# Global Optimisation of Neural Networks Using a Deterministic Hybrid Approach

**Gleb Beliakov and Ajith Abraham***

School of Computing and Mathematics, Deakin University, 662 Balckburn Road, Clayton, Melbourne, Vic. 3168, Australia. Email: gleb@deakin.edu.au

* School of Computing and Information Technology, Monash University, Gippsland Campus, Churchill 3842, Australia, Email: ajith.abraham@ieee.org

***Abstract***

*Selection of the topology of a neural network and correct parameters for the learning algorithm is a tedious task for designing an optimal artificial neural network, which is smaller, faster and with a better generalization performance. In this paper we introduce a recently developed cutting angle method (a deterministic technique) for global optimization of connection weights. Neural networks are initially trained using the cutting angle method and later the learning is fine-tuned (meta-learning) using conventional gradient descent or other optimization techniques. Experiments were carried out on three time series benchmarks and a comparison was done using evolutionary neural networks. Our preliminary experimentation results show that the proposed deterministic approach could provide near optimal results much faster than the evolutionary approach.*

## 1. Introduction

Artificial neural networks are capable of performing many classification, learning and function approximation tasks, yet in practice sometimes they deliver only marginal performance. Inappropriate topology selection and weight training are frequently blamed. Increasing the number of hidden layer neurons helps improving network performance, yet many problems could be solved with very few neurons if only the network took its optimal configuration. Unfortunately, the inherent non-linearity of ANN results in the existence of many sub-optimal networks, and the great majority of training algorithms converge to these sub-optimal configurations.
The problem of multiple local minima in neural networks has been widely addressed [8, 12, 13, 14, 17, 22, 25, 29, 28, 30, 31, 32, 36, 35]. Proposed solutions include multiple start from randomly chosen initial points, simulated annealing, random perturbation, diffusion techniques and evolutionary computing [14, 25, 29, 28, 30, 31, 36, 35]. The majority of these methods are probabilistic in nature: they can find the globally optimal solution with a certain probability, which depends on

the number of iterations of the algorithm. In contrast, deterministic techniques allow one to find guaranteed optimal configuration. The price for this guarantee is enormous computational cost. The fact that the non-linear optimisation problem is NP-hard makes the possibility of designing a quick reliable technique very unlikely. Deterministic methods include tabu search, branch-and-bound, generalised cutting plane and systematic search [26, 33].

In this paper we investigate a recently developed cutting angle method of deterministic global optimisation [3, 4, 27] applied to optimising neural networks. The cutting angle method (CAM) is based on the theory of abstract convexity [27] and it arises as a particular case of generalised cutting plane method in Lipschitz programming. It has been successfully applied to solving other problems with many local minima (free-knot spline approximation, molecular conformations, clustering and classification [6, 7]), and the design of a relatively fast computational algorithm [5] makes this technique practical. In section 2, we present the multiple minima problem related to optimization of weights. The importance of global optimization of weights and the proposed cutting angel method and evolutionary learning method is presented in Section 3 and 4. Evolutionary design of neural networks is presented in Section 5. Experimentation data, set up and results are presented in Section 6 and 7 and some conclusions are provided towards the end.

## 2. Multiple Minima Problem

If we consider a network with differentiable activation functions, then the activation functions of the output units become differentiable functions of both the input variables and of the weights and biases. If we define an error function ($E$), such as sum of squares function, which is a differentiable function of the network outputs, then this error function is itself a differentiable function of the weights. We can therefore evaluate the derivatives of the error with respect to the weights, and these derivatives can then be used to find weight values, which minimize the error function, by using any of the learning algorithms like backpropagation (BP), conjugate gradient, quazi-Newton and Levenberg-Marquardt (LM) approach [9]. Viewed from mathematical programming perspective [22, 35], supervised batch training of a neural network is a classical non-linear optimisation problem: find the minimum of the error function given some set of training data. Traditionally this is accomplished by a suitable local descent technique, such as backpropagation. The independent variables are the weights $\mathbf{w}$, and the objective function is usually the sum of squared errors (although other measures of error are also used). It is formulated mathematically as

$$\min_{\mathbf{w}} E(\mathbf{w}_o, \mathbf{w}_h) = \sum_{k=1}^{K} \left( f\left( {\mathbf{w}_o}^T \mathbf{z}_k \right) - y_k \right)^2 \text{, where } \mathbf{z}_k = f\left( {\mathbf{w}_h}^T \mathbf{x}_k \right) \qquad (1)$$

