Chapter 4

VARIABLE NEIGHBOURHOOD SEARCH

José Andrés Moreno Pérez[1], Nenad Mladenović[2], Belén Melián Batista[1] and Ignacio J. García del Amo[1]

[1]Grupo de Computación Inteligente, Instituto Universitario de Desarrollo Regional, ETS Ingeniería Informática. Universidad de La Laguna, 38271 La Laguna, Spain;

[2]School of Mathematics, Brunel University, West London, United Kingdom

Abstract: The basic idea of VNS is the change of neighbourhoods in the search for a better solution. VNS proceeds by a descent method to a local minimum exploring then, systematically or at random, increasingly distant neighbourhoods of this solution. Each time, one or several points within the current neighbourhood are used as initial solutions for a local descent. The method jumps from the current solution to a new one if and only if a better solution has been found. Therefore, VNS is not a trajectory following method (as Simulated Annealing or Tabu Search) and does not specify forbidden moves. In this work, we show how the variable neighbourhood search metaheuristic can be applied to train an artificial neural network. We define a set of nested neighbourhoods and follow the basic VNS scheme to carry out our experiments

Key words: Variable neighbourhood search; neural networks.

1. INTRODUCTION

Artificial neural networks constitute a tool to approximate non-linear mappings from several input variables to several output variables. In order to perform the approximation, the structure of the network has to be determined and a set of parameters, known as weights, has to be tuned. Depending on the domain of the output values, two kinds of problems can be tackled: approximation or prediction problems, for which the output values of the network are continuous variables, and classification problems, for which the output is a single categorical variable. Most of the key issues in the net functionality are common to both.

The main goal in the fitting process is to obtain a model that makes good predictions for new inputs (i.e. to provide good generalization). Once the structure of the network is given, the problem is to find the values of the weights $\boldsymbol{w}$ that optimize the performance of the network in the classification or prediction tasks.

In the supervised learning approach, given a training data set, the network is trained for the classification or prediction tasks by tuning the values of the weights in order to minimize the error across the training set. The training set T consists of a series of input patterns and their corresponding outputs. If the function f to be approximated or predicted has an input vector of variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and the output is represented by $f(\boldsymbol{x})$, the error of the prediction is the difference between the output $p(\boldsymbol{w},\boldsymbol{x})$ provided by the network and the real value $f(\boldsymbol{x})$. The usual way to measure the total error is by the root mean squared difference between the predicted output $p(\boldsymbol{w},\boldsymbol{x})$ and the actual output value $f(\boldsymbol{x})$ for all the elements x in T (*RMSE*; Root Mean Squared Error).

$$RMSE(T,w) = \sqrt{\frac{1}{|T|}\sum_{x\in T}(f(x) - p(w,x))^2}$$

Therefore, the task of training the net by tuning the weights is interpreted as the non-linear optimization problem of minimizing the *RMSE* on the training set through an optimal set of values w^* for the weights. This is equivalent to solve the following problem:

$$RMSE(T,w^*) = \min_{w} RMSE(T,w)$$

To solve this problem, one can apply specific or general optimization techniques. However, the main goal in the development of an artificial neural network is to obtain a design that makes good predictions for future inputs (i.e. which achieves the best possible generalization). Therefore, the design must allow the representation of the systematic aspects of the data rather than their specific details. The standard way to evaluate the generalization provided by the network consists of introducing another set of input/output pairs, V, in order to perform the validation. Once the training has been performed and the weights have been chosen, the performance of the design is given by the *RMSE* across the validation set V, i.e., the *validation error*, stated as:

$$RMSE(V,w^*) = \sqrt{\frac{1}{|V|}\sum_{y\in V}(f(y) - p(w^*,y))^2}$$

The net must exhibit a good fitting between the target values and the output (prediction) in both the training set and the testing set. If the *RMSE* in *T* is significantly higher than that one in *V*, we will say that the net has memorized the data, instead of learning them (i.e., the net has over-fitted the training data). In order to avoid over-fitting, the training process should stop before the network starts to memorize the data instead of learning the general characteristics of the instances. This can be done by introducing a third disjoint set of instances V_T, the test set, for which the *RMSE* is calculated after a certain number of training iterations. If this error increases instead of decreasing, then the training stops.

