

Chapter 1

SUPERVISED TRAINING USING GLOBAL SEARCH METHODS

V.P. Plagianakos

*Department of Mathematics, University of Patras,
GR-26110, Patras, Greece.*
vpp@math.upatras.gr

G.D. Magoulas

*Department of Information Systems and Computing, Brunel University,
Uxbridge, UB8 3PH, United Kingdom.*
George.Magoulas@brunel.ac.uk

M.N. Vrahatis

*Department of Mathematics, University of Patras,
GR-26110, Patras, Greece.*
vrahatis@math.upatras.gr

Abstract Supervised learning in neural networks based on the popular backpropagation method can be often trapped in a local minimum of the error function. The class of backpropagation-type training algorithms includes local minimization methods that have no mechanism that allows them to escape the influence of a local minimum. 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. This work investigates the use of global search methods for batch-mode training of feedforward multilayer perceptrons. Global search methods are expected to lead to “optimal” or “near-optimal” weight configurations by allowing the neural network to escape local minima during training and, in that sense, they improve the efficiency of the learning process. The paper reviews the fundamentals of simulated annealing, genetic and evolutionary algorithms as well as some recently proposed deflection procedures. Simulations and comparisons are presented.

Keywords: Local minima, simulated annealing, genetic algorithms, evolutionary algorithms, deflection procedure, feedforward neural networks, supervised training.

Introduction

In neural network training the objective is usually to minimize a cost function defined as the multi-variable error function of the network. More specifically, supervised training of a feed-forward neural network (FNN) can be viewed as the minimization of an error function that depends on the weights of the network [9]. This perspective gives some advantage to the development of effective training algorithms, because the problem of minimizing a function is well known in the field of numerical analysis.

It is well known that the supervised training using a Backpropagation (BP) based learning rule can be trapped in a local minimum of the error function [17]. 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 [4]. The insufficient number of hidden nodes as well as improper initial weight settings can cause convergence to a local minimum, which prevents the network from learning the entire training set and results in inferior network performance. Gradient-based backpropagation training algorithms are local minimization methods and have no mechanism that allows them to escape the influence of a local minimum.

Recently, 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 [4, 8, 22]. However, conditions such as the linear separability of the patterns and the pyramidal structure of the FNN [4] 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.

This paper presents techniques that 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. The remaining of this paper is organized as follows. In section 1 the training problem is formulated. In section 2 a recently proposed deflection procedure [10] is briefly presented. The fundamentals of simulated annealing [2, 3, 7] are presented in section 3, while genetic and evolutionary algorithms [12] are reviewed in section 4 and 5 respectively. Section 6 summarizes and discusses our results as well as presents simulations and comparisons with the standard backpropagation algorithm.

1. PROBLEM FORMULATION

Let us consider an FNN whose l -th layer contains N_l neurons, for $l = 1, \dots, L$. The network operation is based on the following equations:

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

where net_j^l is for the j -th neuron in the l -th layer ($j = 1, \dots, N_l$), 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:

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

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 σ is a nonlinear activation function, such as the well known sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$. The weights in the network can be expressed using vector notation as:

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

where θ_j^l denotes the bias of the j -th neuron ($j = 1, \dots, N_l$) at the l -th layer ($l = 2, \dots, L$). 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 training algorithm. The weight update vector describes a direction in which the weight vector will move in order to reduce the network training error. The commonly used training methods are gradient-based algorithms such as the popular Backpropagation algorithm [17]. The BP procedure minimizes the error function $E(w)$ using the steepest descent with constant stepsize μ :

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

The optimal value of the stepsize μ depends on the shape of the N -dimensional error function. The gradient, ∇E , is computed by applying the chain rule on the layers of the FNN (see [17]). It is well known that the BP algorithm leads to slow training and often yields suboptimal solutions [4].

Attempts to speed up back-propagation training have been made by dynamically adapting the stepsize μ during training [9, 20], or by using second derivative related information [11, 13, 19]. However, these BP-like training algorithms occasionally converge to local minima which affect the efficiency of the learning process.