Here $f$ denotes the transfer function, $\mathbf{w}_o$ denote output weights, $\mathbf{w}_h$ denote hidden layer weights, $x_k$ are input training data, $y_k$ is the desired output and $z_k$ denote activations of hidden neurons. Despite its popularity, backpropagation has been widely criticised for its inefficiency [24, 23], and more advanced minimisation

techniques, such as conjugate gradient and Levenberg-Marquardt methods are available [24]. Yet all these techniques converge to the closest local minimum of the error function, which is very unlikely to be the global one. As a consequence, the network trained with a local algorithm may exhibit marginal performance. In this connection, the primitive backpropagation may result in a better solution than more sophisticated methods, because its disadvantages turn to the benefits of avoiding some shallow local minima [24]. The problem of many local minima has been widely addressed in the past [14, 35]. It was shown that training even a simple perceptron with non-linear transfer function may result in multiple minima [13]. The remedies include starting local descent from several random points, using tabu search, simulated annealing and genetic algorithms. The new stochastic optimisation algorithms significantly outperform the local methods, yet they do not provide any guarantee that their solution is the global minimum indeed. What is more, the number of local minima of the error function grows exponentially with the number of neurons, and the likelihood that these stochastic methods will find the global minimum is not that high.

Deterministic global optimisation techniques exist [19, 26, 33]. They are based on more or less systematic exploration of the search space, and involve some assumptions about the class of the error function, such as Lipschitz properties. With a suitable choice of neuron transfer functions, these properties are satisfied. The biggest problem of deterministic techniques is their computational complexity, which grows exponentially with the number of variables (weights). Hence they are applicable only to small dimensional problems. The cutting angle method, recently developed in [4, 3, 27], is no exception to this rule: its computational complexity grows very rapidly with the number of variables. It is therefore imperative to simplify the optimisation problem in order to reduce the size of the search space. On the other hand, it is in the systems with few neurons where global optimisation techniques are most needed. Indeed one of the goals of using global optimisation in ANN training is to reduce the number of neurons without sacrificing the performance, and this has been achieved in many cases [14]. Yet some analysis of neural network structure will be performed in the next section with the purpose of further reducing the search space.

The combinations of neural networks and Evolutionary Computation (EC) procedures have been widely explored [2, 16, 34] . It covers a wide range of topics, such as weight training, architecture design, learning the learning rule, input feature selection, genetic reinforcement learning, initial weight selection etc. The shortcomings of the multiple minima could be overcomed by formulating the search process as the evolution of connection weights in the environment determined by the architecture and the learning task. The evolution of connection weights provides an alternative approach to training neural networks. Such an evolutionary approach consists of two major stages. The first stage, is to decide the genotype representation of connection weights, i.e., whether in the form of binary strings or not. The second one is the evolution itself driven by GAs or other evolutionary search procedures, in which genetic operators like crossover and

mutation have to be decided in conjunction with the representation scheme. Different representation schemes and genetic operators can lead to very different training performance. One of the major problems of evolutionary algorithm is their inefficiency in fine tuned local search although they are good at global search. The efficiency of evolutionary training can be improved significantly by incorporating a local search procedure into the evolution, i.e. combining GA's global search ability with local search's fine tuning ability. The evolutionary algorithm will be used to locate a good region in the space and then a local search procedure is used to find a near optimal solution in this region.

# 3. Global Optimisation of Weights

Consider a neural network with *N* input neurons, one output neuron and one hidden layer with *M* neurons. Let the transfer function of all hidden and the output neurons be the standard sigmoid

$$f(t) = \left(1 + e^{-t}\right)^{-1} \qquad (2)$$

We partition the weight space into hidden weights $\mathbf{w}_h$ and output weights $\mathbf{w}_o$, as has been suggested in the past [18, 23, 15, 36]. With respect to the output weights $\mathbf{w}_o$, with hidden weights $\mathbf{w}_h$ fixed, this is a problem of training a one-layer perceptron. Even though this problem may possess multiple local minima, it is still simpler that that of training all weights at once, and special techniques are available.

