Aggregation Algorithms for Regression. A Comparison with Boosting and SVM Techniques

P.M. Granitto, P.F. Verdes and H.A. Ceccatto

Instituto de Física Rosario - CONICET/UNR
Boulevard 27 de Febrero 210 Bis, 2000 Rosario, Argentina

Abstract. Classification and regression ensembles show generalization capabilities that outperform those of single predictors. We present here a further evaluation of two algorithms for ensemble construction recently proposed by us. In particular, we compare them with Boosting and Support Vector Machine techniques, which are the newest and most sophisticated methods to treat classification and regression problems. We show that our comparatively simpler algorithms are very competitive with these techniques, showing even a sensible improvement in performance in some of the standard statistical databases used as benchmarks.

1 Introduction

In general, the combination of outputs of several predictors improves on the performance of a single generic one [11]. For this, they must be *diverse*, *i.e.*, they must have independently distributed predictions for the test points. This is possible when the learning algorithm is *unstable*[1], that is, very sensitive to small changes in the structure of the data and/or in the parameters defining the learning process. Classical examples in this sense are classification and regression trees and artificial neural networks (ANNs). In particular, in the case of ANNs the instability comes naturally from the inherent data and training process randomness, and also from the intrinsic non-identifiability of the model.

The combination of strong instability of the learning algorithm with the need for good individual generalization capabilities requires an adequate selection of the ensemble members. Attempts to achieve a sensible compromise between the above mentioned properties include elaborations of two general techniques: *Bagging* [1] and *Boosting* [4]. These standard methods for predictors aggregation follow two different strategies: Bagging (short for 'bootstrap aggregation'), and variants thereof,train independent predictors on bootstrap re-samples L_n $(n = 1, M)$ of the available data D, employing the unused examples $V_n = D - L_n$ for validation purposes. These predictors are then aggregated according to different rules (for instance, simple or weighted average). Boosting and its variants are, instead, stagewise procedures that, starting from a predictor trained on D, sequentially train new aggregate members on bootstrap re-samples drawn with modified probabilities. According to the general approach, each example in D is given a different chance to appear in a new training set by prioritizing patterns poorly learnt on previous stages. In the end, the predictions of the different

members so generated are weighted with a decreasing function of the error each predictor makes on its training data.

For regression problems, on which we will focus here, Boosting is still a construction area, where no algorithm has emerged yet as 'the' proper way of implementing this technique [4, 2, 5]. Consequently, Bagging is the most common method for regressors aggregation. In this work, two different Bagging-like strategies for ensemble construction recently proposed by us –SECA[6] and SimAnn[7]– are tested against Boosting and other sophisticated techniques in the literature. We will restrict ourselves to work in the regression setting, using ANNs as learning method. These restrictions are not essential; in principle, our analysis can be extended to classification problems and to other regression/classification methods. Our main purpose is to assess the efficacy of these two simple methods by comparison among themselves and with other modern techniques on several synthetic and real-world data sets.

The organization of this work is the following: In Section 2 we re-discuss the methods proposed in Refs. [6, 7]. In Section 3 we introduce the synthetic and real-world databases considered in this study, and describe the experimental settings used to learn from them. In Section 4 we obtain empirical evidence on the relative efficacy of the methods discussed in Section 2 by applying them to these databases. Finally, in Section 5 we summarize and draw some conclusions.

2 Ensemble Construction Algorithms

The simplest way of generating a regressor aggregate is Bagging [1]. According to this method, from the data set D containing N examples $(t, \mathbf{x})$ one generates bootstrap re-samples L_n $(n = 1, M)$ by drawing with replacement N training patterns. Thus, each training set L_n will contain, on average, $0.63N$ different examples, some of them repeated one or more times. The remaining $0.37N$ examples in $V_n = D - L_n$ are used for validation purposes in the regressor learning phase (backpropagation training of the ANN in our case). In this way one generates M different members f_n of the ensemble, whose outputs on a test point $\mathbf{x}$ are finally averaged to produce the aggregate prediction $\Phi(\mathbf{x}) = w_1 f_1(\mathbf{x}) + ... + w_M f_M(\mathbf{x})$. The weights w_n are usually taken equal to $1/M$ (simple averaging). Other options will be discussed below. Notice that, according to this method, all the regressors are trained independently and their performances individually optimized using the "out-of-bag" data in V_n. Then, although there is no fine-tuning of the ensemble members' diversity, the method frequently improves largely on the average performance of the single regressors f_n.

