Approximating Unknown Mappings: An Experimental Evaluation

Rafael Martí [#], Francisco Montes, Abdellah El-Fallahi

Departamento de Estadística e Investigación Operativa
Universitat de València, 46100 Burjassot (Valencia), Spain

Abstract

Different methodologies have been introduced in recent years with the aim of approximating unknown functions. Basically, these methodologies are general frameworks for representing non-linear mappings from several input variables to several output variables. Research into this problem occurs in applied mathematics (multivariate function approximation), statistics (nonparametric multiple regression) and computer science (neural networks). However, since these methodologies have been proposed in different fields, most of the previous papers treat them in isolation, ignoring contributions in the other areas. In this paper we consider five well known approaches for function approximation. Specifically we target polynomial approximation, general additive models (Gam), local regression (Loess), multivariate additive regression splines (Mars) and artificial neural networks (Ann).

Neural networks can be viewed as models of real systems, built by tuning parameters known as *weights*. In training the net, the problem is to find the weights that optimize its performance (i.e. to minimize the error over the training set). Although the most popular method for Ann training is back propagation, other optimization methods based on metaheuristics have recently been adapted to this problem, outperforming classical approaches. In this paper we propose a short term memory tabu search method, coupled with path relinking and BFGS (a gradient-based local NLP solver) to provide high quality solutions to this problem. The experimentation with 15 functions previously reported shows that a feed-forward neural network with one hidden layer, trained with our procedure, can compete with the best-known approximating methods. The experimental results also show the effectiveness of a new mechanism to avoid overfitting in neural network training.

KeyWords: Data Mining, Prediction, Metaheuristics, Statistical regression.

[#] Corresponding author. Rafael.Marti@uv.es

1. Introduction

The term *function approximation* encompasses a wide range of methodologies from computer science to statistics. It basically refers to adjusting a model that represents a mapping (generally non-linear) from several input variables to one or more output variables. In this paper we restrict our attention to mappings with one output variable. In mathematical terms, the goal is to model the dependence of the response variable y on one or more predictor variables $x_1, \ldots, x_n$ from a set of data observations $E=\{y_i, x_{1i}, \ldots, x_{ni}\}$ ($i = 1,\ldots, T$) generated with an unknown function f. The aim is to use the data to construct a function g that approximates f over the domain of interest.

In the process of adjusting the model (constructing g), the problem is to find the values of the parameters p that minimize the error across the set E of input/output observations (named training set). The error measure is typically the root mean squared difference between the predicted output $g(x,p)$ and the actual output value $f(x)$ for all the elements x in E (RMSE); therefore, the adjusting process can be viewed as an unconstrained nonlinear optimization problem, where the decision variables are the parameters and the objective is to reduce the fitting error *error(E,p)* according to the expression:

$$\underset{p}{Min}\ \ error(E,p)=\sqrt{\frac{\sum_{x\in E}(f(x)-g(x,p))^2}{|E|}}$$

The main goal in the fitting process is to obtain a model which makes good predictions for new inputs (i.e. to provide good generalization). Therefore g must represent the systematic aspects of E rather than its specific details. The standard way to measure the generalization provided by g consists of considering a second set of points called validation set V, in the domain of f. Once the fitting has been performed and the parameters have been set to minimize the error in E $(p=p^*)$, the error across V, *error(V,p*)*, is computed. The model must exhibit a good fit between the target f-values and the output (prediction) in the E as well as in V. If the RMSE in V is significantly higher than that in E, we will say that the model has over-fitted the data.

In this article we present some methods that may be used to identify the relationship between the input data and the outcome. These are selected from the more flexible statistical and mathematical methods. Specifically, in the second section we consider a first approach to the prediction problem called polynomial approximation and we illustrate the over-fitting problem. Section three is devoted to neural networks and our proposal for training (adjusting the net's parameters) and over-fitting prevention. In section four we target three modern statistical models, general additive models (Gam), local regression (Loess), and multivariate additive regression splines (Mars). The paper finishes with a computational comparison of 15 instances of the five models previously introduced and the associated conclusions.