Firstly, by considering a different measure of error,

$$\tilde{E}(\mathbf{w}_o) = \sum_{k=1}^{K} \left(\mathbf{w}_o{}^T \mathbf{z}_k - f^{-1}(y_k)\right)^2 , \qquad (3)$$

we make the problem linear with respect to the output weights $\mathbf{w}_o$. Here the inverse of the transfer function is given by

$$f^{-1}(t) = \ln\left(\frac{t}{1-t}\right) \qquad (4)$$

This technique has been used previously in [18, 23]. It reduces the problem to solving a linear system of equations using QR or SVD factorisation. Its disadvantage is that it changes the measure of error, and the output weights found by this method are not necessarily the optimal with respect to the original error function (3) [11]. Even though some algorithms take the solution of this linear problem as the optimal output weights [18, 23], it would be beneficial to optimise the original measure of error (3) using a local descent technique, taking the solution of the linear problem as the starting point. The second technique proposed in [15] is to approximate the transfer function with its truncated Taylor series expansion. A local descent algorithm to minimize (3) with respect to $\mathbf{w}_o$ is given in [36]. The second observation about neural network weights is related to the hidden layer weights. It is usually assumed that the domain is $\mathbf{R}^m$, where m is the number of

connections between the input and the bias and the hidden layer. Of course any systematic exploration of this infinite domain will fail. Thus we usually restrict the domain to a hypercube in $\mathbf{R}^m$. In fact, the true domain is even smaller. Let us swap any two hidden layer neurons. The output of the network will not change (we assume the same transfer function for all hidden layer neurons). This means that there are several equivalent solutions, whose number is the number of permutations of hidden neurons. So in fact the problem has several global minima, and the error as a function of network weights possesses symmetry. It is not a problem for local search methods: it is not important to which of the equivalent minima the algorithm has converged. But for global search presence of equivalent global minima becomes an extra computational burden. The smaller the search space, the faster the algorithm converges. Taking into account that every m-dimensional hypercube contains m! simplices, the reduction in computing time is by the factor of m!. If we number the neurons in the order of increasing values of weights between any input neuron (or the bias) and the hidden layer, the domain in respect to these variables becomes a simplex. With respect to the other variables the domain does not change. So we can formulate the constrained minimisation problem as

$$\min_{\mathbf{w}} E(\mathbf{w}), \qquad (5)$$

subject to $-a \le w_1^0 \le w_2^0 \le \ldots \le w_M^0 \le a$,

$$-a \le w_m^n \le a, n = 1,\ldots,N, m = 1,\ldots,M,$$

where $E(\mathbf{w}) = \min_{w_O} E(\mathbf{w}_o, \mathbf{w}_h)$, and $E(\mathbf{w}_o,\mathbf{w}_h)$ is given in (4),

Here $w_m^n$ denote weights between the *n*-th input and *m*-th hidden neurons, and $w_m^0$ denote weights between bias and the *m*-th hidden neurons. Solution of the inner problem

$$\min_{w_O} E(\mathbf{w}_o, \mathbf{w}_h)$$

is performed separately using linear or local technique, as discussed above.

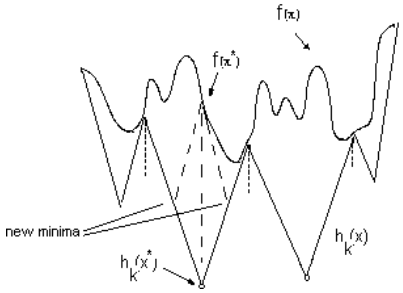


**Figure 1.** Illustration of the cutting angle method for univariate functions. The saw tooth cover of the objective function $f(x)$ is built using known function values.

# 4. Basics of the Cutting Angle Method

The cutting angle method is based on theoretical results in abstract convexity [27]. Without going much into details, we present some of its features. It systematically explores the whole domain by calculating the values of the objective function $f(\mathbf{x})$ at certain points. The points are selected in such a way that the algorithm does not return to unpromising regions where function values are high. The new point is chosen where the objective function can potentially take the lowest value. The function is assumed to be Lipschitz, and the value of the potential minim is calculated based on both the distance to the neigbouring points and function values at these points. This is illustrated on Fig. 1 for one-dimensional case.