If the design of the model has too few parameters, it is difficult to fit the network to the data. On the other hand, if it has too many parameters and the structure is general enough, it would over-fit the training data, decreasing the training error and increasing both the test and validation errors. However, there is not a well-established criterion to determine the appropriate number of parameters, and it is difficult to reach a consensus on what the good architectures of the networks are.

The usual artificial neural networks used for the approximation or prediction of a function consist of a series of input nodes or neurons (one for each variable of the function) and one output neuron (or several if the function is multi-dimensional). These input-output neurons are interconnected by additional sets of hidden neurons, which number are variable and have to be determined at design time. A kind of structure that has been widely used is the multilayer architecture, where the neurons are organized in *layers* (*input*, *hidden* and *output*) with connections (weights) only between neurons of different layers. These connections are usually set in a *feed-forward* way, which means that the output of the neurons in one layer is transformed by the weights and fed to the neurons in the next layer.

The typical model consists of a network with only one hidden layer with h neurons. Therefore, the set of neurons N of the network is divided into n input neurons (n is the number of variables of the function to be approximated) in the input layer N_I, h neurons in the hidden layer N_H, and a single neuron in the output layer N_O. The connections follow the *feed-forward* model, i.e., all the connections go from the neurons in the input layer to the neurons in the hidden layer and from the neurons in the hidden layer to the neuron in the output layer.

Blum and Li (1991) proved that a neural network having two layers and sigmoid hidden units could approximate any continuous mapping arbitrarily well. As consequence, regarding to the classification problem, two layer networks with sigmoid units can approximate any decision boundary with 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.

Within a multilayer neural network, the neurons can be enumerated consecutively through the layers from the first to the last one. So we consider a network with a hidden layer to predict a real valued function with n variables consisting of a set of input neurons $N_I = \{ 1, 2, \dots, n \}$, a set of hidden neurons $N_H = \{ n+1, n+2, \dots, n+h \}$ and the output neuron $n+h+1$. The links are:

$$L = \{ (i,n+j): i = 1, \dots, n, j = 1, \dots, h \} \cup \{ (n+j,n+h+1): j = 1, \dots, h \}.$$

This network is shown in *Figure 4-1.*

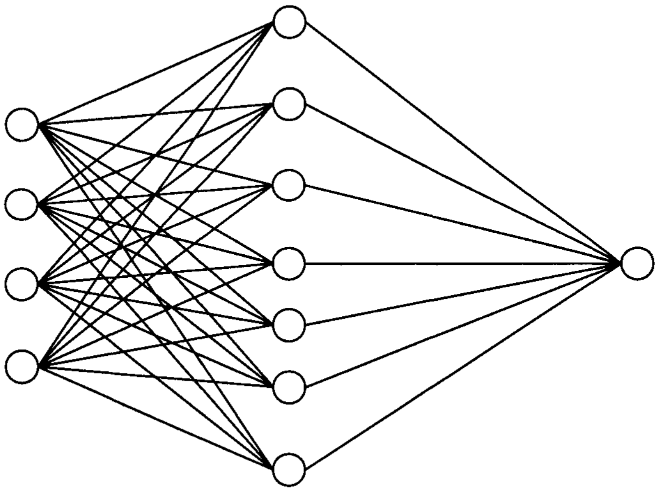

Figure 4-1. General scheme for an artificial neural network.

Given an input pattern $x = (x_1, x_2, \dots, x_n)$ for the input neurons of the network, each hidden neuron receive an input from each input neuron to which it is connected and sends its output to the output neuron.