Bagging can be viewed as a first stage of more sophisticated algorithms for building a composite ANN regressor. Let's consider training to convergence M ANNs on the bootstrap re-samples L_n, saving the intermediate states $f_n(\tau)$ at each training epoch τ. Building an ensemble can be then translated to the task of selecting a combination of one state $f_n(\tau_n^{\rm opt})$ from each of the M runs to create an optimal ensemble. In this light, Bagging solves the problem by choosing the state

using only information on the given run ($\tau_n^{\rm opt}$ is the number of training epochs for which the validation error on V is minimum). In more advanced algorithms, the regressors are not optimized individually but as part of the aggregate. Within this framework, we have recently proposed[7] an 'optimal' way of constructing an ensemble based on the minimization of

$$E(\boldsymbol{\tau}) = \sum_{p=1}^{N} [t_p - \Phi_p(\mathbf{x}_p, \mathbf{w}(\boldsymbol{\tau}))]^2, \tag{1}$$

as a function of the set of training epochs $\boldsymbol{\tau} = \{\tau_n; n = 1, M\}$ for all networks. Here $\Phi_p(\mathbf{x}_p, \mathbf{w}(\boldsymbol{\tau})) = \sum_{n=1,M} w_{pn} f_n[\mathbf{x}_p, \mathbf{w}(\boldsymbol{\tau})]$ is the aggregate regressor built with those networks that have not seen pattern $(t_p, \mathbf{x}_p)$ in their training phase, *i.e.*

$$w_{pn} = \frac{\gamma_{pn}}{\sum_n \gamma_{pn}}, \tag{2}$$

where $\gamma_{pn} = 1$ if $(t_p, \mathbf{x}_p) \in V_n$ and 0 otherwise. Notice that the validation procedure generated by Eq. (1) amounts to effectively optimizing the performances of several subsets of the M trained ANNs, each subset including on average $0.37M$ networks. The advantage is that, like in the description of Bagging at the beginning of this section, no sub-utilization of data for validation purposes is necessary.

The minimization of Eq. (1) can be accomplished by using simulated annealing in $\boldsymbol{\tau}$-space. That is, starting from networks trained $\boldsymbol{\tau}_0$ epochs, we randomly change τ_{0n} and check whether the ensemble generalization error (1) increases or decreases when network n is trained up to $\tau_{0n} + \Delta\tau$. As usual, we accept the move with probability 1 when $E(\boldsymbol{\tau})$ decreases, and with probability

$$\frac{\exp\{-\beta[E(\boldsymbol{\tau}) - E(\boldsymbol{\tau}_0)]\}}{1 + \exp\{-\beta[E(\boldsymbol{\tau}) - E(\boldsymbol{\tau}_0)]\}} \tag{3}$$

when $E(\boldsymbol{\tau})$ increases. This is repeated many times considering different networks n (chosen either at random or sequentially), while the annealing parameter β is conveniently increased at each step; the algorithm runs until $E(\boldsymbol{\tau})$ settles in a deep local minimum. In practice we have taken $\Delta\tau = r\tau^{\rm max}/20$, where $\tau^{\rm max}$ is the maximum number of training epochs and r is a random number in the interval $[-1, 1]$. The annealing temperature was decreased according to $\beta^{-1} = 0.995^q E(\boldsymbol{\tau}_0)/2$, where q is the annealing step. We point out that the minimization problem is simple enough not to depend critically on these choices. Notice that for the implementation of this algorithm –that we have called "SimAnn"– one is forced to store all the intermediate networks $f_n[\mathbf{w}(\tau)]$. However, given the large storage capacity in computers nowadays, in most applications this requirement is not severe.

The SimAnn strategy for ANN aggregation minimizes some particular error function in a global way. A different approach is to adapt the typical hill-climbing search method to this problem. In a previous work[6] we proposed a simple way of generating a ANN ensemble through the sequential aggregation of individual

predictors, where the learning process of a new ensemble member is validated by the previous-stage aggregate prediction performance. That is, the early-stopping method is applied by monitoring the generalization capability on V_{n+1} of the n-stage aggregate predictor plus the $n+1$ network being currently trained. In this way we retain the simplicity of independent network training and only the validation process becomes slightly more involved, leading to a controlled over-training ("late-stopping") of the individual networks. Notice that, despite the stepwise characteristic of this algorithm (called SECA, for Stepwise Ensemble Construction Algorithm), it can be implemented after the parallel training of networks if desirable. Alternatively, if implemented sequentially it avoids completely the burden of storing networks at intermediate training times like in SimAnn.

