Chapter 15

# IMPROVED LEARNING OF NEURAL NETS THROUGH GLOBAL SEARCH

V.P. Plagianakos
*Department of Mathematics, University of Patras, University of Patras Artificial Intelligence Research Center–UPAIRC, GR–26110 Patras, Greece*
vpp@math.upatras.gr

G.D. Magoulas
*School of Computer Science and Information Systems, Birkbeck College, University of London, Malet Street, London WC1E 7HX, UK*
gmagoulas@dcs.bbk.ac.uk

M.N. Vrahatis
*Department of Mathematics, University of Patras, University of Patras Artificial Intelligence Research Center–UPAIRC, GR–26110 Patras, Greece*
vrahatis@math.upatras.gr

**Abstract** Learning in artificial neural networks is usually based on local minimization methods which have no mechanism that allows them to escape the influence of an undesired local minimum. This chapter presents strategies for developing globally convergent modifications of local search methods and investigates the use of popular global search methods in neural network learning. The proposed methods tend to lead to desirable weight configurations and allow the network to learn the entire training set, and, in that sense, they improve the efficiency of the learning process. Simulation experiments on some notorious for their local minima learning problems are presented and an extensive comparison of several learning algorithms is provided.

**Keywords:** Global search, local minima, simulated annealing, genetic algorithms, evolutionary algorithms, neural networks, supervised training, swarm intelligence.

## Introduction

Scientific interest in models of neuronal networks or artificial neural networks mainly arises from their potential ability to perform interesting computational tasks. Nodes, or artificial neurons, in neuronal network models are usually considered as simplified models of biological neurons, i.e. real nerve cells, and the connection weights between nodes resemble to synapses between neurons. In fact, artificial neurons are much simpler than biological neurons. But, for the time being, it is far from clear how much of this simplicity is justified because, as yet, we have only poor understanding of neuronal functions in complex biological networks. Artificial neural nets (ANNs) provide to computing an alternative algorithmic model, which is biologically motivated: the computation is massively distributed and parallel and the learning replaces a priori program development, i.e. ANNs develop their functionality based on training (sampled) data

In neural net learning the objective is usually to minimize a cost function defined as the multi–variable error function of the network. This perspective gives some advantage to the development of effective learning algorithms, because the problem of minimizing a function is well known in the field of numerical analysis. However, due to the special characteristics of the neural nets, learning algorithms can be trapped in an undesired local minimum of the error function: they are based on local search methods and have no mechanism that allows them to escape the influence of an undesired local minimum.

This chapter is focused on the use of Global Optimization (GO) methods for improved learning of neural nets and presents global search strategies that aim to alleviate the problem of occasional convergence to local minima in supervised training. Global search methods are expected to lead to "optimal" or "near-optimal" weight configurations by allowing the network to escape local minima during training.

In practical applications, GO methods can detect just *sub–optimal solutions* of the objective function. In many cases these sub–optimal solutions are acceptable but there are applications where the optimal solution is not only desirable but also indispensable. Therefore, the development of robust and efficient GO methods is a subject of considerable ongoing research.

It is worth noting that, in general, GO–based learning algorithms possess strong theoretical convergence properties, and, at least in principle, are straightforward to implement and apply. Issues related to their numerical efficiency are considered by equipping GO algorithms with a "traditional" local minimization phase. Global convergence, however, needs to be guaranteed by the global–scope algorithm component which, theoretically, should be used in a complete, "exhaustive" fashion. These remarks indicate the inherent computational demand of the GO algorithms, which increases non–polynomially, as a function of problem–size, even in the simplest cases.

The remaining of this chapter is organized as follows. Section 1 formulates the learning problem in the optimization context. In section 2, deterministic monotone and nonmonotone strategies for developing globally convergent modifications of learning algorithms are presented. Section 3 focuses on global search methods and error function transformations to alleviate convergence to undesired local minima. Section 4 presents simulations and comparisons with commonly used learning algorithms, and discusses the results.

## 1. Learning in neural nets

Let us consider an ANN whose $l$-th layer contains $N_l$ neurons ($l = 1, \ldots, L$). The neurons of the first layer receive inputs from the external world and propagate them to the neurons of the second layer (also called hidden layer) for further processing. The operation of the neurons for $l = 2, \ldots, L$ is usually based on the following equations:

$$net_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^{l-1,l} y_i^{l-1}, \qquad y_j^l = f(net_j^l),$$

where $net_j^l$ is for the $j$-th neuron in the $l$-th layer ($l = 2, \ldots, L; j = 1, \ldots, N_l$), the sum of its weighted inputs. The weights from the $i$-th neuron at the $(l-1)$ layer to the $j$-th neuron at the $l$-th layer are denoted by $w_{ij}^{l-1,l}$, $y_j^l$ is the output of the $j$-th neuron that belongs to the $l$-th layer, and $f(net_j^l)$ is the $j$-th's neuron activation function.

If there is a fixed, finite set of input–output pairs, the square error over the training set, which contains $P$ representative cases, is:

$$E(w) = \sum_{p=1}^{P} \sum_{j=1}^{N_L} \left(y_{j,p}^L - t_{j,p}\right)^2 = \sum_{p=1}^{P} \sum_{j=1}^{N_L} \left[\sigma^L(net_j^L + \theta_j^L) - t_{j,p}\right]^2.$$

This equation formulates the error function to be minimized, in which $t_{j,p}$ specifies the desired response at the $j$–th neuron of the output layer at the input pattern $p$ and $y_{j,p}^L$ is the output at the $j$–th neuron of the output layer $L$ that depends on the weights of the network and $\sigma$ is a nonlinear activation function, such as the well known sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$. The weights of the network can be expressed in a vector notation:

$$w = \left(\ldots, w_{ij}^{l-1,l}, w_{i+1\,j}^{l-1,l}, \ldots, w_{N_{l-1}\,j,}^{l-1,l} \theta_j^l, w_{i\,j+1}^{l-1,l}, w_{i+1\,j+1}^{l-1,l}, \ldots\right)^\top,$$

where $\theta_j^l$ denotes the bias of the $j$–th neuron ($j = 1, \ldots, N_l$) at the $l$–th layer ($l = 2, \ldots, L$). This formulation defines the weight vector as a point in the $N$–dimensional real Euclidean space $\mathbb{R}^N$, where $N$ denotes the total number of weights and biases in the network.

Minimization of $E(w)$ is attempted by updating the weights using a learning algorithm. The weight update vector describes a direction in which the weight vector will move in order to reduce the network training error. The weights are modified using the iterative scheme:

$$w^{k+1} = w^k + \Delta w^k, \quad k = 0, 1, \ldots ,$$

where $w^{k+1}$ is the new weight vector, $w^k$ is the current weight vector and $\Delta w^k$ the weight update vector.

Various choices of the correction term $\Delta w^k$ give rise to distinct learning algorithms, which are usually first–order or second–order methods depending on the derivative–related information they use to generate the correction term. Thus, first–order algorithms are based on the first derivative of the learning error with respect to the weights, while second–order algorithms on the second derivative (see [5] for a review on first–order and second–order training algorithms).

A broad class of first–order algorithms, which are considered much simpler to implement than second–order methods, uses the correction term $-\mu\nabla E(w^k)$; $\mu$ is a heuristically chosen constant that usually takes values in the interval $(0, 1)$ (the optimal value of the stepsize $\mu$ depends on the shape of the $N$–dimensional error function) and $\nabla E(w^k)$ defines the gradient vector of the ANN obtained by applying the chain rule on the layers of the network [50].