2. Polynomial Approximation

A first approach to the approximation problem is the well-known polynomial fitting. If we want to approximate a two-variable function $y=f(x_1, x_2)$ with a polynomial of order two we will use the following expression:

$$g(x_1,x_2,p)=p_0+p_1x_1+p_2x_2+p_3x_1x_2+p_4x_1^2+p_5x_2^2$$

The coefficients are computed with the least square method to minimize the RMSE across the set of observations E. As expected, the larger the order of the polynomial the better the fitting. On the other hand, when we are trying to approximate an unknown mapping with an expression with a large number of independent terms, in particular a polynomial, we can find that many of them do not have a great effect on the prediction of the dependent variable. Therefore, in this context it can be desirable to choose a subset that parsimoniously predicts the dependent variable from among such terms or variables.

There are different methods for selecting the best subset of variables in order to obtain the adequate regression model. Specifically, we have considered the implementation of the *leaps* method given in the well-known R software package to select those terms that are relevant in the approximation from a generic polynomial. The R-function *leaps* evaluates all possible models and produces the *best* subset of independent variables based on the choice of a non-monotonic criterion. We have considered the R-squared criterion which is equivalent to the RMSE described above.

As mentioned in the introduction, one of the main problems when approximating a mapping is over-fitting. We will use the polynomial approximation to illustrate it in a simple case. Figure 1 shows a diagram with two sets of 10 observations in each one. The 10 bigger points constitute the training set E used for adjusting the function, while the 10 smaller points form the validation set V.

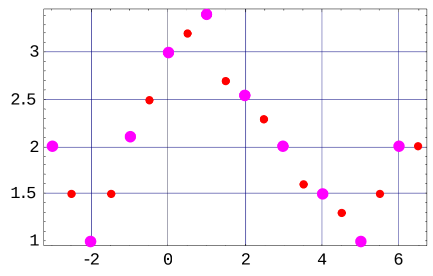

Figure 1. Training and validation data

Figure 2 shows the RMSE of six polynomial approximations, from order 5 to 10. As is known, since E contains 10 points the polynomial of order 10 fits them perfectly. However, in this case significant over-fitting (poor generalization) occurs. This figure depicts the RMSE (y-axis) across the training set (bottom line) as well as across the validation set V, for the different polynomial considered (the x-axis represents the order). We observe that both training and validation error decrease as the order increases, but when a certain level of accuracy is reached (order 6 in the figure), the validation error starts to increase (overfitting). Therefore, in this diagram, the best approximation (in terms of validation error) is obtained with the polynomial of order 6.

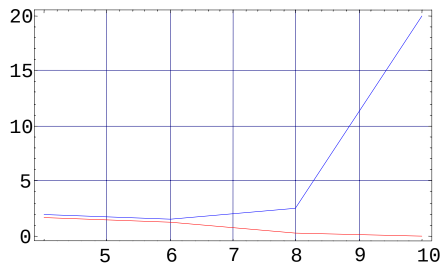

Figure 2. RMSE in E and V

We do not present the polynomial approximation as a competitive method, but as a baseline for future experiments as well as a *first try* that can provide good results in some cases. It is important to point out that we face the problem of approximating a set of data from an unknown function and we cannot reject this fitting method a priori.

3. Artificial Neural Networks

In this section we consider a different paradigm for function approximation: Neural networks. We briefly describe the network architecture, a method for training (adjusting the net's weights) and a proposal to prevent over-fitting in this context. Although some studies claim that any function can be approximated with a neural network (Hornik et al. 1989), in practice the picture is rather different, as the quality of the approximation depends on many different factors. We refer the reader to the book by Bishop (1995) for a deeper study of this methodology.

3.1 Architecture

We have considered the most widely-used architecture for prediction and classification: a multilayer feed-forward network with a single hidden layer. In particular, we target a two layer feed-forward network, with sigmoid activation function in the hidden nodes and linear activation in the output node.