In the usual models, the inputs of each neuron of the hidden and output layers are linear combinations of the weights of the links and the output of the previous layer. So, the input for the j-th neuron of the hidden layer is:

$$x_j = w_j + \sum_{i=1}^{n} x_j w_{ij}; j = n+1, n+2, \ldots, n+h.$$

Here, w_j is the weight associated to the bias of the previous layer (usually, implemented as an "extra" neuron which always outputs a value of 1.0). Each neuron of the hidden layer, transform its input into an output by the expression $y_j = g(x_j)$ being g the sigmoid function $g(x) = 1 / (1 + exp(-x))$ one of the most used function. The sigmoid's output is in the range (0, 1), but other variants of this function may be used in order to adjust the output to a different range. For prediction problems, it is also usual to consider a linear activation function for the output layer.

2. THE VNS METHODOLOGY

An optimization problem consists in finding the minimum or maximum of a real valued function f defined on an arbitrary set X. If it is a minimization problem, it can be formulated as follows:

$$\min\{f(x) : x \in X\} \qquad (1)$$

In this notation, X denotes the *solution space*, x represents a *feasible solution* and f the *objective function* of the problem. It is a combinatorial optimization problem if the solution space is discrete or partially discrete. An *optimal solution* x^* (or a global minimum) of the problem is a feasible solution for which the minimum of (1) is reached. Therefore, $x^* \in X$ satisfies that $f(x^*) \leq f(x)$, $\forall x \in X$. A *Neighbourhood Structure* in X is defined by a function $N: X \rightarrow 2^X$ where, $\forall x \in X$, $N(x) \subseteq X$ is the set of *neighbours* of x. Then, a *local minimum* x' of (1), with respect to (w.r.t. for short) the neighbourhood structure N, is a feasible solution $x' \in X$ that satisfies the following property: $f(x') \leq f(x)$, $\forall x \in N(x')$. Therefore any *local* or neighbourhood *search* method (i.e., method that only moves to a better neighbour of the current solution) is trapped when it reaches a local minimum.

Several metaheuristics, or frameworks for building heuristics, extend this scheme to avoid being trapped in a local optimum. The best known of them are *Genetic Search*, *Simulated Annealing* and *Tabu Search* (for discussion of these metaheuristics and others, the reader is referred to the books of surveys edited by Reeves (1993) and Glover and Kochenberger (2003)). *Variable Neighbourhood Search* (VNS) (Mladenović and Hansen (1997), Hansen and Mladenović (2001, 2003)) is a recent metaheuristic that exploits systematically the idea of neighbourhood changes with the purpose of both reaching local minima and escaping the valleys which contain them.

VNS runs a descent method from a starting point to reach a local minimum. It explores then a series of different predefined neighbourhoods of this solution. Each time one or several points of the current neighbourhood are used as starting points to run the local descent method it stops at a local minimum. The search jumps to the new local minimum if and only if it is better than the incumbent. In this sense, VNS is not a trajectory following method (that allows non-improving moves within the same neighbourhood) as Simulated Annealing or Tabu Search.

Unlike many other metaheuristics, the basic schemes of VNS and its extensions are simple and require few parameters, and sometimes none at all. Therefore, in addition to providing very good solutions, often in simpler ways than other methods, VNS gives insight into the reasons for such a performance, which in turn can lead to more efficient and sophisticated implementations. Despite its simplicity, it proves to be effective. VNS exploits systematically the following observations:

1. A local minimum with respect to one neighbourhood structure is not necessary so for another.
2. A global minimum is a local minimum with respect to all possible neighbourhood structures.
3. For many problems local minima with respect to one or several neighbourhoods are relatively close to each other.

The last observation, which is empirical, implies that a local optimum often provides some information about the global one. There may for instance be several variables with the same value in both. However, it is usually not known which ones are such. An organized study of the neighbourhoods of this local optimum is therefore performed in order, until a better one is found.