For the sake of completeness, we summarize the implementation of SECA as follows:

Step 1: Generate a training set L_1 by a bootstrap re-sample from dataset D, and a validation set $V_1 = D - L_1$ by collecting all instances in D that are not included in L_1. Produce a model f_1 by training a network on L_1 until a minimum $e_f(V_1)$ of the generalization error on V_1 is reached.

Step 2: Generate new training and validation sets L_2 and V_2 respectively, using the procedure described in Step 1. Produce a model f_2 training a network until the generalization error on V_2 of the aggregate predictor $\Phi_2 = (f_1 + f_2)/2$ reaches a minimum $e_\Phi(V_2)$. In this step the parameters of model f_1 are kept constant and the model f_2 is trained with the usual (quadratic) cost function on L_2.

Step 3: Iterate the process until a number M of models is produced. A suitable M can be estimated from the behavior of $e_\Phi(V_n)$ as a function of n, since this error will stabilize when adding more networks to the aggregate becomes useless.

Let's consider a simple analysis of the computational cost involved in the implementation of the abovedescribed algorithms. Once the M ANNs have been independently trained and T networks saved along each training evolution, which is common to all the algorithms, Bagging requires a computational time $t \sim M \times T$ to select the best combination (essentially, the evaluation of the T ANN's validation errors for each of the M networks to find the corresponding minima). SECA and SimAnn require $\frac{(M+1)}{2} \times M \times T$ and $p \times M \times T$ network evaluations, respectively. Here we have written the number of simulated annealing steps $N_{\rm sa} = pT$, with p an arbitrary integer, to facilitate the comparison. In the following we will take $p \sim M$ to have a fair comparison between SECA and SimAnn. Notice, however, that the major demand from a computational point of view is the ANN training and not the network selection to build the ensemble. In practice, in the algorithms' evaluations in Section 4 we have taken $M = 20$, $T = 200$ and $p = 15$, with all the networks trained a maximum of $10T$ to $100T$ epochs, depending on the database.

As mentioned in the Introduction, a completely different strategy for building composite regression/classification machines is Boosting. For classification

problems, its main difference with Bagging is the use of modified probabilities to re-sample the training sets L_n. At stage n, the weights associated to examples in D are larger for those examples poorly learnt in previous stages, so that they eventually appear several times in L_n. In this way, the new predictor f_n trained on L_n specializes on these hard examples. Finally, the inclusion of f_n in the ensemble with a suitably-chosen weight allows the exponential decrease with boosting rounds n of the ensemble's training error on the whole dataset D. Notice that, in addition to the above mentioned modification of re-sampling probabilities, other differences with Bagging are: i) Boosting is essentially a stage-wise approach, which requires a sequential training of the aggregate members f_n, and ii) in the final ensemble these members are weighted according to their performances on the respective training sets L_n (using a decreasing function of the training error). A further consideration of this last characteristic will be done in the next Subsection, where we discuss a weighting scheme for bagged regressors alternative to the simple average considered in this section.

While Boosting is, as explained above, a well defined procedure in the classification setting, for regression problems there are several ways of implementing its basic ideas. Unfortunately, none of them has yet emerged as "the" proper way of boosting regressors. Without the intention of exhausting all the proposed implementations, we can distinguish two Boosting strategies for solving regression problems: i) by reducing them to classification problems and essentially changing example weights [4, 2], and ii) by forward stage-wise additive modelling, which modifies the target values to effectively fit residual errors[5, 3]. In order to compare with SECA and SimAnn algorithms described above, in this work we will implement the Boosting techniques from [2] and [5] as examples of these two different strategies.