2. THE DEFLECTION PROCEDURE

It is well known that in order to minimize the error function E we require a sequence of weight vectors $\{w^k\}_0^\infty$, where k indicates iterations, that converges to a minimizer of E . Following the *deflection procedure* proposed in [10], when this sequence converges to a local minimum $r \in \mathbb{R}^N$ the following function is formulated:

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

where $S(w; r, \lambda)$ is a function depending on a weight vector w and on the local minimizer r of E ; λ is a relaxation parameter. In case there exist m local minima $r_1, \dots, r_m \in \mathbb{R}^N$, the above relation is reformulated as:

$$F(w) = S(w; r_1, \lambda_1)^{-1} \cdots S(w; r_m, \lambda_m)^{-1} E(w).$$

The deflection procedure suggests to find a “proper” $S(\cdot)$ such that $F(w)$ will not have a minimum at $r_i, i = 1, \dots, m$, 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 r_i (a local minimizer of E) will not produce a minimum of F at $w = r_i$. In addition, this function F will retain all other minima of E . This is *the deflection property* [10], and, as it will be explained below, the function:

$$S(w; r_i, \lambda_i) = \tanh(\lambda_i \|w - r_i\|),$$

provides F with this property.

Let us assume that a local minimum r_i has been determined, then

$$\lim_{w \rightarrow r_i} \frac{E(w)}{\tanh(\lambda \|w - r_i\|)} = +\infty,$$

which means that r_i is no longer a local minimizer of F . Moreover, it is easily verified that for $\|w - r_i\| \geq \varepsilon$, where ε is a small positive constant,

it holds that:

$$\lim_{\lambda \rightarrow +\infty} F(w) = \lim_{\lambda \rightarrow +\infty} \frac{E(w)}{\tanh(\lambda \|w - r_i\|)} = E(w), \quad (1.1)$$

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 λ . For an arbitrary value of λ there is a small neighborhood $\mathcal{R}(r, \rho)$ with center r and radius ρ , with $\rho \propto \lambda^{-1}$, that for any $x \in \mathcal{R}(r, \rho)$ it holds that $F(x) > E(x)$. To be more specific, when the value of λ is small (say $\lambda < 1$) the denominator in the above relation becomes one for w “far” from r . Thus, the deflection procedure affects a large neighborhood around r in the weight space. On the other hand, when the value of λ is large, new local minima is possible to be created near the computed minimum r , like a Mexican hat. These minima have function values greater than $F(r)$ and can be easily avoided by taking a proper stepsize or by changing the value of λ . To better visualize the effect of the deflection procedure, we provide

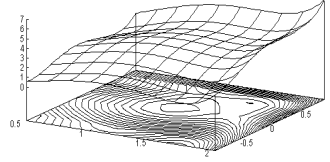


Figure 1.1 The original Six Hump Camel Back test function

an application example on the well-known Six Hump Camel Back test function:

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

The plot of the original function is shown in Fig. 1.1. The application of the deflection procedure for $\lambda = 1.5$ is illustrated in Fig. 1.2 (left) and for $\lambda = 10$ in Fig. 1.2 (right); note the Mexican hat that is generated.

Notice that the deflection procedure can be incorporated in any training 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).

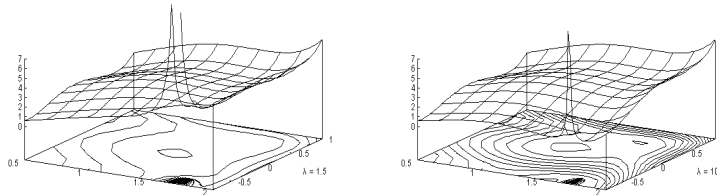


Figure 1.2 Applying the deflection procedure to the Six Hump Camel Back test function for $\lambda = 1.5$ (left) and $\lambda = 10$ (right)

3. THE SIMULATED ANNEALING

Simulated Annealing (SA) [7] 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. In the numerical optimization framework, SA is a procedure that has the capability to move out of regions near local minima. SA is based on random evaluations of the cost 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, the method is able to discriminate between “gross behavior” of the function and finer “wrinkles”. First, it reaches an area in the function domain where a global minimum 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 minimum, if not the global minimum itself.

The performance of the SA [3], as observed on typical neural network training problems, is not the appropriate one. SA is characterized by the need for a number of function evaluations greater than that commonly required for a single run of common training algorithms and by the absence of any derivative related information. In addition, 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 [21].

In [2], it has been suggested to incorporate SA in the weight update rule:

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

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

our experiments we have applied this technique for updating the weights from the beginning of the training as proposed by Burton *et al.* [2]. Alternatively, we update the weights using BP until convergence to a global or local minimum is achieved. In the latter case, we switch to SA. This combined BP with SA is named BPSA.

4. 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 [5]. GAs have been used primarily in optimization and machine learning problems.