Variable Neighbourhood Descent (VND) is a deterministic version of VNS. It is based on the *observation 1* mentioned above, i.e., *a local optimum for a first type of move* $x \leftarrow x'$ (either heuristic or within the neighbourhood $N_1(x)$) *is not necessary one for another type of move* $x \leftarrow \widetilde{x}$ (within neighbourhood $N_2(x)$). It may thus be advantageous to combine descent heuristics. This leads to the basic VND scheme presented in *Figure 4-2*. It is assumed that the series of neighbourhood structures N_k, $k = 1, \cdots, k_{max}$, that will be used in the algorithm are given.

```
VND method
1. Find an initial solution $x$.
2. Repeat the following sequence until no improvement is
   obtained:
   (i) Set $l \leftarrow 1$;
   (ii) Repeat the following steps until $l = l_{\max}$:
      (a) Find the best neighbor $x'$ of $x$ ($x' \in N_l(x)$);
      (b) If the solution $x'$ thus obtained is better than $x$, set $x \leftarrow x'$
          and $l \leftarrow 1$; otherwise, set $l \leftarrow l+1$;
```

Figure 4-2. Variable Neighbourhood Descent (VND).

Another simple application of the VNS principles appears in the reduced VNS. It is a pure stochastic search method: solutions from the pre-selected neighbourhoods are chosen at random. Its efficiency is mostly based on *observation 3* described above. A set of neighbourhoods $N_1(x)$, $N_2(x)$, $\cdots$, $N_{k_{max}}(x)$ around the current point x (which may be or not a local optimum) is considered. Usually, these neighbourhoods are nested, i.e., each one contains the previous. Then, a point x' is chosen at random in the first neighbourhood. If its value is better than that of the incumbent (i.e., $f(x') < f(x)$), the search is re-centred there ($x \leftarrow x'$). Otherwise, it proceeds to the next neighbourhood. After all neighbourhoods have been considered, the search begins again with the first one, until a stopping condition is satisfied (usually it is the maximum computing time since the last improvement, or the maximum number of iterations). The description of the steps of *Reduced VNS* is as shown in *Figure 4-3*. It is assumed that the neighbourhood structures N_k, $k = 1, \cdots, k_{max}$, that will be used in the shake are given.

```
RVNS method
1. Find an initial solution $x$; choose a stopping condition;
2. Repeat the following sequence until the stoping
   condition is met:
   (i) Set $k \leftarrow 1$;
   (ii) Repeat the following steps until $k = k_{\max}$:
      (a) Shake. Take at random a solution $x'$ from $N_k(x)$;
      (b) If the solution $x'$ is better than the incumbent,
          move there ($x \leftarrow x'$) and continue the search with
          $N_1$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k+1$;
```

Figure 4-3. Reduced Variable Neighbourhood Search (RVNS).

In the two previous methods, we examined how to use variable neighbourhoods in descent to a local optimum and in finding promising regions for near-optimal solutions. Merging the tools for both tasks leads to the *General Variable Neighbourhood Search* (GVNS) scheme. We first discuss how to combine a local search with systematic changes of neighbourhoods around the local optimum found. We then obtain the *Basic VNS* scheme of *Figure 4-4*. Here also, it is assumed that that the neighbourhood structures N_k, $k = 1, \cdots, k_{max}$, that will be used in the shaking are given.

```
BVNS method
1. Find an initial solution $x$; choose a stopping condition;
2. Repeat the following sequence until the stoping condition is met:
   (i) Set $k \leftarrow 1$ ;
   (ii) Repeat the following steps until $k = k_{\max}$ :
      (a) Shaking. Generate a point $x'$ at random from the $k$-th neighborhood of $x$ ( $x' \in N_k(x)$);
      (b) Local Search. Apply some local search method with $x'$ as initial solution; denote with $x''$ the so obtained local optimum;
      (c) Move or not. If the local optimum $x''$ is better than the incumbent $x$, move there ($x \leftarrow x''$) and continue the search with $N_1$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k+1$;
```

Figure 4-4. Basic Variable Neighbourhood Search (BVNS).

