Path Relinking and GRG for Artificial Neural Networks*

Abdellah El-Fallahi, Rafael Martí #

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

Leon Lasdon

Management Science and Information Systems Department
The University of Texas at Austin, Austin, TX, 78712, USA

Abstract

Artificial neural networks (ANN) have been widely used for both classification and prediction. This paper is focused on the prediction problem in which an unknown function is approximated. ANNs 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 training these networks is back propagation, other optimization methods such as tabu search or scatter search have been successfully applied to solve this problem. In this paper we propose a path relinking implementation to solve the neural network training problem. Our method uses GRG, a gradient-based local NLP solver, as an improvement phase, while previous approaches used simpler local optimizers. The experimentation shows that the proposed procedure can compete with the best-known algorithms in terms of solution quality, consuming a reasonable computational effort.

KeyWords: Optimization, Metaheuristics, Prediction.

* Research partially supported by the Ministerio de Ciencia y Tecnología of Spain: TIC2000-1750-C06-01.

Corresponding author. Rafael.Marti@uv.es

1. Introduction

Artificial neural networks offer a general framework for representing non-linear mappings from several input variables to several output variables. They are built by tuning a set of parameters known as *weights,* and can be considered as an extension of the many conventional mapping techniques. In classification or recognition problems the net's outputs are categories, while in prediction or approximation problems they are continuous variables. Although this paper is focused on the prediction problem, most of the key issues in the net functionality are common to both.

In the process of training the net (*supervised learning*), the problem is to find the values of the weights w that minimize the error across a set of input/output pairs (patterns) called the training set E. For a single output and input vector x, the error measure is typically the root mean squared difference between the predicted output $p(x,w)$ and the actual output value $f(x)$ for all the elements x in E (RMSE); therefore, the training is an unconstrained nonlinear optimization problem, where the decision variables are the weights and the objective is to reduce the training error. Ideally, the set E is a representative sample of points in the domain of the function f that we are approximating; however, in practice it is usually a set of points for which we know the f-value.

$$\min_{w} \; error(E,w) = \sqrt{\frac{\sum_{x \in E} (f(x) - p(x,w))^2}{|E|}}$$

The main goal in the design of an ANN is to obtain a model which makes good predictions for new inputs (i.e. to provide good generalization). Therefore the net must represent the systematic aspects of the training data rather than their specific details. The standard way to measure the generalization provided by the net consists of introducing a second set of points in the domain of f called the testing set T. We assume that no point in T belongs to E and $f(x)$ is known for all x in T. Once the optimization has been performed and the weights have been set to minimize the error in E $(w=w^*)$, the error across the testing set T is computed ($error(T,w^*)$). The net must exhibit a good fit between the target f-values and the output (prediction) in the training set and also in the testing set. If the RMSE in T is significantly higher than that one in E, we will say that the net has *memorized* the data, instead of *learning* them (i.e., the net has over-fitted the training data).

If we consider a net with too few weights, it will not be able to fit the training data, while too many weights may provide a perfect fit to the training data, but a poor fit over the test set T. There is no direct rule to compute the optimum number of parameters when designing a neural net. The tradeoff between the flexibility and generalization properties of the system must be empirically determined in each particular case.

Several models inspired by biological neural networks have been proposed throughout the years, beginning with the *perceptron* introduced by Rosenblatt (1962). He studied a simple architecture where the output of the net is a transformation of a linear combination of the input variables and the weights. The transformation g is a threshold activation function where $g(x) = -1$ for $x<0$ and $g(x) = 1$ for $x>0$. This model has been the subject of extensive study in the statistics literature under the generic name of " logistic discrimination" where the activation function is given by the sigmoid function $g(x) = 1/(1+e^{-x})$, which produces a smooth approximation to the original step function.

Minskey and Papert (1969) showed that the perceptron can only solve linearly separable classification problems and is therefore of limited interest. A natural extension to overcome its limitations is given by the so called *multi-layer-perceptron* or simply multilayer neural networks, consisting of a set of nodes *N* and a set of arcs *A*. In 2-layered networks, *N* is partitioned into three subsets: N_I, input nodes, N_H, hidden nodes and N_O, output nodes. The arcs go from N_I to N_H or from N_H to N_O; in this sense we say that the net is a layered graph. We assume that there are *n* input variables, so $|N_I| = n$, and a single output. The neural network has *m* hidden neurons ($|N_H| = m$) with a bias term in each hidden neuron.