The most popular first–order algorithm is called Backpropagation (BP) and uses the steepest descent [36] with constant stepsize $\mu$:

$$w^{k+1} = w^k - \mu\nabla E(w^k), \quad k = 0, 1, \ldots .$$

It is well known that the BP algorithm leads to slow network learning and often yields suboptimal solutions [16]. Attempts to speed up back–propagation training have been made by dynamically adapting the stepsize $\mu$ during training [29, 55], or by using second derivative related information [32, 34, 54]. Adaptive stepsize algorithms are more popular due to their simplicity. The stepsize adaptation strategies that are usually suggested are: (i) start with a small stepsize and increase it exponentially, if successive iterations reduce the error, or rapidly decrease it, if a significant error increase occurs [4, 55], (ii) start with a small small stepsize and increase it, if successive iterations keep gradient direction fairly constant, or rapidly decrease it, if the direction of the gradient varies greatly at each iteration [9], (iii) for each weight, an individual stepsize is given, which increases if the successive changes in the weights are in the same direction and decreases otherwise [21, 52], and (iv) use a closed formula to calculate a common stepsize for all the weights at each iteration [29, 46] or a different stepsize for each weight [14, 31]. Note that all the above–mentioned strategies employ heuristic parameters in an attempt to enforce the decrease of the learning error at each iteration and to secure the converge of the learning algorithm.

Methods of nonlinear optimization have also been studied extensively in the context of NNs [32, 54, 56]. Various Levenberg–Marquardt, quasi–Newton and trust–region algorithms have been proposed for small to medium size neural nets [18, 25]. Variations on the above methods, limited-memory quasi-Newton and double dogleg, have been also proposed in an attempt to reduce the memory requirements of these methods [1, 6]. Nevertheless, first–order methods, such as variants of gradient descent [27, 41] and conjugate-gradient algorithms [34] appear to be more efficient in training large size neural nets.

At this point it is worth mentioning an important consideration for adopting an iterative scheme in practical learning tasks is its susceptibility to ill–conditioning: the minimization of the network's learning error is often ill–conditioned, especially when there are many hidden units [51]. Although second–order methods are considered better for handling ill–conditioned problems [5, 32], it is not certain that the extra computational/memory cost these methods require leads to speed ups of the minimization process for nonconvex functions when far from a minimizer [35]; this is usually the case with the neural network training problems, [5], especially when the networks uses a large number of weights [27, 41].

Moreover, BP–like learning algorithms, as well as second–order algorithms, occasionally converge to undesired local minima which affect the efficiency of the learning process. Intuitively, the existence of local minima is due to the fact that the error function is the superposition of nonlinear activation functions that may have minima at different points, which sometimes results in a nonconvex error function [16]. The insufficient number of hidden nodes as well as improper initial weight settings can cause convergence to an undesired local minimum, which prevents the network from learning the entire training set and results in inferior network performance.

Several researchers have presented conditions on the network architecture, the training set and the initial weight vector that allow BP to reach the optimal solution [26, 60]. However, conditions such as the linear separability of the patterns and the pyramidal structure of the ANN [16] as well as the need for a great number of hidden neurons (as many neurons as patterns to learn) make these interesting results not easily interpretable in practical situations even for simple problems.

## 2. Globally Convergent Variants of Local Search Methods

A local search learning algorithm can be made globally convergent by determining the stepsize in such a way that the error is exactly minimized along the current search direction at each iteration, i.e. $E(w^{k+1}) < E(w^k)$. To this end, an iterative search, which is often expensive in terms of error function evaluations, is required. It must be noted that the above simple condition does not guarantee global convergence for general functions, i.e. converges to a local minimizer from any initial condition (see [11] for a general discussion of globally convergent methods).

### Monotone Learning Strategies

In adaptive stepsize algorithms, monotone reduction of the error function at each iteration can be achieved by searching a local minimum with small weight steps. These steps are usually constrained by problem–dependent heuristic learning parameters.

The use of heuristic strategies enforces the monotone decrease of the learning error and secures the convergence of the training algorithm to a minimizer of $E$. However, the use of inappropriate values for the heuristic learning parameters can considerably slow down the rate of training or even lead to divergence and to premature saturation [26, 49]; there is a trade–off between convergence speed and stability of the training algorithm. Additionally, the use of heuristics for bounding the stepsize prevents the development of efficient algorithms with the property that starting from any initial weight vector the weight updates will converge to a local minimum, i.e. globally convergent training algorithms.

A monotone learning strategy, which does not apply heuristics to bound the length of the minimization step, consists in accepting a positive stepsize $\eta^k$ along the search direction $\varphi^k \neq 0$, if it satisfies the *Wolfe conditions*:

$$E(w^k + \eta^k\varphi^k) - E(w^k) \leqslant \sigma_1 \eta^k \left\langle \nabla E(w^k), \varphi^k \right\rangle, \qquad (1.1)$$

$$\left\langle \nabla E(w^k + \eta^k \varphi^k), \varphi^k \right\rangle \geqslant \sigma_2 \left\langle \nabla E(w^k), \varphi^k \right\rangle, \qquad (1.2)$$

where $0 < \sigma_1 < \sigma_2 < 1$ and $\langle \cdot, \cdot \rangle$ stands for the usual inner product in $\mathbb{R}^n$. The first inequality ensures that the error is reduced sufficiently, and the second prevents the stepsize from being too small. It can be shown that if $\varphi^k$ is a descent direction and $E$ is continuously differentiable and bounded below along the ray $\{w^k + \eta\varphi^k \mid \eta > 0\}$, then there always exists a stepsize satisfying (1.1)–(1.2) [11, 35]. Relation (1.2) can be replaced by:

$$E(w^k + \eta^k \varphi^k) - E(w^k) \geqslant \sigma_2 \eta^k \left\langle \nabla E(w^k), \varphi^k \right\rangle, \qquad (1.3)$$

where $\sigma_2 \in (\sigma_1, 1)$ (see [11]). The strategy based on Wolfe's conditions provides an efficient and effective way to ensure that the error function is globally reduced sufficiently. In practice, conditions (1.2) or (1.3) are generally not needed because the use of a backtracking strategy avoids very small learning rates [31, 57].

An alternative strategy has been proposed in [47]. It is applicable to any descent direction $\varphi^k$ and uses two parameters $\alpha, \beta \in (0, 1)$. Following this approach the stepsize is $\eta^k = \beta^{m_k}$, where $m_k \in \mathbb{Z}$ is any integer such that:

$$E(w^k + \beta^{m_k} \varphi^k) - E(w^k) \leqslant \beta^{m_k} \alpha \left\langle \nabla E(w^k), \varphi^k \right\rangle, \qquad (1.4)$$

$$E(w^k + \beta^{m_k - 1} \varphi^k) - E(w^k) > \beta^{m_k - 1} \alpha \left\langle \nabla E(w^k), \varphi^k \right\rangle. \qquad (1.5)$$

To ensure global convergence, monotone strategies that employ conditions (1.1)–(1.2) or (1.4)–(1.5) must be combined with stepsize tuning subprocedures. For example, a simple subprocedure for tuning the length of the minimization step is to decrease the stepsize by a reduction factor $q^{-1}$, where $q > 1$ [36], so that it satisfies conditions (1.1)–(1.2) at each iteration. This *backtracking strategy* has the effect that the stepsize is decreased by the largest number in the sequence $\{q^{-m}\}_{m=1}^{\infty}$, so that condition (1.1) is satisfied. When seeking to satisfy (1.1) it is important to ensure that the stepsize is not reduced unnecessarily so that condition (1.2) is not satisfied. Since in training, the gradient vector is known only at the beginning of the iterative search for a new weight vector, condition (1.2) cannot be checked directly (this task requires additional gradient evaluations at each iteration), but is enforced simply by placing a lower bound on the acceptable values of the stepsize. This bound on the stepsize has the same theoretical effect as condition (1.2), and ensures global convergence [11].

## Nonmonotone Learning Strategies

Although monotone learning strategies provide an efficient and effective way to ensure that the error function is reduced sufficiently, they have the disadvantage that no information, which might accelerate convergence, is stored and used [15]. To alleviate this situation we propose a nonmonotone learning strategy that exploits the accumulated information with regard to the $M$ most recent values of the error function. The following condition is used to formulate the new approach and to define a criterion of acceptance of any weight iterate:

$$E\left(w^k - \eta^k \nabla E(w^k)\right) - \max_{0\leqslant j\leqslant M} E(w^{k-j}) \leqslant \gamma\ \eta^k \left\langle \nabla E(w^k), \phi^k \right\rangle, \quad (1.6)$$

where $M$ is a nonnegative integer, named *nonmonotone learning horizon*, $0 < \gamma < 1$, $\eta^k$ indicates the learning rate and $\phi^k$ is the search direction at the $k$th iteration. The above condition allows for an increase in the function values without affecting the global convergence properties, as it has been proved theoretically in [17, 48].