2.1 Weighting Ensemble Members

For the Bagging-like ensemble construction algorithms above described, the final aggregate prediction on a test point is simply the mean of the individual predictions, without weighting the outputs of the ensemble members ($w_n = 1/M$, $n = 1, M$). This is not particularly wise for SECA, since some of these members may have poor generalization capabilities.In [6] we gave a typical example of the problems SECA can run into, and following general ideas from Boosting we proposed to modify the algorithm so that the output of the ensemble at the m-th stage becomes

$$\Phi_m(\mathbf{x}) = \sum_{n=1}^{m} w_n f_n(\mathbf{x}), \tag{4}$$

where w_n is a decreasing function of e_n, the MSE of the n-th member over D; *i.e.*, we weighted each ensemble member according to its individual performance on the whole dataset. This is the way in which Boosting reduces the importance of overfitted members in the final ensemble. We explored two different weighting

functions:

$$w_i = \frac{e_i^{-\alpha}}{\sum_j e_j^{-\alpha}}, \quad w_i = \frac{\exp(-\alpha e_i)}{\sum_j \exp(-\alpha e_j)}, \tag{5}$$

and showed that, for small to intermediate values of α, weighting produces better results than simply averaging the individual predictions. Since in [7] we found no major differences between both weighting laws, we will use, like in this reference, potential weighting with $\alpha = 2$ to aggregate regressors (the corresponding algorithms have been called W-SECA and W-SimAnn).

In Sections 4 we will show how efficient W-SECA and W-SimAnn are by comparing them with Boosting and other techniques in the literature, on real and synthetic databases. In the next section we briefly describe these databases and the experimental settings considered for this comparison.

3 Benchmark Databases and Experimental Settings

We have evaluated the algorithms described in the previous section by applying them to several benchmark databases: the synthetic Friedman #1, 2, 3 data sets and chaotic Ikeda map, and the real-world Abalone, Boston Housing, Ozone and Servo data sets. In the cases of the Friedman data sets we can control the (additive) noise level, which allows us to investigate its influence on the different algorithm's performances. We present the results for the Ikeda map together with those of real-world sets because the level of noise in this problem is fixed by its intrinsic dynamics. In addition, at the end of next section we will present results on the Mackey-Glass equation, which affords a more general comparison with other regression methods in the literature. In the following we give brief descriptions of the databases and the ANN architectures used.

- Friedman #1
  The Friedman #1 synthetic data set corresponds to training vectors with 10 input and one output variables generated according to

$$t = 10\sin(x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon, \tag{6}$$

  where ε is Gaussian noise and $x_1, \ldots x_{10}$ are uniformly distributed over the interval $[0, 1]$. Notice that $x_6, \ldots x_{10}$ do not enter in the definition of t and are only included to check the prediction method's ability to ignore these inputs. In order to explore the algorithm's performances in different situations we used different noise levels and training set lengths. The noise component was set to three levels: No noise (*i.e.*, $\varepsilon = 0$, labeled "free"), low noise (ε with normal distribution $N(\mu = 0, \sigma = 1)$), and high noise (ε with normal distribution $N(\mu = 0, \sigma = 2)$). We generated 1200 sample vectors for each noise level and randomly split the data in training and test sets. The data sets D had alternatively 50, 100 and 200 patterns, while the test set contained always 1000 examples. We considered ANNs with 10:h:1 architectures, where the number of hidden units $h = 6, 10$ and 15 for increasing number of patterns in the training set.

– Friedman #2
Friedman #2 has four independent variables and the target data are generated according to

$$y = x_1^2 + \sqrt{x_2 x_3 - (x_2 x_4)^{-2}} + \varepsilon \tag{7}$$

where the zero-mean, normal noise is adjusted to give noise-to-signal power ratios of 0 (no noise), 1:9 (low noise) and 1:3 (high noise). The variables x_i are uniformly distributed in the ranges

$$0 < x_1 < 100, \quad 20 < \frac{x_2}{2\pi} < 280, \quad 0 < x_3 < 1, \quad 1 < x_4 < 11 \tag{8}$$

The training sets contained 20, 50 and 100 patterns, and the test set had always 1000 patterns. We considered 4:h:1 ANNs, with $h = 4, 6$ and 8 according to the training set length.
– Friedman #3
Friedman #3 has also four independent variables distributed as above but the target data are generated as

$$y = \tan^{-1}\left[\frac{x_2 x_3 - (x_2 x_4)^{-2}}{x_1}\right] + \varepsilon \tag{9}$$