Kolmogorov (1957) proved that every continuous function of several variables can be represented as the superposition of a small number of one-variable functions. This result explains, in a theoretical sense, why neural networks work, but in practice it is of limited interest. If we put Kolmogorov's theorem in neural network terms, we will obtain a net with non-smooth activation functions that depend on the mapping that we are trying to approximate and provide poor generalization. Recent studies have proved that multilayer networks with one hidden layer and specific "squashing" functions are able to approximate any function. Hornik et al. (1989) show that neural networks are universal approximators and they conclude that any lack of success in applications must arise from inadequate learning or an insufficient number of hidden units. However, practical results show that in some "difficult" functions, neural networks provide approximations of low quality.

Blum and Li (1991) proved that a neural network having two layers and sigmoidal hidden units can approximate any continuous mapping arbitrarily well. As consequence, regarding the classification problem, two layer networks with sigmoidal units can approximate any decision boundary to arbitrary accuracy. However, Gallant and White (1992) showed that, from a practical point of view, the number of hidden units must grow as the size of the data set to be approximated (or classified) grows.

Barron (1993) showed that the training error decreases as the number of parameters increases, independently of the number of input variables. In other approximation models, such as polynomial approximation, the reduction in the error is considerably lower and depends on the dimension of the input space. Moreover, the number of parameters in these other models usually grows exponentially with the number of input variables, while the number of weights grows linearly or quadratically in the neural networks models. However, as it has been mentioned, the objective of these models is to make good predictions and increasing the number of weights reduces the networks ability to generalize the data. A view on these theoretical aspects and proofs of the results above are outlined in the excellent book due to Bishop (1995).

In this paper we will consider two layer feed-forward networks since this is the most common architecture used. We explore the behavior of the most relevant optimization methods developed for training neural networks. These include back propagation, a well-known gradient descent technique, as well as other non-linear optimization methods such as conjugate directions set or conjugate gradients (Martí and El-fallahi, 2003). Recently, metaheuristics such as tabu search (Sexton, 1998) or scatter search (Laguna and Martí, 2002) have been also adapted to this context, for the special case of optimizing simulations.

We propose a new training method based on the path relinking methodology. Path relinking starts from a given set of elite solutions obtained during a previous search process. Following the notation in Martí et al. (2003) we will call this set RefSet. Section 2 describes the construction of the initial RefSet using the tabu search algorithm for on-line training by Martí and El-Fallahi (2003), its updating, as well as the

outline of our path relinking method. Path relinking and its "cousin" scatter search are mainly based on two elements: combinations and local search. Path Relinking generalizes the concept of combination beyond its usual application to consider paths between solutions. Local search intensifies the search by seeking local optima. Section 3 explores two different relinking elements while Section 4 describes two well known local optimizers for non linear optimization: the (nonlinear) Simplex search method and GRG (Generalized Reduced Gradient method). The paper concludes with an empirical comparison of the proposed method with the best previous approaches and the associated conclusions.

2. Path Relinking and the Reference Set

Basic scatter search and path relinking implementations are designed to check that the reference set does not contain duplications; however, sometimes we can find that although the values of two points differ, they represent the same solution of the problem. This is the case with neural network training, where different symmetries are present (Bishop, 1995).

If we change the sign of all weights of the arcs from the input layer to a particular hidden unit, then for a given net input, we will obtain the same input value in this unit but with the opposite sign. Since the activation is an odd function, the output of this hidden layer will be the same as in a net with the original weights but with the opposite sign. If we also change the sign of the weight from this hidden unit to the output unit, the change in sign is compensated and the net's output is unchanged. Therefore, given a solution, for each hidden unit, we can obtain another solution which gives rise to the same mapping. Since we can do this for each hidden unit, or groups of units, we can obtain 2^m solutions representing the same mapping.

