Optimization and global minimization methods suitable for neural networks

Włodzisław Duch† and Jerzy Korczak‡
†Department of Computer Methods, Nicholas Copernicus University,
Grudziądzka 5, 87-100 Toruń, Poland
‡Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, CNRS,
Université Louis Pasteur, Blvd. Sebastien Brant, 67400 Illkirch, France

Abstract

Neural networks are usually trained using local, gradient-based procedures. Such methods frequently find suboptimal solutions being trapped in local minima. Optimization of neural structures and global minimization methods applied to network cost functions have strong influence on all aspects of network performance. Recently genetic algorithms are frequently combined with neural methods to select best architectures and avoid drawbacks of local minimization methods. Many other global minimization methods are suitable for that purpose, although they are used rather rarely in this context. This paper provides a survey of such global methods, including some aspects of genetic algorithms.

CONTENTS

[0]We would like to thank the Polish Committee for Scientific Research, grant no. 8T11F 014 14, and French Telecom, CNET, Belfort, for partial support. Updates, corrections, and comments should be sent to W. Duch at duch@phys.uni.torun.pl.

1 INTRODUCTION

Soft computing methods compete with traditional pattern recognition and statistical methods in many applications. For neural networks with predetermined structure, for example Multilayer Perceptrons (MLPs) with fixed architectures, finding an optimal set of parameters (weights and thresholds) requires a solution of a non-linear optimization problem. Such problems in general are NP-complete and the chance to find the best solution using typical, gradient-based learning techniques starting from a large, multi-layered network, is minimal. Nevertheless neural networks are able to learn many non-trivial problems, since even non-optimal solutions connected to the local minima of the error function may sometimes be satisfactory in practice. Optimization of neural architectures and global minimization methods applied to neural network cost functions may in many cases improve dramatically the quality of networks measured by their performance.

There are many global minimization methods (GMM) suitable for use in minimization of the neural cost functions and optimization of neural architectures. Unfortunately the information about these methods is scattered in the mathematical literature and therefore it is not accessible easily. We hope that a survey of GMM will stimulate research in this direction. Of course global minimization is not the only solution to the local minima problem. One of the simplest and the most widely used method is based on the momentum terms added to gradients, discussed in all good books on neural networks (cf. [1, 2, 3, 4, 5, 6]). Although the use of momentum may not lead to a global optimum it helps to

avoid poor local solutions. Perhaps the most obvious, although rarely used method to find optimal solution, is based on good initialization, followed by gradient optimization [7, 8, 9]. Initialization procedures should bring adaptive parameters into the vicinity of global minimum. Random initialization with small weights and biases, commonly used for neural networks, may not be sufficient if the optimal parameters are large. Recently in a long series of computer experiments Schmidhuber and Hochreiter [10] observed that repeating random initialization ("guessing" the weights) many times leads to faster convergence than using sophisticated versions of gradient methods. Gradient learning procedures are usually not able to compensate for bad initial values of weights and biases, getting stuck in local minima. Therefore a good strategy is to abandon training as soon as it slows down significantly and start again from random weights. Even better strategy is to use a good initialization or a global minimization method to solve the non-linear optimization problem.

The direct approach to finding optimal network structures is to use 'educated guesses' for good structures, and then select the most promising structures. Some minimization method, such as genetic algorithms [11] or simulated annealing [12], are used to generate new network architectures using the estimated quality ('fitness' in genetic terms) of previous networks. This strategy is based on an assumption (rarely spelled out explicitly) that the quality of the network, measured by the error on the training or sometimes on the validation set, is a smooth function of the topology and adaptive parameters of the network. Adding one neuron or deleting one connection, followed by retraining of the network, should not have a dramatic effect on the quality of the network. If the quality function is chaotic all minimization methods will ultimately fail to find the global optimum. Still, since performance of many neural architectures is evaluated and the best one selected, such strategy may create networks of higher quality than those postulated by many human experts. It should be interesting to check the assumption about the smoothness of the network quality, and characterize the space of all possible architectures, either in a theoretical way, or by doing simulations for some real-world problems. Unfortunately we are not aware of any systematic study targeting this topic.

Perhaps one caveat is in order. Although GMM may solve some problems which are beyond capabilities of the backpropagation algorithm even the best error minimization procedure cannot compensate the drawbacks of the network architecture selected or the mathematical model chosen. For example, classification borders conforming to a simple logical rule $x_1 > 1 \wedge x_2 > 1$ are easily represented by two hyperplanes but there is no way to represent them accurately using a sum of two soft sigmoidal functions in an MLP network. Increasing slopes of sigmoidal functions to improve representation of such decision borders around the (1,1) point leads to problems with learning by backpropagation, or by any other gradient-based method, since the volume of the input space in which sigmoids change rapidly (and thus gradients are non-zero) is rapidly shrinking. In the limit sigmoidal functions become step-functions but gradient techniques like backpropagation cannot be used to make this transition. As a result for some datasets no change in the gradient-based learning rule or in the network architecture will improve the accuracy of neural solutions. Many global minimization procedures do not have such drawbacks and may optimize slopes as well as other adaptive parameters without running into numerical instabilities. A good real-world example is the hypothyroid dataset [13], for which the best optimized MLPs still give about 1.5% of error [14] while logical rules [15] reduce it to 0.64% (since 3428 cases are provided for testing this is a significant improvement). Most research on neural networks is concentrated on network architectures and learning rules, but the selection of neural transfer functions may have strong impact on the complexity and performance of neural models [16].

Selection of the minimization procedure may lead to great improvement in the quality of the network and in

the speed of convergence of the learning algorithm itself. Global minimization [17] replacing the gradient-based backpropagation algorithms (for good overview of local minimization algorithms used in neural networks see [18, 19]) is able to find solutions to hard problems using smaller, compact neural networks. For example, Shang and Wah [20], using a NOVEL method that combines local gradient approach with global exploration of the search space, have found good solutions to the two-spiral benchmark problem using just 4-6 hidden neurons. Previous smallest MLPs (build using the cascade correlation algorithm [21]) had to use for this problem 9 neurons and 75 weights, and the training process was very sensitive to initial conditions. As Shang and Wah illustrate [20], a chance to find a good local minimum using a large network is greater than using a small network, because the error surface of a small network may be extremely rugged. Therefore using one of the gradient-based methods it is often easier to obtain good results with larger networks than with small ones. Global minimization should be especially useful for smaller networks. The use of global methods should also improve the quality of logical rules extracted with the help of neural networks [22]. Only a few global optimization methods have been applied so far to neural networks. Many methods are buried in the literature on engineering, financial, physical or chemical optimization problems and are virtually unknown to neural networks experts.

The problem of unconstrained global minimization is stated as follows: given a vector of initial parameters P, including such data as the value of weights, biases, other adaptive parameters, characterization of the structure of the network, and given a function $E(P)$ evaluating the quality of this vector for some dataset $D = \{X^{(i)}, Y^{(i)}\}$ (this may be either training or validation dataset), where $X^{(i)}$ are input vectors and $Y^{(i)}$ are the desired target vectors, generate a sequence of new vectors of parameters $P^{(k)}$ until the global minimum of $E(P)$ function is found. The simplest strategy of generation of these $P^{(k)}$ parameters is based on Monte Carlo approach, but in the context of neural networks the most commonly used global minimization and network architecture optimization methods are based on genetic algorithms. Applications of the genetic algorithms to optimization of neural networks and minimization of the cost functions are certainly more advanced than applications of other GM methods, and therefore these methods deserve a long section in our survey. The popularity of genetic methods should not prevent anyone from trying other global minimization methods. They are described in the subsequent sections starting from the simplest methods: Monte Carlo, simulated annealing, reactive tabu search, NOVEL algorithm, multisimplex, methods based on deterministic dynamics exploring the search space, smoothing methods, branch and bound methods, and interval methods. In the final sections a few new global minimization methods are proposed, some issues related to optimization of neural architectures discussed, and a short comparison of results of several GM methods given.

2 MONTE CARLO AND ITS IMPROVEMENTS

In the simplest Monte Carlo (MC) approach new vector of parameters P is randomly generated by changing a single parameter or a group of parameters. For optimization of neural network structures the change may involve adding or deleting one connection or one neuron with some connections and may be followed by gradient-based learning. Using gradient learning in connection with MC optimization of architecture does not guarantee that globally optimal solution will be found (but the use of genetic algorithms does not guarantee it either), but is relatively fast. After a specified number of networks is generated and trained the best are selected and used in the crossvalidation tests. Although this method is clearly not so sophisticated as genetic algorithms it is much simpler to implement.