This process can be seen as constructing the piecewise linear lower approximation of the objective function $f(\mathbf{x})$. With the addition of new points, the approximation $h_k(\mathbf{x})$ becomes closer to the objective function, and the global minimum of the approximating function $\mathbf{x}^*$ converges to the global minimum of the objective function. The lower approximation, the auxiliary function $h_k(\mathbf{x})$, is called the saw-tooth cover of $f$. The cutting angle method is formulated for optimisation of Lipschitz functions $f(\mathbf{x})$ in the interior of the $n$-dimensional unit simplex

$$S_n = \left\{ \mathbf{x} \in R_{++}^n : \sum_{i=1,\ldots,n} x_i = 1 \right\}.$$

$R_{++}$ denotes the set of strictly positive real numbers. If the domain of $f$ is a hypercube, it can be transformed to simplex with the change of variables

$$y_i = \frac{\ln x_{i+1}}{\ln x_i}, i = 1,2,\ldots, n,$$

where $x_i$ are variables in $S_n$, and $y_i$ are variables in $\mathbf{R}^{n-1}$.

We can formulate the cutting angle algorithm as follows.

*Step 0. Initialisation.*
Calculate the values of $f(\mathbf{x})$ at $n$ vertices of the unit simplex.
Set k=$n$
Build the auxiliary function $h_k(\mathbf{x})$ using all k known function values.

*Step 1.* Find all local minima of $h_k(\mathbf{x})$.
Choose the smallest local minimum $\mathbf{x}^*$ (the global minimum of $h_k(\mathbf{x})$)

*Step 2.* Calculate $f(\mathbf{x}^*)$.
Build $h_{k+1}(\mathbf{x})$ using $h_k(\mathbf{x})$ and the new point $\mathbf{x}^*$

*Step 3.* Set k=k+1

Go to Step 1.

The algorithm stops when the smallest computed value of $f(\mathbf{x})$ is sufficiently close to the global minimum of $h_k(\mathbf{x})$, or when the number of iterations reaches the maximum permitted value. Calculation of the local minima of the auxiliary function $h_k(\mathbf{x})$ is the most computationally expensive step. Instead of actually building this piecewise linear function, its minima can be calculated using combinatorial approach [4, 27], as combinations of $n$ out of $k$ vectors satisfying certain properties. An effective solution to this combinatorial problem was found in [5], where the actual computational algorithm is described. Still the number of function evaluations to solve this NP-hard problem with a given accuracy is very big. It was therefore proposed to use a combination of the CAM and local descent techniques. A specified number of iteration is performed using cutting angle, and then the best-known function value is improved by a local search method. This combination was shown to be effective in solving some difficult problems with many local minima. Any local descent method can be used, in particular backpropagation, conjugate gradient and Levenberg-Marquardt methods are all suitable. Hence we can use the following algorithm to train the neural networks.

*Step 1*. Perform K iterations of the cutting angle method to optimise the hidden layer weights, with the objective function E(**w**) given in (3)

*Step 2*. Starting from the best vector of weights from Step 1, find the optimal weights by using backpropagation, conjugate gradient or Levenberg-Marquardt methods.

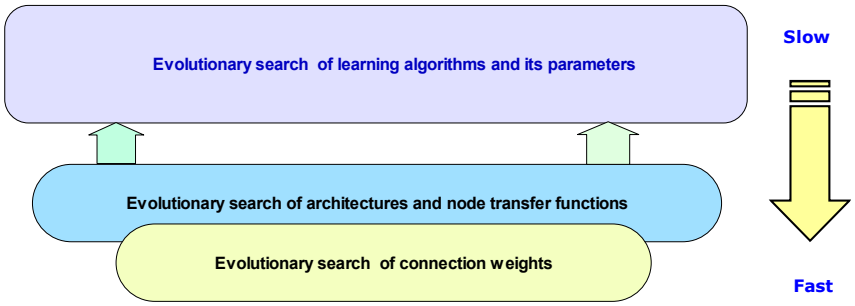


**Figure 2.** Meta-learning framework using evolutionary computation

# 5. Evolutionary Neural Networks and Meta-Learning

Evolutionary computation has been widely used for training and automatically designing neural networks. Figure 2 illustrates the meta-learning framework with the learning mechanism of the ANN evolving at the highest level on the slowest time scale [1]. All the randomly generated architecture of the initial population are trained by different learning algorithms and evolved separately. Parameters of the learning algorithms will be adapted (example, learning rate and momentum for BP)

according to the problem. Figure 3 depicts the basic algorithm of proposed meta-learning approach using evolutionary computation.

