Multilayer Neural Networks: An Experimental Evaluation of On-Line Training Methods

Rafael Martí and Abdellah El-Fallahi

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

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

Latest version: July 4, 2002

Abstract

Artificial neural networks (ANN) are inspired by the structure of biological neural networks and their ability to integrate knowledge and learning. In ANN training, the objective is to minimize the error over the training set. The most popular method for training these networks is back propagation, a gradient descent technique. Other non-linear optimization methods such as conjugate directions set or conjugate gradient have also been used for this purpose. Recently, metaheuristics such as simulated annealing, genetic algorithms or tabu search have been also adapted to this context.

There are situations in which the necessary training data are being generated in real time and an extensive training is not possible. This "on-line" training arises in the context of optimizing a simulation. This paper presents extensive computational experiments to compare 12 "on-line" training methods over a collection of 45 functions from the literature within a short term horizon. We propose a new method based on the tabu search methodology, which can compete in quality with the best previous approaches.

Scope and Purpose

Artificial neural networks present a new paradigm for decision support that integrates knowledge and learning. They are inspired by biological neural systems where the nodes of the network represent the neurons and the arcs the axons and dendrites. In recent years there has been an increasing interest in ANN since they had proven very effective in different contexts. In this paper we will focus on the prediction/estimation problem for a given function, where the input of the net is given by the values of the function variables and the output is the estimation of the function image. Specifically we will consider the optimization problem that arises when training the net in the context of optimizing simulations (i.e. when the training time is limited). As far as we know, partial studies have been published, where a few training methods are compared over a limited set of instances. In this paper we present extensive computational experimentation of twelve different optimization methods over a set of 45 well-known functions.

1. Introduction

Neural networks have been widely used for both classification and prediction. This paper is focused on the prediction problem in which an unknown function is approximated. The input of the net is given by the values of the function variables and the output is the estimation of the function image. In mathematical terms, given a real function $f:\Re^n \rightarrow \Re$ and a neural net *NN*, the objective is to find appropriate values for the arc weights *w* of the net, such as its output *NN(x,w)* from an input vector *x*, approximates the value *f(x)*.

When training the net, the problem is to find the weights that optimize its performance (i.e. that allows the ANN to most accurately fit the data). The most common error measure to report the quality of the network performance is the Root Mean Squared Error (RMSE). Let $E=\{x^1, x^2,..x^T\}$ be a random sample of points in the domain of *f*, and suppose that the value of *f(x)* is known for all *x* in *E*. Given the values *w* for the net weights, for each *x* in *E* the error is computed as $error(x,w) = [f(x)- NN(x,w)]^2$, and the RMSE across all the elements in the training set *E* is given by the expression:

$$Error(E,w) = \sqrt{\frac{\sum_{i=1}^{T} error(x^i, w)}{T}} .$$

Therefore, for a fixed set E, training the neural network can be formulated as minimizing this expression over the set of *w*-values.

Neural network training is typically considered an offline activity where the training procedures may be run for hours of CPU time in order to achieve the best possible results. There are, however, situations where an extensive training is not possible. This situation is one in which the necessary training data are being generated in real time. That is, the data do not reside in large databases from which training sets can be constructed. Instead, the same process that will make use of the neural network output generates the data. We refer to this situation as on-line training of the neural network. One important application of on-line training arises in the context of optimizing a simulation (Glover and Kelly, 1999).

The basic idea of simulation is to model a physical process on the computer, incorporating the uncertainties that are inherent in all real systems. The model is then executed to simulate the effects of the physical process and to determine their consequences. Since simulations are generally computationally expensive, the optimization process would be able to search the solution space more extensively if it were able to quickly eliminate from consideration low-quality solutions, where quality is based on the performance measure being optimized. This is where a neural network becomes useful. Specifically, a neural network can be used to filter out solutions that are likely to perform poorly when the simulation is executed. Some optimization software packages are based on this filter strategy to perform the optimization of time-consuming evaluation functions (Voss and Woodruff, 2002). In these contexts, the training is limited to one or two minutes of CPU time.

This paper is focused on the neural network on-line training problem. Twelve different methods for solving this unconstrained non-linear problem are computationally compared. We also propose a new procedure based on the tabu search methodology that is able to obtain high quality results and a filtered multi-start framework. As far as we know, most of the previous methods have been only partially compared using a few instances. For example, Denton and Hung (1996) compared four non-linear methods over six classification problems. Sexton (1998) proposed a tabu search based training method and studied its performance over six prediction problems, and Laguna and Martí (2000) proposed a Scatter Search algorithm and compared it with Sexton's method over the same six problems. We have considered a collection of 45 well known functions from the literature and compare the different methods, included our tabu search approach, when training a neural network within a short term horizon. The following non-linear and metaheuristic methods have been considered:

1. Back-propagation
2. Non-linear Simplex method (Nelder and Mead, 1965)
3. Direction set (Powell's) method (Brent, 1973)
4. Conjugate gradient method (Polak, 1971)
5. Simple tabu search (Sexton, 1998)
6. Extended tabu search (Sexton, 1998)
7. Scatter Search (Laguna and Martí, 2000)

Additionally, we implement some of these methods within a Multi-Start procedure. Particularly, we test the performance of:

8. Multi-start back-propagation
9. Multi-start non-linear Simplex
10. Multi-start direction set
11. Multi-start conjugate gradient
12. A new tabu search approach

Both tabu search methods by Sexton (5 and 6) are mainly based on random selections and extensive sampling of the solution space. In this paper we propose a new tabu search method (12) based on context and search information as it is provided by the function's gradient and memory structures.

In the remainder of the paper, we first introduce in Section 2 notation and the network architecture and then describe in Section 3 the previous methods mentioned above. Section 4 is devoted to the multi-start framework and our new tabu search method. Our implementation, as described subsequently, proposes innovative mechanisms for move selection. The computational experiments are reported in Section 5 and the paper ends with the associated conclusions.

2. Neural Network Architecture

We have considered the most employed architecture for prediction and classification: a multilayer feed-forward network with a single hidden layer. A schematic representation of the network appears in Figure 1.