An alternative procedure is to use a separate set of architectural discontinuous parameters P_a and continuous network parameters P_W (such as weights and biases), both of them optimized using Monte Carlo approach. In this case selection of architecture is followed by a much slower search for the global minimum of the cost function. Random generation of parameters leads, after sufficiently long time, to exploration of the whole parameter space, but if the number of parameters is large computing times may become prohibitive. Although in the MC method parameters are selected randomly they may be drawn from a probability distribution and overall constraints favoring small networks may easily be implemented. Another improvement is the use of quantized weights and biases during MC search, followed by short gradient-based training (without quantization) of the most promising networks that have been selected by MC.

An interesting improvement in the Monte Carlo procedure has been proposed by Dittes [23] in the context of N interacting spins, but the method has quite general applicability. The energy, or the error minimized, is assumed to be a sum of single spin terms, plus two-spin interaction terms, up to the k-spin interaction terms. A whole ensemble of energy functions is defined by taking $k = 1 \ldots N$ spin energies. The minimum is searched for on all these energy landscapes: probability distribution is defined for parameter changes and for selection of one of the landscapes. Energy for each of these landscapes should be proportional to the total energy, but local minima should be in a different place. Applications to several spin glass problems and the traveling salesman problem showed superiority of this approach over other Monte Carlo methods, especially for difficult optimization problems.

In context of neural systems this proposal amounts to something between on-line and batch training procedures. In on-line procedures parameters are adjusted after presentation of every new vector, while in the batch learning they are changed after the end of the epoch in which all data is presented. A probability distribution p_E for selection of the number of examples $K = 1..N$ considered simultaneously may be defined by:

$$p_E(x) = \left(1 - x + \frac{x}{N^{\alpha}}\right)^{-1/\alpha} \tag{1}$$

where the α parameter changes the shape of the distribution; for large positive α small K is selected, while for large negative α large K, around N, should be chosen. The optimization procedure requires selection of K and of the elementary change ΔP of network parameters. The change is performed only when the error function

$$E_K(P') = \sum_{i=1}^{K} ||Y^{(i)} - M(X^{(i)}; P')||^2 < E_K(P) \tag{2}$$

for $P' = P + \Delta P$ and randomly selected subset of training data $\{X^{(i)}, Y^{(i)}\}$. The $M(X^{(i)}; P)$ is the response of the network.

This method may be used in conjunction with any other method specifying the change of adaptive parameters, such as genetic algorithms or simulated annealing. It does not focus on the error value only but rather tries to include contributions from various configurations of the training data vectors. So far this method has not been tried in the context of neural networks.

3 SIMULATED ANNEALING AND ITS VARIANTS

This optimization method has been introduced in 1983 by Kirkpatrick, Gellat and Vecchi [12], inspired by the annealing (cooling) processes of crystals that reach the lowest energy, corresponding to the perfect crystal structure, if cooled sufficiently slowly (the process is very hard in practice and even very good crystals may contain some defects, showing that the global minimization process is so difficult that even Nature has problems with it). High temperature allows the atomic configurations to reach higher energy states overcoming energy barriers and avoiding inner tensions due to defects. Simulated annealing (SA) has found numerous applications in all branches of science and technology. In essence it adds to the simple Monte Carlo procedure the importance sampling according to the Boltzmann distribution (known from thermodynamics) to the selection of new parameter vectors, evolving the vector of adaptive parameters $P = (P_1, ...P_N)$ from some initial value to the value minimizing the error function.

There are three types of user-defined functions in the SA procedure: first, $p_P(P)$, describing the probability distribution of parameters; second, $p_E(\Delta E(P))$, the probability of accepting the new set of parameters as the current one, depending on the change of the error function; and third, $T(k)$, the schedule of changing the 'temperature' parameter T in some time steps t. Temperature T determines the scale of fluctuations allowed at a given time step. The Boltzmann annealing schedule is most frequently used because of the statistical mechanics roots of the method. It is defined by:

$$p_E(\Delta E(P)) = \frac{1}{1+\exp(\Delta E(P)/T)} \tag{3}$$

There are various proofs showing that, with the probability approaching one, for $T(t)$ slower than $T_0/\ln t$ a global minimum can be found by this procedure. For the p_P distribution gaussian form is frequently used:

$$p_P(\Delta P) = (2\pi T)^{-N/2} \exp(-\Delta P^2/2T) \tag{4}$$

where ΔP is the vector defining change of parameters from the previous value. Another popular annealing method, called Fast Annealing, is based on Cauchy distribution, defined by:

$$p_P(\Delta P) = \frac{T}{(\Delta P^2 + T^2)^{(N+1)/2}} \tag{5}$$

which assigns higher probabilities to larger changes of parameters. To save time, temperature is frequently reduced by a constant amount, leading to exponential schedule that does not guarantee that the global minimum is found (this variant of SA is often called "simulated quenching', or SQ [24]). In many simulations high temperature (leading to large changes of parameters) in the initial stages of minimization will not allow to sample the minima; to avoid the time waste short sample runs with fast annealing schedule are recommended to determine good initial temperature. In later stages, when local minima are explored, shifting to gradient-based or linear search techniques may significantly reduce cost of calculation.

Mean-field annealing (MFA) is a popular approximation to stochastic search, derive for error functions that are quadratic in quenched variables. Although MFA method may easily be applied to Hopfield-like networks for minimization of their energy – it works well for quasi-linear or quadratic error functions – it is not clear how to use it for MLP or other feedforward networks.

3.1 Adaptive Simulated Annealing

Adaptive Simulated Annealing (ASA), previously called Very Fast Simulated Reannealing (VFSR, [25]), uses different probability distributions for different parameters [24]. Parameters may have different finite ranges determined by physical considerations. For parameter P_i belonging to the $[A_i, B_i]$ range, and a random variable $r_i \in [-1, +1]$ range, the new value at the time step $t+1$ is generated by:

$$P_i^{t+1} = P_i^t + r_i(B_i - A_i); \quad r_i = \mathrm{sgn}(u_i - 0.5) T_i \left[(1 + 1/T_i)^{|2u_i - 1|} - 1 \right] \tag{6}$$

where u_i is uniformly distributed in the $[0,1]$ sector and T_i is the specific temperature for the i-th parameter. The annealing schedule is defined in terms of two free parameters, m_i, n_i:

$$T_i(t) = T_i(0) \exp(-c_i t^{1/N}); \quad c_i = m_i \exp(-n_i/N) \tag{7}$$

and the same type of function is taken for the $p_E(\Delta E(P))$ acceptance probability. In effect ASA has many free parameters that may be tuned to specific problems. Several other mechanisms are build into the publicly available ASA software[1], including re-annealing based on sensitivity of the cost function to parameters changes and self-optimization of some parameters. A new idea that seem to significantly improve the quality of the SA results is based on the rescaling of the error function [26]:

$$E(P) \leftarrow \left(\sqrt{E(P)} - \sqrt{E_{target}} \right)^2 \tag{8}$$

For $E_{target} = 0$ the original error landscape is obtained, but initially E_{target} is relatively large and the error landscape is smoothed, helping to escape from local minima. The modification is trivial but in the tests on the traveling salesman problem rescaled SA significantly outperformed original SA. The method has not yet been used for optimization of neural networks, although it may be useful not only in connection with simulated annealing, but also with standard gradient-based procedures.

SA has not been popular among neural network researchers working on MLPs, except for an interesting paper by Engle [27] in which the network adaptive parameters were discretized. Boltzmann machines and Harmony Networks are based on simulated annealing approach [28, 29]. SA was used with Gibbs sampling techniques in Bayesian approach to neural networks [30]. In one study [31] Mean Field Annealing [32] has been found superior to other minimization techniques in the neural network context. Simulated annealing has been used to solve optimization problems inherent in the vector quantization methods [33], for feature weighting in LVQ networks [34], selection and optimization of reference LVQ vectors [35], optimization of probabilistic networks [36] and in recurrent neural networks exhibiting chaotic behavior [37].

SA could also be combined with Dittes approach [23], although so far it has not been done. For many optimization problems SA was found superior to other techniques [24, 38]. It is certainly worthwhile to make more experiment with ASA applied to neural networks.

