowledgment

The authors would like to thank FAPESP, CAPES, and CNPq for their support.

References

[1] A. Barreto and H. Barbosa, Growing compact RBF networks using a genetic algorithm, in: *VII Brazilian Symposium on Neural Networks*, 2002.
[2] B. Efrom and R.J. Tibshirani, *An Introduction to the Bootstrap*, Chapman and Hall, 1993.
[3] L.J. Eshelman and D.J. Shaffer, Real-coded genetic algorithms and interval-schemata, in: *Foundations of Genetic Algorithms 3*, D.L. Whitley, ed., San Mateo, CA: Morgan Kaufman, 1992, pp. 187–203.
[4] C.M. Fonseca and P.J. Fleming, Genetic algorithms for multiobjective optimization: formulation, discussion and generalization, in *Proceedings of the 5th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, Inc., San Mateo, 1993, pp. 416–423.
[5] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*, Wiley, 2000.
[6] D.E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, 1989.
[7] G.H. Golub, M. Heath and G. Wahba, Generalised cross-validation as a method for choosing a good ridge parameter, *Technometrics* **21**(2) (1979), 215–223.
[8] T. Hastie, R. Tibshirani and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Springer, 2002.
[9] S. Haykin, *Neural Networks: A Comprehensive Foundation*, second edition, Prentice Hall, 1999.
[10] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in *International Joint Conference on Artificial Intelligence* (*IJCAI*), 1995, pp. 1137–1145.
[11] E. Lacerda, A. Carvalho and T. Ludermir, Evolutionary optimization of rbf networks, *International Journal of Neural Systems* **11**(3) (2001), 287–294.
[12] D.J.C. MacKay, Bayesian interpolation, *Neural Computation* **4**(3) (1992), 415–447.
[13] T.M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[14] J. Moody and C.J. Darken, Fast learning in networks of locally-tuned processing units, *Neural Computation* **1**(2) (1989), 281–294.

[15] M.J.L. Orr, Regularisation in the selection of radial basis function centers, *Neural Computation* **7**(3) (1995), 606–623.

[16] M. Powell, The theory of radial basis function approximation in 1990, in: *Advances in Numerical Analysis*, (Vol. 3), W. Light, ed., Clarendon, Oxford, 1992, pp. 105–210.

[17] D.E. Rumelhart, G.E. Hilton and R.J. Williams, Learning internal representations by error propagation, in: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds, Mit Press, Cambridge, MA, 1986, pp. 318–362.

[18] A. Saha and J.D. Keller, Algorithms for better representation and faster learning in radial basis function networks, in: *Advances in Neural Information Processing Systems*, (Vol. 2), D.S. Touretzki, ed., 1990, pp. 482–489.

[19] K.O. Stanley and R. Miikkulainen, Evolving neural networks through augmenting topologies. Technical Report TR-AI-01-290, The University of Texas at Austin – Department of Computer Sciences, 2001.

[20] M. Stone, Crossvalidatory choice and assessment of statistical predictions, *Journal of the Royal Statistical Society* **B2** (1974), 111–147.

[21] X. Yao, Evolving artificial neural networks, *PIEEE: Proceedings of the IEEE* **87** (1999).