Let $NN=(N, A)$ be an ANN where N is the set of nodes (neurons) and A is the set of arcs. N is partitioned into three subsets: N_I, input nodes, N_H, hidden nodes and N_O, output nodes. We assume that there are n variables in the function that we want to predict or approximate, therefore $|N_I| = n$. The neural network has m hidden neurons ($|N_H| = m$) with a bias term in each hidden neuron and a single output neuron (as mentioned in the introduction, in this paper we restrict our attention to real mappings $f: \Re^n \rightarrow \Re$). There is an arc, with an associated weight, from each node in N_I to each node in N_H, and from each node in N_H to the output node.

3.2. Training

Martí and El-Fallahi (2004) proposed a training algorithm based on tabu search (TS) methodology (Glover, 1989) to train a neural network. The authors performed a computational study of 12 different methods for neural network training to compare the best existing procedures, including the well-known backpropagation method (BP). The experiments show that some functions cannot be approximated with a reasonable level of accuracy when training the network for a limited number of iterations. The experimentation also shows that the Scatter Search approach by Laguna and Martí (2002) as well as the proposed TS method provide the best performance in terms of solution quality. We will use the core of their TS approach, called TSProb, to generate the initial set of solutions in our algorithm.

In the process of training the net, the search takes place in the space of weights. An iteration of TSProb begins by randomly selecting a weight from the current solution w. The probability of selecting weight w_i^t at iteration t, is proportional to the absolute value of the partial derivative of the RMSE on E with respect to w_i^t. These derivative values can be efficiently computed with the first phase of the BP method. The neighborhood consists of solutions that are reached from w^t by modifying the value of the selected weight w_i^t. Specifically, three solutions are considered with the following expression:

$$w_i^{t+1} = w_i^t + \alpha \, \beta \, w_i^t \; ; \; w_j^{t+1} = w_j^t \; , \; \forall j \neq i$$

The method selects the best solution from among the three considered (given appropriate α values), and labels it as w^{t+1}. Note that the move is executed even when the error of w^{t+1} is greater than the error of w^t, thus resulting in a deterioration of the current value of the objective function. The moved weight becomes tabu-active for *TabuTenure* iterations, and therefore it cannot be selected during this time. The factor β scales the change in the selected weight according to the status of the search (reducing its value from 1 as long as the current solution is close to a local optimum). Starting from a random initial solution, the TSProb method finishes after a number of k consecutive iterations with no improvement. The search parameters have been set to the values recommended by the authors: *TabuTenure*= $n(m+1)/3$,α=(0.3, 0.5, 0.8), $\beta\in[0,1]$, k=500.

Our procedure is based on the Path Relinking methodology (Laguna and Martí, 2003). It starts with the creation of the Reference Set (*RefSet*), which contains the b elite solutions found during the application of the TSProb method. These b solutions must be different and they must be far enough apart to ensure that the BFGS improvement method (Smith and Lasdon, 1992) starting from any two solutions will converge to different final solutions. Therefore, a solution is admitted to *RefSet* if its Euclidean distance from each solution already in the set is larger than a pre-specified threshold *th_d*. The improvement method is applied to the $b/2$ best solutions in *RefSet* and the improved solutions are ordered according to quality (i.e., to their *error(E,w)* value).

At each iteration of the path relinking algorithm, the set *NewPairs* is constructed with all pairs of solutions in *RefSet* that include at least one new solution (in the first iteration it contains $(b^2-b)/2$ pairs, but in successive iterations this number is usually significatively smaller). For each pair (w', w'') in *NewPairs* a path is initiated from w' to w'', and the best solution found in the path is added to the set *PRSol*. Once all the pairs in NewPairs have been subjected to the path relinking method, the BFGS algorithm is applied to the best b solution in *PRSol*. For each newly created solution we test whether it improves upon the worst solution currently in *RefSet*, in which case the new solution replaces the worst and *RefSet* is reordered. Then, if *RefSet* contains a new solution we perform another iteration of the path relinking algorithm, starting with the creation of the set NewPairs; otherwise, the algorithm terminates.