On the other hand, if we exchange the values of all the weights of the arcs incident to a hidden unit with those of the arcs incident with another hidden unit (considering both output and input arcs in the unit), we will obtain a different solution that represents the same mapping. Since we can interchange the values of the weights among all the hidden units, we can obtain m! different solutions representing the same mapping. If we add the symmetries previously described, we find that for each solution w in the solution space, there are a total of $m!2^m$ solutions representing the same mapping. Therefore, before adding a solution to the RefSet, we check that it is different than those currently in it (where 'different' means that the solution doesn't represent the same mapping of any other solution in the RefSet).

The initial Reference Set consists of the best solutions found during a previous tabu search application. Martí and El-Fallahi (2003) proposed a tabu search approach for neural network training when run time is limited. We will use the core of their approach to generate the initial set of elite solutions in the path relinking algorithm. A description of the TSProb method follows. An iteration 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

$$\left| \frac{\partial error(E, w^t)}{\partial w_i^t} \right| .$$

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 algorithm starts by evaluating two solutions; the first one with α=0,3 and the second one with α=0,5. If the error associated with the first one is lower than that one associated with α=0,5, then α=0,1 provides the last solution considered; otherwise, the method computes the solution generated with α=0,8. The method selects the best solution among the three considered, 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. The algorithm starts with β =1 and reduces the magnitude of this change as long as the current solution is close to a local optimum. Starting from a random initial solution, the *TSProb* phase stops when *b* diverse local solutions have been found.

Figure 1 shows the outline of our path relinking algorithm for neural network training. It starts with the creation of the Reference Set, which contains the *b* elite solutions found during the application of the TSProb method. These *b* solutions must be different as described above (they do not represent the same mapping) and they must be far enough apart to insure that the improvement method (Simplex or GRG) started from any two solutions will converge to different final solutions. This is possible because the training error function *error(E,w) has many local minima.* Therefore, a solution is admitted to the 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 the RefSet and the improved solutions are ordered according to quality (i.e., to their *error(E,w)* value).

0. Select the training set *E* and the testing set *T*. Normalize input and output data.
1. Build *RefSet* = { $w^{(1)}$, …, $w^{(b)}$ } with the best solutions found with the TSProb method. Apply the improvement method (with stopping condition modified as explained below) to the best $b/2$ solutions in *RefSet*.
2. Order *RefSet* according to their objective function value such that $w^{(1)}$ is the best solution and $w^{(b)}$ the worst. Compute *E_best=error(E,*$w^{(1)}$*)* and *T_best* as the minimum of *error(T,*$w^{(i)}$*)* for *i*=1,…,*b*. Set *T_Improve* = 0.

while (*T_Improve* < *T_Limit*) **do**

3. Generate *NewPairs*, which consists of all pairs of solutions in *RefSet* that include at least one new solution. Make *NewSolutions* = ∅.

for (all *NewPairs*) **do**

4. Select the next pair ($w^{(i)}$, $w^{(j)}$) in *NewPairs*.
5. Obtain new solutions in the path from $w^{(i)}$ to $w^{(j)}$ and add the best one to *NewSolutions*.

end for

6. Select the best *b* solutions in *NewSolutions* and apply the improvement method.

for (each improved *w*) **do**

if (*w* is not in *RefSet* and *error(E,w)*<*error(E,*$w^{(b)}$*)*) **then**

7. Make $w^{(b)}$ = *w* and reorder *RefSet*.

end for

8. Make *T_current* = the minimum of *error(T,*$w^{(i)}$*)* for *i*=1,…,*b*.

if (*T_current* < *T_best*) **then**

9. Make *T_best=T_current* and *T_improve*=0.

else

10. Make *T_improve*= *T_improve* +1.

end while

Figure 1. Outline of the path relinking procedure.