Furthermore, it can be shown that the nonmonotone learning strategy generates a globally convergent sequence for any algorithm that follows a search direction $\varphi^k \neq 0$, provided that two positive numbers $c_1, c_2$ exist, such that:

$$\left\langle \nabla E(w^k), \varphi^k \right\rangle \leqslant -c_1 \|\nabla E(w^k)\|, \quad (1.7)$$

$$\|\varphi^k\| \leqslant c_2 \|\nabla E(w^k)\|. \quad (1.8)$$

This follows directly from the convergence theorem in [17].

Next, we summarize the basic steps of the nonmonotone learning strategy at the $k$th iteration:

**1:** Update the weights $w^{k+1} = w^k + \eta^k \varphi^k$.

**2:** If $E(w^{k+1}) - \max_{0\leqslant j\leqslant M^k} E(w^{k-j}) \leqslant \gamma\ \eta^k \left\langle \nabla E(w^k), \phi^k \right\rangle$, store $w^{k+1}$, set $k = k+1$ and go to Step 1; otherwise go to the next step.

**3:** Use a tuning technique for $\eta^k$ and return to Step 2.

Experimental results indicate that the choice of the parameter $M$ is critical for the implementation and depends on the nature of the problem [42, 46]. Therefore, instead of using a user–defined value for the nonmonotone learning horizon $M$, an adaptive procedure can be applied to dynamically evaluate $M$.

To this end, the following procedure, based on the notion of the Lipschitz constant, dynamically adapts the value of the nonmonotone learning horizon $M$ at each iteration:

$$M^k = \begin{cases} M^{k-1}+1, & \Lambda^k < \Lambda^{k-1} < \Lambda^{k-2}, \\ M^{k-1}-1, & \Lambda^k > \Lambda^{k-1} > \Lambda^{k-2}, \\ M^{k-1} \quad , & \text{otherwise,} \end{cases} \tag{1.9}$$

where $\Lambda^k$ is the local estimation of the Lipschitz constant at the $k$th iteration [29]:

$$\Lambda^k = \frac{\|\nabla E(w^k) - \nabla E(w^{k-1})\|}{\|w^k - w^{k-1}\|}, \tag{1.10}$$

which can be obtained without additional error function or gradient evaluations. If $\Lambda^k$ is increased for two consecutive iterations, the sequence of the weight vectors approaches a steep region and the value of $M$ has to be decreased in order to avoid overshooting a possible minimum point. On the other hand, when $\Lambda^k$ is decreased for two consecutive iterations, the method possibly enters a valley in the weight space, so the value of $M$ has to be increased. This allows the method to accept larger stepsizes and move faster out of the flat region. Finally, when the value of $\Lambda^k$ has a rather random behavior (increasing or decreasing for only one iteration), the value of $M$ remains unchanged. It is evident that $M$ has to be positive. Thus, if Relation (1.9) gives a non positive value in $M$, the nonmonotone learning horizon is set equal to 1 in order to ensure that the error function is sufficiently reduced at the current iteration.

At this point it is useful to remark that a simple technique to tune $\eta^k$ at Step 3 is to decrease the stepsize by a reduction factor $1/q$, where $q > 1$, as mentioned in the previous subsection. The selection of $q$ is not crucial for successful learning, however, it has an influence on the number of error function evaluations required to obtain an acceptable weight vector. Thus, some training problems respond well to one or two reductions in the stepsize by modest amounts (such as $1/2$), while others require many such reductions, but might respond well to a more aggressive stepsize reduction (for example by factors of $1/10$, or even $1/20$). On the other hand, reducing $\eta^k$ too much can be costly since the total number of iterations will be increased. The value $q = 2$ is usually suggested in the literature [2] and, indeed, it was found to work effectively and efficiently in the experiments [41, 46]. The above procedure constitutes an efficient method of determining an appropriate stepsize without additional gradient evaluations. As a consequence, the number of gradient evaluations is, in general, less than the number of error function evaluations.

The nonmonotone learning strategy can be used as a subprocedure that secures and accelerates the convergence of a learning algorithm by providing the ability to handle arbitrary large stepsizes, and, in this way, learning by neural nets becomes feasible on a first–time basis for a given problem. Additionally, it alleviates problems generated by poor selection of the user–defined learning parameters, such as decreased rate of convergence, or even divergence and convergence to undesired local minima due to premature saturation [26]. It is worth noting that any stepsize adaptation strategy can be incorporated in Step 1 of the above algorithm model. For example, in [41, 46] the nonmonotone Backpropagation with variable stepsize (NMBPVS) and the nonmonotone Barzilai–Borwein Backpropagation (NMBBP) have been proposed.

The NMBPVS is the nonmonotone version of the Backpropagation with Variable Stepsize (BPVS) [29], which exploits the local shape of the error surface to obtain a local estimate the Lipschitz constant at each iteration and uses this estimate to adapt the stepsize $\eta^k$. The nonmonotone strategy helps to eliminate the possibility of using an unsuitable local estimation of the Lipschitz constant.

With regards to the NMBBP, the nonmonotone strategy helps to secure the convergence of the BBP method [42], even when the Barzilai–Borwein formula [3] gives an unsuitable stepsize. Experimental results show that the NMBBP retains the ability of BBP to escape from undesirable regions in the weight space, i.e. undesired local minima and flat valleys, whereas other methods are trapped within these regions [41, 46].

Furthermore, alternative weight adaptation rules can be used in Step 2 of the above algorithm model to develop their nonmonotone version. For example, in [21, 50] a simple, heuristic strategy for accelerating the BP algorithm has been proposed based on the use of a momentum term. The momentum term can been incorporated in the steepest descent method as follows:

$$w^{k+1} = w^k - (1-m)\eta\nabla E(w^k) + m(w^k - w^{k-1}),$$

where $m$ is the momentum constant. A drawback with the above scheme is that, if $m$ is set to a comparatively large value, gradient information from previous iterations is more influential than the current gradient information in updating the weights. A solution is to increase the stepsize, however, in practice, this approach frequently proves ineffective and leads to instability or saturation. Thus, if $m$ is increased, it may be necessary to make a compensatory reduction in $\eta$ to maintain network stability. Combining the BP with Momentum (BPM) with the nonmonotone learning strategy (this is named NMBPM) helps to alleviate this problem.

## 3. Learning Through Global Search Methods

In this section we focus on global search methods for neural network learning and we propose objective function transformation techniques that can be combined with any search method (either local or global) to alleviate the problem of occasional convergence to undesired local minima.

### Adaptive stochastic search algorithms

Adaptive stochastic search algorithms include, simulated annealing [8, 24], genetic and evolutionary algorithms [33], as well as swarm intelligence [13, 22, 23]. Next, the fundamentals of those methods are reviewed.

**The method of simulated annealing.** *Simulated Annealing* (SA) refers to the process in which random noise in a system is systematically decreased at a constant rate so as to enhance the response of the system [24].

In the numerical optimization framework, SA is a procedure that has the capability to move out of regions near local minima [10]. SA is based on random evaluations of the objective function, in such a way that transitions out of a local minimum are possible. It does not guarantee, of course, to find the global minimum, but if the function has many good near–optimal solutions, it should find one. In particular, SA is able to discriminate between "gross behavior" of the function and finer "wrinkles". First, it reaches an area in the function domain space where a global minimizer should be present, following the gross behavior irrespectively of small local minima found on the way. It then develops finer details, finding a good, near–optimal local minimizer, if not the global minimum itself.

In the context of neural network learning the performance of the classical SA is not the appropriate one: the method needs a greater number of function evaluations than that usually required for a single run of first–order learning algorithms and does not exploit derivative related information. Notice that the problem with minimizing the neural network error function is not the well defined local minima but the broad regions that are nearly flat. In this case, the so–called Metropolis move is not strong enough to move the algorithm out of these regions [59].