Finally, we describe the path relinking method, which constructs a path to join two solutions u and v. Considering the m neurons in the hidden layer in a given order, we construct a path with m solutions from solution u to solution v by performing moves that transform u into v. In the first step we create the first solution in the path, w^1, by replacing in u the values of the weights in the arcs from the n input neurons to the first hidden neuron with their values in v. Similarly, in the second step we create the solution w^2 by replacing in w^1 the values of the weights in the arcs from the n input neurons to the second hidden neuron with their values in v. We proceed in this way until we obtain solution w^m, which only differs from solution v in the values associated with the weights from the hidden layer to the output neuron.

The effectiveness of adding a local search exploration from some of the generated solutions within the relinking path has been well documented (Laguna and Marti, 2003). In the context of neural network training, the application of the BFGS as the improvement method is a time-consuming operation, so we have limited it to the best solution found in the path, as described above.

3.3. Prevention of Overfitting

In order to prevent overfitting (a significantly larger error in the validation set *V* than the error in the training set *E*), we divide *E* into two subsets E_1 and E_2. We use E_1 as a classical training set, i.e. to compute the error during the training in order to guide the search. But when we want to measure the quality of the current solution to check whether it improves the best solution found, we compute the error in E_2 and ignore the value of the error in E_1. So, although the search is conducted to reduce the error in E_1, we store the solution with a minimum error in E_2 as the best. The interplay between these two sets induces a subtle effect to identify solutions with good generalization properties (that present similar error values across different data sets), thus avoiding overfitting. As a result, this method is able to find solutions with similar RMSE values in E_1, E_2 as well as in *V* (and note that V was not used at all in the training process).

4. Modern Statistical Methods

In this section we consider recent regression procedures that make no assumption about the underlying functional relationship between the dependent and independent variables. Specifically we target GAM, Loess and MARS, providing brief descriptions of these methods. Further details can be found in the excellent book by Chambers and Hastie (1991).

4.1 Generalized Additive Models (GAM)

Generalized additive models extend linear models such as those described in the previous section. To this end, they replace the linear combination of variables with a linear combination of functions of these variables as:

$$g(x,p) = p_0 + \sum_j g_j(x_j),$$

where g_j is a non-parametric smooth function.

Given the set of data observations $E=\{y_i, x_{1i}, \ldots, x_{ni}\}$ ($i = 1,\ldots, T$), the functions g_j are computed to minimize the penalized residual sum of squares PRSS given by the expression:

$$PRSS(p_0, g_1, \ldots, g_n) = \sum_{i=1}^{T} \left\{ y_i - p_0 - \sum_{j=1}^{n} g_j(x_{ji}) \right\}^2 + \sum_{j=1}^{n} \lambda_j \int g_j''(t_j)^2 \, dt_j$$

The integral in the second additive term of the expression above measures the *wiggliness* of the g_j functions, and the λ_j are non-negative smoothing parameters that must be set in order to reach an equilibrium between the goodness of the fit (measured by the first additive term) and the wiggliness of the function. Smoother g_j are obtained with larger λ_j values.

The minimization of the PRSS leads to an additive cubic spline model in which each g_j is a cubic spline in x_j. Specifically, the expression of the g_j is obtained with the backfitting algorithm as described in Hastie et al. (2001). In the computational experiments we will use the *gam()* function given in the R software package to apply this method.

4.2 Local Regression (LOESS)

A classical smoothing method in time series is the well-known moving average in which each value is replaced with the average of some close values. These values are typically selected in a symmetrical neighborhood $N(x_0)$ around the value x_0 to be replaced. A first improvement over this generic scheme, called the kernel methods, consists of replacing this simple average with a weighted average by using kernel weighting. This kernel function $K_\lambda(x_0,x_i)$, where $x_i \in N(x_0)$, assigns a weight to x_i based on its distance from x_0. The parameter λ in the kernel specifies the width of the neighborhood. Nevertheless, this technique still has problems on the boundaries of the domain because of the asymmetry of the kernel. A way to overcome this limitation is given by the local regression. It consists of adjusting straight lines rather than the weighted average described above.