The simplest Basic VNS, where the neighbourhood for shaking are fixed, is called *Fixed Neighbourhood Search* (FNS) (see Brimberg et al. (2000)) and sometimes called *Iterated Local Search,* (see Lourenco et al. (2003)). This method selects a neighbour of the current solution by a perturbation, runs a local search from it to reach a local optimum, and moves to it if there has been an improvement. Therefore, the definition of different neighbourhood structures is not necessary, as it can consider only one among them (i.e., by fixing k) and jump (or 'kick the function') in the shaking (or perturbation) step to a point from that fixed neighbourhood. For example in Johnson and Mc-Geosh (1997) a new solution is always obtained from 4-opt (double-bridge) neighbourhood in solving TSP. Thus, k is fixed to 4. The steps of the FNS are obtained considering only one neighbourhood (see *Figure 4-5*).

```
FNS method
1. Initialization:
   Find an initial solution $x$; Set $x^* \leftarrow x$;
2. Iterations:
   Repeat the following sequence until a stopping
   condition is met:
     (a) Shake.
         Take at random a neighbor $x'$ of $x$ ($x' \in N(x)$);
     (b) Local Search.
         Apply the local search method with $x'$ as initial
         solution; denote $x''$ the so obtained local optimum;
     (c) Move or not.
         If $x''$ is better than $x^*$, do $x^* \leftarrow x''$
```

Figure 4-5. Fixed Neighbourhood Search (FNS).

If VND is used instead of simple local search, and the initial solution found by Reduced VNS is improved, the *General Variable Neighbourhood Search* scheme (GVNS) is obtained (shown in *Figure 4-6*). Here, in addition to the set of neighbourhood structures (N_k, $k = 1, \cdots, k_{max}$) to be used in the shaking, the set of neighbourhood structures (N_l, $l = 1, \cdots, l_{max}$) that will be used in the local search are given.

```
GVNS method
1. Initialization:
   Find an initial solution $x$ and improve it by using RVNS.
2. Iterations:
   Repeat the following sequence until the stopping
   condition is met:
   (i) Set $k \leftarrow 1$;
   (ii) Repeat the following steps until $k = k_{max}$;
     (a) Shaking.
         Generate at random a point $x'$ in the $k$-th
         neighborhood of $x$ ($x' \in N_k(x)$);
     (b) Local Search by VND.
         Set $l \leftarrow 1$; and repeat the following steps until $l =$
         $l_{max}$;
         1. Find the best neighbor $x''$ of $x$ in $N_l(x')$
         2. If $f(x'') < f(x')$ set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise
            set $l \leftarrow l + 1$;
     (c) Move or not.
         If this local optimum is better than the incumbent,
         move there ($x \leftarrow x''$), and continue the search with
         $N_1$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$;
```

Figure 4-6. General Variable Neighbourhood Search (GVNS).

Also, a C code for the basic version of the Variable Neighbourhood Search is shown in *Figure 4-7.*

BVNS Code

```
initialize(best_sol) ;
while (t < t_max) {
  k = 0 ;
  while (k < k_max) {
    k++ ;
    cur_sol = shake(best_sol,k) ;
    local_search(cur_sol,best_sol)
    if improved(cur_sol,best_sol) {
      best_sol = cur_sol ;
      k = 0 ;
    } /* if */
  } /* while k */
} /* while t */
```

Figure 4-7. Basic Variable Neighbourhood Search code.

This code of the VNS can be applied to any problem if the user provides the initialization procedure `initialize`, the shake procedure `shake`, the local search procedure `local_search`, and the function `improved` to test if the solution is improved or not.

3. APLICATION OF THE VNS TO THE ARTIFICIAL NEURAL NETWORKS TRAINING PROBLEM

When considering the problem of training an artificial neural network from the metaheuristics point of view, it is useful to treat it as a global optimization problem with the following considerations. The error E is a function of the adaptive parameters of the net, and we can arrange them all (i.e., weights and biases) into a single W-dimensional weight vector $\boldsymbol{w}$ with components $w_1, w_2, \cdots, w_W$. Now, the problem consists in finding the weight vector $\boldsymbol{w}^*$ with the lowest error. All over this chapter we will talk about solutions, meaning simply vectors of weights, as there are no restrictions to the values a weight can have, and thus, every point of the W-dimensional space is a feasible solution. From now on in this section, we will try to explain how a VNS variant can be applied to this problem.