In [8], it has been suggested to incorporate SA in the BP algorithm:

$$w^{k+1} = w^k - \mu \nabla E(w^k) + nc2^{-dk},$$

where $n$ is a constant controlling the initial intensity of the noise, $c \in (-0.5, +0.5)$ is a random number and $d$ is the noise decay constant. In the experiments reported below we have applied this technique for updating the weights from the beginning of the training as proposed by Burton *et al.* [8]. Alternatively, we update the weights using plain BP until convergence to an undesired local minimum is obtained, then we switch to SA. This combined BP with SA is named BPSA.

**Genetic Algorithms.** *Genetic Algorithms* (GA) are simple and robust search algorithms based on the mechanics of natural selection and natural genetics. The mathematical framework of GAs was developed in the 1960s and is presented in Holland's pioneering book [19]. GAs have been used primarily in optimization and machine learning problems and their operation is briefly described as follows. At each *generation* of a GA, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than their progenitors, just as in natural adaptation. For a high level description of the simple GA see Figure 1.1.

More specifically, a simple GA processes a finite population of fixed length binary strings called *genes*. GAs have two basic operators, namely: *crossover* of genes and *mutation* for random change of genes. The crossover operator explores different structures by exchanging genes between two strings at a crossover position and the mutation operator is primarily used to escape the local minima in the weight space by altering a bit position of the selected string; thus introducing diversity in the population. The combined action of crossover and mutation is responsible for much of the effectiveness of GA's search. Another operator associated with each of these operators is the *selection* operator, which produces survival of the fittest in the GA.

The parallel noise–tolerant nature of GAs, as well as their hill–climbing capability, make GAs eminently suitable for training neural networks, as they seem to search the weight space efficiently. The "Genetic Algorithm for Optimization Toolbox (GAOT)" [20] has been used for the experiments reported here. GAOT's default crossover and mutation schemes, and a real–valued encoding of the ANN's weights have been employed.

**Evolutionary Algorithms.** *Evolutionary algorithms* (EA) are adaptive stochastic search methods which mimic the metaphor of natural

```
STANDARD GENETIC ALGORITHM MODEL
{
  //initialise the time counter
  t := 0;
  //initialise the population of individuals
  InitPopulation(P(t));
  //evaluate fitness of all individuals
  Evaluate(P(t));
  //test for termination criterion (time, fitness, etc.)
  while not done do
    t := t + 1;
    //select a sub-population for offspring production
    Q(t) := SelectParents(P(t));
    //recombine the "genes" of selected parents
    Recombine(Q(t));
    //perturb the mated population stochastically
    Mutate(Q(t));
    //evaluate the new fitness
    Evaluate(Q(t));
    //select the survivors for the next generation
    P(t + 1) := Survive(P(t), Q(t));
  end
}
```

*Figure 1.1.* A high level description of the simple GA Algorithm

biological evolution. Differently from other adaptive stochastic search algorithms, evolutionary computation techniques operate on a set of potential solutions, which is called *population*, applying the principle of survival of the fittest to produce better and better approximations to a solution, and, through cooperation and competition among the potential solutions, they find the optimal one. This approach often helps finding optima in complicated optimization problems more quickly than traditional optimization methods.

To demonstrate the efficiency of the EA in alleviating the local minima problem, we have used the *Differential Evolution* (DE) strategies [53]. DE strategies have been designed as stochastic parallel direct search methods that can efficiently handle non differentiable, nonlinear and multimodal objective functions, and require few, easily chosen control parameters. Experimental results [28] have shown that DE algorithms have good convergence properties and outperform other evolutionary methods [44, 45]. To apply DE algorithms to neural network learning we start with a specific number (*NP*) of $N$–dimensional weight vectors, as an initial weight population, and evolve them over time. The number of individuals *NP* is kept fixed throughout the learning process and the weight vectors population is initialized randomly following a uniform probability distribution. As in GAs, at each iteration of the DE algorithm, called *generation*, new weight vectors are generated by the combination of weight vectors randomly chosen from the population, which is called *mutation*. The outcoming weight vectors are then mixed with another predetermined weight vector, the *target* weight vector. This operation is called *crossover* and it yields the so–called *trial* weight vector. This vector is accepted for the next generation if and only if it reduces the value of the error function $E$. This last operation is called *selection*.

Below, we briefly review the two basic DE operators used for ANN learning. The first DE operator, we consider, is mutation. Specifically, for each weight vector $w_g^i$, $i = 1, \dots, NP$, where $g$ denotes the current generation, a new vector $v_{g+1}^i$ (mutant vector) is generated according to one of the following relations:

$$\text{Alg. DE}_1: \quad v_{g+1}^i = w_g^{r_1} + \xi\left(w_g^{r_1} - w_g^{r_2}\right), \tag{1.11}$$

$$\text{Alg. DE}_2: \quad v_{g+1}^i = w_g^{\text{best}} + \xi\left(w_g^{r_1} - w_g^{r_2}\right), \tag{1.12}$$

$$\text{Alg. DE}_3: \quad v_{g+1}^i = w_g^{r_1} + \xi\left(w_g^{r_2} - w_g^{r_3}\right), \tag{1.13}$$

$$\text{Alg. DE}_4: \quad v_{g+1}^i = w_g^i + \xi\left(w_g^{\text{best}} - w_g^i\right) + \xi\left(w_g^{r_1} - w_g^{r_2}\right), \tag{1.14}$$

$$\text{Alg. DE}_5: \quad v_{g+1}^i = w_g^{\text{best}} + \xi\left(w_g^{r_1} - w_g^{r_2}\right) + \xi\left(w_g^{r_3} - w_g^{r_4}\right), \tag{1.15}$$

$$\text{Alg. DE}_6: \quad v_{g+1}^i = w_g^{r_1} + \xi\left(w_g^{r_2} - w_g^{r_3}\right) + \xi\left(w_g^{r_4} - w_g^{r_5}\right), \tag{1.16}$$

where $w_g^{\text{best}}$ is the best member of the previous generation, $\xi > 0$ is a real parameter, called mutation constant, which controls the amplification of the difference between two weight vectors, and

$$r_1, r_2, r_3, r_4, r_5 \in \{1, 2, \dots, i-1, i+1, \dots, NP\}$$

are random integers mutually different and different from the running index $i$.

Relation (1.11) has been introduced as crossover operator for GAs [33] and is similar to Relations (1.12) and (1.13). The remaining relations are modifications which can be obtained by the combination of (1.11), (1.12) and (1.13). It is clear that many more relations of this type can be generated using the above ones as building blocks. In recent works [44, 45], we have shown that the above relations can efficiently be used to train ANNs with arbitrary integer weights as well.

The second DE operator, i.e. the crossover, is applied to increase the diversity of the mutant weight vector. Specifically, for each component $j$ ($j = 1, 2, \dots, N$) of the mutant weight vector $v_{g+1}^i$, we randomly choose a real number $r$ in the interval $[0, 1]$. Then, this number is compared with the crossover constant $\rho$; if $r \leqslant \rho$ we replace the $j$-th component of the trial vector $u_{g+1}^i$ with the $j$-th component of the mutant vector $v_{g+1}^i$; otherwise, we pick the $j$-th component of the target vector $w_g^i$.

**The particle swarm optimization method.** In *Particle Swarm Optimization* (PSO) algorithm the population dynamics simulates a "bird flock's" behavior where social sharing of information takes place and individuals can profit from the discoveries and previous experience of all other companions during the search for food. Thus, each companion, called *particle*, in the population, which is now called *swarm*, is assumed to "fly" over the search space in order to find promising regions of the landscape. For example, in the minimization case, such regions possess lower functional values than other visited previously. In this context, each particle is treated as a point in a $N$–dimensional space which adjusts its own "flying" according to its flying experience as well as the flying experience of other particles (companions).

There are many variants of the PSO proposed so far, after Eberhart and Kennedy introduced this technique [13, 22]. In our experiments we have used a version of this algorithm, which is derived by adding a new inertia weight to the original PSO dynamics [12]. This version is described in the following paragraphs.