The search is initiated by assigning the value of 0 to the variable *T_Improve*. This variable stores the number of consecutive iterations without improvement in the best testing error found *error(T,w)*. In step 1, note that although the improvement method minimizes the *error(E,w)* value, it is stopped when *error(T,w)* has not been improved for the last *T_Limit* iterations to avoid over-training. In step 3, *NewPairs* is constructed with all pairs of solutions in *RefSet* that include at least one new solution, in order to perform a relinking phase. The procedure does not allow two solutions to be subjected to the relinking phase more than once. For each pair (w', w'') a path is initiated from w' to w''. The best solution found in the path is added to the set *NewSolutions*. The cardinality of *NewSubsets* corresponding to the initial reference set is given by $(b^2-b)/2$, which accounts for all pairs of solutions in *RefSet*. In step 6 the improvement method is applied to the best b solution in *NewSolutions*. If a newly created solution improves upon the worst solution currently in *RefSet*, the new solution replaces the worst and *RefSet* is reordered in step 7.

Note that this procedure is very aggressive in trying to improve upon the quality of the solutions in the current reference set, even if it sacrifices search diversity. The updating of the reference set is based on improving the quality of the worst solution and the search terminates when no new solutions are admitted to *RefSet*.

In the step 3 of the algorithm (see Figure 1) we construct the set *NewPairs*, which consists of all pairs of solutions in *RefSet* that include at least one new solution. We apply the relinking process to the pairs in this set in step 5. This prevents considering again a pair already re-linked in previous iterations (note that the relinking method described in the next section is deterministic, so it would produce the same sequence of solutions).

3. The Relinking Phase

Since the training problem is a nonlinear unconstrained problem with continuous variables (weights), a natural way to perform a combination of two solutions u and v is to create a linear combination of them. So, in path relinking terms, we can say that if u is the *initiating* solution and v is the *guiding* solution, we can create a path linking both solutions by giving values to $\alpha \in (0,1)$ in the expression $w = u + \alpha (v - u)$.

Instead of directly producing a single solution when combining two original solutions, this approach produces a set of solutions that can be interpreted as a sequence (according to the values of α). We refer to this method as LINEAR. It was proposed in the scatter search algorithm for neural network training due to Laguna and Martí (2000).

We introduce now a different method of combining that fits better with the relinking concept of incorporating attributes from one solution into another. The path relinking approach is oriented to the goal of choosing moves that introduce the attributes of the guiding solution, in order to create a "good attribute composition" in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from a set of attributes of the guiding solution.

Consider that the weights in a solution w are numbered sequentially starting with those from the n neurons in the input layer to the first hidden neuron: $w_{1,1}$, $w_{2,1}$, …, $w_{n,1}$ and the bias term for this first hidden neuron $w_{n+1,1}$. We will say that these n+1 weights are associated with hidden neuron 1. Therefore, each solution has $(n+1)*m+m+1$ weights where the firsts n+1 are associated with hidden neuron 1, the next n+1 with hidden

neuron 2 ($w_{1,2}$, $w_{2,2}$, …, $w_{n,2}$, $w_{n+1,2}$) and so on, up to those associated with hidden neuron m and, additionally, the m from the hidden layer to the output and the bias of the output.

In our implementation, if the neural network has m hidden neurons, 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 associated with the hidden neuron 1 with their values in v:

$$w_{i,j}^1 = \begin{cases} v_{i,j} & i = 1,..,n+1,\ j = 1 \\ u_{i,j} & otherwise \end{cases}$$

Similarly, in the second step we create the solution w^2 by replacing in w^1 the values of the weights associated with the hidden neuron 2 with their values in v:

$$w_{i,j}^2 = \begin{cases} v_{i,j} & i = 1,..,n+1,\ j = 2 \\ w_{i,j}^1 & otherwise \end{cases}$$

Therefore, in each step we copy into the current solution the v-values of the weights associated with a hidden neuron. Solution w^m only differs from solution v in the values associated with the weights from the hidden layer to the output neuron. We will call to this method, that progressively adds to one solution the characteristics of the other, Block Swap.

The effectiveness of adding a local search exploration from some of the generated solutions within the relinking path (Laguna and Marti, 2003) has been well documented. In the context of neural network training, the application of an improvement method is a time consuming operation, so we will limit it to the best solution found in the path, as is shown in the outline of Figure 1.