1. *Set t=0 and randomly generate an initial population of neural networks with architectures, node transfer functions and connection weights assigned at random.*
2. *Train the randomly generated neural networks (architectures and associated weights) using conventional learning algorithms (backpropagation, conjugate gradient etc.)*
3. *Evaluate the fitness and select parents for reproduction*
4. *Apply mutation to the parents and produce offspring (s) for next generation. Refill the population back to the defined size.*
5. *Repeat step 2*
6. *STOP when the required solution is found or number of iterations has reached the required limit.*

**Figure 3.** Meta-learning algorithm for optimising artificial neural networks

# 6. Experimentation Data

To evaluate the performance of the optimization techniques, we used the following 3 different time series for training and evaluating the neural network performance.

**a) Waste Water Flow Prediction**

The problem is to predict the wastewater flow into a sewage plant [20]. The water flow was measured every hour. It is important to be able to predict the volume of flow *f(t+1)* as the collecting tank has a limited capacity and a sudden increase in flow will cause to overflow excess water. The water flow prediction is to assist an adaptive online controller. The data set is represented as [*f(t), f(t-1), a(t), b(t), f(t+1)*] where *f(t), f(t-1)* and *f(t+1)* are the water flows at time *t,t-1, and t+1* (hours) respectively. *a(t)* and *b(t)* are the moving averages for 12 hours and 24 hours. The time series consists of 475 data points. The first 240 data sets were used for training and remaining data for testing.

**b) Mackey-Glass Chaotic Time Series**

The Mackey-Glass differential equation is a chaotic time series for some values of the parameters *x(0)* and $\tau$ [21].

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1\,x(t). \quad (5)$$

We used the value *x(t-18), x(t-12), x(t-6), x(t)* to predict *x(t+6)*. Fourth order Runge-Kutta method was used to generate 1000 data series. The time step used in

the method is 0.1 and initial condition were $x(0)$=1.2, $\tau$=17, $x(t)$=0 for $t$<0. First 500 data sets were used for training and remaining data for testing.

**c) Gas Furnace Time Series Data**

This time series was used to predict the $CO_2$ (carbon dioxide) concentration $y(t+1)$ [10]. In a gas furnace system, air and methane are combined to form a mixture of gases containing $CO_2$. Air fed into the gas furnace is kept constant, while the methane feed rate $u(t)$ can be varied in any desired manner. After that, the resulting $CO_2$ concentration y(t) is measured in the exhaust gases at the outlet of the furnace. Data is represented as [$u(t)$, $y(t)$, $y(t+1)$] The time series consists of 292 pairs of observation and 50% of data was used for training and remaining for testing.

# 7. Experimentation Results Using CAM and EC

**Cutting Angle Method**

We chose relatively small number of hidden layer neurons, because as we mentioned earlier, both CAM and ALEC are computationally very expensive. We summarise the results in the following tables. Computations were performed on COMPAQ Alpha EV68 single processor at 833 MHz, with 2 Gb of RAM (800Mflops/sec) for experiments using CAM and a Pentium II, 450 MHz with 256 MB of RAM for performing the experiments using evolutionary technique.

**Table 1.** Performance of ANN trained using Cutting Angle method. After the specified number of iterations of CAM, the error was improved using further 300 iterations of Levenberg-Marquardt method from 30 best minima provided by CAM.

| Data set | Architecture | RMSE (train) | RMSE (test) | Iterations | Time (sec) |
|---|---|---|---|---|---|
| Mackey-Glass | 4:4:1 Sigmoid | 0.0085 | 0.0091 | 3000 | 410 |
| Gas Furnace | 2:3:1 Sigmoid | 0.0173 | 0.0384 | 1000 | 12 |
| Waste water flow | 4:4:1 Sigmoid | 0.057 | 0.066 | 2000 | 216 |

**Evolutionary Computation (EC) Method**

We have applied the EC method to the three-time series prediction problems mentioned in Section 6. The parameters used in our experiments were set to be the same for all the 3 problems. Fitness value is calculated based on the RMSE achieved on the test set. The best-evolved ANN is taken to be the best individual in the last generation. As the learning process is evolved separately, user has the option to pick the best ANN (e.g. less RMSE, fast convergence, less computational expensive etc.) among the different learning algorithms. Genotypes were represented using binary coding and the initial populations of network architectures were randomly created based on the following parameters settings.