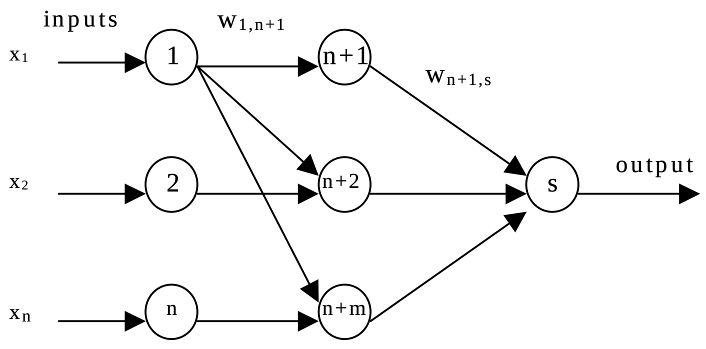

Figure 1. Neural network diagram

Let *NN=(N, A)* be an ANN where *N* is the set of nodes 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 an output neuron (we restrict our attention to real functions $f: \Re^n \rightarrow \Re$, then they only have one single value as the function's image). Figure 1 shows a net where $N_I = \{1, 2, ..., n\}$ and $N_H = \{n+1, n+2,..., n+m\}$.

Given an input pattern $x=(x_1,...,x_n)$, the neural network provides the user with an associated output *NN(x,w)*, which is a function of the weights *w*. Each node *i* in the input layer receives an amount of signal x_i that it sends through all its incident arcs to the nodes in the hidden layer. Each node *j* in the hidden layer receives a signal *input(j)* according to the expression:

$$Input(j) = w_j + \sum_{i=1}^{n} x_i w_{ij}$$

where w_j is the bias value for node *j*, and w_{ij} is the weight value on the arc from node *i* in the input layer to node *j* in the hidden layer. Each hidden node transforms its input by means of a nonlinear activation function: *output(j)=sig(input(j))*. The most popular choice for the activation function is the sigmoid function $sig(x)= 1/(1+e^{-x})$. Laguna and Martí (2000) test two activation functions for the hidden neurons and conclude that the sigmoid presents superior performance. Each hidden node *j* sends the amount of signal *output(j)* thorough the arc *(j,s)*. The node *s* in the output layer receives the weighted sum of the values coming from the hidden nodes. This sum, *NN(x,w),* is the net's output according to the expression:

$$NN(x,w) = w_s + \sum_{j=1}^{m} output(n+j)\, w_{n+j,s}$$

In the process of training the net (supervised learning), the problem is to find the values of the weights (including the bias factors) that minimizes the error (RMSE) across the training set *E*. Once the optimization has been performed and the weights have been set ($w=w^*$), the net is ready to produce the *output* for any *input* value. The testing error (TE) computes the Root Mean Squared Error across the elements in the testing set $TS= \{y^1, y^2,..,y^S\}$ where no one belongs to the training set *E*.

$$TE = \sqrt{\frac{\sum_{i=1}^{S} error(y^i, w^*)}{S}}\ .$$

3. Previous Approaches

3.1 Back Propagation

This was the first method for neural network training, and it is still the most widely used algorithm in practical applications. It is a gradient descent method that searches for the global optimum of the network weights. Each iteration *t*, consists of two steps. First, partial derivatives $\partial Error/\partial w$ are computed for each weight in the net. Given that the network is acyclic, this can be exactly done by starting the computation at the output node *s*, and then successively computing the derivatives for the arcs from N_H to *s*, and finally for the arcs between N_I and N_H (i.e.: the gradient information is successively moved from the output layer back towards the input layer). In the second step, weights are modified according to the expression:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \frac{\partial Error(E, w(t))}{\partial w_{ij}}$$

where the step α is the learning rate. Then the error is computed for the new weights, *t* is increased in one unit and a new iteration begins. The most significant modification to the method is the addition of a momentum term β. Each new search direction is computed as a weighted sum of the current gradient and the previous search direction:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \frac{\partial Error(E, w(t))}{\partial w_{ij}} + \beta(w_{ij}(t) - w_{ij}(t-1))$$

The values of both parameters α and β are adjusted in each particular application. For instance, Riedmiller (1994) states that sometimes better results can be achieved using no momentum term. We will test some values for these parameters in our computational experiments. For a detailed description of the method we refer the reader to Masters (1993) or El-Fallahi (2002).

3.2 Non-Linear Methods

Since the neural network training problem can be expressed as a nonlinear unconstrained optimization problem, we might use more sophisticated nonlinear methods than the gradient descent to solve it. We consider three well-known methods which, as it is stated in Press et al. (1997), constitute a selection of the best established algorithms in unconstrained non-linear optimization. They are:

- Non-linear Simplex method (Nelder and Mead, 1965)
- Direction set (Powell's) method (Brent, 1973)
- Conjugate gradient method (Polak, 1971)

We follow the implementation of the three methods given in Press et al. (1997).

3.3 Tabu Search

The simple Tabu Search method by Sexton et al. (1998) is mainly based on a random search. Neighborhoods are randomly drawn points from uniform distribution. The neighborhood is a region restricted to ±0.1% for each weight in the current solution. 500 solutions are randomly generated in this region and the best one is chosen.

An initial solution is randomly generated in the range [-10,10] and its neighborhood is examined. The best solution in the neighborhood replaces the initial one and the process is repeated. A tabu list is generated by adding the last solution to the beginning of the list and discarding the oldest solution from it. To be rejected, all the weights of a new solution would need to be within ±0.01% for any of the solutions in the tabu list. The size of this list is evaluated at three levels (1,50, 100). This algorithm is used as a baseline for comparison with the extended TS method. In both methods, and in the Scatter Search described in 3.4, the search takes place only over the weights from the input to the hidden layer and the bias factor of the hidden neurons. Weights from the hidden layer to the output neuron, $w_{n+j,s}$ as well as the bias factor of node s, w_s, are obtained by linear regression.

The extended TS algorithm (Sexton et al., 1998) follows. An initial solution x_0 is randomly drawn from a uniform distribution in the range [-10,10] and the current best solution x_{best} is initialised to x_0. Solutions are randomly generated in this range for a given number of iterations. When generating a new point x_{new}, aspiration level and tabu conditions are checked. If $f(x_{new})<f(x_{best})$ then the point is automatically accepted and both x_{best} and $f(x_{best})$ are updated; otherwise the tabu conditions are

tested. If there is one solution x_i in the tabu list (TL) such as $f(x_{new}) \in [f(x_i)-0.01*f(x_i), f(x_i)+0.01*f(x_i)]$, then the complete test is applied to x_{new} and x_i; otherwise the point is accepted. The test checks if all the weights in x_{new} are within ±0.01 from x_i, in this case the point is rejected, otherwise the point is accepted and x_{new} and $f(x_{new})$ are entered into TL. This process continues for 1000 iterations of accepted solutions. Then, another cycle of 1000 iterations of random sampling begins. These cycles will continuously repeat while $f(x_{best})$ improves.