[1] Available at http://www.alumni.caltech.edu/ ingber/

3.2 Alopex

A special form of simulated annealing is used in the Alopex algorithm [39]. Since the result of this approach seem to be very good (probably due to the global minimization) it is described here. Alopex algorithm is based on a very simple idea which is competitive to the backpropagation. The weight W_{ij} is changed by a constant amount δ with probability defined by the sigmoidal (Boltzmann) factor, $p_{ij} = \sigma(\Delta W_{ij} \cdot \Delta E / T)$, where the weight change and the error change computed in the previous iteration is used. The annealing temperature is changed every epoch consisting of K steps, using the sum of error changes in the previous epoch:

$$T(n) = \frac{\delta}{K} \sum_{t=n-K}^{n-1} |\Delta E(t)| \tag{9}$$

For large temperature, probabilities of $\pm\delta$ are close to 0.5 and the weights are randomly changed until a large change of energy is detected (correlations between changes of weights and changes of error are large) and the temperature is reduced. During an iteration all weights are updated simultaneously. No assumptions are made about the structure of the network, the error measure being minimized or the transfer functions, no gradients are computed, the same algorithm may be used in feedforward as well as recurrent networks, and there is even some neurobiological plausibility of this algorithm (at least it is more plausible than backpropagation). There are 3 parameters: the step-size δ, which is taken as 0.01-0.001 times the dynamic range of weights, the initial temperature, and the number of steps per epoch $K = 10 - 100$. For some problems instead of the standard quadratic measure of error the information-theoretic cost function for $[0,1]$ target $Y_j^{(i)}$ and output $M(X^{(i)};P)$ values gives better results:

$$E(P) = \sum_{i=1}^{N} \sum_{j=1} Y_j^{(i)} \log \frac{Y_j^{(i)}}{M(X^{(i)};P)_j} + (1 - Y_j^{(i)}) \log \frac{1 - Y_j^{(i)}}{1 - M(X^{(i)};P)_j} \tag{10}$$

This function is much smoother than the quadratic error function and for networks without hidden layers using sigmoidal transfer functions it contains only one minimum. It is amazing that it has not been used more often and there are no comparisons with the standard quadratic cost functions.

Obviously many improvements can be proposed, such as the variable K, fast and slow weights (corresponding to different δ, or fast and slow synapses), different annealing schedules etc. Alopex may be quite easily used in connection with other global minimization methods, for example with genetic algorithms. One disadvantage of the Alopex algorithm seems to be that the weights are always updated and therefore saturate around large positive or negative values. To prune the small weights and enable feature selection it is better to define conditions when they may vanish, for example by using penalty functions described later in this article.

The Alopex algorithm has been tested so far only on a few problems with very good results, for example it has learned to solve quite large parity problems, it also solved all the standard machine learning benchmark, i.e. the 3 Monk's problems [40], with 100% accuracy (except for our MLP2LN approach [22, 41] this is the only network that was able to do it), but no results on the real-world noisy data have been reported so far.

4 REACTIVE TABU SEARCH

The reactive tabu search (both spellings, "tabu" and "taboo" are in use) is based on a very simple idea [42, 43]. The search is started at a random point and the best elementary move is selected; cycles are avoided by keeping the

trajectory of the search and discouraging the system from visiting the same regions again. In context of neural networks the values of the adaptive parameters P are kept with finite precision and the neighborhood $N(P)$ is defined by single-bit change operations. The error function $E(P)$ is therefore defined on a finite set of points. The best operation for which $E(P'), P' \in N(P)$ has the lowest value, is selected (even if the error grows) and the ties are broken in a random way. The inverses of most recent moves are prohibited to avoid cycles, hence the 'tabu' name for the method – regions already visited should be avoided. If there are too many possibilities only a restricted subset of moves are randomly sampled and the best one selected. The tabu is put on the moves, not on the values of P, and kept for a limited number of T time steps. The value of T should be large enough to avoid cycles and small enough to avoid overconstraining the search. Reactive Tabu Search (RTS) optimizes the prohibition period T adjusting it to the local structure of the problem. This requires remembering the points P along the trajectory and counting how many times each point has been encountered. RTS algorithm is presented below:

1. Set up initial parameters and initial point P
2. Check whether P is repetition: check how many times it was repeated and what is the length of the cycle; if it is short increase T to avoid it, if repeated too many times start the escape sequence; otherwise if T was constant for a number of iterations that was greater than the moving average or repetition intervals, decrease it by a constant factor (smaller T save computational time).
3. Select best move from moves that are available (not on the prohibited list); store the new point P'; use hashing to reduce size, remember the $E(P)$ and P that were the best so far.
4. Escape sequence: start diversified search, or a sequence of random steps to get out of the cyclic or chaotic area.

The dynamics defined here gives probability of visiting distant points which is much higher than given by a random walk. In application to neural networks [44] weights are mapped to binary strings using Gray encoding, and concatenated afterwards. The proper choice of this encoding, and selection of elementary operations, are very important for the success of the method. The need to use binary strings seems to be the weakness of RTS, since an elementary step may correspond to a small change in weights and after each change rather costly decoding has to be done to evaluate $E(P')$, even though only one weight is changed at a time. Very long strings of bits are obtained for coding weights, therefore it is recommended that only a few bits per weight are used (4 bits gave results comparable to full precision, while 2 or even 1 bit may already give very good results). Since derivatives are not used sigmoids may be replaced by multistep functions. To maximize generalization the function is minimized until the error on the validation set, not on the training set, is minimal.

The tabu search was used with very good result on a large number of combinatorial optimization problems. It was used [44] to discriminate interesting events in High Energy Physics data, with the best results obtained for a one-bit representation of weights (interpreted as $W_i = \pm 5$ weight values). The generalization levels reached 90%, while in the standard MLP they reached only 62%. Unfortunately the authors did not try to include 0 weight values – this would allow for feature selection. These results show that in a large network it may be more important to explore the whole search space than to find the precise values of weights.

5 THE NOVEL ALGORITHM

Recently a new global optimization method has been proposed for neural networks [20]. It is a hybrid, global/local trajectory based method, exploring the solution space, locating promising regions and using local search to locate promising minima. Trajectory $P(t)$ in the global search stage is defined by a differential equation:

$$\dot{P}(t) = A(\nabla_P M(P(t))) + B(T(t), P(t)) \tag{11}$$

where t plays the role of time, T is the trace function and $A,\ B$ are in general non-linear functions. The first component allows local minima to attract the trajectories, and the second component allows to walk out from the local minima. In the simplest case used in the practical NOVEL algorithm A and B functions are constants:

$$\dot{P}(t) = -\mu_g \nabla_P M(P(t)) + \mu_t (T(t) - P(t)) \tag{12}$$

The trace function T should assure that all space is finally traversed; it may either partition the space into regions that are explored in details or make first coarse and than fine searches. In the NOVEL algorithm a non-periodic function is used in each dimension i:

$$T_i(t) = \rho \sin\left[2\pi (0.5t)^{1-(0.05+0.45(i-1)/N)} + 2\pi(i-1)/n\right] \tag{13}$$

where N and n are two integer parameters. The differential equation is either solved in its original form by standard ODE computer package [20] or in a discretized form as a difference equation:

$$P(t+\delta t) = P(t) + \delta t\,[-\mu_g \nabla_P M(P(t)) + \mu_t (T(t) - P(t))] \tag{14}$$

Shang and Wah noted [20] that ODE solutions are slightly better although discretized equations are faster to simulate. The method has been tested on the two-spiral problem, training 5 hidden units in 100 time steps, starting from zero weights. This is a very hard problem for most MLP networks. The slope of sigmoids was unusually large (100), and $\mu_g = 1,\ \mu_t = 20$ was taken after some experimentation. Unfortunately finding a solution for 4 hidden units required a total of one week of Sun SS20/71 workstation time. The discretized version was about 10 times faster but didn't find the absolute minimum.

Deterministic algorithms, such as NOVEL, have some advantages over the stochastic versions. They find all deep minima contained in some bound region of the parameter space.

6 MULTISIMPLEX METHODS

Linear Least Squares SIMplex (LLSSIM) is another interesting global minimization method based on multisimplex minimization, recently presented by Gupta *et al.* [45]. Results of applications to the 3-layer neural networks are very interesting. The input-hidden layer weights W^h are estimated using the Multi-Start Downhill Simplex (MSDS) method, while the hidden-output weights W^o are estimated using the Linear Least Squares (LLS) approach.