**Table 2.** Parameters used for evolutionary design of artificial neural networks

| | |
|---|---|
| Population size | 40 |
| Maximum no of generations | 40 |
| Number of hidden nodes | 4 neurons (maximum) |
| Activation functions | tanh (T), logistic (L), sigmoidal (S), tanh-sigmoidal (T*), log-sigmoidal (L*) |
| Output neuron | linear |
| Training epochs | 400 epochs for LM algorithm |
| Initialisation of weights | +/- 0.3 |
| Ranked based selection | 0.50 |
| Mutation rate | 0.40 |

For LM we used 1 as the factor for memory/speed trade off to converge faster, adaptive learning rate of 0.001 (+/- 100%) and learning rate increasing and decreasing factor of 10 and 0.1 respectively. The experiments were repeated three times and the worst RMSE values are reported.

**Table 3.** Experiment results using evolutionary approach

| Data set | RMSE (train) | RMSE (test) | Architecture | Time (minutes) |
|---|---|---|---|---|
| Mackey-Glass | 0.0056 | 0.0061 | 2 T, 2 T* | 602 |
| Gas Furnace | 0.0181 | 0.0290 | 1 T, I L, 1 T* | 132 |
| Waste water flow | 0.0567 | 0.0591 | 2 L, 1 T, 1 T* | 294 |

# 8. Discussions and Future Research Directions

Our computational results suggest that both the cutting angle method and evolutionary design technique are capable of training neural networks, and of locating if not global, very deep local minima of the error function. Yet both techniques are computationally very expensive. This is due to the fact that the underlying non-convex optimisation problem is NP-hard, and hence exponential time to solve the problem is needed. This is the reason why the globally optimal weights can be found easily only for relatively small networks. Along with simulated annealing (discussed elsewhere [14,23,24,28,30,36]), genetic algorithms and CAM are feasible alternatives to network training, which are superior to traditional local techniques. Being deterministic method, CAM is also capable of confirming the global optimum for small networks, and as our experiments show, is performing much faster than evolutionary approach on the problems considered. Our future research will concentrate on comparing various global and local methods for network training. We will standardise measuring algorithm performance using various measures (number of network error evaluations, computing time, quality of the solution), and will formally compare the methods on

a suit of training datasets. We will also investigate combinations of various global and local techniques.

# 9. Conclusion

The problem of multiple local minima, persistent in neural networks training, can be dealt with using various global optimisation techniques. This paper investigates one such technique, evolutionary computing, and introduces another one, the cutting angle method. We show that both methods can be successfully used for network training. If allowed to run sufficiently long, CAM can also confirm that the global minimum has been reached. Because the underlying optimisation problem is NP-hard, any general global training method is bound to be computationally very expensive. Hence in practice some global optimisation methods are applicable to relatively small networks. An important advantage of the evolutionary design of neural network is the complete adaptation of the network architecture, node transfer functions, connection weights, learning algorithm and its parameters according to the problem environment. CAM technique optimises the connection weights in a pre-defined architecture depending upon the designer's knowledge of the problem domain. Hence EC technique (even parallel evolutionary algorithms) might be helpful to solve complicated problems where not much knowledge about the network is available.

Because of increased computational complexity, it is important to reduce the domain of the variables both in dimensionality and size. Using output weights as linear or local variables reduces the dimension of the problem, and using the symmetry of the problem to restrict some weights to a unit simplex reduces the size of the domain by a factor of $m!$ ($m$ is the number of hidden neurons).

## References


[1] A. Abraham and B. Nath, *ALEC -An Adaptive Learning Framework for Optimizing Artificial Neural Networks*, in V. N. Alexandrov, ed., *Computational Science*, Springer-Verlag, San Francisco, USA, 2001, pp. 171-180.

[2] A. Abraham and B. Nath, *Optimal Design of Neural Nets Using Hybrid Algorithms*, *6th Pacific Rim International Conference on Artificial Intelligence*, Springer Verlag, Melbourne, 2000, pp. 510-520.

[3] M. Andramonov, A. Rubinov and B. Glover, *Cutting angle methods in global optimization*, Applied Mathematics Letters, 12 (1999), pp. 95-100.

[4] A. Bagirov and A. Rubinov, *Global minimization of increasing positively homogeneous function over the unit simplex*, Annals of Operations Research, 98 (2000), pp. 171-187.

[5] L. M. Batten and G. Beliakov, *Fast algorithm for the cutting angle method of global optimization*, Journal of Global Optimization, accepted (2001).

[6] G. Beliakov, *Fuzzy clustering of non-convex patterns using global optimisation*, *FUZZ-IEEE*, Melbourne, 2001.

[7] G. Beliakov, *Least squares splines with free knots: global optimisation approach*, IMA Journal of Numerical Analysis, submitted (2001).