When the random sampling ends, the process of intensification starts by performing a search from the best solution found x_{best}. The new points are drawn by modifying the x_{best} by a small *step* value, where:

$$step=((0.1*x_{best})-(0.2*x_{best})*random)/change.$$

Each cycle of the intensification phase generates 1000 new points. This phase makes a maximum of 20 cycles as long as there is at least one reduction in the $f(x_{best})$. Once this phase finishes, the diversification process begins in order to expand the search area. The step value is now computed as:

$$step=((0.1*x_{best})-(0.2*x_{best})*random)*change$$

This diversification phase generates new points by modifying x_{best} with this step value. As in the intensification phase, cycles of 1000 iterations are performed up to a maximum of 20. Both phases, intensification and diversification, are alternated for a maximum of 5 consecutive iterations. Then, the whole process is repeated for 10 global iterations. The *random* variable is a random number drawn from a uniform distribution in the range [0, 1], the *change* variable is initialised to one, and is increased in one after each intensification phase.

3.4 Scatter Search

In this section we describe the adaptation of Laguna and Martí (2000) of the Scatter Search methodology (Glover, Laguna and Martí, 1999) to the net training problem.

After the data normalization, an initial reference set (*RefSet*) of *b* solutions is created. A set *P* of *PSize* solutions *w* (bounded between *wlow* and *whigh*) is built with the diversification method, based on a controlled randomization scheme, given in Glover, Laguna and Martí (1999). The *RefSet* is filled with the best *b*/2 solutions in *P* to which the improvement method is applied. The *RefSet* is completed with *b*/2 more solutions generated as perturbations of the first *b*/2. The perturbation consists of multiplying each weight by 1 + U[-0.05,0.05], where U is the uniform distribution.

In step 2, the solutions in *RefSet* are ordered according to quality, where the best solution is the first one in the list. Then, the *NewPairs* set is constructed. *NewPairs* consists of all the new pairs of solutions that can be obtained from *RefSet*, where a "new pair" contains at least one new solution. The pairs in *NewPairs* are selected one at a time in lexicographical order to create linear combinations. The best *b* solutions created as linear combinations are subjected to the improvement method. Each improved solution is then tested for admission into *RefSet*. 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 3 the procedure intensifies the search around the best-known solution. At each intensification iteration, the best-known solution is perturbed and the improvement method is applied. The best solution is updated if the perturbation plus improvement generates a better solution. After *IntLimit* intensification iterations without improving the best solution, the procedure abandons the intensification phase and returns to step 2. Previously, the improvement method is applied to the best $b/2$ solutions in *RefSet* and the worst $b/2$ solutions are replaced with perturbations of the best $b/2$. The training procedure stops when the number of objective function evaluations reaches the total allowed. After experimentation, the parameters *wlow, whigh, b* and *IntLimit* were set to –2, 2, 10 and 20 respectively.

4. New Approaches

4.1 Multi Start Methods

Multi-start methods have two phases: the first one in which the solution is generated and the second one in which the solution is typically (but not necessarily) improved. Then, each global iteration produces a solution (usually a local optimum) and the best overall is the algorithm's output.

Figure 1 shows a pseudo-code of the multi-start procedure. A solution x_i is constructed in Step 1 at iteration i. This is typically performed with a constructive algorithm. Step 2 is devoted to improving this solution, obtaining solution x_i'. A simple improvement method can be applied. However, this second phase has recently become more elaborate and, in some cases, is performed with a complex method that may or may not improve the initial solution x_i (in this latter case we set $x_i'=x_i$).

```
Initialise $i=1$
while(Stopping condition is not satisfied)
{
        Step 1. (Generation)
            Construct solution $x_i$
        Step 2. (Search)
            Apply a search method to improve $x_i$
            Let $x_i'$ be the solution obtained
        if($x_i'$ improves the best )
            Update the best
        $i=i+1$
}
```

Figure 1. Multi-start procedure

In recent years, many heuristic algorithms had been proposed to solve some combinatorial optimization problems following the outline given in Figure 1. A basic multi-start method generates uniformly distributed points in the solution space, and starts the search procedure from each of these points. This is well known to converge to a global solution with probability one as the number of points approaches infinity. This algorithm is very inefficient because the same local solution is reached many times, and some of these can be of a very low quality. We consider a filter method based on the proposed in Ugray et al. (2001), to selectively apply the Step 2. Solutions are randomly generated in Step 1, and then two filters are considered, if the solution is accepted by both filters, then the search method is applied; otherwise it is discarded and a new solution is generated.

The distance filter helps ensure that the starting points x_i are diverse, in the sense that they are not too close to any previously found local optimum. Its goal is to prevent the step 2 search method from starting more than once within the basin of attraction of any local optimum. When a local solution x_i' is found, it is stored with the euclidean distance between it and the starting point x_i that led to it. Let *maxdist*, be the maximum of these distances. For each trial point x_i, if the distance between x_i, and any local optimum already found is less than *distfactor*maxdist,* the solution is rejected. Otherwise, the quality filter is applied.

The quality filter helps ensure that the starting points have a relative high quality, by not starting from candidate points whose value is greater than a threshold *QTH*.

This threshold is set initially to the value of the first solution generated. If trial points are rejected by this test for more than *maxiter* consecutive iterations, the threshold is increased by the updating rule $QTH=QTH+\delta(1+|QTH|)$. On the other hand, when a trial point is accepted by this filter, QTH is decreased by setting it to the value of that point. Both filters were proposed by Ugray et al. (2001) in the context of multi start non-linear optimization. We have implemented them with the parameter values suggested by the authors: $\delta=0.2$, *maxdist=0.75* and *maxr=20*.

4.2 Tabu Search

The solution approach that we have developed for training the neural network is based on the tabu search methodology. Our method consists of three phases: MultiRSimplex, TSProb and TSFreq. After the initialization with the *MultiRSimplex* phase, the procedure performs iterations in a loop consisting in alternating both phases, TSProb and TSFreq, to intensify and diversify the search respectively. After each phase, the Simplex method is applied to the best solution found. The procedure terminates when a pre-specified number of iterations (or function evaluations) is met.

MultiRSimplex

The initialization phase, MultiRSimplex, generates pseudo-random trial points and starts the Simplex optimizer (Nedler and Mead, 1965) with a subset of the points determined by the two filters. Weights from the input to the hidden layer and the bias factor of the hidden neurons are randomly generated from a uniform distribution in the range of [-0.5, 0.5]. Weights from the hidden layer to the output neuron, $w_{n+j,s}$ and the bias factor of node s, w_s, are obtained with linear regression. Therefore, although this solutions are partially randomly generated, they present a relatively good value. The linear regression is only used in this phase; thus, in the other two phases, the search takes place over the entire set of weights.