Supposed that there are N_h hidden neurons, $(X^l, Y^l), l = 1..m$ training patterns, with N inputs and N_y outputs. The error function is:

$$E(W) = \frac{1}{2(m-1)} \sum_{l=1}^{m} \sum_{i=1}^{N_y} \left(Y_i^l - M_i(X^l) \right)^2 \tag{15}$$

and

$$M_k(X) = \sigma \left(\sum_{j=0}^{N_h} W_{jk}^o y_j(X) \right); \quad y_j(X) = \sigma \left(\sum_{i=0}^{N} W_{ij}^h X_i \right) \tag{16}$$

The total number of non-linear parameters here is $d = (N+1)N_h + (N_h+1)N_y$. For 3-layer networks one can estimate the output weights W^o by inverting the sigmoidal functions:

$$S_j(X) = \sigma^{-1}(Y_j) = \ln \frac{Y_j}{1-Y_j} \tag{17}$$

$$Z_j(X) = \sigma^{-1}(M_j) = \ln \frac{M_j(X)}{1-M_j(X)} = \sum_{i=0}^{N_h} y_i(X) \tag{18}$$

Here $S_j(X)$ is the activation of the output unit that will give expected results Y_j, and $Z_j(X)$ is the actual activation. Therefore the error function could be written as:

$$\begin{aligned} E(W) &= \frac{1}{2(m-1)} \sum_{l=1}^{m} \sum_{i=1}^{N_y} \left(S_i^l - Z_i(X^l) \right)^2 \\ &= \frac{1}{2(m-1)} \sum_{l=1}^{m} \sum_{i=1}^{N_y} \left(S_i^l - \sum_{j=0}^{N_h} W_{ij}^o \sigma \left(\sum_{n=0}^{N} W_{jn}^h X_n^l \right) \right)^2 \end{aligned} \tag{19}$$

The MSDS minimization for non-linear optimization is therefore restricted only to the input-hidden weights. The hidden-output layer weights are set up solving linear equations resulting from the least square problem. This approach has been tested using two function approximation problems and one real-world dataset (prediction of rainfall) and compared to backpropagation with the momentum, adaptive learning rates and conjugate gradient. The time was only 2-5 times longer than for the gradient-based procedure but the results were significantly better.

The multisimplex method of global optimization is similar to the multi-level single-linkage stochastic methods, which are a particular type of clustering methods [46]. Cluster is defined here as a set of points corresponding to a basin containing exactly one minimum to which descent methods should converge. Single linkage methods evaluate function on a set of sample points, find the best solution, apply local minimization and create a cluster by adding points around the minimum. Some of the initial sample points will fall into the cluster (meaning that local optimization follows into the minimum contained in the cluster), while others will be further than a critical distance to the cluster (i.e. to the closest point in the cluster). These far points should form new clusters. The space is thus partitioned into clusters, or basins of local minima gradient dynamics attractors. The sample set is continually expanding and therefore even with the finite sampling inside each cluster all minima will eventually be found. In the Multi Level Single Linkage (MLSL) local optimization is applied to all initially sampled points if they are not closer than some critical distance. The method has not yet been applied to neural networks.

7 HYBRID LOCAL-GLOBAL OPTIMIZATION METHODS

Baba [47] and Baba *et al.* [48] described one of the first hybrid algorithm for global minimization in neural networks. Conjugate gradient method with line search is used to find a local minimum and when the error decrease becomes smaller than a given threshold the method switches to a global mode to escape from local minimum. Once the error decrease becomes sufficiently large local minimization is turned on again. Random optimization method of Solis and Wets has been used by Baba [47]; the method guarantees convergence to a global minimum. The following steps are distinguished:

1. Select initial weight vector W_0, assuming that each weight is bounded by ± 10 or some other number. M is the maximum number of steps, $k = 0, b_0 = 0$.

2. Use normal distribution $N(b_k, \gamma)$ to generate random vector ξ_k of the same dimension K as the weight vector; use these vectors only if elements $W_k + \xi_k$ are within bounds.

3. If $E(W_k + \xi_k) < E(W_k)$ take $W_{k+1} = W_k + \xi_k$ and $b_{k+1} = 0.4\xi_k + 0.2b_k$;
   else if $E(W_k - \xi_k) < E(W_k)$ take $W_{k+1} = W_k - \xi_k$ and $b_{k+1} = -0.4\xi_k + b_k$;
   otherwise take $W_{k+1} = W_k$ and $b_{k+1} = 0.5b_k$;

Baba method has been applied to 3 problems with good results, but the number of iterations has been quite large. It has been slightly improved by Liang *et al.* [49] who combined it with Rosario [50] algorithm for local minimization (faster than the conjugate gradient search) and used it for blind equalization achieving significantly better results than using standard methods. The global Solis and Wets minimization seems to have drawbacks in context of neural networks training [51]: Gaussian distribution chooses areas around the local minimum with too high probability; results are strongly dependent on the choice of variances γ; the mean of this distribution is computed using arbitrary parameters 0.2, 0.4, 0.5; experience with neural networks show that good solutions are obtained if W_0 is in the ± 0.77 sector, while here arbitrary starts are used; previous values of weights are not used to improve the search. Other variants of the hybrid methods are described in [52, 53].

Orsier [51] has presented another hybrid optimization method called P^*_{SCG}. It is based on Random Line Search (RLS) combined with the Scaled Conjugate Gradient (SCG) method. The algorithm contains the following operations:

1. Select a random initial weight vector W_0.

2. From the current point W_i choose a random line in the weight space.

3. Minimize the error along the line and move to the new point if the error there is lower than in the current point.

This algorithm converges to the global minimum in the probabilistic sense. In applications to neural networks local minimum is found first using SCG method starting from weights in the ± 0.77 range. Random lines are generated always from the origin (in RLS algorithm they are generated from the current W_i). One of the most effective one-dimensional search techniques is called P^* [51]. It uses a sophisticated strategy creating statistical model (Wiener process) to estimate the position of the global minimum and quadratic approximation to find the final value. The length of the line is scaled by a factor proportional to stochastic Gaussian variable, while the components are uniformly

randomly selected in the specified range; thus most of the lines generated are rather short. The cost of the global strategy part in the P^* method is relatively small (usually about 10%) comparing to the cost of the local searches. The main parameter that the user may change is the range of the weights (in some experiments even ± 10 was too small – this depends on preprocessing of the input data).

The P^*_{SCG} method has been implemented in the SNNS simulator [54]. Comparison of this method with the results of Baba [48] on the parity problem and on one other problem showed its superiority. A deterministic hybrid method called LGOTA has been proposed by Tang and Kohler [55]. Comparison of P^*_{SCG} with this method on the 8-parity problem was initially not too successful but allowed to identify several problems. First, interesting minima did not lie near the origin, therefore renormalization of the random line vector by a Gaussian component was dropped and random lines were drawn between a point and its opposite (instead the origin at 0). This version of P^*_{SCG} worked very well for the 8-parity problem, finding global minima with a few random lines only. Although these results are preliminary the method shows great promise.

Hu *et al.* [56] introduced a novel hybrid random search scheme RasID (Random Search with Intensification and Diversification), based on an interesting probability density function for generation of the random search vectors. Their random search vectors are generated using the formula:

$$x_m = \begin{cases} \frac{1}{\beta} \ln\left(\frac{z_m}{1-q_m}\right) & \text{if } 0 < z_m \leq 1-q_m \\ \frac{-1}{\beta} \ln\left(\frac{1-z_m}{q_m}\right) & \text{if } 1-q_m < z_m \leq 1 \end{cases} \tag{20}$$

where z_m are random values uniformly distributed in $[0,1]$ interval and q_m and β are two parameters, first controlling asymmetry in searching in positive and negative direction and second controlling the range of random searching. It seems that such random search algorithms guarantee convergence [57]. Some heuristics are given for the choice of q_m and β parameters. The system observes recent success/failure rations and if the results improve local search is 'intensified' by the random search or performed by a gradient-based algorithm, otherwise diversified search sequence is started to escape from local minimum.

8 SMOOTHING ALGORITHMS

This algorithm has been developed and used for searching the minima of potential energy functions [58]. The idea is to transform the minimized function to a simpler one with smaller number of local minima. The function is smoothed and shallow minima disappear, leaving only the deep ones. Adding a second derivative of a function to the function itself leads to a combination that has the same inflection points (second derivative is zero in the inflection point) but the maxima decrease and the minima grow. A series of functions is defined as:

$$F^{(k)}(X) = (1+\beta\nabla^2)F(X), \qquad \beta > 0 \tag{21}$$

i.e. a trace of the Hessian matrix is added to the original function. The deformation is most effective if k grows to infinity and β goes to zero, for example by taking:

$$F(X,t) = \lim_{k\to\infty} \left(1+\frac{t}{k}\nabla^2\right)^N F(X) = e^{t\nabla} F(X) \tag{22}$$

It is easy to see that the exponential operator $\hat{T}(t) = \exp(t\nabla^2)$ flattens high frequency components of the function since its eigenvalues in one dimension are:

$$\hat{T}(t)\sin\omega x = e^{-\omega^2 t}\sin\omega x \tag{23}$$

and therefore for larger t all higher Fourier components will be very small. The $\hat{T}$ operator preserves the degree of the polynomial it acts on. Unfortunately it may lead to a divergent series. The $F(X,t)$ function fulfills the diffusion equation:

$$\nabla^2 F(X,t) = \frac{\partial F(X,t)}{\partial t} \tag{24}$$

For finite t it may be easier to use this equation instead of applying the exponential operator to the function directly, with $F(X,0) = F(X)$ as initial condition. Once the deformed function becomes simple enough to find the remaining minima by a gradient method one should perform the reverse transformation to obtain the positions of the minima at the original surface. This is done by following the minimum of $F(X,t) = F(X,k\Delta t)$ back to $F(X,0)$, using gradient procedure from the initial point found at $t_k = k\Delta t$ through $t_{k-1}, t_{k-2} \ldots t_1 = \Delta t$.