First let us define the notation used: the $i$-th particle of the swarm is represented by the $N$–dimensional vector $X_i = (x_{i1}, x_{i2}, \dots, x_{iN})$ and the best particle in the swarm, i.e. the particle with the smallest function value, is denoted index $g$. The best previous position (the position giving the best function value) of the $i$-th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, \dots, p_{iN})$, and the position change (velocity) of the $i$-th particle is $V_i = (v_{i1}, v_{i2}, \dots, v_{iN})$.

The particles are manipulated according to the equations

$$v_{in} = w\, v_{in} + c_1 r_1 (p_{in} - x_{in}) + c_2 r_2 (p_{gn} - x_{in}), \tag{1.17}$$

$$x_{in} = x_{in} + v_{in}, \tag{1.18}$$

where $n = 1, 2, \dots, N$; $i = 1, 2, \dots, NP$ and $NP$ is the size of population; $w$ is the inertia weight; $c_1$ and $c_2$ are two positive constants; $r_1$ and $r_2$ are two random values in the range $[0, 1]$.

The first equation is used to calculate $i$-th particle's new velocity by taking into consideration three terms: the particle's previous velocity, the distance between the particle's best previous and current position, and, finally, the distance between swarm's best experience (the position of the best particle in the swarm) and $i$-th particle's current position. Then, following the second equation, the $i$-th particle flies toward a new position. In general, the performance of each particle is measured according to a predefined fitness function, which is problem–dependent.

The role of the inertia weight $w$ is considered very important in PSO convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current velocity. In this way, the parameter $w$ regulates the trade–off between the global (wide–ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine–tuning the current search area. A suitable value for the inertia weight $w$ usually provides balance between global and local exploration abilities and consequently a reduction on the number of iterations required to locate the optimum solution. A general rule of thumb suggests that it is better to initially set the inertia to a large value, in order to make better global exploration of the search space, and gradually decrease it to get more refined solutions, thus a time decreasing inertia weight value is used.

From the above discussion it is obvious that PSO, to some extent, resembles EAs. However, in PSO, instead of using genetic operators, each individual (particle) updates its own position based on its own search experience and other individuals (companions) experience and discoveries. Adding the velocity term to the current position, in order to generate the next position, resembles the mutation operation in evolutionary programming. Note that in PSO, however, the "mutation" operator is guided by particle's own "flying" experience and benefits by the swarm's "flying" experience. In another words, PSO is considered as performing mutation with a "conscience", as pointed out by Eberhart and Shi [12].

In general, PSO has been proved very efficient in a plethora of application in science and engineering [23, 38–40]

## Transforming the objective function

Let a point $\bar{w}$ such that there exists a neighborhood $\mathcal{B}$ of $\bar{w}$ with

$$E(\bar{w}) \leqslant E(w), \quad \forall\, w \in \mathcal{B}. \tag{1.19}$$

This point is a local minimizer of the error function and, as already mentioned above, many methods get stuck in such undesired local minima. The main idea of applying a transformation to the error function is to make some undesired local minima disappear, while keeping the location of the global minimizer unchanged. The techniques that will be described below aim at transforming the error function in such a way that convergence to a global minimizer is enhanced for *any* learning algorithm that is equipped with them. Two methods are described: the *deflection procedure* and the *function stretching* technique.

**The deflection procedure.** Following the *deflection procedure* proposed in [30], when the sequence of weight vectors $\{w^k\}_0^\infty$ converges to a local minimum $\bar{w} \in \mathbb{R}^N$ the error function $E(w)$ is reformulated as follows:

$$F(w) = S(w; \bar{w}, \lambda)^{-1} E(w),$$

where $S(w; \bar{w}, \lambda)$ is a function depending on a weight vector $w$ and on the local minimizer $\bar{w}$ of $E$; $\lambda$ is a relaxation parameter. In case there exist $m$ local minima $\bar{w}_1, \ldots, \bar{w}_m \in \mathbb{R}^N$, the above relation is reformulated as:

$$F(w) = S(w; \bar{w}_1, \lambda_1)^{-1} \cdots S(w; \bar{w}_m, \lambda_m)^{-1} E(w).$$

The deflection procedure suggests to find a "proper" $S(\cdot)$ such that $F(w)$ will not have a minimum at $\bar{w}_i, i = 1, \ldots, m$, while keeping all other minima of $E$ locally "unchanged". In other words, we have to construct functions $S$ that provide $F$ with the property that any sequence of weights converging to $\bar{w}_i$ (a local minimizer of $E$) will not produce a minimum of $F$ at $w = \bar{w}_i$. In addition, this function $F$ will retain all other minima of $E$. This is *the deflection property* [30]. For example, the function:

$$S(w; \bar{w}_i, \lambda_i) = \tanh\left(\lambda_i \|w - \bar{w}_i\|\right),$$

provides $F$ with this property, as it will be explained below.

Let us assume that a local minimum $\bar{w}_i$ has been determined, then

$$\lim_{w \to \bar{w}_i} \frac{E(w)}{\tanh\left(\lambda \|w - \bar{w}_i\|\right)} = +\infty,$$

which means that $\bar{w}_i$ is no longer a local minimizer of $F$. Moreover, it is easily verified that for $\|w - \bar{w}_i\| \geqslant \varepsilon$, where $\varepsilon$ is a small positive constant, it holds that:

$$\lim_{\lambda \to +\infty} F(w) = \lim_{\lambda \to +\infty} \frac{E(w)}{\tanh\left(\lambda \|w - \bar{w}_i\|\right)} = E(w), \tag{1.20}$$

since the denominator tends to unity. This means that the error function remains unchanged in the whole weight space.

It is worth noticing that the effect of the deflection procedure is problem–dependent and is related to the value of $\lambda$. For an arbitrary value of $\lambda$ there is a small neighborhood $\mathcal{R}(\bar{w}, \rho)$ with center $\bar{w}$ and radius $\rho$, with $\rho \propto \lambda^{-1}$, that for any $x \in \mathcal{R}(\bar{w}, \rho)$ it holds that $F(w) > E(w)$. To be more specific, when the value of $\lambda$ is small (say $\lambda < 1$) the denominator in the above relation becomes one for $w$ "far" from $\bar{w}$. Thus, the deflection procedure affects a large neighborhood around $\bar{w}$ in the weight space. On the other hand, when the value of $\lambda$ is large, new local

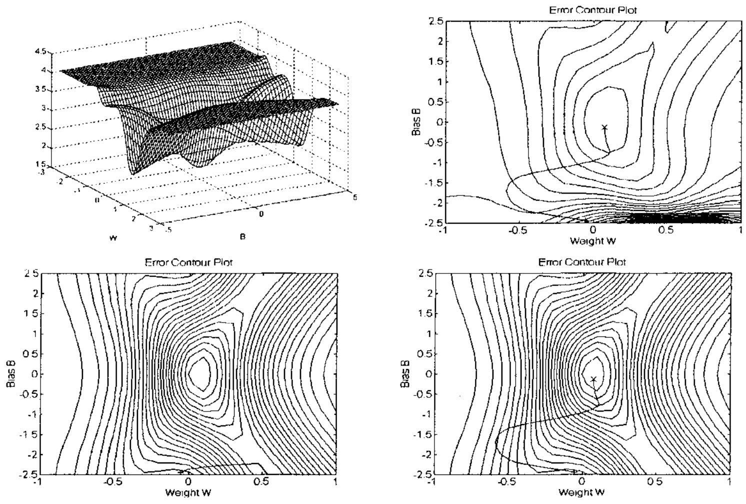

*Figure 1.2.* Applying deflection to a simple learning task

minima is possible to be created near the computed minimum $\bar{w}$, like a "Mexican hat". These minima have function values greater than $F(\bar{w})$ and can be easily avoided by taking a proper stepsize or by changing the value of $\lambda$.