This phase generates solutions and applies the merit filter as well as the quality filter to accept or reject them. Once 50 solutions have been accepted, the Simplex method is applied to the best 5 of them. The best final solution is labeled as w_{best}.

TSProb

An iteration in the TSProb phase begins by randomly selecting a weight. 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.

Factor β scales the change in the selected weight according to the status of the search. The algorithm starts with $\beta = 1$ and reduces the magnitude of this change as long as the current solution is close to a local optimum. Specifically, we define $S\nabla(w)$ as the sum of absolute values of partial derivatives in solution w. In mathematical terms, the expression is:

$$S\nabla(w) = \sum_k \left| \frac{\partial Error(E,w)}{\partial w_k} \right| .$$

Let $S\nabla(w^0)$ be the value for the initial solution w^0 of this phase. At iteration t, the value $S\nabla(w^t)$ is computed, and β is randomly selected in the range:

$$\left[\frac{S\nabla(w^t)}{S\nabla(w^0)} - 0.1, \frac{S\nabla(w^t)}{S\nabla(w^0)} + 0.1 \right] .$$

Starting from the w_{best} solution, the *TSProb* phase stops when *NonImp* moves are performed without improving the best solution found.

TSFreq

The number of times that weight w_i has been chosen to be moved is accumulated in the value freq(i). This frequency information is used for diversification purposes. In each iteration of the *TSFreq* phase, a weight is randomly selected, where the probability of selecting weight w_i is inversely proportional to the frequency count *freq(i)*. This phase carries out the same steps than *TSProb* to perform a move for a selected weight. Starting from the last solution generated by *TSProb*, the *TSFreq* phase stops when *NonImp* moves are performed without improving the best solution found.

The calculations of the values of the partial derivatives in the current solution are performed with the well known expressions used in the back-propagation method. These values, however, are not updated after the execution of a move, because it is a computationally expensive calculation. Note that we are only using them as a way to discriminate among weights and we consider that it is not absolutely necessary to update them after each move because most of these values either remain the same or their relative merit remains almost unchanged. The notion of not updating key values after every iteration is based on the *elite candidate list* suggested in Glover and Laguna (1997). The application of this strategy is particularly useful when the updating of the move values is computationally expensive, as in our context. The partial derivatives are updated each *DeriveUpdate* iterations.

5. Computational Experiments

The procedures described in the previous sections were implemented in C, and all the experiments were performed on a Pentium III-750 personal computer. Table 8 in the Appendix shows summary information of the 45 test problems considered in our computational testing that are based on a set of nonlinear objective functions, which can be found in the following web pages:

http://www.maths.adelaide.edu.au/Applied/llazausk/alife/realfopt.htm
http://solon.cma.univie.ac.at/~neum/glopt/my_problems.html
http://www-math.cudenver.edu/~rvan/phd/node32.html

As it was done in previous works (Sexton et al., 1998; Laguna and Martí, 2000), the training set consists of 50 randomly drawn observations and the testing set of 150, the training is limited to $4*10^6$ function evaluations, and the number of hidden nodes is set to 6. Limiting the number of hidden nodes to a relative low value is necessary in “on-line” training since it is performed within a short period of time. A model with more hidden nodes or even with two hidden layers could provide better results, but would significantly increase the number of variables where the search takes place, thus increasing the training time. Since we are focusing on the online training, we need a suitable model to be trained in one or two minutes of computer time. All the procedures considered for training, start with the input and output data normalization as it is described in Laguna and Martí (2000) and in *www.neuralware.com*. We have computationally found that the effectiveness of a training method increases if the data are previously normalized.

In our first experiment we explore the effect of changes in the two search parameters, α and β, of the back-propagation algorithm (*BP(α,β)*). The instances had been divided in four groups according to the number of input variables (2, 3-9, 10-20 and 21-30). Table 1 reports for α=0.3 and β=0, the average RMSE in the training set for the instances in each group (*Avg. Train Error*), the minimum and maximum of those values (*Min Train Error* and *Max Train Error*), the average running time in seconds (*Avg. CPU*) and the average RMSE in the testing (*Avg. Test Error*), respectively.

	Train Error				Test Error
N. Vars.	Avg.	Min.	Max.	Avg. CPU	Avg.
2	1.96E+08	1.50E-01	3.33E+09	49.1	$8.15*10^8$
3-9	1.07E+09	1.05E-01	1.26E+10	70.2	$1.84*10^9$
10-20	5.07E+08	1.62E-01	5.50E+09	163.3	$2.18*10^8$
21-30	7.88E+05	3.23E-01	2.79E+06	283.1	$7.48*10^5$
summary	4.83E+08	1.05E-01	1.26E+10	109.02	$8.53*10^8$

Table 1. BP(α=0.3, β=0)

Table 1 shows that in some instances the training error is very high, while in others it is relatively low. For example, in graphs with two variables (first row in the table) the maximum train error obtained in one instance is $3.33*10^9$ while the minimum is 0.15. Considering the 45 instances, 26 of them present a training error inferior to 50 units (they appear with a "*" symbol in the Appendix table), and the other 19 present high error values. We have trained the neural net to approximate these 19 instances with all the methods described in this paper, and none of them is able to approximate them properly (i.e. to obtain a reduced train error). Even if we run the methods longer, the error is not significantly reduced. Therefore, we can say that these functions cannot be approximated with online training with this net architecture. From now on, we will report the results of our experiments in the reduced set of 26 mentioned instances, that we will refer as set F. Nevertheless, this section ends with a summary table where the best solution found in each of the 45 instances is shown.

Table 2 shows the results of the back-propagation method with α=0.3 and β=0 over the set F. These values show that the number of variables is not a main influential factor in the "difficulty" of the prediction. On the other hand, it is shown that as the number of variables increases, the running times also increases.

	Train Error				Test Error
N. Vars.	Avg.	Min.	Max.	Avg. CPU	Avg.
2	32.73	0.15	96.21	49.00	67.81
3-9	10.22	0.11	46.61	71.75	24.52
10-20	41.84	0.16	90.73	157.80	94.25
21-30	32.47	0.32	64.62	303.00	55.48
summary	27.54	0.11	142.51	96.46	58.63

Table 2. Back-Propagation (α=0.3, β=0) over F

Table 3 shows different combinations of the search parameters. First, we compare three values of α in the BP version with no momentum term (β=0). Specifically we consider the values 0.3, 0.5 and 0.7. The best results are obtained with α=0.3, which presents an average train error of 27.54. We then fix α=0.3 and compare three versions of the BP method with momentum: β = 0.5, 0.7 and 1. As expected, the addition of the momentum term significantly reduces the error. The best results are obtained with α=0.3 and β =1, which presents an average train error of 9.92 obtained in 89.35 seconds. It should be mentioned that the testing errors are of a greater magnitude than the training ones.