Briefly, 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. Another operator associated with each of these operators is the *selection* operator, which produces survival of the fittest in the GA. 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. 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)” [6] has been used for the experiments reported in this paper. GAOT’s default crossover and mutation schemes, and a real-valued encoding of the FNN’s weights have been employed.

5. EVOLUTIONARY ALGORITHMS

Evolutionary algorithms (EA) are stochastic search methods which mimic the metaphor of natural biological evolution. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, 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.

To demonstrate the efficiency of the Evolutionary Algorithms (EA) in alleviating the local minima problem, we have used the *Differential Evolution* (DE) strategies [18]. This is because DEs can efficiently handle non differentiable, nonlinear and multimodal objective functions, and require few, easily chosen control parameters. Experimental results have shown that DEs have good convergence properties and outperform other evolutionary algorithms [14, 15, 16]. To apply DEs to neural network training we start with a specific number (NP) of N -dimensional weight vectors, as an initial weight population, and evolve them over time. NP is fixed throughout the training process. The weight vectors population is initialized randomly following a uniform probability distribution.

As in GAs, at each iteration, called *generation*, new weight vectors are generated by the combination of weight vectors randomly chosen from the population. This operation is called *mutation*. The outcoming weight vectors are then mixed with another predetermined weight vector, the *target* weight vector, and this operation is called *crossover*. This operation yields the so-called *trial* weight vector. The trial 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*.

We now briefly review the two basic DE operators used for FNN training. 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:

$$v_{g+1}^i = w_g^{r_1} + \xi (w_g^{r_1} - w_g^{r_2}) \quad (1.2)$$

$$v_{g+1}^i = w_g^{\text{best}} + \xi (w_g^{r_1} - w_g^{r_2}) \quad (1.3)$$

$$v_{g+1}^i = w_g^{r_1} + \xi (w_g^{r_2} - w_g^{r_3}) \quad (1.4)$$

$$v_{g+1}^i = w_g^i + \xi (w_g^{\text{best}} - w_g^i) + \xi (w_g^{r_1} - w_g^{r_2}) \quad (1.5)$$

$$v_{g+1}^i = w_g^{\text{best}} + \xi (w_g^{r_1} - w_g^{r_2}) + \xi (w_g^{r_3} - w_g^{r_4}) \quad (1.6)$$

$$v_{g+1}^i = w_g^{r_1} + \xi (w_g^{r_2} - w_g^{r_3}) + \xi (w_g^{r_4} - w_g^{r_5}) \quad (1.7)$$

where w_g^{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 (1.2) has been introduced as crossover operator for genetic algorithms [12] and is similar to relations (1.3) and (1.4). The remaining

relations are modifications which can be obtained by the combination of (1.2), (1.3) and (1.4). It is clear that many more relations of this type can be generated using the above ones as building blocks. In recent works [14, 15, 16], we have shown that the above relations can efficiently be used to train FNNs 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 ρ ; if $r \leq \rho$ 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 .

6. EXPERIMENTS AND DISCUSSION

Several experiments have been performed to evaluate the training methods mentioned in the previous sections and compare their performance. Below, we exhibit preliminary 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 $E(w^k) \leq 0.04$. Convergence to a local minimizer is related to the magnitude of the gradient vector, i.e. when the stopping condition $|\nabla E(w^k)| \leq 10^{-3}$ is met, w^k is taken as a local minimizer r_i of the error function E .

We call DE1 the algorithm that uses relation (1.2) as mutation operator, DE2 the algorithm that uses relation (1.3), and so on. We note here that a key feature of the GAs and the DE algorithms is that *only* error function values are needed. No gradient information is required, so there is no need of backward passes. We made no effort to tune the mutation and crossover parameters, ξ and ρ 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 $2N \leq NP \leq 4N$. 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.

1) *The XOR classification problem:* classification of the four XOR patterns in one of two classes, $\{0, 1\}$, using a 2–2–1 FNN is a classical test

problem [17, 19]. The XOR problem is sensitive to initial weights and presents a multitude of local minima [1]. The stepsize is taken equal to 1.5 and the heuristics for SA and BPSA are tuned to $n = 0.3$ and $d = 0.002$. In all instances, 100 simulations have been run and the results are summarized in Table 1.1.

2) *The three bit parity problem* [17]: a 3–3–1 FNN 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 FNN 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 and BPSA have been tuned to $n = 0.1$ and $d = 0.00025$. In all instances, 100 simulations have run and the results are summarized in Table 1.1.