4. The Improvement Method

The Simplex Search procedure is a popular and effective method for unconstrained minimization which does not use derivatives. A good description is found in [Avriel, 1976]. It maintains a set of n+1 points, located at the vertices of a n-dimensional simplex. Each major iteration attempts to replace the worst point by a new and better one using *reflection, expansion, and contraction* steps. The reflection step moves from the centroid of all points but the worst, in a direction away from the worst toward the centroid, to a new point x^r. If x^r is better than all others, the expansion step moves farther in this direction. If x^r is worse than all points in the original simplex but the worst, then the contraction step replaces the original simplex by a new one that retains the best point, but with the other vertices some fraction of their original distances from this best point. Some variants can be shown to converge to a local minimum of a smooth function, but the rate of convergence is at most linear, like steepest descent.

The GRG implementation used here (Lasdon and Smith, 1993) applies the BFGS Quasi-Newton procedure to unconstrained problems (Nash and Sofer, 1996). This gradient-based algorithm is known to have a superlinear convergence rate, and has been found empirically to have the best combination of reliability and efficiency among all gradient-based methods for unconstrained minimization of smooth functions. When applied to the problem of minimizing the training error, BFGS will almost always find solutions with lower training error than the Simplex search procedure, and will do so faster. This

is confirmed in the results of the next section. Recall, however, that both Simplex Search and GRG procedures are terminated when the testing error has not improved for *T_limit* consecutive iterations, so they may not converge to a local minimum of the training error.

5. Computational Experiments

For our computational testing, we implemented, in C, the classical Back-Propagation method (BP), the extended tabu search method, ETS, of Sexton et al. (1998), the SS method of Laguna and Martí (2000) and our procedure described in the previous sections (PR). We have also implemented the Simplex search procedure, and have used the LSGRG2 implementation of GRG described in [Smith and Lasdon, 1992).

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 2 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.

It is well known that, for most applications, it is advantageous to apply pre-processing transformations to the input data. It is often useful to process the data with a linear re-scaling to ensure that all inputs are of similar magnitude. Specifically, for each input variable x we calculate its mean $\bar{x}$ and variance σ^2 in the training set. Then, we define the scaled variable

$$x' = \frac{x - \bar{x}}{\sigma},$$

which has zero mean and unit standard deviation. A similar re-scaling is made for the function values (net's output).

The training set consists of 200 observations with data randomly drawn from [-100, 100] for x_1 and [-10,10] for x_2. The testing set consists of 100 observations drawn from the same uniform distributions. Since we use this testing set to stop the methods, we will use an additional set (the validation set) of 100 observations to test the ability of the neural network to predict *f(x)* for *x* values that were not used in the search process at all (neither in the training set or in the testing set).

Back-propagation is one of the first methods for neural network training, and is the most widely used algorithm in practical applications. It is a gradient descent procedure that computes the derivatives' values in a very efficient way (from the output layer back towards the input layer), and modifies the weights according to a parameter known as 'learning rate'. The original algorithm has been modified in many ways; the most popular consists in adding a 'momentum' term (Rumelhart and McClelland, 1986) when the weights are updated. The inclusion of this term leads to significant improvements, although it introduces a second parameter in the algorithm. Jacobs (1988) suggested a different modification called 'delta-bar-delta rule' which introduces a separate learning rate for each weight. It has been shown (Bishop, 1995) that this rule increases the convergence of the method in some cases, but does not work well in practice across different instances due to some stability problems. Several methods have been proposed to compute the learning rate. This is the case of the quickprop (Fahlman,

1988) and the Rprop (Riedmiller and Heinrich, 1993) methods but they share, in general, the limitations associated with first derivative based methods. Our adaptation of the back-propagation algorithm (BP) includes the momentum term and compares favorably with commercial implementations of this method (El-Fallahi, 2002).