		Train Error				Test Error
α	β	Avg.	Min.	Max.	Avg. CPU	Avg.
0.3	0	27.54	0.11	142.51	96.46	58.63
0.5	0	32.59	0.11	153.72	96.69	59.58
0.7	0	35.55	0.15	182.09	96.69	63.07
0.3	0.5	14.23	0.01	118.99	89.77	56.81
0.3	0.7	14.72	0.02	118.99	89.77	56.43
0.3	1.0	9.92	0.02	35.72	89.35	47.09

Table 3. Back-Propagation over F

The box and whiskers plot in Figure 2 shows the RMSE of the training set for the 26 instances of the set F obtained with *BP(0.3,1)*. It is shown in the diagram that a 50% of them present errors that are lower than 5 units.

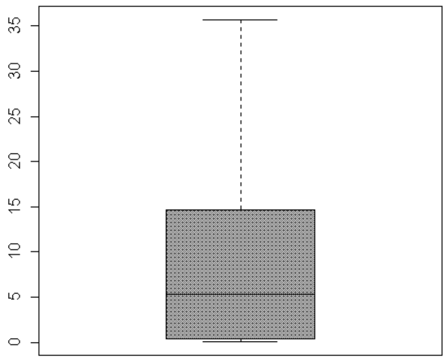

Figure 2. Train Error with *BP(0.3,1)* over F

To finish our first experiment, we consider the multi-start procedure in which solutions are randomly generated in step 1 and improved with the *BP(0.3,1)* method in step 2. We have fixed the same number of evaluations for this Multi-BP method as in the previous experiments. We do not produce tables for this experiment; however, we report that the BP procedure outperforms this multi-start-BP method within this limit of iterations. On the other hand, Figure 3 depicts the average training error over F, and it shows that Multi-BP clearly dominates the simple BP approach as the procedures are allowed to run longer. Anyway, since we are interested in the training problem within a limited run time, we will focus on the “stand-alone” BP method.

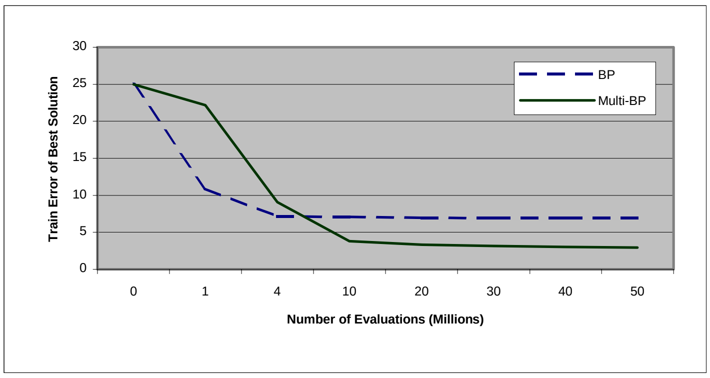

Figure 3. BP and Multi-BP average best solution over time

In our second experiment, we have considered the three non-linear methods previously introduced: the non-linear Simplex method by Nelder and Mead (Simplex), the direction set method by Powell (Powell) and the Conjugate gradient method by Polak and Ribiere (PR). We have also considered the Multi-Start procedure where solutions are randomly generated in step 1 and improved with each of these methods in the step 2. Table 4 reports the results for the six methods.

	Train Error				Test Error
Methods	Avg.	Min.	Max.	Avg. CPU	Avg.
Simplex	61.73	0.00	494.60	55.68	96.42
Powell	35.21	0.05	249.87	53.40	$2.04*10^{6}$
PR	122.62	0.29	494.60	66.52	142.16
Multi-Simplex	79.52	0.00	494.60	106.20	108.42
Multi-Powell	28.94	0.02	184.77	96.72	$6.5*10^{11}$
Multi-PR	76.03	0.14	580.72	106.40	88.92

Table 4. Non-linear methods over F

Table 4 shows that the best quality solution with respect to the train error is obtained by the Multi-Start procedure with the Powell improvement method (Multi-Powell), considering its average value of 28.94. However, this algorithm presents the worst quality solution with respect to the testing error, with an average value of $6.5*10^{11}$. The Back-Propagation outperforms this six methods on average, since it obtains 9.92 and 47.09 for the training and testing errors respectively. The Simplex method is probably the best non-linear method, considering its average train error of 61.73 and its average testing error of 96.42. Although the performance of the Multi-Simplex method is inferior to the Simplex, Figure 4 shows that the former approach dominates the latter as the procedures are allowed to run longer (out of the scope of an online training).

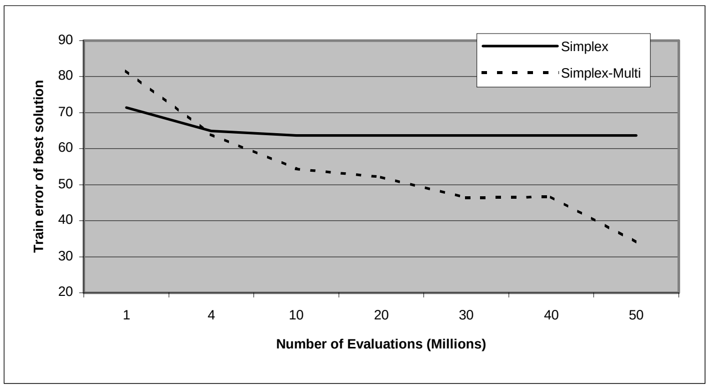

Figure 3. Simplex and Multi-Simplex average best solution over time

In our last experiment we compare the metaheuristic methods for the training problem. Table 5 reports the results on the set F for the basic and extended TS methods by Sexton et al. (B-TS and E-TS), the SS method by Laguna and Martí (SS) and the proposed TS method (TS).