Loess implements a nonparametric method for estimating local regression surfaces (Cleveland, 1979). In the loess method, the weighted least squares method is used to fit linear functions of the predictors at the centers of neighborhoods of each x according to the expression:

$$\min_{p_0(x_0),p_1(x_0)} \sum_{i=1}^{n(x_0)} K_\lambda(x_0,x_i)\left[y_i - p_0(x_0) - p_1(x_0)x_i\right]^2 .$$

Note that we only use the fitted linear regression to evaluate the point x_0 to be replaced.

Compared to classical parametric approaches, local regression substantially increases the domain of surfaces that can be estimated without distortion. This method also increases the flexibility compared with traditional modeling tools because it can be used in situations in which we do not know a suitable parametric for the regression surface. We will use the implementation *loess()* given in the R software in our computational testing.

4.3 Multivariate Adaptive Regression Splines (MARS)

MARS is a spline regression model. The basic principle of splines is to divide the space of data into different regions and to approximate the function in each one by linear, quadratic or cubical regression. The main problem with this method, called curse of dimensionality, arises when there are several regions associated to each variable, when a large number of regions in the solutions space makes the method inapplicable.

MARS uses a specific class of linear splines to deal with the curse of dimensionality problem. This class C is formed with the functions

$$b(x-t) = (x-t)_+ \quad , \quad b(t-x) = (t-x)_+ ,$$

in which $b(x-t)$ takes the value $x-t$ when $x<t$ and 0 otherwise. Symmetrically, we define functions $b(t-x)$.

The model-building strategy is like a forward stepwise linear regression, but instead of using the original input we use M functions from the class C of splines considered. The model has the form

$$g(x,p) = p_0 + \sum_{j=1}^{M} p_j g_j(x) ,$$

where *gj(x)* belongs to the class *C* in the first step. In any step, new basis functions are created by means of products between the *gj(x)* functions already in the model and the original functions in *C*.

MARS is one of the most flexible nonparametric regression modeling techniques that attempts to overcome the limitations presented in other classical regression models. It can be viewed either as a generalization of the recursive partitioning regression strategy, or as a generalization of the additive modeling approach of Friedman and Silverman (1989). We will use the implementation *mars()* given in the R software in our computational testing.

5. Computational Experiments

For our computational testing, we implemented in C the most relevant previous training methods and the procedure described in section 3. We have also considered the statistical methods described in sections 2 and 4 as they are implemented in the R sofware package (1.8.1, 2003). Specifically, we consider the *leaps*() function for selective polynomial approximation and the *gam*(), *loess*() and *mars*() functions. All the experiments have been performed on a Pentium IV 2.8 Ghz personal computer.

1.Sexton 1:	$f(x) = x_1 + x_2$
2. Sexton 2:	$f(x) = x_1 * x_2$
3. Sexton 3:	$f(x) = \dfrac{x_1}{\lvert x_2 \rvert + 1}$
4. Sexton 4:	$f(x) = x_1^2 - x_2^3$
5. Sexton 5:	$f(x) = x_1^3 - x_1^2$
6. Branin:	$f(x) = \left(x_2 - \left(\dfrac{5}{4\pi^2} \right) x_1^2 + \left(\dfrac{5}{\pi} \right) x_1 - 6 \right)^2 + 10\left(1 - \dfrac{1}{8\pi} \right) \cos(x_i) + 10$
7. B2:	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$
8. Easom:	$f(x) = -\cos(x_1)\cos(x_2)\exp\left(-\left((x_1 - \pi)^2 + (x_2 - \pi)^2\right)\right)$
9. Goldstein:	$f(x) = \left(1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right) \left(30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right)$
10. Shubert:	$f(x) = \left(\sum_{j=1}^{5} j\cos((j+1)x_1 + j) \right) \left(\sum_{j=1}^{5} j\cos((j+1)x_2 + j) \right)$
11. Beal:	$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$
12. Booth:	$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$
13. Matyas:	$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$
14. SixHumpCamelB:	$f(x) = 4x_1^2 - 2.1x_1^4 + \dfrac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
15. Schwefel:	$f(x) = 418.9829n + \sum_{i=1}^{n} \left(-x_i \sin\sqrt{\lvert x_i \rvert} \right)$