1.Sexton 1:	$f(x) = x_1 + x_2$
2. Sexton 2:	$f(x) = x_1 * x_2$
3. Sexton 3:	$f(x) = \frac{x_1}{\lvert x_2 \rvert + 1}$
4. Sexton 4:	$f(x) = x_1^{\,2} - x_2^{\,3}$
5. Sexton 5:	$f(x) = x_1^{\,3} - x_1^{\,2}$
6. Branin:	$f(x) = \left(x_2 - \left(\frac{5}{4\pi^2}\right)x_1^2 + \left(\frac{5}{\pi}\right)x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_i) + 10$
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:	$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
15. Schwefel:	$f(x) = 418.9829n + \sum_{i=1}^{n}\left(-x_i \sin\sqrt{\lvert x_i \rvert}\right)$

Figure 2. Testing Functions.

Martí and El-Fallahi (2003) compared three well known non-linear methods: Simplex, Powell's direction set (Brent, 1973) and Conjugate gradient method (Polak, 1971) as they appear in Press et al. (1992). Their experimentation shows that the Simplex is probably the best of the three in terms of computational efficiency and solution quality (considering both training and testing error).

In our first experiment we compare two local optimizers, Simplex and GRG. We run both procedures from the same random initial point until the testing error does not improve in 20 consecutive iterations. Table 1 reports, for both methods, the average RMSE in the training set (Train Error), in the testing set (Test Error) and in the validation set (Valid. Error) for problems 3, 5, 6 and 12 in Figure 2 above. In this

experiment we also explore the effect of changes in the number of hidden neurons (*m*). Table 1 also reports the average running time in seconds (CPU time) of the methods.

Method	m	Train Error	Test Error	Valid. Error	CPU time
Simplex	6	7.801	3.137	7.158	204.30
	9	3.722	1.408	3.686	160.04
	20	8.842	2.831	6.785	156.77
GRG	6	0.394	0.115	0.125	123.65
	9	0.092	0.074	0.068	149.33
	20	0.233	0.090	0.098	220.87

Table 1. Performance of Improvement methods

As expected, GRG clearly outperforms the Simplex method with respect to the training error, since it is a more powerful optimizer. Considering the testing and the validation errors, GRG is also able to obtain much better results than the Simplex method. The Simplex obtains an average validation error of 3.686 with 9 hidden neurons while GRG obtains 0.068 with similar running times.

This experiment confirms what is well known about the parameters in neural networks; i.e. the performance of the net is highly dependent on the number of hidden neurons. Table 1 shows that the performance of the training methods improves as the net has more hidden neurons up to one point (that we have empirically found for our functions when *m* is equal to 9) and then it begins to deteriorate. Therefore, from now on, we will use a net with 9 hidden neurons and GRG as the improvement method of our path relinking algorithm.

A common search scheme (Sexton et al., 1998 or Laguna and Marti, 2002) consists of applying the training method to the set of weights associated with hidden neurons and then using linear regression to find the weights associated with the output neuron. We have implemented the linear regression combined with the GRG optimizer. If we apply this method to solve the problems in the previous experiment, we obtain a reduction in the errors with a modest increment in the computational time (from 149.3 to 199.1 seconds). Specifically, the average training error decreases from 0.092 to 0.048, the testing error from 0.074 to 0.044 and the validation error from 0.068 to 0.051. Therefore, we will use this combined optimizer in the path relinking method.

In our second experiment we compare three relinking mechanisms in the same four instances considered in the previous experiment: the method based on linear combinations where all pairs in the RefSet are considered (Linear All), the method based on linear combinations where only relinking between the best element in RefSet and other solutions are considered (Linear Best), and the method based on block swapping given by the weights associated with each node (Block Swap). In this experiment we also want to isolate and measure the relative contribution of the relinking method in the algorithm. Therefore, no improvement methods have been used. We have generated a random initial RefSet and we have performed the relinking method for 500 iterations, updating, if necessary, the RefSet. Table 2 shows the average training and testing error for the solutions in the initial RefSet as well as for the solutions in the final RefSet after the application of each of the three methods under consideration.