	Train Error				Test Error
Methods	Avg.	Min.	Max.	Avg. CPU	Avg.
B-TS	142.67	0.00	1528.38	19.64	180.03
E-TS	94.56	0.00	1173.16	287.96	124.27
SS	7.62	0.002	65.39	278.03	57.66
TS	8.65	0.00	47.71	254.00	55.30

Table 5. MetaHeuristics over F

The basic TS procedure is clearly inferior in terms of solution quality, although the simplicity and low CPU time of the approach remains appealing. SS and TS variants, on the other hand, present a similar performance since SS has an average training error of 7.62 and an average testing error of 57.66, while TS has 8.65 and 55.30 respectively. The performance of the E-TS method is inferior compared with these two methods, given its training average value of 94.56 and its testing average error of 124.27.

Tables 6 and 7 report, respectively, the average training error and average testing error of the best methods considered. He have considered two versions of our proposed TS method; the first one (TS0) does not incorporate the TSFreq phase, thus the iterations are performed with the TSProb phase alone. The second one (TS) consists of the whole procedure as it is described in the previous section. All the methods are run for two minutes of computer time. The last column in both tables reports, for each instance, the name of the method which is able to obtain the best result.

Table 6 shows that the SS method obtains the best results in terms of the training error in 24 instances, while our both TS variants together are able to obtain the minimum training error in 18 out of 45 instances. The other methods obtain the best solution in none or 1 instance. If we restrict our attention to the 26 instances in the set F (those with an "*" symbol in the identification number), the SS obtains the best results in 12 of them, while TS and TS0 together obtain 13 best solutions.

Num.	BP(0.3,1)	Simplex	Powell	E-TS	SS	TS0	TS	Best
1*	0.528	0.019	2.592	0.004	0.000	0.000	0.000	TS0
2*	7.689	9.385	19.436	4.801	0.012	0.013	0.358	SS
3*	0.259	0.060	1.155	0.201	0.141	0.002	0.001	TS0
4*	31.260	142.534	114.478	42.538	0.378	6.951	9.520	SS
5*	14.310	172.771	154.903	22.654	3.245	0.010	0.045	TS
6*	14.693	156.042	57.265	33.099	237.016	1248.859	47.664	BP
7*	9.679	34.446	44.018	49.008	0.206	493.735	47.711	SS
8*	0.265	0.001	0.067	0.001	0.000	0.000	0.000	TS0
9	3.3E+09	3.0E+09	1.9E+09	3.9E+09	3.87E+15	2.2E+17	1.6E+17	Powell
10*	17.627	17.338	19.450	14.974	20.646	11.500	9.024	TS0
11	6.7E+06	6.3E+06	5.2E+06	4.2E+06	3.29E+08	1.9E+12	1.3E+12	E-TS
12	88.538	200.507	118.385	134.525	0.023	3140.629	85.836	SS
13*	7.919	12.980	16.409	9.639	0.001	239.415	1.759	SS
14	5.9E+04	5.7E+04	2.9E+04	2.4E+04	1.22E+07	0.0E+00	8.0E+05	TS
15*	1.953	1.576	1.538	77.918	16.936	0.000	0.000	TS0
16	2.6E+05	2.4E+05	1.0E+05	9.3E+04	19586.970	1.1E+05	2.1E+04	SS
17	3.2E+03	3.5E+03	2.3E+03	2.9E+03	737.096	3.2E+06	1.7E+05	SS
18*	14.435	32.545	41.935	42.928	0.603	0.000	1.111	TS
19*	0.032	0.000	0.183	0.000	0.000	0.000	0.000	TS0
20	3.7E+05	2.9E+05	2.2E+05	2.7E+05	3.57E+04	7.0E+04	5.3E+04	SS
21*	0.375	0.021	0.054	0.037	0.010	0.003	0.002	TS0
22*	0.377	0.023	0.084	0.044	0.007	0.006	0.003	TS0
23*	0.371	0.050	0.090	0.059	0.013	0.010	0.006	TS0
24	1.1E+08	7.9E+07	6.6E+07	8.1E+07	4.33E+07	3.3E+07	1.8E+07	TS0
25	1.3E+10	8.2E+09	6.5E+09	1.1E+10	6.28E+09	3.7E+09	3.0E+09	TS0
26	8.2E+07	5.3E+07	4.5E+07	6.9E+07	2.25E+07	2.1E+07	2.1E+07	TS0
27*	0.022	0.000	0.081	0.000	0.000	0.000	0.000	TS0
28*	3.109	494.600	3.535	1173.156	1.226	351.626	1.492	SS
29*	34.149	60.834	66.483	107.905	6.894	20.682	12.767	SS
30*	35.723	98.534	120.344	157.439	0.473	17.711	9.489	SS
31*	26.946	86.749	97.180	186.258	0.180	27.236	15.355	SS
32*	0.152	0.016	0.149	0.483	0.001	0.132	0.008	SS
33	318.467	628.155	407.891	956.793	1.231	146.864	36.699	SS
34	1.0E+06	6.3E+05	4.2E+05	5.8E+05	487.743	5.9E+04	4.8E+04	SS
35	7.6E+07	6.9E+07	5.7E+07	8.1E+07	9.29E+04	9.8E+05	2.4E+06	SS
36*	27.874	183.198	84.532	401.886	0.551	72.723	16.844	SS
37*	0.228	0.026	0.210	0.567	0.001	0.149	0.006	SS
38	435.836	1745.650	972.866	3708.566	8.429	435.388	350.416	SS
39	2.2E+06	6.5E+05	6.2E+05	1.0E+06	4.28E+03	1.4E+05	1.6E+05	SS
40	5.5E+09	4.7E+09	4.5E+09	1.8E+09	9.82E+07	2.1E+07	1.4E+08	TS
41	1.1E+06	8.6E+05	4.4E+05	6.7E+05	5.79E+03	1.1E+05	1.4E+05	SS
42	2.8E+06	1.8E+06	1.5E+06	1.3E+06	3.15E+03	2.2E+05	1.4E+05	SS
43*	7.499	100.762	93.065	188.510	0.742	30.691	19.760	SS
44	100.108	398.721	129.856	510.660	1.002	80.716	53.433	SS
45*	0.360	0.440	1.186	7.227	3.200	1.822	0.136	TS0