Figure 3. Testing Functions

Sexton et al. (1998) perform a computational study over 5 functions with 2 variables. Laguna and Martí (2000) use the same set of instances. We have extended this set, adding 10 more functions of 2 variables from the literature. Figure 3 shows the 15 functions used to compare the performance of the methods under consideration. It should be noted that the new functions, from 6 to 15, present more complex expressions than the five previously reported. All functions have multiple local minima, and hence are difficult to fit.

The training set E consists of 200 observations with data randomly drawn from [-100, 100] for x_1 and [-10,10] for x_2. The validation set V consists of 100 observations drawn from the same uniform distributions. In all the methods presented in this section, we use the same set E to adjust the parameters of the model, and the same set V to compute the error on an independent set not used in the adjustment.

In our first experiment we have considered the polynomial approximation. Table 1 shows the RMSE on E (EE) and V (EV) in two different cases, polynomials of order 3 and 5. As described in section 2 we use the *leaps()* function in which the best subset of the terms in the polynomial is selected.

Np	**Order 3**		**Order 5**	
	EE	**EV**	**EE**	**EV**
1	0.00	0.00	0.00	0.00
2	0.00	0.63	0.00	0.06
3	4.53	7.06	15.15	30.48
4	0.00	1.03	0.00	1.03
5	0.00	98.21	0.00	98.21
6	5.09E8	1.94E9	4.76	1.3E6
7	0.32	1.04	0.34	1.04
8	0.00	0.00	0.00	0.00
9	1.20E17	2.10E17	1.44E16	1.75E17
10	26.24	31.33	26.24	31.33
11	2.34E8	7.32E8	1.02E8	4.08E8
12	0.00	5.04	0.00	5.04
13	0.00	0.25	0.00	0.25
14	1.32E10	5.46E10	2.21E9	2.22E10
15	18.56	43.93	9.28	21.47

Table 1. Polynomial approximation

As expected, some functions can be satisfactorily approximated with a polynomial while others present large error values. Examples 1, 2, 4, 5, 12 and 13, in which the data have been generated with polynomials of order lower than 5 (see Figure 3), present very low training and validation errors, except for example 5 where the validation error is several orders of magnitude larger than the training error (over-training). It should be noted that examples 7 and 8 also present good approximations although their data have been generated with a highly non-linear expression. Finally, note that examples 9, 11 and 14 generated by a polynomial of order larger than 5 have not been correctly approximated.

In our second experiment we compare the three statistical methods described in Section 4. Table 2 shows the EE and EV values for each of the 15 instances considered. This table shows that GAM and MARS methods are of a modest quality while Loess presents a better performance. If we compare the validation error of these methods with the polynomial approximation (order 5), Loess is able to obtain the lowest RMSE values in the validation set in 11 out of 15 instances, while the polynomial approximation, GAM and MARS, obtain the best results in 7, 4 and 5 cases respectively. Note that none of the methods can approximate examples 9, 11 and 14 satisfactorily. We perform a Friedman test for paired samples and obtain a p-value of 0.0001 which indicates that there are significant differences among the methods.