Training Method	<i>XOR Problem</i>			<i>Parity Problem</i>		
	Succ.	Mean	s.d.	Succ.	Mean	s.d.
BP	42%	144.1	112.6	91%	932.0	1320.8
SA	43%	424.2	420.8	22%	805.4	2103.1
BPSA	65%	1661.9	2775.7	66%	2634.0	6866.8
GA	95%	422.3	397.5	73%	1091.5	766.2
DE1	75%	192.9	124.7	91%	622.6	522.1
DE2	80%	284.9	216.2	61%	1994.1	657.6
DE3	97%	583.9	256.3	99%	896.3	450.6
DE4	98%	706.1	343.7	98%	1060.2	716.6
DE5	85%	300.5	250.2	26%	2112.0	644.9
DE6	93%	482.9	264.9	44%	2062.5	794.8
BPD	100%	575.1	387.3	100%	760.0	696.4

Table 1.1 Comparative results

The results of Table 1.1 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 SA 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 DE3 and DE4 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 DE5 and DE6. For a discussion on the generalization capabilities of the networks generated by the DE algorithms see [14, 15].

In conclusion, global search methods provide techniques that alleviate the problem of occasional convergence to local minima in feedforward neural network training. 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. Preliminary results on two notorious for their local minima problems are promising.

References

- [1] E.K. Blum, "Approximation of Boolean functions by sigmoidal networks: Part I: XOR and other two variable functions", *Neural Computation*, vol.1, 1989, 532–540.
- [2] M. Burton Jr. and G.J. Mpitsos, "Event dependent control of noise enhances learning in neural networks", *Neural Networks*, vol.5, 1992, 627–637.
- [3] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the Simulated Annealing algorithm", *ACM Trans. Math. Soft.*, vol.13, 1987, 262–280.
- [4] M. Gori and A. Tesi, "On the problem of local minima in backpropagation", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.14, 1992, 76–85.
- [5] J.H. Holland, "Adaptation in Neural and Artificial Systems", University of Michigan Press, 1975.
- [6] C. Houck, J. Joines, and M. Kay, "A Genetic Algorithm for Function Optimization: A Matlab Implementation", NCSU-IE TR, 1995, 95–09.
- [7] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, "Optimization by simulated annealing", *Science*, vol.220, 1983, 671–680.
- [8] Y. Lee, S.H. Oh, and M. Kim, "An analysis of premature saturation in backpropagation learning", *Neural Networks*, vol.6, 1993, 719–728.

- [9] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis, “Effective backpropagation training with variable stepsize”, *Neural Networks*, vol.10, No.1, 1997, 69–82.
- [10] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis, “On the alleviation of local minima in backpropagation”, *Nonlinear Analysis, Theory, Methods and Applications*, vol.30, 1997, 4545–4550.
- [11] G.D. Magoulas, M.N. Vrahatis, T.N. Grapsa, and G.S. Androulakis, “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, 1997, 245–249.
- [12] Z. Michalewicz, “Genetic algorithms + data structures = evolution programs”, Springer, 1996.
- [13] M.F. Möller, “A scaled conjugate gradient algorithm for fast supervised learning”, *Neural Networks*, vol.6, 1993, 525–533.
- [14] V.P. Plagianakos and M.N. Vrahatis, “Training neural networks with 3-bit integer weights”, *Proceedings of Genetic and Evolutionary Computation Conference (GECCO’99)*, 1999, 910–915.
- [15] V.P. Plagianakos and M.N. Vrahatis, “Neural network training with constrained integer weights”, *Proceedings of Congress on Evolutionary Computation (CEC’99)*, 1999, 2007–2013.
- [16] V.P. Plagianakos and M.N. Vrahatis, “Training Neural Networks with Threshold Activation Functions and Constrained Integer Weights”, *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN’2000)*, (2000).
- [17] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, “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, 1986, 318–362.
- [18] R. Storn and K. Price, “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”, *Journal of Global Optimization*, vol.11, 1997, 341–359.
- [19] P.P. Van der Smagt, “Minimisation methods for training feedforward neural networks”, *Neural Networks*, vol.7, 1994, 1–11.
- [20] T.P. Vogl, J.K. Mangis, A.K. Rigler, W.T. Zink, and D.L. Alkon, “Accelerating the convergence of the back-propagation method”, *Biological Cybernetics*, vol.59, 1988, 257–263.
- [21] S.T. Weslstead, “Neural network and fuzzy logic applications in C/C++”, Wiley, 1994.
- [22] X.-H. Yu, G.-A. Chen, “On the local minima free condition of back-propagation learning”, *IEEE Trans. Neural Networks*, vol.6, 1995, 1300–1303.