[8] M. Bianchini, M. Gori and M. Maggini, *On the problem of local minima in recurrent neural networks*, IEEE Transactions on Neural Networks, 5 (1994), pp. p167(11).

[9] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford Press, 1995.

[10] G. E. P. Box and G. M. Jenkins, *Time Series Analysis, Forecasting and Control*, Holden Day, San Francisco, 1970.

[11] M. Brown and C. Harris, *Neurofuzzy adaptive modelling and control*, Prentice Hall, London, 1994.

[12] P. A. Castillo, J. J. Merelo, A. Prieto, V. Rivas and G. Romero, *G-Prop: Global optimization of multilayer perceptrons using GAs*, Neurocomputing, 35 (2000), pp. 149-163.

[13] F. M. Coetzee and V. L. Stonick, *On the uniqueness of weights in single-layer perceptrons*, IEEE Transactions on Neural Networks, 7 (1996), pp. p318(8).

[14] W. Duch and J. Korczak, *Optimization and global minimization methods suitable for neural networks*, Neural computing surveys (1999).

[15] S. Ergezinger and E. Thomson, *An accelerated learning algorithm for multilayer perceptron: optimizing layer by layer*, IEEE Transactions on Neural Networks, 6 (1995), pp. 31-42.

[16] D. Fogel, *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*, IEEE press, 1999.

[17] M. Forti, *A note on neural networks with multiple equilibrium points*, IEEE Transactions on Circuits and Systems-I: Fundamental Theory, 43 (1996), pp. p487(5).

[18] H. V. Gupta, K. Hsu and S. Sorooshian, *Superior training of artificial neural networks using weight-space partitioning*, *International conference on neural networks*, Houston, USA, 1997, pp. 1919-1923.

[19] R. Horst and P. M. Pardalos, *Handbook of global optimization*, Kluwer Academic Publishers, Dordrecht ; Boston, 1995.

[20] N. Kasabov, *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*, The MIT Press, 1996.

[21] M. C. Mackey and L. Glass, *Oscillation and Chaos in Physiological Control Systems*, Science, 197 (1977), pp. 287-289.

[22] O. L. Mangasarian, *Mathematical programming in neural networks*, ORSA Journal on Computing, 5 (1993), pp. 349--360.

[23] T. Masters, *Advanced algorithms for neural networks : a C++ sourcebook*, Wiley, New York, 1995.

[24] T. Masters, *Practical neural network recipes in C++*, Academic Press, Boston, 1993.

[25] V. V. Phansalkar and M. A. L. Thathachar, *Local and Global Optimization Algorithms for Generalized Learning Automata*, Neural Computation, 7 (1995), pp. 950-973.

[26] J. PintÈr, *Global optimization in action : continuous and Lipschitz optimization--algorithms, implementations, and applications*, Kluwer Academic Publishers, Dordrecht ; Boston, 1996.

[27] A. M. Rubinov, *Abstract convexity and global optimization*, Kluwer Academic Publishers, Dordrecht ; Boston, 2000.

[28] R. Sexton, R. Dorsey and J. Johnson, *Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing*, European Journal of Operational Research, 114 (1999), pp. 589-601.

[29] R. Sexton, R. Dorsey and J. Johnson, *Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation*, Decision Support Systems, 22 (1998), pp. 171-185.

[30] R. S. Sexton, R. E. Dorsey and J. D. Johnson, *Beyond Backpropagation: Using Simulated Annealing for Training Neural Networks*, Journal of End User Computing, 11 (1999), pp. 3.

[31] Y. Shang and B. W. Wah, *Global optimization for neural network training*, Computer, 29 (1996), pp. p45(10).

[32] K. K. Shukla and Raghunath, *An efficient global algorithm for supervised training of neural networks*, Computers and Electrical Engineering, 25 (1999), pp. 195(2).

[33] A. Tˆrn and A. Zhilinskas, *Global optimization*, Springer-Verlag, Berlin ; New York, 1989.

[34] X. Yao, *Evolving Artificial Neural Networks*, Proceedings of the IEEE, 87 (1999), pp. 1423-1447.

[35] X. M. Zhang and Y. Q. Chen, *Ray-guided global optimization method for training neural networks*, Neurocomputing, 30 (2000), pp. 333-337.

[36] X.-S. Zhang, *Neural networks in optimization*, Kluwer Academic Publishers, Boston, Mass., 2000.