	Training Error (EE)			**Validation Error (EV)**		
Np	**GAM**	**Loess**	**MARS**	**GAM**	**Loess**	**MARS**
1	0.00	0.00	0.00	5.52	0.00	0.05
2	30.77	0.07	31.48	44.25	0.05	29.37
3	0.92	0.69	0.94	1.73	1.16	1.5
4	2.57	113.8	17.08	60.13	75.68	20.57
5	2.75	49.45	23.09	91.23	50.54	20.53
6	120.58	24.22	124.25	222.8	23.25	138.13
7	0.34	0.32	6.52	19.26	0.36	8.19
8	0.00	0.01	0.00	0.00	0.005	0.00
9	3.50E+10	3.66E+10	6.01E+09	1.34E+10	6.74E+09	8.20E+09
10	96.01	377.5	96.09	688.01	385.6	102.45
11	3.31E+06	3.18E+06	5.80E+06	6.76E+12	4.90E+06	7.74E+06
12	175.87	6.88E-08	185.22	229.75	8.37E-08	1.82E+02
13	163.33	1.71E-09	172.79	389.99	2.03E-09	172.68
14	3.89E+03	2.62E+04	6.79E+04	1.03E+05	2.97E+04	7.80E+04
15	48.55	0.69	222.4	410.07	0.74	231.34

Table 2. Statistical Methods

In the third experiment we are going to compare the training methods in neural networks. Specifically, we consider our training method based on tabu search and path relinking methodologies (TS_PR), the Back-Propagation method (BP), the extended tabu search method, ETS, of Sexton et al. (1998) and the Scatter Search method, SS, of Laguna and Martí (2002). SS implements the nonlinear Simplex method (Press et al., 1992) as the local search optimizer. BP and TS_PR are implemented with the method to avoid the overfitting described in section 3.3. It should be mentioned that our adaptation of the back-propagation algorithm (BP) includes the momentum term and compares favorably with commercial implementations and with that given in the R software package.

NP	**BP**	**ETS**	**SS**	**TS_PR**
1	1.69	0.00	0.00	0.00
2	6.30	1.68	0.01	0.02
3	1.82	0.67	0.36	0.08
4	148.36	11.45	0.05	0.16
5	9.80	14.79	17.54	0.33
6	17.79	43.25	89.53	0.25
7	64.19	60.63	31.53	0.38
8	0.11	0.01	0.02	0.07
9	1.22E+10	7.72E+09	3.68E+09	4.87E+09
10	13.66	20.13	33.51	20.39
11	4.19E+6	6.62E+06	3.75E+06	2.69E+06
12	104.34	152.45	0.00	0.06
13	3.94	9.03	0.55	0.00
14	5.96E+4	4.46E+04	1.87E+04	1.08E+03
15	2.14	523.79	79.18	0.12

Table 3. Neural Network Validation Errors

Tables 3 shows that the best solution quality is obtained with the tabu search method with path relinking, TS_PR, in most of the cases. Considering the validation errors, the PR method is able to obtain the best solutions in 8 instances, while the SS method obtains 4 best solutions. BP and ETS obtain results of a lower quality. This experiment also shows that none of the methods can effectively handle problems 9, 11 and 14 as is the case with the previous techniques. If we remove these three instances, the average of the RMSE on the testing set *E,* EE, on the thirteen remaining instances is 28.50, 78.75, 0.57 and 1.59 for the BP, ETS, SS and TS_PR respectively. The average of the RMSE on the validation set *V*, EV, on these thirteen instances is 31.18, 69.82, 21.02 and 1.82 for the BP, ETS, SS and TS_PR respectively. We can see that in the SS method the average error value significantly increases from EE=0.57 to EV=21.02, while in the BP and TS_PR it is of a similar magnitude. We also perform a Friedman test for paired samples and obtain a p-value of 0.0001 which indicates that there are significant differences among the methods.

If we rank all the methods considered in the paper according to the quality in the validation set we see that the neural network trained with our TS_PR implementation is able to obtain 9 out of 15 best solutions, followed by the Loess with 6 best results. The polynomial approximation obtains 5 best results and the remaining of the methods present lower quality solutions.

In our last experiment we want to measure the effect of the initial solution in the performance of the proposed training method TS_PR for neural networks. Table 4 reports the average and standard deviation of the RMSE on *E* and *V* in 20 independent runs of the method.