The key point in the VNS is that it searches over several neighbourhood structures in the solution space. A neighbourhood of a solution s_1 is simply a group of solutions that are related to s_1 through a neighbourhood function. Usually, this function is the classical Euclidean distance, so a solution s_2 belongs to the k-th neighbourhood of a solution s_1 if the Euclidean distance between s_2 and s_1 is less than or equal to k. From this example we can also deduce that this function generates nested neighbourhoods, in the sense that if a solution s_2 belongs to the k-th neighbourhood of s_1, it also belongs to its (k+1)-th neighbourhood. This is a desirable property, because if we find that a certain solution s_3 is a local minimum of the k-th neighbourhood, it is also most likely a local minimum of all the previous neighbourhoods $1, \cdots, k-1$.

If the solution space has no inner structure, or this structure is unknown to us, the Euclidean distance is a neighbourhood function as good as any other can be, with the advantage that it is an intuitive function and generates nested neighbourhoods. However, if the solution space does have some kind of inner structure and some information about it is known, it can be used to generate a neighbourhood function that is better fitted to the problem. In the artificial neural network training problem, it seems obvious that the solutions do have such a structure. In fact, we know the structure, as the elements of a solution (the weights) have to be arranged spatially according to the architecture of the net. Then, the goal is to use this information to generate a good neighbourhood function.

With the purpose of determining a possible neighbourhood function that allows us to define a set of nested neighbourhoods, we use some ideas proposed by El Fallahi et al. (2005). The basic idea consists in sorting the neurons in descending order according to the partial errors of their incoming weights respect to the global network error. This can be done by making use of the classical *Backpropagation* method (Rumelhart et al., 1986) to calculate the contribution of each weight to the global error of the network. Then, each neuron is assigned a value equal to the error ratio of its incoming weights. Lastly, we define the k-th neighbourhood of a solution as all the solutions that can be reached by changing the incoming weights of the k first neurons in the ordering (see *Figure 4-8*).

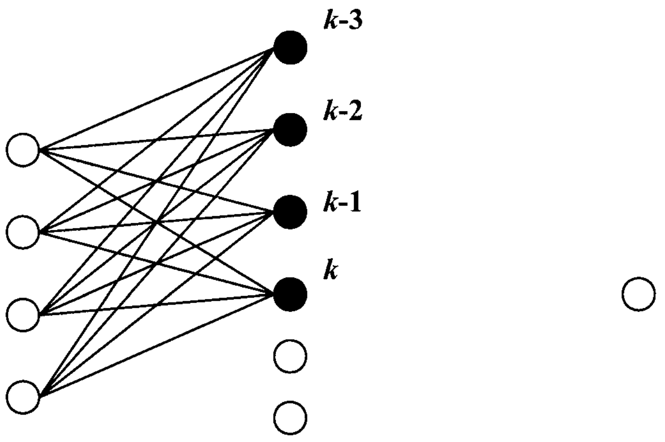

Figure 4-8. Neighbourhood structures for the VNS.

The neighbourhood definition given above satisfies the property of generating nested neighbourhoods, since when increasing the value of k, all the previous neurons, and consequently their incoming weights, are considered. Once a local optimum is reached in any of the neighbourhoods, the partial errors of some weights change and so the neurons ordering. In addition, following the variable neighbourhood scheme, the value of k is set to 1. If the local optimum is good enough, the procedure will eventually consider a neighbourhood containing all the neurons of the net, which is in fact a global search through all the search space.

After deciding the way in which we are going to select a neighbourhood, the next step is to consider the stopping criterion of the search process. Usually, the criterion for stopping the training process for an artificial neural network is the increasing of the error of the test set V_t. However, this condition can be introduced in the "improved" method at the end of each iteration, so a solution would only be accepted if the error of the training set T decreases, and the error of the test set V_t is, at most, the error of the previous best solution for this set. Hence, there is no need for considering it in the stopping criterion, so, in the experimental tests of this chapter, we will use the CPU time. Also, this stopping condition may require more iterations than neurons available in the net, which can lead to inconsistencies in the way neighbourhoods are selected (k can reach a value larger than the neurons in the net, producing an undefined neighbourhood). To solve this, we can slightly modify the neighbourhood function to make it modular, thus selecting the k-th neighbourhood modulus the number of neurons in the net.