Method	Train Error	Test Error	CPU time
Initial RefSet	313.52	319.24	----
Linear All	188.36	291.27	277.84
Linear Best	238.85	301.67	202.56
Block Swap	247.54	273.76	200.84

Table 2. Relinking methods

Table 2 shows that all three methods are able to reduce the average error in the RefSet. Specifically, the "Linear All" method yields an average percent reduction from the initial testing error of 39% (from 313.52 to 188.36), while the "Linear Best" and "Block Swap" achieve a 23.8% and 25% respectively. As it was expected, the "Linear Best" method has lower computational time, since it performs less relinking steps than the other two methods. "Linear All" has a better average performance than the other two methods with respect to the training error. However, the "Block Swap" method is able to obtain the best solutions (closely followed by the "Linear All") in terms of the testing error, although it requires more computational effort than the other methods. We will consider the "Block Swap" method as the relinking method in our algorithm given that our objective is to design a method that has good generalization properties.

In the last experiment we are going to compare our path relinking algorithm PR, as it appears in the outline given in Figure 1, with the Back-Propagation method (BP), the extended tabu search method, ETS, of Sexton et al. (1998) and the SS method of Laguna and Martí (2000). The path relinking algorithm uses GRG with linear regression as the improvement method.

Tables 3, 4 and 5 report, respectively, the training, testing and validation error obtained with the four methods in the 15 problems considered. All the methods were run for ten minutes on average. In all the cases, we have considered the same training, testing and validation sets.

Id. Prob.	BP	ETS	SS	PR
1	0.43	0.00	0.00	0.00
2	7.46	6.15	0.09	0.00
3	0.06	0.22	0.08	0.02
4	25.30	45.78	0.51	0.00
5	14.30	37.70	1.17	0.00
6	14.25	39.45	2.37	0.05
7	8.45	43.60	0.51	0.29
8	0.16	0.00	0.00	0.00
9	3.33E+09	1.81E+09	4.04E+07	1.20E+09
10	16.85	17.10	5.38	20.07
11	6.75E+06	3.04E+06	8.82E+05	5.79E+09
12	88.50	113.12	0.63	0.00
13	7.75	9.81	0.03	0.00
14	5.88E+05	3.53E+04	7.55E+01	8.69E+01
15	1.65	65.12	2.30	0.05

Table 3. Training Error across different methods

Id. Prob.	BP	ETS	SS	PR
1	1.25	0.00	0.01	0.00
2	14.50	11.45	73.50	0.00
3	0.65	0.51	4.45	0.02
4	63.45	62.60	54.45	0.00
5	25.80	47.46	0.00	0.00
6	91.10	87.70	727.46	0.05
7	17.60	59.45	35.55	0.28
8	0.33	263.00	0.00	0.00
9	1.10E+10	1.03E+10	6.59E+08	1.06E+09
10	31.50	33.45	1342.25	19.50
11	8.50E+06	2.56E+45	4.24E+07	6.12E+05
12	155.15	175.25	75.15	0.00
13	11.50	11.39	1.01	0.00
14	7.36E+04	50400.00	3.12E+05	8.31E+01
15	1.58	1.52E+46	1475.25	0.05

Table 4. Testing Error across different methods

Tables 3 and 4 show that the best solution quality is obtained by the path relinking method (PR) in most of the cases. Table 3 shows that the PR method is able to obtain the best solutions with respect to the training error in 11 instances, while the SS method obtains 3 best solutions and the ETS only obtains 1. Table 4 shows similar results since PR obtains 14 best solutions with respect to the testing error and SS is able to obtain the other one.

This experiment shows that none of the methods can effectively handle problems 9, 11 and 14 within the run time considered. (We also ran the training methods in these three instances for around half an hour of CPU time with no significant improvement). Therefore, we can say that in practice this neural network is not able to approximate these functions. If we remove these three instances, we found that the SS algorithm is the best with 1.09 average training error, closely followed by PR with a value of 1.7, and ETS and BP obtain 31.5 and 15.4 respectively. However, considering the testing error, the PR clearly outperforms the other methods, since it obtains an average of 1.64 while SS, ETS and BP obtain 316, 1.27E47 and 34.5 (note that the high value of the ETS is given by instance 15, and if we removed it, it would be 68.4). It should be mentioned that the SS method was designed for online training within short computational time, and it does not take advantage of the extra time considered. Moreover, it uses as the improvement method the Simplex algorithm, which was shown in previous experiment to be inferior to GRG. On the other hand, although the BP results are only of medium quality, this method is very simple to implement compared with the other metaheuristics approaches.