To better visualize the effect of the deflection procedure, we provide an application example. It concerns training a single neuron using the BP algorithm to associate 8 input–output pairs. The error surface of the problem is shown in Fig. 1.2 (top–left). The desired minimum is located at the center and there are two valleys that lead to undesired local minima. In Fig. 1.2 (bottom–left) we illustrate the weight trajectory when the initial conditions lead the learning algorithm to converge to an undesired local minimum. In Fig. 1.2 (top–right) and in Fig. 1.2 (bottom–right) we present the deflected trajectory of weights drawn on the contour lines of the original and the error function subject to deflection, respectively.

Notice that the deflection procedure can be incorporated in any learning algorithm to help escaping the influence of local minima. In the experiments reported below, the classical BP method has been equipped with the deflection procedure. The resulting scheme is named BP with deflection (BPD).

**The function "stretching" technique.** The *function "stretching" technique* [37] consists of a two–stage transformation in the form of the original error function $E(w)$ and can be applied soon after a local minimum $\bar{w}$ of the function $E$ has been detected:

$$G(w) = E(w) + \frac{\gamma_1}{2}\, \|w - \bar{w}\| \,(\operatorname{sign}(E(w) - E(\bar{w})) + 1)\,, \quad (1.21)$$

$$H(w) = G(w) + \gamma_2 \frac{\operatorname{sign}(E(w) - E(\bar{w})) + 1}{2\tanh(\mu(G(w) - G(\bar{w})))}, \quad (1.22)$$

where $\gamma_1, \gamma_2$ and $\mu$ are arbitrary chosen positive constants, and sign$(\cdot)$ defines the well known three valued sign function. Note that the sign function can be approximated by the well known logistic function:

$$\operatorname{sign}(w) \approx \operatorname{logsig}(w) = \frac{2}{1 + \exp(-\nu w)} - 1 \equiv \tanh\left(\frac{\nu}{2}\, w\right),$$

for a large value of $\nu$. This sigmoid function is continuously differentiable and is widely used as a transfer function in artificial neurons.

It is worth noticing that the first transformation stage elevates $E(w)$ and makes disappear all the local minima located above $\bar{w}$. The second stage stretches the neighborhood of $\bar{w}$ upwards, since it assigns higher function values to those points. Both stages do not alter the local minima located below $\bar{w}$; thus, the global minimizer is left unchanged.

At this point it is useful to provide an application example of this technique in order to illustrate its effect. The problem considered is a notorious two dimensional test function, called the *Levy No. 5*:

$$\begin{aligned} f(x) &= \sum_{i=1}^{5} i\cos[(i+1)x_1 + i] \times \sum_{j=1}^{5} j\cos[(j+1)x_2 + j] + \\ &\quad + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2, \end{aligned} \quad (1.23)$$

where $-10 \leqslant x_i \leqslant 10, i = 1, 2$. There are about 760 local minima and one global minimum with function value $f^* = -176.1375$ located at $x^* = (-1.3068, -1.4248)$. The large number of local optimizers makes extremely difficult for any method to locate the global minimizer. In Fig. 1.3, the original plot of the *Levy No. 5* into the cube $[-2, 2]^2$ is shown.

After applying the transformation of Eq. 1.21 (first stage of function "stretching") to the *Levy No. 5*, the new form of the function is shown in Fig. 1.4 (left). As one can see, local minima with higher functional values than the "stretched" local minimum disappeared, while lower minima as well as the global one have been left unaffected. In Fig. 1.4 (right), the final landscape, derived after applying the second transformation stage to the *Levy No. 5*, is presented. It is clearly shown how the whole neighborhood of the local minimum has been elevated; thus, the former local

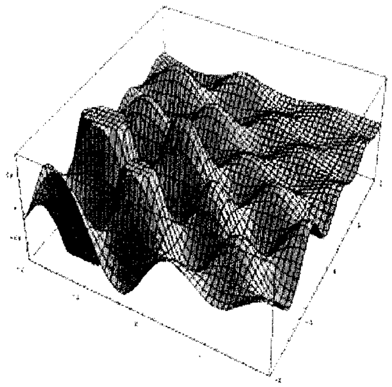

*Figure 1.3.* The original plot of the function *Levy No. 5.*

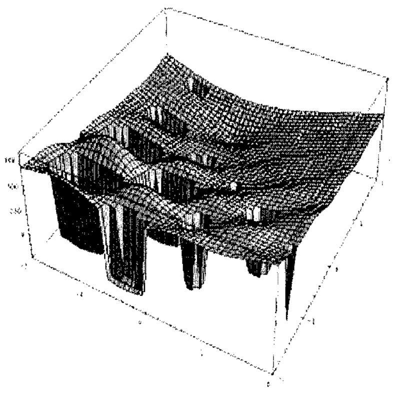
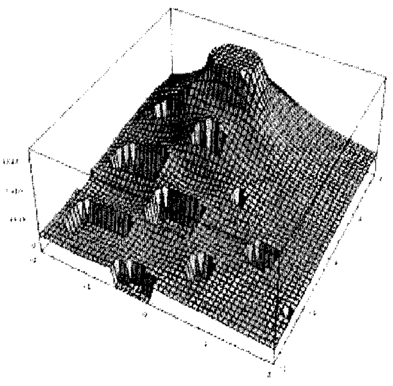

*Figure 1.4.* Plot of the *Levy No. 5* after the first stage (left) and after the second stage (right) of the function "stretching" technique.

minimum has now turned to be a local maximum of the function. Details on the performance of the PSO algorithm combined with the function "stretching" technique (SPSO) on two well known test problems, as well as suggestions for selecting parameter values, are presented in the next section.

## 4. Experiments and discussion

Experiments have been performed to evaluate the learning methods mentioned in the previous sections and compare their performance. Below, we exhibit results on two notorious for their local minima problems. The algorithms have been tested using initial weights chosen from the uniform distribution in the interval $(-1, 1)$. Note that BPSA and BPD update the weights using BP until convergence to a global or local minimum is obtained: the weight vector $w^k$ is considered as a global minimizer when $E(w^k) \leqslant 0.04$. Convergence to a local minimizer is related to the magnitude of the gradient vector, i.e. when the stopping condition $\|\nabla E(w^k)\| \leqslant 10^{-3}$ is met, $w^k$ is taken as a local minimizer $\bar{w}_i$ of the error function $E$.

No effort has been made to tune the mutation and crossover parameters, $\xi$ and $\rho$ respectively. We have used the fixed values $\xi = 0.5$ and $\rho = 0.7$, instead. The weight population size *NP* has been chosen to be twice the dimension of the problem, i.e. *NP*$= 2N$, for all the simulations considered. Some experimental results have shown that a good choice for *NP* is $2N \leqslant NP \leqslant 4N$. It is obvious that the exploitation of the weight space is more effective for large values of *NP*, but sometimes more error function evaluations are required. On the other hand, small values of *NP* make the algorithm inefficient and more generations are required in order to converge to the minimum.

In all the PSO simulations reported, the values of $\gamma_1, \gamma_2$ and $\mu$ were fixed: $\gamma_1 = 10000, \gamma_2 = 1$ and $\mu = 10^{-10}$. The balance between the global and local exploration abilities of the SPSO is mainly controlled by the inertia weights, since the particles' positions are updated according to the classical PSO strategy. A time decreasing inertia weight value, i.e. start from 1 and gradually decrease towards 0.4, has been found to work better than using a constant value. This is because large inertia weights help to find good seeds at the beginning of the search, while, later, small inertia weights facilitate a finer search.

Notice that for the BP, BPM, BBP, SA, BPSA and BPD methods each iteration corresponds to one gradient and one error function evaluation, differently from the BPVS, NMBPM, NMBBP and NMBPVS where, in general, the number of error Function Evaluations (FE) is larger than the number of Gradient Evaluations (GE), due to the use of the line search. In the table below, there are two rows for these algorithms; the first one indicates the statistics for the FE and the second for the GE. On the other hand, a key feature of GA, DE, PSO and SPSO algorithms is that *only* error function values are needed.