The VNS relies on the local search procedure to find a local minimum in a certain neighbourhood. The main advantage of the VNS is that it greatly reduces the search space for the local search procedure. Local search procedures may use either no gradient information at all (as, for example, the "*Flexible Simplex*" method, Nelder and Mead, 1965), first order derivative information (gradient descent, conjugate gradients) or second order derivatives (Newton methods, Levenberg-Marquardt method, etc). The key concept is that gradient information is fast to calculate with the backpropagation technique, and that it is independent of the method used to search the weights-space.

4. EXPERIMENTAL TESTING

In order to test the performance of the VNS for training neural networks, we trained a network to approximate different functions. We compared the results given by a variant of the VNS (the BVNS) and the classical backpropagation, modified to perform a multistart strategy whenever a local miminum is obtained. The problems used consist of several functions, each with different grades of complexity and multiple local minima. These functions are listed below:

$$sexton1: f(x) = x_1 + x_2$$

$$sexton2: f(x) = x_1 * x_2$$

$$sexton3: f(x) = \frac{x_1}{|x_2| + 1}$$

$$sexton4: f(x) = x_1^2 - x_2^3$$

$$sexton5: f(x) = x_1^3 - x_2^2$$

$$Branin: f(x) = \left(x_2 - \left(\frac{5}{4\pi^2}\right)x_1^2 + \left(\frac{5}{\pi}\right)x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$$

$$B2: f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

$$Easom: f(x) = -\cos(x_1)\cos(x_2)\exp\left(-\left((x_1 - \pi)^2 + (x_2 - \pi)^2\right)\right)$$

$$Shubert: f(x) = \left(\sum_{j=1}^{5} j\cos((j+1)x_1 + j)\right)\left(\sum_{j=1}^{5} j\cos((j+1)x_2 + j)\right)$$

$$Booth: f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

$$Matyas: f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$$

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

It is widely known that normalizing both the inputs and the outputs to have zero mean and unit standard deviation improves the training process. Therefore, we normalize the data for the training and test phases using Eq. (2), but we undo the normalization to carry out the validation phase and calculate the errors. The rationale behind this fact is to obtain the final error using the same units as the original data.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{2}$$

An additional consideration has to be taken into account. The normalization of the training set *T* is done by means of calculating the mean and the standard deviation of each of the input and output variables through all the patterns in the pattern set. However, the net modifies its weights to match the output values of the patterns considering a specific mean and standard deviation vector. So, in order to obtain comparable results both for the test and validation sets V_t, V, they must be rescaled with those same mean and standard deviation vectors used for the training set *T*.

The training, test and validation sets consist of 200 observations with data randomly drawn from [-100, 100] for x_1 and [-10, 10] for x_2. The architecture of the net consists of two neurons in the input layer (plus the bias), nine neurons in the hidden layer (plus the bias) and one neuron for the output layer.

In order to get local minima after setting the weights, we have used as local optimizers the Flexible Simplex and the GRG (described in Smith and Lasdon, 1992), although only the results of the GRG are shown, due to the higher quality of the solutions obtained.

The stopping criterion was the CPU time, which was set to 10 minutes for statistical purposes, in order to obtain low variance in the results. However, it is interesting to mention that, in most of the cases, a 40 seconds run was enough to obtain a solution which error was roughly 2 units above the error obtained by the 10 minutes run average solution.

For our computational testing, all the programs were implemented in C and run on a Pentium 4 at 2.4 Ghz.

Table 4-1 summarizes the results of the validation set both for the backpropagation and variable neighbourhood search. First column shows the function name. Second and third columns report the mean squared validation errors over ten runs obtained by the BP and VNS, respectively.