This procedure has been applied to some problems in physical chemistry [58], but never in the context of neural systems. If $E(W,t)$ containing a few deepest minima could be found and the minima traced back to $E(W,0)$ an optimal set of parameters could be found. Direct application of the exponential operator requires the ability to systematically compute high-order derivatives. Perhaps this is feasible. The diffusion equation may be solved in special cases, for example for expolynomial error functions (combination of polynomial factors multiplied by exponential factors), which is not quite the case of the neural networks. Several other minimization methods used in physical chemistry are also worth investigating [59, 60, 61].

9 BRANCH AND BOUND METHODS

These methods provide lower bounds on the objective function and are similar to the discrete minimization methods used in combinatorial AI searches [62]. The history of the branch and bound (B&B) methods is described in [63]. The advantage of these methods is that they do not require any information about the function minimized and they can be combined with many heuristic techniques to explore the search space. These methods may also characterize the error surface completely finding all local minima and saddle points around the global minimum, which may be useful in some applications. B&B methods may require exponential amount of work to find the global optimum, but in many practical problems this is not the case.

The branch and bound methods work in a finite domain ($W_i \in [-A,+A]$ for some constant A). The problem in the whole domain is called the root problem. A procedure for calculating lower and upper bound should be defined and if both bounds match for the current problem than a minimum has been found; otherwise the domain is partitioned into smaller subdomains and bounds checked for them. A graph with the search nodes is defined is this way and recursively expanded. Finding a local minimum in some subdomain allows to prune the tree removing all node with the lower bound above the local minimum found. The crucial point is the ability to compute bounds.

It is not clear how to apply these methods in a rigorous way to neural optimization. Probabilistic formulation, in which simulated annealing is used to estimate the bounds, is relatively straightforward, but so far has not been used.

10 INTERVAL-BASED METHODS

The interval-based methods [64], in which information about minimized function is computed over a box, cone or simplex-shaped regions, is another exciting possibility that can be used in conjunction with the branch and bound method[2]. Interval arithmetic is gaining popularity and has been implemented in Maple, Mathematica and extensions of Fortran 90. Interval methods used for global optimization are sometimes faster than point methods. Instead of single real number (which cannot be exactly represented in a computer) an interval $X = [\overline{x}, \underline{x}]$ is used and all basic operations are defined on intervals. Brouver fixed point theorem combined with Newton's method allows to check if a solution in a given interval exists.

Let $x_0 \in X$, evaluate $f'(X)$ and compute the Newton operator $N(f;X,x_0) = x_0 - f(x_0)/f'(X)$; then if $N(f;X,x_0) \subset X$ a unique solution $f(x) = 0$ exist in X. Newton operator is used in the interval iteration algorithm [63], consisting of the following steps:

1. Select a box B in which global minimum is searched from a list of boxes L; while L is not empty do:
2. If $N(B,\tilde{x}) \subset B$ is not true for some $\tilde{x} \in B$ than discard B;
3. else if $N(B,\tilde{x})$ is smaller than tolerance put B on the *Done* list;
4. else if $N(B,\tilde{x})$ is sufficiently smaller than B put it on L list as a new box;
5. else split the $N(B,\tilde{x})$ box into k pieces and put them on the list L.

Boxes on the *Done* list contain local and global optima. This algorithm has been modified by Hansen [64], adding different techniques to eliminate portions of boxes. The Back-Boxing method [63] formulated very recently seems to be the most efficient partitioning scheme based on interval arithmetic and rectangular partitions. It is used with constrained damped Newton's method for real local optimization, with some provisions to stay within the interval. The use of such local method is necessary because branch-and-bound algorithms spend most of their time around local minima. Newton method requires calculation of the interval Hessian which is done similarly as the usual Hessian calculation, but for neural networks applications it is rather expensive. Back-boxing is the process of identifying a box surrounding the region such that the error function on the box is convex. Finding the largest box is a non-trivial problem. Boxes around saddle points are treated as prohibited areas and avoided. There are 3 lists of boxes in the algorithm. First, boxes of undetermined contents; second, the finished boxes, reduced to smallest size and containing minima; third, the list of convex boxes, in which the error function is convex. The algorithm itself is rather complex and is described in details in [63].

Global interval minimization methods have not yet been used for neural networks although their application should not be too difficult. An applications of interval arithmetic to deal with the problem of the missing values in classification and regression were reported [65], but only local minimization technique has been used by the authors.

[2] See also http://cs.utep.edu/interval-comp/main.html

11 GENETIC ALGORITHMS

Great popularity of genetic algorithms in neural network optimization seems to stem from the biological inspirations of both methods. Of course in practical applications it does not matter if a given method was biological inspired or not. Mutations of candidates for good local minima correspond to random steps in Monte Carlo algorithms, so it may seem that the principle of "survival of the fittest" should help, especially that crossovers enhance the probability to leave local minima. However, in contrast to simulated annealing and a few other GM methods genetic approaches do not guarantee global convergence. Therefore one should carefully compare the results obtained with genetic algorithms with other global optimization techniques because it is not a priori clear that they should perform better in case of neural problems. Success in using genetic algorithms still depends on careful analysis of the nature of the problem at hand and may require tedious fiddling with the genetic rules.

One of the major domains of application of genetic algorithms (GA) is searching in a large space for good solutions and the optimization of hard problems [11, 66, 67, 68]. Genetic algorithms were inspired by the adaptation and learning capabilities of natural species. Compared to other methods, GA is perfectly capable of exploring discontinued spaces of solutions (which is common to many other global minimization methods, but is not possible using the gradient-based techniques) with a minimum background knowledge and domain-based information. GA techniques explore the space of parameters being guided by a fitness function, and enabling many solutions in the population to evolve in parallel, rather than focusing on a single best solution.

In order to explain the principle of GA one should consider a problem of optimization P and its space of solutions $S(P)$. At the beginning, an initial population G_0 is created, containing a family of chromosomes describing the elements $S(P)$. This population is evaluated in terms of its adaptation capacity to a given environment, by use of the fitness function $f(\cdot)$, which measures the capacity of chromosomes for solving the problem P. The main task of the fitness function is to guide the search for best solutions, thereby promoting good chromosomes instead of bad ones. Good chromosomes will then be selected as candidates for genetic operations. After one epoch of genetic operations is finished a new population G_1 is created, and the process is continued. Since the genetic search may run infinitely to control the evolution a condition for termination must be defined. Frequently, the level of satisfaction of the fitness function, the maximal number of generations, or even a measure of homogeneity of solutions may terminate the process. One may also define more sophisticated criteria such as measuring the convergence of the populations towards an acceptable solution.

Many practitioners state that GA are a robust optimization method with a large extent of applications. The scope of GA applications is restricted to those problems only where it is possible to encode the set of solutions as chromosomes and where a fitness function may be defined. The fitness function measures the chromosome capacity for solving a problem P assigning better chromosomes higher values of the fitness function. The fittest chromosomes are promoted in the evolution. The evolution process is supposed to improve the quality of populations gradually, but there is no guarantee that the optimal solution will be found.

During the last decade there has been a growing interest in the evolutionary approach to neural network design. This subject slowly enters the textbooks on neural networks [5, 69]. The search space of possible network structures is enormous, even for a particular type of neural networks. It is also obvious that a random search or an exhaustive search for "good networks" is practically impossible, especially in complex domains such as image processing, voice recognition, robot control, signal processing or financial forecasting.

Among many types of network models MLP networks are the most common, not only because of their universality, but also because of their good performance [70, 71, 72]. The efficiency of learning and the quality of generalization is strongly related to the neural network topology. The number of neurons, their organization in layers, as well as their connection scheme, have a considerable effect on network learning and its capacity for generalization [73]-[76]. Using a non-suitable network architecture has influence on the network performance and various quality factors, such as the training time, convergence in the training phase and the capacity for generalization.

One of the central issues in neural network research is the question of how to find an optimal MLP architecture. Frequently a naive approach is used: the network architecture is arbitrarily designed and the network trained for some time. Depending on the result of training neurons and/or connections are manually inserted or deleted, and then the network is trained further. The learning capacity of the modified network is observed and, if necessary, the process is repeated. In general, because of a large number of trial-and-error experiments required to find a good solution (which is sometimes still poor in comparison with an optimal solution), this intuitive approach is not acceptable. Genetic algorithms find near-optimal solutions through evolving populations of networks, which encode the candidate solutions to a given problem. Each individual, in our case a neural network, is assigned a fitness value evaluating how well it solves the problem. This process is iterative, and the selection of the best networks is based on the fitness function evaluation. Genetic algorithms are capable of solving difficult real-life optimization problems.