*1) The XOR classification problem:* classification of the four XOR patterns in one of two classes, $\{0, 1\}$, using a 2–2–1 ANN is a classical test problem [50, 54]. The XOR problem is sensitive to initial weights and presents a multitude of local minima [7]. The stepsize is taken equal to 1.5 and the heuristics for SA, BPSA and PSO are tuned to $n = 0.3$, $d = 0.002$ and $c_1 = c_2 = 0.5$. In all instances, 100 simulations have been run and the results are summarized in Table 1.1.

*2) The three bit parity problem* [50]*:* a 3–3–1 ANN receives eight, 3–dimensional binary input patterns and must output an "1" if the inputs have an odd number of ones and "0" if the inputs have an even number of ones. This is a very difficult problem for an ANN because the network must determine the proper parity (the value at the output) for input patterns which differ only by Hamming distance 1. It is well known that the network's weight space contains "bad" local minima. The stepsize has been taken equal to 0.5 and the heuristics for SA, BPSA and PSO have been tuned to $n = 0.1$, $d = 0.00025$, $c_1 = 0.1$ and $c_2 = 1$. In all instances, the results of 100 simulations are summarized in Table 1.1.

The results suggest that combination of local and global search methods like BPSA and BPD provide a better probability of success than the BP. Note that the performance of BPSA is not the appropriate one although derivative related information has been used. On the other hand, BPD escapes local minima and converges to the global minimum in all cases. A consideration that is worth mentioning is that the number of function evaluations in BPSA and BPD contains the additional evaluations required for BP to satisfy the local minima stopping condition. The results also indicate that the GA and the DE are promising and effective, even when compared with other methods that require the gradient of the error function, in addition to the error function values. For example, GAs as well as $DE_3$ and $DE_4$ have exhibited very good performance for the test problems considered. On the other hand, there have been cases where a discrepancy has been found in DE's behavior; see for example $DE_5$ and $DE_6$. For a discussion on the generalization capabilities of the networks generated by the DE algorithms see [43, 45]. Finally, the PSO algorithm combined with the function "stretching" technique (SPSO) has exhibited improved success rate, although it needed additional iterations to converge.

In conclusion, global search methods provide techniques that alleviate the problem of occasional convergence to local minima in neural network learning. Escaping from local minima is not always possible, however these methods exhibit a better chance in locating appropriate solutions and, in that sense, they improve the efficiency of the learning process. Experiments indicate that learning algorithms equipped with the proposed error function transformation techniques are capable to escape from undesired local minima and locate a desired one effectively. The deflection procedure and the function "stretching" technique provide stable convergence and thus a better probability of success for a learning algorithm. In general, the results exhibited by the proposed methods on two notorious for their local minima problems are promising.

| Training Method | | *XOR Problem* | | | *Parity Problem* | | |
|---|---|---|---|---|---|---|---|
| | | Mean | s.d. | Succ. | Mean | s.d. | Succ. |
| BP | | 144.1 | 112.6 | 42% | 932.0 | 1320.8 | 91% |
| BPM | | 249.7 | 322.1 | 49% | 219.9 | 198.9 | 93% |
| BBP | | 93.3 | 201.5 | 71% | 150.3 | 137.3 | 94% |
| NMBPM | (FE) | 260.4 | 287.8 | 68% | 244.3 | 205.9 | 99% |
| | (GE) | 254.4 | 287.3 | | 235.1 | 204.4 | |
| NMBBP | (FE) | 191.6 | 328.9 | 80% | 106.6 | 123.1 | 99% |
| | (GE) | 102.1 | 173.4 | | 99.2 | 164.5 | |
| BPVS | (FE) | 199.1 | 373.1 | 78% | 105.8 | 186.9 | 98% |
| | (GE) | 185.2 | 343.3 | | 100.4 | 171.6 | |
| NMBPVS | (FE) | 208.4 | 395.2 | 80% | 102.1 | 109.9 | 99% |
| | (GE) | 201.3 | 378.8 | | 95.3 | 183.5 | |
| SA | | 424.2 | 420.8 | 43% | 805.4 | 2103.1 | 22% |
| BPSA | | 1661.9 | 2775.7 | 65% | 2634.0 | 6866.8 | 66% |
| GA | | 422.3 | 397.5 | 95% | 1091.5 | 766.2 | 73% |
| $DE_1$ | | 192.9 | 124.7 | 75% | 622.6 | 522.1 | 91% |
| $DE_2$ | | 284.9 | 216.2 | 80% | 1994.1 | 657.6 | 61% |
| $DE_3$ | | 583.9 | 256.3 | 97% | 896.3 | 450.6 | 99% |
| $DE_4$ | | 706.1 | 343.7 | 98% | 1060.2 | 716.6 | 98% |
| $DE_5$ | | 300.5 | 250.2 | 85% | 2112.0 | 644.9 | 26% |
| $DE_6$ | | 482.9 | 264.9 | 93% | 2062.5 | 794.8 | 44% |
| PSO | | 1459.7 | 1143.1 | 77% | 6422.4 | 2992.1 | 42% |
| SPSO | | 7869.6 | 13905.4 | 100% | 9803.6 | 5436.6 | 95% |
| BPD | | 575.1 | 387.3 | 100% | 760.0 | 696.4 | 100% |

*Table 1.1.* Comparative results

## Acknowledgments

The authors would like to thank the European Social Fund, Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the Program PYTHAGORAS for funding the above work. Dr V.P. Plagianakos and Prof. M.N. Vrahatis also acknowledge the financial support of the University of Patras Research Committee through a "Karatheodoris" research grant.

## References

[1] N. Ampazis and S.J. Perantonis, (2002). Two Highly Efficient Second Order Algorithms for Training Feedforward Networks, *IEEE Transactions on Neural Networks*, **13**, 1064–1074.

[2] L. Armijo, (1966). Minimization of Functions Having Lipschitz–continuous First Partial Derivatives, *Pacific Journal of Mathematics*, **16**, 1–3.

[3] J. Barzilai and J.M. Borwein, (1988). Two Point Step Size Gradient Methods, *IMA Journal of Numerical Analysis*, **8**, 141–148.

[4] R. Battiti, (1989). Accelerated Backpropagation Learning: Two Optimization Methods, *Complex Systems*, **3**, 331–342.

[5] R. Battiti, (1992). First– and Second–order Methods for Learning: Between Steepest Descent and Newton's Method, *Neural Computation*, **4**, 141–166.

[6] D.P. Bertsekas, (1995). Nonlinear Programming, Belmont, MA: Athena Scientific.

[7] E.K. Blum, (1989). Approximation of Boolean Functions by Sigmoidal Networks: Part I: XOR and Other Two Variable Functions, *Neural Computation*, **1**, 532–540.

[8] M. Burton Jr. and G.J. Mpitsos, (1992). Event Dependent Control of Noise Enhances Learning in Neural Networks, *Neural Networks*, **5**, 627–637.

[9] L.W. Chan and F. Fallside, (1987). An Adaptive Training Algorithm for Back–propagation Networks, *Computers Speech and Language*, **2**, 205–218.

[10] A. Corana, M. Marchesi, C. Martini, and S. Ridella, (1987). Minimizing Multimodal Functions of Continuous Variables with the Simulated Annealing Algorithm, *ACM Transactions on Mathematical Software*, **13**, 262–280.

[11] J.E. Dennis and R.B. Schnabel, (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, Prentice–Hall.

[12] R.C. Eberhart and Y.H. Shi, (1998). Evolving Artificial Neural Networks, *Proceedings International Conference on Neural Networks and Brain*, Beijing, P.R. China.

[13] R.C. Eberhart, P.K. Simpson and R.W. Dobbins (1996). *Computational Intelligence PC Tools*, Academic Press Professional, Boston, MA.

[14] S.E. Fahlman (1988). Faster–learning Variations on Back–propagation: An Empirical Study, D.S. Touretzky, G.E. Hinton and T.J. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School*, 38–51, San Mateo, Morgan Koufmann.

[15] A.V. Fiacco and G.P. McCormick (1990). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, Philadelphia, SIAM.

[16] M. Gori and A. Tesi, (1992). On the Problem of Local Minima in Backpropagation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**, 76–85.