Table 4-1. Computational results

Problem	BP	VNS
Sexton 1	1.77 ± 1.60	0.00 ± 0.00
Sexton 2	8.59 ± 3.94	0.05 ± 0.01
Sexton 3	1.68 ± 0.22	3.69 ±0.18
Sexton 4	43.89 ± 9.86	0.17 ± 0.06
Sexton 5	14.30 ± 5.50	0.05 ±0.02
Branin	15.32 ± 0.87	5.36 ± 1.15
B2	21.56 ± 11.97	0.41 ± 0.02
Easom	0.19 ± 0.06	0.00 ± 0.00
Shubert	13.67 ± 0.02	24.95 ± 0.37
Booth	111.17 ± 6.93	0.05 ± 0.02
Matyas	5.25 ± 2.56	0.02 ± 0.01
Schwefel	2.93 ± 0.53	0.97 ± 0.04

5. CONCLUSIONS

In this chapter we have described the implementation of a variable neighbourhood search for training a feed-forward neural network with one hidden layer.

The results show that a significant improvement on the quality of the solutions is obtained when using the VNS with the GRG as local optimizer. The existence of several variants of the VNS gives a wide set of possible implementations, each of them with its own properties (e. g., VND for faster, lower quality solutions; GVNS for slower, higher quality solutions). However, and independently of the variant used, the definition of the neighbourhood structures provided in this chapter allows for a better understanding of the optimization process (related to neurons, instead of isolated weights), reducing at the same time the average search space size in an efficient way.

ACKNOWLEDGEMENTS

The authors want to acknowledge Rafael Martí and Abdellah El Fallahi for their support and for supplying the GRG code for the computational tests.

This research has been partially supported by the projects TIC2002-04242-C03-01, TIN2005-24056-E (70% of which are FEDER funds) and TIN2004-20061-E (Spanish Heuristic Network).

REFERENCES

Blum, E. K., and Li, L. K., 1991, Approximation theory and feedforward networks, *Neural Networks* **4**(4):511-515.

Brimberg, J., Hansen, P., Mladenović, N., and Taillard, E., 2000, Improvements and comparison of heuristics for solving the multisource Weber problem, *Operations Research* **48**:444-460.

El–Fallahi, A., Martí, R., and Lasdon, L., 2005, Path relinking and GRG for artificial neural networks, *European Journal of Operational Research* (to appear).

Gallant, A. R., and White, H., 1992, On learning the derivatives of an unknown mapping with multilayer feedforward networks, *Neural Networks* **5**:129-138.

Glover, F., and Kochenberger, G., 2003, *Handbook of Metaheuristics*, Kluwer.

Hansen, P., and Mladenović, N., 2001, Variable neighbourhood search: principles and applications, *European Journal of Operational Research* **130**:449-467.

Hansen, P., and Mladenović, N., 2003, Variable neighbourhood search, in: *Handbook of Metaheuristics,* F. Glover, and G. Kochenberger, eds., Kluwer, pp. 145-184.

Johnson, D. S., and McGeoch, L. A., 1997, The travelling salesman problem: A case study in local optimization, in: *Local Search in Combinatorial Optimization,* E. H. L. Aarts, and J. K. Lenstra, eds., John Wiley & Sons, pp. 215-310.

Lourenco, H. R., Martin, O., and Stuetzle, T., 2003, Iterated Local Search, in: *Handbook of Metaheuristics,* Glover, F., and Kochenberger, G., eds., Kluwer, pp. 321-353.

Mladenović, N., and Hansen, P., 1997, Variable neighbourhood search, *Computers and Operations Research* **24**:1097-1100.

Nelder, J. A., and Mead, R., 1965, A Simplex method for function minimization, *Computer Journal* 7:308-313.

Reeves, C. R., 1993, *Modern Heuristic Techniques for Combinatorial Problems,* Blackwell Scientific Press.

Rumelhart, D. E., Hilton, G. E., and Williams, R. J., 1986, Learning internal representations by error propagation, in: D. E. Rumhelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, Cambridge, MA: MIT Press. Reprinted in Aderson and Rosenfeld (1988), pp. 318-362.

Smith, S., and Lasdon L., 1992, Solving large nonlinear programs using GRG, *ORSA Journal on Computing* **4**(1):2 - 15.