Genetic connectionism is based on the integration of evolution and learning within one system, by combining connectionist methods and genetic search techniques. Evolving neural networks have already been applied in a number of research projects [69, 73]-[88]. Taking into consideration the level and the way of integration of connectionist and genetic methods, this variety of approaches can be divided into four classes. The first class is formed by the approaches which use genetic algorithms to pre-process the training data, e.g. to select relevant input features [89, 90]. The second class employs genetic algorithms to train neural networks. Typically, this involves optimizing the weights in a neural network with a predefined topology [81, 83, 91]-[94]. The third class of approaches uses genetic algorithms to select a neural network topology [75, 79, 84, 95]-[98]. Finally, the fourth class is a combination of the previous methods [99, 100].

Our own approach can be considered as a hybrid method [101]-[104]. The novelty here is to emphasize not only the network performance aspects in terms of domain-oriented measures, but also the network simplification achieved by reducing the network topology and by the elimination of irrelevant and redundant variables (reducing the problem dimension). To find an acceptable network not only the training and generalization quality is taken into account, but also the number of neurons and connections, as well as the number of input variables (of course the fewer the better). All these genetic manipulations on the network architecture should not decrease the neural network performance. A smaller set of carefully chosen parameters may improve the performance of a neural network model and also reduce computation costs.

11.1 Overview of genetic connectionism methods

It is impossible to review all contributions to the vast field of evolutionary computation and genetic connectionism. A large biography on evolutionary design of neural architectures may be found in the Internet [105]. The discussion here is focused mainly on the MLP optimization methods. Many such methods have been elaborated upon in literature [74, 76, 78, 85, 93, 106, 107, 108]. The constructive neural models that modify network topologies during the learning

process [1, 2, 5, 3] are sometimes called ontogenic networks [108]. An ontogenic network has some advantages in comparison to the classical MLP: its architecture is not designed *ad hoc* or by a trial-and-error experiments, its performance is usually better, the computing time is reduced because adding neurons one after another requires little re-training, and memory requirements are lower. One may divide these methods into 4 main classes: methods which grow the network, prune network connections and remove neurons, methods with variable topology and pseudo-ontogenic methods.

The growing network methods increase the complexity of network topology starting the learning process from a very simple network, inserting new neurons and connections [21, 109, 110] (depending on the required accuracy). Contrary to this, the pruning methods decrease the complexity of topology starting from networks of large sizes, and then trying to simplify them [70]. Methods with variable topology use a combination of these two techniques, growing and pruning the network during the learning phase [111]-[114]. Some methods can not be considered as pure ontogenic, because they use fixed size networks determined by a user before the learning process starts. However, since they turn some units off and on they may be considered as a variant of an ontogenic algorithm [115, 116].

The methods that modify the network architecture after the end of the training phase are called non-ontogenic [70, 107, 108]. In general, the domain of applications of these methods is reduced to simple problems, for example problems involving Boolean functions, where one may easily find a simple and good neural network in a very short time by trial-and-error. A lot of research has been done on network optimization using genetic algorithms [76, 78, 85, 93]. These methods may be grouped into three classes: supportive methods, collaborative methods and hybrid methods.

11.2 Supportive methods

Here either genetic algorithms are used to help neural networks or vice versa, neural networks are used to optimize genetic algorithms. A few examples are given below.

- Genetic algorithms assisted by neural networks.

The XROUT algorithm, developed by Kadaba, Nygard and Juell [117, 118], was used to resolve the Vehicle Routing Problem. The problem consists in minimizing the distance traveled by vehicles, by assigning stopping points and a servicing order. Kadaba designed a hybrid system in which genetic algorithms have to find a set of good parameters of heuristic procedure, which searches the stopping points, and to construct an efficient set of heuristics designated to select the tour. The parameters for finding the minimum tour are encoded in chromosomes and determine the heuristics. Neural networks are used to generate an initial population for the two types of search implemented by genetic algorithms.

- Neural network assisted by genetic algorithms.

GA can be used to support neural network research in three different ways: the first is to select the input data or to transform the feature space, the second is to select a network learning rule and its parameters, and the third is to analyze a neural network [90, 119].

- Data preprocessing: Kelly and Davis [120] used GA to find rotations of data vectors and the scaling factors for each attribute, improving the performance of neural classifier. Other approaches are focused on data reduction. Frequently the data reduction improves the network performance and reduces the computing time. Chang and

Lippmann [121] used GA to reduce the volume of data. The algorithm creates new input data set from the original one; for instance, new features may be generated from raw data using polynomial functions. Drabe *et al.* [122] used GA to cluster subtasks of a complex task that neural network should learn. This is one of a few papers that try to apply neural networks to complex problems involving combinatorial optimization. GA are frequently applied to feature selection before training MLPS [123], Kohonen networks or vector quantization algorithms (cf. articles in [87]).

- Modification of parameters and learning rules: Belew, McInerney and Schraudolph [96] used GA to determine the learning coefficient and momentum for training of the MLP network. The speed of convergence has been improved in comparison with hand-assigned values. Harp, Samad and Guha [79] applied GA to modify learning coefficients dynamically, depending on the number of epochs. Schaffer, Caruana and Eshelman [85] adapted the learning coefficient, the momentum and the connection weights. Chalmers [78] encoded the learning rule in a chromosome, changing the rule using observation of network performance in previous epochs. GA was also used to initialize the Radial Basis Networks [124, 125] and to train cellular neural networks [126].

- Neural network analysis using GA: Opitz and Shavlik [127] used GA to explain the behavior of neural networks by defining a function linking the network inputs and outputs.

11.3 Collaborative methods

The idea of network optimization using genetic algorithms during the training process is quite popular [79, 83, 85, 128]. GA are used here to determine weights of connections using the mean-square quadratic error function as the fitness function. However, this is a very time consuming process, even if some heuristics are used to reduce the computing time. Other global minimization methods have been applied in neural networks almost exclusively for minimization of the error function. A more natural way to combine GA with neural networks is to use genetic search techniques to find an optimal network topology. Such search requires the following elements:

- A representation of a genotype which is defined as the function mapping of the genotype into a phenotype.
- A protocol allowing to establish a link between the phenotype and the environment related to the problem at hand.
- A method of training capable of verifying the quality of a specified network.
- A measure of fitness obtained.
- A method generating new genotypes (a set of genetic operators).

11.4 Hybrid methods

Methods using GA to modify the weight of neural network connections are usually considered as less efficient and more computing-intensive than those based on gradient backpropagation. Another possible application of GAs is for initialization of adaptive parameters, as well as determination of learning coefficient and momentum. In some

applications of neural networks, where the backpropagation cannot be used, for example in recurrent networks, GA offers a solution replacing gradient learning algorithms [129].

Encouraging results have been obtained in GA application to optimize MLP network topologies [80, 93, 104, 121, 130], optimization of RBF network parameters [125] and structures of Kohonen networks [131, 132]. A specification of a mapping function that encodes a network into a string of genes, known as 'a chromosome', is required for such optimization. A number of methods to encode MLP networks in chromosomes [84, 86, 103, 133, 134] have been used and a few of these methods will be described below. Another important problem is to define a fitness function that will take into consideration various aspects of a hybrid optimization system. Before discussing these two questions, let us describe a process of genetic search of an optimal neural network at a very general level.

11.5 Process of genetic search

The process of genetic search can be divided into five main phases: initialization, selection, genetic operations, evaluation and replacement. The aim of the first phase is to build the initial generation of neural networks. In the domains where no background knowledge is available, it is important that the construction procedure ensures an equiprobable sampling in the search space. Therefore, to begin searching efficiently, one needs to start with representatives chosen from the whole search space. However, in domains where a priori knowledge is available, the user may insert the networks that are "close" to the optimal solution into the initial population.

Once the initial population of networks has been created and trained, one has to assign a certain probability of reproduction in the next generation to each network. The selection operation creates a genetic patrimony, called a mating pool. The selection of the networks chosen for reproduction is carried out with probabilities directly proportional to their quality. There are many selection strategies [135, 136, 137], often referred to in statistics as the sampling techniques. One of the most interesting techniques is the stochastic universal sampling [135] which is simple and theoretically sound.

Having taken out two networks from the mating pool, the new networks are produced by applying genetic operations. Two operations are fundamental: the crossover and the mutation. The crossover operation allows two new networks, which are composed of a mixture of genes inherited from each parent-network, to be created. To preserve properties of the best networks over the successive generations, the crossover is not applied systematically. The purpose of the mutation operation is to provoke a jump inside the search space by modifying a randomly selected gene in the network population. Generally, it prevents the GA from converging towards a local minimum. The mutation creates some noise in the evolution process; therefore its probability is usually quite small. In our own system a large number of network-oriented operations have been designed [101].