The noise-to-signal ratios were chosen as before, but in this case the training sets contained 100, 200 and 400 patterns. Accordingly we considered $h = 6, 8$ and 12. As in the previous cases, the test sets had always 1000 patterns.
– Abalone
The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope. To avoid this boring task, other measurements easier to obtain are used to predict the age. Here we considered the data set that can be downloaded from the UCI Machine Learning Repository (ftp to ics.uci.edu/pub/machine-learning-databases), containing 8 attributes and 4177 examples without missing values. Of these, 1045 patterns were used for testing and 3132 for training. The ANNs used to learn from this set had a 8:5:1 architecture.
– Boston Housing
This data set consists of 506 training vectors, with 11 input variables and one target output. The inputs are mainly socioeconomic information from census tracts on the greater Boston area and the output is the median housing price in the tract. These data can also be downloaded from the UCI Machine Learning Repository.
We considered 450 training examples and 56 data points for the test set. The ANNs used had a 11:5:1 architecture.
– Ozone
The Ozone data correspond to meteorological information (humidity, temperature, etc.) related to the maximum daily ozone (regression target) at a location in Los Angeles area. Removing missing values one is left with 330 training vectors, containing 8 inputs and one target output in each one. The

data set can be downloaded by ftp (to ftp.stat.berkeley.edu/pub/users/breiman) from the Department of Statistics, University of California at Berkeley.
We considered ANNs with 8:5:1 architectures and performed a (random) splitting of the data in training and test sets containing, respectively, 295 and 35 patterns.

- Servo
  The servo data cover an extremely non-linear phenomenon –predicting the rise time of a servomechanism in terms of two (continuous) gain settings and two (discrete) choices of mechanical linkages. The set contains 167 instances and can be downloaded from the UCI Machine Learning Repository.
  We considered 4:15:1 ANNs, using 150 examples for training and 17 examples for testing purposes.
- Ikeda
  The Ikeda laser map [8], which describes instabilities in the transmitted light by a ring cavity system, is given by the real part of the complex iterates

$$z_{n+1} = 1 + 0.9 z_n \exp\left[0.4i - \frac{6i}{(1+|z_n|^2)}\right]. \tag{10}$$

  Here we have generated 1100 iterates, using 100 in the training set and 1000 for testing purposes. After some preliminary investigations, we choose an embedding dimension 5 for this map and considered ANNs with a 5:10:1 architecture.

For each one of these databases we trained $M = 20$ independent networks, storing $T = 200$ intermediate weights and biases $\mathbf{w}(\tau)$ on long training experiments until convergence ($10T$ to $100T$ epochs, depending on the database). We considered this number of networks after checking on preliminary evaluations that there were no sensible performance improvements with bigger ensembles. With these 20 ANNs we implemented W-SECA and W-SimAnn ensemble construction algorithms. We tested these Bagging-like techniques and also boosted ANNs according to the Friedman [5] and Drucker [2] Boosting algorithms, considering in this case a maximum of 20 rounds for comparison.

The results given in the following section correspond to an average over 50 independent runs of the above-described procedures, without discarding any anomalous case (for Boston, Ozone and Servo databases we averaged over 100 experiments because the smaller test sets allow larger sample fluctuations). We will not indicate the variance of average errors, since these deviations only characterize the dispersion in performances due to different realizations of training and test sets. They have no direct relevance in comparing the average performances of different methods (in each run all the algorithms use the same 20 networks). This procedure guarantees that differences in the final ensemble performances are only due to the aggregation methods and/or validation settings.

Finally, at the end of Section 4 we make a final comparison of W-SECA and W-SimAnn with several other methods in the literature. This is done using as a test bed the chaotic Mackey-Glass time series:

– Mackey-Glass

The Mackey-Glass time-delay differential equation is a model for blood cell regulation. It is defined by

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{11}$$

When $x(0) = 1.2$ and $\tau = 17$, we have a non-periodic and non-convergent time series that is very sensitive to initial conditions (we assume $x(t) = 0$ when $t < 0$).

In order to compare with the results in [9] and [10], we have downloaded the database used by these authors and considered, like in these works, an embedding dimension $d = 6$ and 1194 patterns for training and 1000 patterns for testing purposes. After some preliminary investigations to optimize the number of hidden units, we considered ANNs with a 6:40:1 architecture.

4 Evaluation Results

The results quoted below are given in terms of the normalized mean-squared test error:

$$NMSE_T = \frac{MSE_T}{\sigma_D^2}, \tag{12}$$

defined as the mean-squared error on the test set T divided by the variance of the total data set D. According to this definition, $NMSE \simeq 1$ for a constant predictor equal to the data average and 0 for a perfect one. Then, its value allows to appraise both the predictor's performance and the relative complexity of the different regression tasks. Notice that, as indicated in the table captions, the results are given in units of 10^{-2}, so that all the errors are much smaller than 1 and, consequently, the predictions much better than the trivial data average.