Table 6. Average Training Error with Bests methods

Num.	BP(0.3,1)	Simplex	Powell	E-TS	SS	TS0	TS	Best
1*	1.609	0.336	3.731	0.006	0.000	0.000	0.000	TS
2*	14.547	21.735	13.227	7.527	0.026	0.451	0.011	TS
3*	0.658	0.172	1.67E+05	0.912	1.465	1.128	1.227	Simplex
4*	63.936	252.485	247.323	53.905	0.862	34.752	9.435	SS
5*	25.754	201.740	275.055	29.286	8.405	2.169	0.060	TS
6*	91.133	246.418	209.300	44.453	557.301	222.784	1332.833	E-TS
7*	17.647	55.400	74.703	55.218	0.371	57.201	247.324	SS
8*	0.325	0.066	0.526	0.099	0.000	0.023	0.056	SS
9	1.10E+10	1.34E+10	1.21E+10	1.12E+10	1.71E+18	1.57E+17	5.31E+17	BP
10*	31.487	35.778	33.609	23.929	39.528	29.470	30.924	E-TS
11	8.50E+06	1.09E+07	1.01E+07	3.88E+07	1.03E+09	1.09E+12	1.35E+12	BP
12	155.416	334.625	255.698	153.811	1.401	86.021	8218.651	SS
13*	11.489	22.445	13.406	9.709	0.001	173.412	278.955	SS
14	7.36E+04	6.50E+04	6.00E+04	5.72E+04	4.02E+07	1.17E+06	1.33E+06	E-TS
15*	2.580	86.724	164.898	2.437	35.424	1.32E+15	5.20E+06	E-TS
16	2.87E+05	2.54E+05	1.92E+05	1.34E+05	6.03E+04	4.66E+04	1.44E+05	TS0
17	6.77E+03	7.92E+03	5.86E+03	4.50E+03	1.62E+03	5.06E+06	5.22E+06	SS
18*	37.826	59.467	51.426	57.458	1.334	1.656	59.441	SS
19*	0.093	0.000	5.07E+07	0.000	0.000	0.000	0.024	TS0
20	4.76E+05	3.46E+05	3.30E+05	4.07E+05	3.41E+05	2.54E+05	1.52E+06	TS0
21*	0.521	0.082	1820.838	0.071	0.031	0.025	0.029	TS0
22*	0.523	0.116	1100.655	0.090	0.044	0.064	6.758	SS
23*	0.539	0.100	2.488	0.081	0.055	0.130	0.794	SS
24	1.73E+08	1.39E+08	1.11E+08	1.52E+08	1.42E+08	1.93E+08	1.47E+08	Powell
25	2.16E+10	1.74E+10	1.96E+10	1.59E+10	1.67E+10	2.06E+10	2.30E+10	E-TS
26	1.37E+08	1.13E+08	9.92E+07	1.14E+08	1.13E+08	1.37E+08	1.46E+08	Powell
27*	0.049	0.000	0.250	0.000	0.000	0.000	0.000	TS
28*	25.105	662.425	760.161	1538.863	11.631	5.823	815.709	TS1
29*	144.208	138.474	124.541	115.172	108.273	122.305	171.038	SS
30*	230.148	181.694	11081.671	222.450	416.352	287.447	285.407	Simplex
31*	178.991	118.810	114.017	295.231	184.445	187.791	209.434	Powell
32*	0.426	0.060	119.474	0.725	0.091	0.371	6705.030	Simplex
33	859.632	816.584	659.899	1331.280	3.30E+05	752.662	892.877	Powell
34	1.08E+06	7.98E+05	1.27E+07	7.57E+05	1.20E+06	8.39E+05	1.84E+06	E-TS
35	8.08E+07	3.48E+07	4.69E+09	1.20E+08	1.51E+08	1.13E+08	9.57E+07	Simplex
36*	165.164	287.022	175.563	592.486	244.177	393.154	241.095	BP
37*	0.333	0.399	0.609	0.799	0.066	0.051	0.046	TS
38	1.90E+03	2.28E+03	2.02E+03	5.89E+03	3.01E+03	2.47E+03	2.93E+03	BP
39	1.99E+06	9.52E+05	1.06E+06	1.28E+06	2.05E+06	1.38E+06	1.29E+06	Simplex
40	6.52E+09	2.89E+09	6.01E+09	4.19E+09	3.49E+09	3.05E+09	5.07E+09	Simplex
41	1.10E+06	1.39E+06	8.33E+05	6.70E+05	1.26E+06	8.22E+05	7.76E+05	E-TS
42	2.68E+06	2.14E+06	1.20E+06	1.73E+06	1.48E+06	1.59E+06	1.32E+06	Powell
43*	178.099	133.168	127.358	267.273	177.907	148.182	150.243	Powell
44	304.797	668.804	226.190	614.845	335.063	572.758	356.889	Powell
45*	1.120	1.997	4.369	11.420	72.400	1.082	4.109	TS0

Table 7. Average Interpolation Error with Bests Methods

Table 7 shows that, with the respect to the interpolation error, the BP method obtains the best results in 4 instances, the Simplex in 5, Powell and E_TS in 7, SS in 10 and our both variants together in 12 out of the 45 instances under consideration. With respect to the 26 instances in the set F, the number of best solutions found is 1 for BP, 2 for Simplex and Powell's methods, 3 for E-TS, 8 for SS and 9 for our both tabu search variants (TS0 and TS).

6. Conclusions

A computational study of 12 methods for neural network training has been presented. Specifically, the on-line problem where only a limited training time is allowed has been considered. We have also proposed a new training method based on the tabu search methodology as well as a filtered multi-start framework that can be superimposed to other methods. Overall experiments with 45 functions from the literature were performed to compare the procedures.

Our experiments show that some functions cannot be approximated with a reasonable accuracy level when training the net for a limited number of iterations. The experimentation also shows that the Scatter Search approach by Laguna and Martí(2000) as well as the proposed TS method provide the best performance in terms of solution quality. The proposed filtered multi-start method is able to improve the methods in a long term horizon, but in the context of on-line training seems not to be adequate.

Acknowledgement

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

References

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

Glover, F. (1998) "A Template for Scatter Search and Path Relinking," in *Artificial Evolution, Lecture Notes in Computer Science* 1363, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (Eds.), Springer-Verlag, pp. 13-54.

Glover, F. and J. Kelly (1999) "Combining Simulation and Optimization for Improved Business Decisions," *Colorado Business Review*.

Glover, F., M. Laguna and R. Martí (1999) "Scatter Search," to appear in *Theory and Applications of Evolutionary Computation: Recent Trends*, A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag.

Laguna, M. (2000) "Scatter Search," to appear in *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford Academic Press.

Masters, T. (1995) *Neural, Novel & Hybrid Algorithms for Time Series Prediction*, John Wiley.

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).

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.

Voss, S. and Woodruff D. (2002), Optimization Software Class Libraries, Kluwer Academic Publishers.

Appendix

Table 8 reports the number of variables, the identification number, the name (and parameters) and the lower and upper limit of each function variable. The number in parentheses associated with some of the problems are the parameter values for the corresponding objective function. The set F contains the 26 instances with the symbol "*" in the identification number.