The mean squared error values in Table 5, which corresponds to the validation set, are in line with those obtained in the training and testing sets. Specifically, PR is able to obtain 14 out of 15 best solutions, while BP obtains the other one. Regarding average values (removing instances 9, 11 and 14), PR obtains 1.75 while SS, ETS and BP obtain 321.9, 1.26E45 and 17.6 respectively. As in the previous experiment if we removed instance 15, the average value of the ETS method would be reduced to 69.8.

Id. Prob.	BP	ETS	SS	PR
1	0.56	0.02	0.00	0.00
2	13.47	15.65	67.28	0.00
3	0.19	1.03	3.16	0.01
4	27.26	63.76	61.82	0.00
5	9.29	46.04	0.00	0.00
6	16.44	91.45	842.78	0.03
7	7.97	57.03	32.43	0.29
8	0.16	270.25	0.09	0.00
9	1.22E+10	1.03E+10	5.00E+09	1.33E+09
10	13.64	35.05	1330.86	20.64
11	4.20E+06	2.56E+45	1.99E+06	5.44E+05
12	117.02	175.05	72.12	0.00
13	3.62	13.04	0.78	0.00
14	5.96E+04	5.04E+04	6.39E+04	9.06E+01
15	2.01	1.52E+46	1452.36	0.06

Table 5. Validation Error across different methods

Conclusions

In this paper we have described the implementation of path relinking for training a single-layer feed-forward neural network. The proposed method was compared with the well known Back-Propagation algorithm as well as a recently developed scatter search and tabu search procedures. Our experiments show that some few functions cannot be approximated by the neural network considered (with 9 hidden units) trained with these methods. For those that can be approximated with a reasonable accuracy level, the best results are obtained by the PR method coupled with an improvement phase based on the GRG optimizer.

Acknowledgement

The authors want to thank Eric Hartman (Pavilion Technologies) for his useful comments to improve the paper.

References

Avriel, M. (19760, Nonlinear Programming, Analysis and Methods, Prentice-Hall, Englewood Cliffs, New Jersey.

Barron, A.R. (1993) “Universal approximation bounds for superposition of a sigmoidal function”, IEEE trans. On information theory, vol. 39 (3), pp. 930-945.

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

Blum, E.K. and L.K. Li (1991) “Approximation theory and feedforward networks”, Neural Networks, vol. 4(4), pp 511-515.

El-Fallahi A. (2002), Entrenamiento de Redes Neuronales, Trabajo de Investigación, Dpto Estadística e I.O. Universidad de Valencia.

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.

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.

Kolmogorov, A.N. (1957) “On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition”, Doklady Academiia Nauk SSRR, vol. 114 (5), pp. 953-956

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 (2003) “Multilayer neural networks: an experimental evaluation of on-line training methods” Computers and Operations Research, forthcoming.

Martí, R., M. Laguna and F. Glover (2003) “Principles of Scatter Search”, European Journal of Operational Research, forthcoming.

Minsky, M.L. and S.A. Papert (1969) Perceptrons, Cambridge, MA: MIT Press. Expanded edition 1990.

Nelder, J. A. and R. Mead (1965) “A Simplex Method for Function Minimization,” Computer Journal, vol. 7., pp. 308-313.

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).

Riedmiller, M. and B. Heinrich (1993) “A Direct Adaptive Method for Faster Back-propagation Learning: The RPROP Algorithm” Proc. of the IEEE Intl. Conf. on Neural Networks.

Rosemblatt, F. (1962) Principles of Neurodynamics: Perceptrons and theory of Brain Mechanisms, Washington DC: Spartan.

Rumelhart, D.E. and McClelland, J.L. (1986), Parallel distributed processing: explorations in the microstructure of cognition, Cambridge, MA: The MIT Press.

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.

Sexton, R. S., R. E. Dorsey and J. D. Johnson (1999) “Optimization of Neural Networks: A Comparative Analysis of the Genetic Algorithm and Simulated Annealing,” European Journal of Operational Research, vol. 114, pp. 589-601.

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