[17] L. Grippo, F. Lampariello, and S. Lucidi, (1986). A Nonmonotone Line Search Technique for Newton's Method, *SIAM Journal on Numerical Analysis*, **23**, 707–716.

[18] M.T. Hagan and M. Menhaj, (1994). Training Feedforward Networks with the Marquardt Algorithm, *IEEE Transactions on Neural Networks*, **5**, 989–993.

[19] J.H. Holland, (1975). *Adaptation in Neural and Artificial Systems*, University of Michigan Press.

[20] C. Houck, J. Joines, and M. Kay, (1995). *A Genetic Algorithm for Function Optimization: A Matlab Implementation*, NCSU–IE TR, 95–09.

[21] R.A. Jacobs, (1988). Increased Rates of Convergence Through Learning Rate Adaptation, *Neural Networks*, **1**, 295–307.

[22] J. Kennedy and R.C. Eberhart, (1995). Particle Swarm Optimization, *Proceedings IEEE International Conference on Neural Networks*, Piscataway, NJ, IV:1942–1948.

[23] J. Kennedy and R.C. Eberhart, (2001). *Swarm Intelligence*, Morgan Kaufmann Publishers.

[24] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, (1983). Optimization by Simulated Annealing, *Science*, **220**, 671–680.

[25] S. Kollias and D. Anastassiou, (1989). An Adaptive Least Squares Algorithm for the Efficient Training of Multilayered Networks, *IEEE Transactions on Circuits Systems*, **36**, 1092–1101.

[26] Y. Lee, S.H. Oh, and M. Kim, (1993). An Analysis of Premature Saturation in Backpropagation Learning, *Neural Networks*, **6**, 719–728.

[27] G.D. Magoulas, V.P. Plagianakos, and M.N. Vrahatis, (2002). Globally Convergent Algorithms with Local Learning Rates, *IEEE Transactions Neural Networks*, **13**, 774–779.

[28] G.D. Magoulas, V.P. Plagianakos, and M.N. Vrahatis, (2004). Neural Network-based Colonoscopic Diagnosis Using On-line Learning and Differential Evolution, *Applied Soft Computing*, **4**, 369–379.

[29] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis, (1997). Effective Back–propagation with Variable Stepsize, *Neural Networks*, **10**, 69–82.

[30] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis, (1997). On the Alleviation of Local Minima in Backpropagation, *Nonlinear Analysis, Theory, Methods and Applications*, **30**, 4545–4550.

[31] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis, (1999). Improving the Convergence of the Back–propagation Algorithm Using Learning Rate Adaptation Methods, *Neural Computation*, **11**, 1769–1796.

[32] G.D. Magoulas, M.N. Vrahatis, T.N. Grapsa, and G.S. Androulakis,(1997). *Neural Network Supervised Training Based on a Dimension Reducing Method, Mathematics of Neural Networks, Models, Algorithms and Applications*, S.W. Ellacott, J.C. Mason, I.J. Anderson Eds., Kluwer Academic Publishers, Boston, 245–249.

[33] Z. Michalewicz, (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer.

[34] M.F. Möller, (1993). A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, *Neural Networks*, **6**, 525–533.

[35] J. Nocedal, (1992). Theory of Algorithms for Unconstrained Optimization, *Acta Numerica*, **1**, 199–242.

[36] J.M. Ortega and W.C. Rheinboldt, (1970). *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York.

[37] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas and M.N. Vrahatis, (2001). Objective Function "Stretching" to Alleviate Convergence to Local Minima, *Nonlinear Analysis, Theory, Methods and Applications*, **47**, 3419–3424.

[38] K.E. Parsopoulos and M.N. Vrahatis, (2002). Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization, *Natural Computing*, **1**, 235–306.

[39] K.E. Parsopoulos and M.N. Vrahatis, (2004). On the Computation of All Global Minimizers Through Particle Swarm Optimization, *IEEE Transactions on Evolutionary Computation*, **8**, 211–224.

[40] N.G. Pavlidis, K.E. Parsopoulos and M.N. Vrahatis, (2004). Computing Nash Equilibria Through Computational Intelligence Methods, *Journal of Computational and Applied Mathematics*, in press.

[41] V.P. Plagianakos, G.D. Magoulas and M.N. Vrahatis, (2002). Deterministic Nonmonotone Strategies for Effective Training of Multi–Layer Perceptrons, *IEEE Transactions on Neural Networks*, **13**, 1268–1284.

[42] V.P. Plagianakos, D.G. Sotiropoulos, and M.N. Vrahatis, (1998). Automatic Adaptation of Learning Rate for Backpropagation Neural Networks, N.E. Mastorakis, (Ed.), *Recent Advances in Circuits and Systems* 337–341, Singapore, World Scientific.

[43] V.P. Plagianakos and M.N. Vrahatis, (1999). Neural Network Training with Constrained Integer Weights, *Proceedings of Congress on Evolutionary Computation (CEC'99)*, 2007–2013, Washington D.C.

[44] V.P. Plagianakos and M.N. Vrahatis, (2000). Training Neural Networks with Threshold Activation Functions and Constrained Integer Weights, *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'2000)*, Vol. **5**, pp.161–166, Como, Italy.

[45] V.P. Plagianakos and M.N. Vrahatis, (2002). Parallel Evolutionary Training Algorithms for "Hardware–Friendly" Neural Networks, *Natural Computing*, **1**, 307–322.

[46] V.P. Plagianakos, M.N. Vrahatis, and G.D. Magoulas (1999). Nonmonotone Methods for Backpropagation Training with Adaptive Learning Rate, *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'99)*, Vol. **3**, pp.1762–1767, Washington D.C.

[47] E. Polak, (1997). *Optimization: Algorithms and Consistent Approximations*, New York, Springer–Verlag.

[48] M. Raydan, (1997). The Barzilai and Borwein Gradient Method for the Large Scale Unconstrained Minimization Problem, *SIAM Journal on Optimization*, **7**, 26–33.

[49] A.K. Rigler, J.M. Irvine, and T.P. Vogl, (1991). Rescaling of Variables in Backpropagation Learning, *Neural Networks*, **4**, 225–229.

[50] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, (1986). Learning Internal Representations by Error Propagation, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1*, D.E. Rumelhart, J.L. McClelland Eds., MIT Press, 318–362.

[51] S. Saarinen, R. Bramley, and G. Cybenko, (1993). Ill-conditioning in Neural Network Training Problems, *SIAM Journal on Scientific Computing*, **14**, 693–714.

[52] F. Silva and L. Almeida, (1990). Acceleration Techniques for the Back–propagation Algorithm, *Lecture Notes in Computer Science*, **412**, 110–119, Berlin, Springer–Verlag.

[53] R. Storn and K. Price, (1997). Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization*, **11**, 341–359.

[54] P.P. Van der Smagt, (1994). Minimisation Methods for Training Feedforward Neural Networks, *Neural Networks*, **7**, 1–11.

[55] T.P. Vogl, J.K. Mangis, A.K. Rigler, W.T. Zink, and D.L. Alkon, (1988). Accelerating the Convergence of the Back–propagation Method, *Biological Cybernetics*, **59**, 257–263.

[56] M.N. Vrahatis, G.S. Androulakis, J.N. Lambrinos and G.D. Magoulas, (2000). A Class of Gradient Unconstrained Minimization Algorithms with Adaptive Stepsize, *Journal of Computational and Applied Mathematics*, **114**, 367–386.

[57] M.N. Vrahatis, G.D. Magoulas and V.P. Plagianakos, (2000). Globally Convergent Modification of the Quickprop Method, *Neural Processing Letters*, **12**, 159–169.

[58] M.N. Vrahatis, G.D. Magoulas and V.P. Plagianakos, (2003). From Linear to Nonlinear Iterative Methods, *Applied Numerical Mathematics*, **45**, 59–77.

[59] S.T. Weslstead, (1994). *Neural Network and Fuzzy Logic Applications in C/C++*, Wiley.

[60] X.-H. Yu, G.-A. Chen, (1995). On the Local Minima Free Condition of Backpropagation Learning, *IEEE Transactions on Neural Networks*, **6**, 1300–1303.