The evaluation operation allows the fitness of each neural network in the population to be measured. The operation is composed of two steps. The first is chromosome decoding, leading to a construction of the corresponding network. The second consists of computing the fitness function described previously. As a result a value is attributed to each network, allowing to compare it with other networks generated during the selection and the genetic manipulation processes.

Once evaluated, the replacement operation selects neural networks for the next population. The new population is created by using similar algorithms to those used in the selection strategy. Therefore, in many genetic systems, the replacement is treated as a selection operation. In our system a variety of strategies to create new populations has been

implemented. The two most important aspects of genetic evolution of neural networks, the network encoding scheme and the fitness function, are described in details below.

11.6 Network representation and encoding schemes

During the evolution each generated network is first decoded from a chromosome representing a neural network, then it is trained and tested, and finally the network is encoded back into the chromosome (Fig. 1). Of course after training neural network changes not only its connection weights, but also its quality.

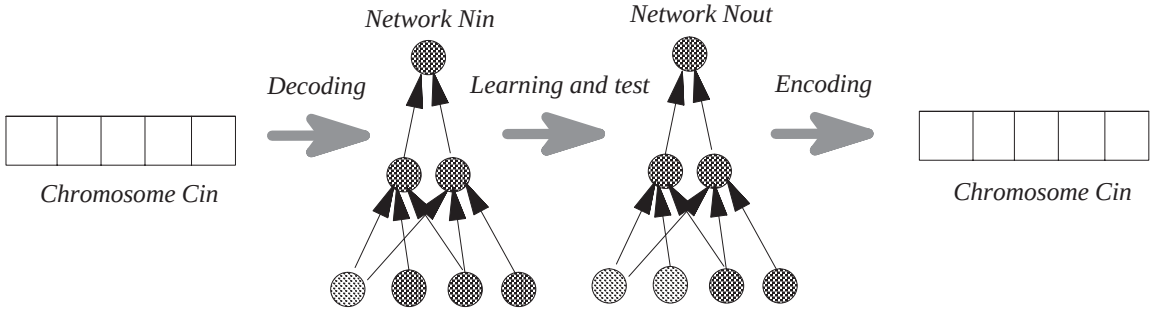

Figure 1: Network encoding.

Various information may be encoded in a chromosome. The most common practice is to store only a network topology, but one may also encode the initialization parameters, connection weights, transfer functions, learning coefficients, etc. The more parameters are encoded the bigger is the search space. More precisely, if one encodes n parameters $p_1,...,p_n$, and if each of them takes $val(p_i)$ different values, then the size of the solution space is equal to $\prod_{i=1}^{n} val(p_i)$. In consequence, taking too many parameters will at least slow down the convergence of the genetic process. This excess cost, no doubt prohibitive, will decrease the quality of solutions, especially if the inserted parameters are irrelevant. An ideal network coding scheme does not exist, but a number of requirements should be fulfilled by a good encoding scheme. For example, the network-coding scheme should be bijective, otherwise it would be impossible to control the evolution process. A detailed discussion of all such requirements may be found in [84, 102].

To introduce the problem of coding, let us consider one of most common methods used to encode MLP networks. The encoding scheme consists of mapping the network structure onto a binary connection matrix where each cell of the matrix determines whether a connection between two neurons exists or not. Each hidden and output neuron receives connections from at least one neuron. The complete network structure is represented by the list of neurons with their incoming connections (Fig. 2).

The direct encoding scheme has a number of advantages. It is simple, easy to implement and manipulate. In addition, large number of genetic operators may be defined, including generic operators of crossover and mutation, as well as network-oriented operators. One of the drawbacks of this method is that it may result in very long codes for large networks. The main goal of an encoding scheme is to represent neural networks (phenotypes) in a population as a collection of chromosomes (genotypes). There are many approaches to genetic representation of neural networks [80, 84, 103, 133, 134, 138].

Besides the direct encoding scheme many other interesting methods exist. Several authors proposed an encoding

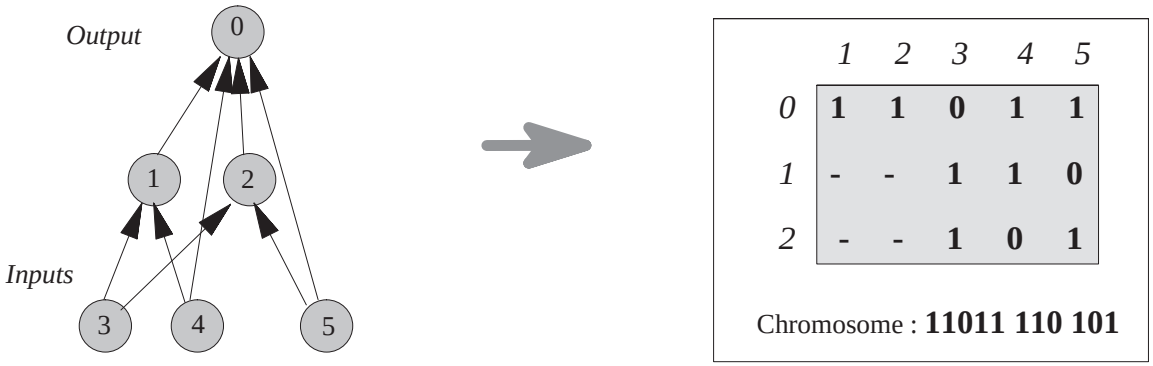

Figure 2: Direct encoding scheme.

scheme based on grammar rules. Kitano [133] developed a method founded on rewriting rules. Fig. 3 illustrates his approach for MLP network encoding. The matrix of connections is recursively created by applying rewriting rules. For n neurons the connection matrix should have the size $2^k \geq n$ (in Fig. 3 $n = 6$ and $k = 3$) and the additional elements of the matrix are filled with zeros. Each 2×2 matrix is replaced by a symbol according to a given rule and the 2×2 matrix of symbols is replaced by a higher order symbol. This recursive process ends with a single symbol for the whole connection matrix. Each rewriting rules is treated as a gene in the chromosome representing the whole network structure.

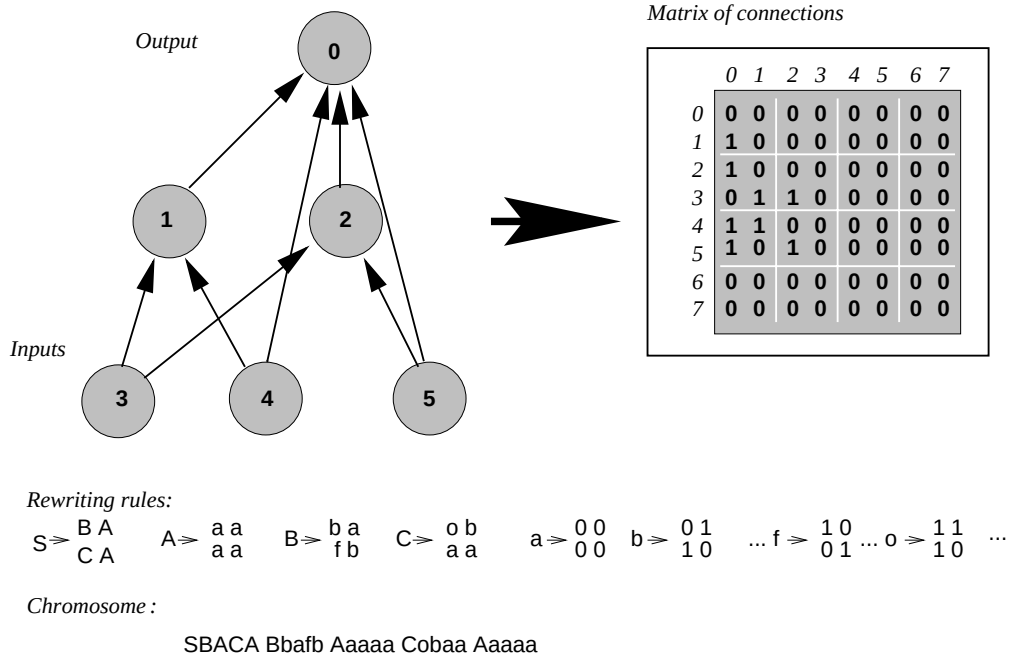

Figure 3: Example of encoded network using rewriting rules.

A more sophisticated version of this method is provided by Korczak and Dizdarevic [74, 103]. Each network is described by an encoding scheme based on the network decomposition according to 'genes'. Each such gene, called a *'parcours'*, encodes a subgraph of a neural network containing one output neuron, one attached connection and all antecedents to this connection (Fig. 4).