NP	EE	EV
1	0,00± 0,00	0,00 ± 0,00
2	0,00± 0,00	0,01 ± 0,01
3	0,00± 0,00	0,15 ± 0,10
4	0,00± 0,00	0,30 ± 0,09
5	0,00± 0,00	0,41 ± 0,12
6	0,09± 0,04	0,26 ± 0,11
7	0,25± 0,00	0,39 ± 0,10
8	0,00± 0,00	0,05 ± 0,02
9	1,37E9±1,6E8	8,5E9±5,6E1
10	16,14±1,67	19,14 ± 1,41
11	1,8E6±1,36E5	2,93E6±4,36E5
12	0,01±0,00	0,13 ± 0,12
13	0,00±0,00	0,01 ± 0,02
14	1,34E4±8,34E3	5,7E4±5,7E3
15	0,02 ± 0,00	0,16 ± 0,04

Table 4. 20 independent runs for TS_PR

Table 4 shows the robustness of the proposed method given that the standard deviation presents relatively low values. Moreover, the similarity between the EE and EV values confirms the effectiveness of the proposed method to avoid overfitting.

Conclusions

A computational comparison of 5 existing methodologies for function approximation has been presented. Experiments with 15 problems were performed to compare the procedures. This experimentation allows us to conclude that Neural Networks trained with metaheuristics seem better suited to tackling this problem. Specifically, our implementation based on tabu search methodology (including the path relinking strategy) and coupled with the BFGS local solver is able to obtain the best solutions in terms of quality. We have also implemented a new mechanism to prevent overfitting that has been proven to be effective in this context.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* (Grant Refs. TIC2003-C05-01) and by the *Agencia Valenciana de Ciencia y Tecnologia* (Grant Refs. GRUPOS2003/189).

References

Bishop, C.M. (1995), Neural Networks for Pattern Recognition, Oxford University Press, New York.

Chambers, J. and T. Hastie (1991), Statistical models in S, Wadsworth/Brooks Cole, Pacific grove, CA.

Cleveland, W.S. (1979) "Robust Locally Weighted Regression and Smoothing Scatterplots," Journal of the American Statistical Association, Vol. 74, pp. 829-836.

Fahlman, S.E. (1988) "An empirical study of learning speed in back-propagation networks", In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, Connectionist Models Summer School, San Mateo, CA, Morgan Kaufmann, pp. 38-51.

Friedman, J. H. and Silverman, B. W. (1989), Flexible parsimonious smoothing and additive modelling (with discussion), Technometrics, 31, 3-39.

Glover, F. (1989) "Tabu Search-Part 1", ORSA Journal on Computing, vol 1, pp. 190-206.

Hastie, T., R. Tibshirami and J. Friedman (2001), The elements of statistical learning, Springer, New-York.

Hornik, K., M. Stinchcombe and H. White (1989) "Multilayer feedforward networks are universal approximators", Neural networks, vol. 2, pp. 359-366.

Jacobs, R.A., (1988) "Increased Rates of Convergence Through Learning Rate Adaptation", Neural Networks, 1, pp. 295-307.

Laguna, M. and R. Martí (2002) "Neural Network Prediction in a System for Optimizing Simulations," IIE Transactions, vol. 34(3), pp. 273-282.

Laguna, M. and R. Martí (2003) Scatter Search: Methodology and Implementations in C, Kluwer Academic Publishers.

Martí, R. and A. El-Fallahi (2004) "Multilayer neural networks: an experimental evaluation of on-line training methods" Computers and Operations Research 31, pp. 1491-1513.

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1992) Numerical Recipes: The Art of Scientific Computing, Cambridge University Press (www.nr.com).

R Development Core Team. (2003), "R: A Language and Environment for Statistical Computing", "http://www.R_project.org ".

Sexton, R. S., B. Alidaee, R. E. Dorsey and J. D. Johnson (1998) “Global Optimization for Artificial Neural Networks: A Tabu search Application,” European Journal of Operational Research, vol. 106, pp. 570-584.

Smith, S. and L. Lasdon (1992), "Solving Large Nonlinear Programs Using GRG," ORSA Journal on Computing, Vol. 4, No. 1, pp. 2-15.