In Tables 1 and 2 we present results obtained with two Boosting algorithms ("Drucker" [2] and "Residual" [5]). For these algorithms we used a maximum of 20 boosting rounds, which should produce a fair test considering the 20 ANNs ensembled in the Bagging-like W-SECA and W-SimAnn methods. Notice that for the 27 Friedman datasets the Boosting algorithms perform better than W-SECA and W-SimAnn only in three cases, and in these few cases the "Drucker" implementation is always the best performer. For the real-world databases and Ikeda map this implementation and W-SimAnn are the top performers.

Friedman #1	**Noise Free**			**Low Noise**			**High Noise**		
Length	50	100	200	50	100	200	50	100	200
W-Bagging	3.23	1.88	0.13	4.17	2.75	1.62	5.73	4.63	3.27
W-SECA	**3.13**	**1.76**	0.12	**4.10**	**2.49**	**1.47**	**5.69**	**4.40**	3.07
W-SimAnn	3.24	1.77	**0.11**	4.13	2.54	1.50	5.73	4.44	**3.07**
Residual	3.63	2.46	0.53	4.46	3.24	2.15	6.13	4.90	4.00
Drucker	3.32	1.98	0.60	4.10	2.68	1.67	5.78	4.50	3.21
Friedman #2	**Noise Free**			**Low Noise**			**High Noise**		
Length	20	50	100	20	50	100	20	50	100
W-Bagging	0.65	0.0076	0.0041	3.28	1.86	**1.53**	7.52	5.64	**4.90**
W-SECA	0.62	0.0079	0.0042	2.94	1.84	1.55	7.50	**5.56**	4.91
W-SimAnn	**0.49**	**0.0076**	**0.0041**	**2.93**	1.82	1.57	7.50	5.59	4.94
Residual	0.83	0.0116	0.0050	3.09	1.98	1.64	7.74	5.80	5.14
Drucker	1.20	0.0081	0.0044	3.14	**1.81**	1.56	**7.49**	5.57	4.96
Friedman #3	**Noise Free**			**Low Noise**			**High Noise**		
Length	100	200	400	100	200	400	100	200	400
W-Bagging	1.71	0.67	0.37	6.42	5.09	4.36	**12.50**	**11.08**	10.14
W-SECA	**1.64**	0.66	0.36	6.16	4.93	4.28	13.38	11.39	10.08
W-SimAnn	1.69	**0.65**	**0.35**	6.24	**4.90**	**4.26**	14.83	12.14	**10.07**
Residual	2.37	0.86	0.55	7.47	5.80	4.75	17.16	13.37	10.74
Drucker	1.77	0.72	0.40	**6.15**	5.00	4.28	13.24	11.52	10.10

Table 1 Normalized mean-squared test errors (in units of 10^{-2}) for the weighted versions of the algorithms indicated. These figures correspond to an average over 50 experiments, using out-of-bag data for validation purposes. The results of two different boosting algorithms are included for comparison.

Database	**Abalone**	**Boston**	**Ozone**	**Servo**	**Ikeda**
W-Bagging	4.644	2.503	3.931	1.840	16.64
W-SECA	4.626	2.482	3.887	1.845	15.10
W-SimAnn	4.631	2.498	**3.865**	1.823	**15.10**
Residual	4.646	2.638	4.028	2.172	22.07
Drucker	**4.624**	**2.479**	3.920	**1.778**	16.19

Table 2 Same as Table 1 for the real-world databases indicated.

As a further investigation on W-SECA and W-SimAnn, we have considered the Mackey-Glass problem. This allows us to make a comparison with seven other regression methods based on Support Vector Machines (SVM) and regularized Boosting using Radial Basis Function (RBF) networks, as described in [9] and [10]. Following these works, we introduced three levels of uniform noise to the *training* set, with signal-to-noise ratios of 6.2%, 12.4% and 18.6% respectively, and Gaussian noise with signal-to-noise ratios of 22.15% and 44.30% respectively. The test set is kept noiseless to measure the true prediction error. As mentioned in Section 4, to have a fair comparison all the experimental settings (training and test set lengths, embedding dimension, etc.) are the same as in [9] and [10]. Table

3 presents the corresponding results, which show that W-SECA and W-SimAnn are among the top performers in almost all cases, with only a performance edge in favor of SVM methods for the largest Gaussian noise case (we are disregarding the CG-k result for the largest uniform noise since it seems to be abnormally small).