Num. Variables	Id. Number	Name and parameter values	*Low limit*	*Upper limit*
2	1*	Sexton 1	-100	100
	2*	Sexton 2	-100	100
	3*	Sexton 3	-100	100
	4*	Sexton 4	-100	100
	5*	Sexton 5	-100	100
	6*	Branin	-5	15
	7*	B2	-50	100
	8*	Easom	-100	100
	9	Goldstein and Price	-2	2
	10*	Shubert	-10	10
	11	Beale	-4.5	4.5
	12	Booth	-10	10
	13*	Matyas	-5	10
	14	SixHumpCamelback	-5	5
	15*	Schwefel(2)	-500	500
	16	Rosenb rock (2)	-10	10
	17	Zakharov(2)	-5	100
3	18*	De Joung	-2.56	5.12
	19*	Hartmann(3,4)	0	1
4	20	Colville	-10	100
	21*	Shekel(5)	0	10
	22*	Shekel(7)	0	10
	23*	Shekel(10)	0	10
	24	Perm(4,0.5)	-4	04
	25	Perm0(4,10)	-4	4
	26	Powersum(8,18,44,114)	0	4
6	27*	Hartmann(6,4)	0	1
	28*	Schwefel(6)	-500	500
	29*	Trid(6)	*-36*	36
10	30*	Trid(10)	*-100*	100
	31*	Rastrigin(10)	-2.56	5.120
	32*	Griewank(10)	-300	600
	33	Sum Squares(10)	-5	10
	34	Rosenbrock(10)	-10	10
	35	Zakharov(10)	-5	10
20	36*	Rastrigin(20)	-2.56	5.120
	37*	Griewank(20)	-300	600
	38	Sum Squares(20)	-5	10
	39	Rosenbrock(20)	-10	10
	40	Zakharov(20)	-5	10
> 20	41	Powell(24)	-4	5
	42	Dixon and Price(25)	-10	10
	43*	Levy(30)	-10	10
	44	Sphere(30)	-2.56	5.12
	45*	Ackley(30)	-15	30

Table 8. Test Functions

Functions' Description:

1. Sexton 1:

$$f(x) = x_1 + x_2$$

2. Sexton 2:

$$f(x) = x_1 * x_2$$

3. Sexton 3:

$$f(x) = \frac{x_1}{|x_2| + 1}$$

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 and Price:

$$\begin{aligned} f(x) = \ & \left(1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right) \\ & \left(30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right) \end{aligned}$$

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. SixHumpCamelBack :

$$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

15, 23. Schwefel(n):

$$f(x) = 418.9829n + \sum_{i=1}^{n} \left(-x_i \ \sin\sqrt{|x_i|} \right)$$

16, 29, 34. Rosenbrock(n):

$$f(x) = \sum_{i=1}^{n/2} 100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2$$

17, 30, 35. Zakharov(n)

$$f(x) = \sum_{j=1}^{n} x_j^2 + \left(\sum_{j=1}^{n} 0.5 j x_j\right)^2 + \left(\sum_{j=1}^{n} 0.5 j x_j\right)^4$$

18. De Joung :

$$f(x) = x_1^2 + x_2^2 + x_3^2$$

19. Hartmann(3,4):

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right)$$

i	a_{ij}			c_i	p_{ij}		
1	3.0	10.0	30.0	1.0	0.3689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.0381	0.5743	0.8828

20. Colville:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + \\ 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$$

21-23. Shekel(n):

$$f(x) = -\sum_{i=1}^{n} \left((\boldsymbol{x} - \boldsymbol{a}_i)^T(\boldsymbol{x} - \boldsymbol{a}_i) + c_i\right)^{-1} \quad \boldsymbol{x} = (x_1, x_2, x_3, x_4)^T; \boldsymbol{a}_i = \left(a_i^1, a_i^2, a_i^3, a_i^4\right)^T$$

i	a_i^T				c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

24. Perm(n):

$$f(x) = \sum_{k=1}^{n} \left(\sum_{i=1}^{n} \left(i^k + \beta\right)\left(\left(\frac{x_i}{i}\right)^k - 1\right)\right)^2$$

25. Perm0(n):

$$f(x) = \sum_{k=1}^{n} \left(\sum_{i=1}^{n} (i + \beta)\left(x_i^k - \left(\frac{1}{i}\right)^k\right)\right)^2$$

26. PowerSum(b_1,...,b_n):

$$f(x) = \sum_{k=1}^{n} \left(\left(\sum_{i=1}^{n} x_i^k\right) - b_k\right)^2$$

27. Hartmann(6,4):

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij} \left(x_j - p_{ij}\right)^2 \right)$$

i	a_{ij}						c_i
1	10.0	3.0	17.0	3.5	1.7	8.0	1.0
2	0.05	10.0	17.0	0.10	8.0	14.0	1.2
3	3.0	3.5	1.7	10.0	17.0	8.0	3.0
4	17.0	8.0	0.05	10.0	0.1	14.0	3.2

p_{ij}					
0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

29-30. Trid(*n*):

$$f(x) = \left(\sum_{i=1}^{n} \left(x_i - 1\right)^2 \right) - \sum_{i=2}^{n} x_i x_j$$

31, 36. Rastrigin(*n*):

$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos\left(2\pi x_i\right) \right)$$

32, 37. Griewank(*n*):

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$$

33, 38. Sum Squares (*n*):

$$f(x) = \sum_{i=1}^{n} i x_i^2$$

41. Powell(*n*):

$$f(x) = \sum_{j=1}^{n/4} (x_{4j-3} + 10x_{4j-2})^2 + 5(x_{4j-1} - x_{4j})^2 + (x_{4j-2} - 2x_{4j-1})^4 + 10(x_{4j-3} - x_{4j})^4$$

42. Dixon and Price(*n*):

$$f(x) = \sum_{i=1}^{n} i(2x_i^2 - x_{i-1})^2 + (x_1 - 1)^2$$

43. Levy(*n*):

$$f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{k-1} \left(y_i - 1\right)^2 (1 + 10\sin^2(\pi y_i + 1)) + (y_k - 1)^2 (1 + \sin^2(2\pi x_k))$$

where $y_i = 1 + \dfrac{x_i - 1}{4}$ for i=1,…, n

44. Sphere(*n*):

$$f(x) = \sum_{i=1}^{n} x_i^2$$

45. Ackley(*n*):

$$f(x) = 20 + e - 20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$