Mackey-Glass	**Uniform Noise**			**Gaussian Noise**	
Noise Level	6.20%	12.40%	18.60%	22.15%	44.30%
CG-k	0.11	0.35	0.31	-	-
CG-ak	0.10	0.35	0.65	-	-
BAR-k	0.13	0.32	0.51	-	-
BAR-ak	0.12	0.27	0.66	-	-
SVM e-ins	0.07	0.28	0.57	0.58	3.23
SVM Huber	0.13	0.38	0.71	0.58	3.23
RBF-NN	0.16	0.38	1.54	0.65	3.90
W-Bagging	0.07	0.25	0.58	0.69	4.00
W-SECA	0.07	0.25	0.53	0.66	3.67
W-SimAnn	0.08	0.24	0.55	0.57	3.78

Table 3: Test set prediction errors (in units of 10^{-2}) for the Mackey-Glass problem using W-SECA and W-SimAnn. For comparison, we give the results of other methods in the literature taken from [10].

5 Summary and Conclusions

We have performed a comparison of simple methods for construction of neural network ensembles with more sophisticated Boosting and SVM methods for regression. In particular, we considered the W-SECA and W-SimAnn algorithms, previously proposed by us in Refs. [6, 7], which can be implemented with an independent (parallel) training of the ensemble members. The W-SimAnn algorithm uses simulated annealing to minimize the error on unseen data with respect to the number of training epochs for each individual ensemble member.

From this comparison we found that the algorithms we proposed are among the top performers in almost all situations considered (Tables 1-3). Given this competitive behavior of weighted Bagging-like algorithms, one is tempted to speculate that, for regression, the success of Boosting ideas might no be mainly related to the modification of resampling probabilities but to the final error weighting of ensemble members.

We had previously found [6, 7] that in general W-SECA and W-SimAnn produce better results than other similar algorithms in the literature (Bagging, NeuralBAG, Epoch). Although this holds true in several cases with more than 95% of statistical significance, the performance improvement obtained depends largely on the problem considered. The answer to the question as to whether these performances justify the use of W-SECA and W-SimAnn instead of, for instance, the simpler Bagging method, depends then on the concrete application.

However, a priori there is always a chance that using these algorithms one might obtain fairly large improvements. In any case, the best justification is perhaps the fact that not much additional computational time is required to implement them.

References

1. L. Breiman. Bagging predictors. *Machine Learning* 24:123-140, 1996.
2. H. Drucker. Boosting using Neural Networks. In *Combining Artificial Neural Nets*, Amanda J. C. Sharkey, editor, pages 51-77, Springer-Verlag, London, 1999.
3. N. Duffy and D. Helmbold. Leveraging for regression. In *COLT'00*, pages 208-219, 2000.
4. Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Seond European Conference on Computational Learning Theory*, pages 23-37, Springer Verlag, 1995.
5. J. Friedman. Greedy function approximation: A gradient boosting machine. Technical Report, Department of Statistics, Stanford University, 1999.
6. P. M. Granitto, P. F. Verdes, H. D. Navone and H. A. Ceccatto. A late-stopping method for optimal aggregation of neural networks. *International Journal of Neural Systems* 11:305-310, 2001.
7. P.M. Granitto, P.F. Verdes, H.D. Navone and H.A. Ceccatto. Aggregation Algorithms for Neural Network Ensemble Construction. SBRN 2002, VII Brazilian Symposium on Neural Networks, Recife, Brazil, 2002.
8. K. Ikeda. Multiple valued stationarity state and its instability of the transmited light by a ring cavity system. *Opt. Commun.* 30:257-261, 1979.
9. K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen and V. Vapnik. Predicting time series with support vector machines. In B. Schölkopf, C. J. C. Burges and A. J. Smola, editors, *Advances in Kernel Methods–support vector learning*, pages 243-254, Cambridge, MA: MIT Press, 1999.
10. G. Rätsch, A. Demiriz and K.P. Bennett, Sparse regression ensembles in infinite and finite hypothesis spaces. Machine Learning 48(1-3):189-218, 2002.
11. A. J. C. Sharkey, editor. *Combining Artificial Neural Nets.* Springer-Verlag, London, 1999.