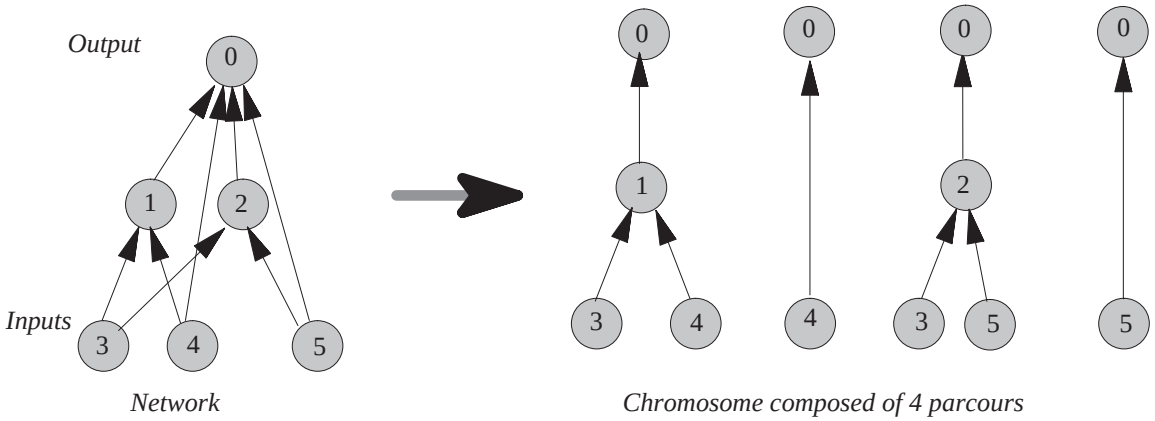

Figure 4: Example of network encoding using "parcours".

From a semantic point of view, a *parcours* represents a function, defined on a given set of network inputs. Each *parcours* thus includes a part of the computing operations carried out by the network. These partial functions are independent, therefore there is no semantic interaction between genes. This feature makes the manipulations on network structures by the crossover operation particularly easy. Nevertheless, there are two types of syntactic interactions between genes that are helpful in the design of genetic algorithm: interaction between genes concerning the same output, and interaction between the genes possessing a common element of architecture. The size of a chromosome coding network is not fixed; it is proportional to the number of adjacent connections to the output neurons.

Other indirect encoding schemes are based on clustering methods. The *parcours* method of encoding neural networks has been designed to achieve many objectives. On the one hand, the efficiency and simplicity of the encoding scheme was very important. Large number of genetic operators should allow for the manipulation of the smallest syntactic and functional elements of network architectures. On the other hand, this encoding scheme allows for the application of a wide variety of genetic operators and ensures that network coherence is easily maintained.

11.7 Fitness function

Together with the encoding scheme the definition of a fitness function is of crucial importance for the genetic evolution. This function is in charge of guiding the genetic search process by assigning a score to each neural network. The fitness function may incorporate many criteria evaluating the network quality. In our system [74] three criteria have been proposed: the performance, the complexity of network topology and the number of network inputs. More precisely, we have looked for networks that:

- minimize the number of training errors and validation errors;

- have a simple topology, in terms of a number of inputs, neurons and connections;
- are able to learn rapidly.

The last factor is particularly important when large data files are manipulated. Given a population of networks, the fitness value of each network is computed as a linear combination of different criteria, where each criterion is normalized and weighted according to its importance in a given problem. Frequently, the limits of the highest values for the different criteria are unknown, but the user can define the acceptable limits for performance level and network complexity. Formally, the fitness function $F[i]$ of the i-th network is a weighted sum of the network quality $F_{quality}$ and the network complexity $F_{complexity}$:

$$F[i] = aF_{quality}[i] + bF_{complexity}[i]; \text{ where } a+b=1, \quad a,b>0 \tag{25}$$

The network evaluation is carried out for all networks in the population. After complete evaluation the values of $F_{quality}$ and $F_{complexity}$ are renormalized. The normalization is necessary because the criteria have different and heterogeneous domains. The formulas used are as follows:

$$F_{quality}[i] = \sum_{j=1}^{m} p_j^{(q)} \frac{C_{ij}}{\max(C_{1j} \ldots C_{nj})} \tag{26}$$

and

$$F_{complexity}[i] = \sum_{k=1}^{r} p_k^{(c)} \frac{C_{ik}}{\max(C_{1k} \ldots C_{nk})} \tag{27}$$

where $p_j^{(q)}$ and $p_k^{(c)}$ are weights assigned by the user to the evaluation criteria C_{ij} and C_{ik}, and:

$$\sum_{j=1}^{m} p_j^{(q)} = \sum_{k=1}^{r} p_k^{(c)} = 1 \tag{28}$$

As a measure of the network quality one can also use the training efficiency calculated as the speed of the decrease of the training error. To introduce this measure one can define a threshold for the training error t, which should be reached at a given time in the training process.

The complexity component of the fitness function measures the topological complexity of a neural network in terms of the number of neurons, the number of connections and the number of used input features. Usually, for each such criterion C a maximal acceptable value $\max(C)$ is known. This value may be imposed by the application, e.g. it may be the number of input features, or it may be defined by a user as a constraint such as the maximum number of the hidden layers or the maximum number of neurons in a layer. To compute a total value of the network complexity the following indicators are computed for each chromosome (equivalent to a network) i and each criterion C:

$$Comp(C,i) = 1 - \frac{Used(C,i)}{\max(C)} \tag{29}$$

For instance, to compute an indicator of used inputs for a network corresponding to a chromosome using only 8 of 16 inputs, one would obtain: $Comp(UsedInp,i) = 1 - 8/16 = 0.5$

Finally, to compute the total network complexity one can use the following formula:

$$F_{complexity}[i] = \sum_{k=1}^{r} p_k^{(c)} \frac{Comp(C_k, i)}{\max(C_k)} \qquad \text{where } \sum_{k=1}^{r} p_k^{(c)} = 1 \tag{30}$$

The formula proposed here uses only values relative to a given population, and therefore it should be applied to indicators for which the global limits are difficult to estimate a priori. In a case where it would be possible to define these limits for a whole set of solutions, it is preferable to apply a global normalization.

To summarize, three categories of optimization criteria have been identified: the quality, the complexity and the performance. Each category is represented by one component of the fitness function and each component of the fitness function might be calculated using some sub-criteria, which in turn may have their own weights.

Applying the fitness formula 25 may lead to a problem resulting from the compensation of two components of the fitness function. A network of poor quality may have its fitness compensated by a low complexity, or inversely, a network of high complexity may be compensated by its very good quality. In other words a rather poor network may have good chances of being selected for reproduction. To avoid such problems thresholds for quality and complexity have been defined. If a network has quality or complexity lower than the threshold a strong penalty factor is applied to the fitness function to neutralize the effect of possible compensation. Suppose that we have two sets of critical thresholds $\left\{S_t^{(q)}, 1 \le t \le m\right\}$ and $\left\{S_t^{(c)}, 1 \le t \le r\right\}$, one for the quality and the second for the complexity. The complexity and the quality penalties may be applied in the following way:

$$F_{quality}[i] \leftarrow F_{quality}[i] \times Pr_q\left(S_1^{(q)} \wedge S_2^{(q)} \ldots \wedge S_m^{(q)}\right) \tag{31}$$

$$F_{complexity}[i] \leftarrow F_{complexity}[i] \times Pr_c\left(S_1^{(c)} \wedge S_2^{(c)} \ldots \wedge S_r^{(c)}\right) \tag{32}$$

where $Pr_q(True) = Pr_c(True) = 1$ and $Pr_q(False) = Const_q,\ Pr_c(False) = Const_c$. If one of these conditions is not satisfied, then a penalty is inflicted. The penalties may also be defined as additive coefficients. Usually the network quality has more weight as a fitness criterion, and it is natural to penalize networks having low quality. However, it is difficult to define what is an acceptable network complexity. As a consequence it will be harder to estimate the threshold for complexity than for quality and the penalty for the complexity should probably be lower than that for a poor quality.

The computed fitness is a decreasing function and that its codomain is in the interval $[0,1]$. To obtain an increasing function, whose codomain is identical, one may transform F using the formula:

$$\bar{F}[i] = \frac{1 - F[i]}{1 + \alpha \cdot F[i]}, \qquad \text{where } \alpha \ge 0 \tag{33}$$

The performance of a given network represents its capacity to solve a problem. All sorts of performance evaluation functions have already been applied. They can be based on the training error, generalization error, or other domain related measures. In our system the user may select one or many performance measures. The complexity of the network topology is measured by taking into account the cost of basic operations $(+, -, \times, /)$ required to propagate results from input to output neurons in the framework of the gradient-based backpropagation algorithm. Thus, the number of neurons and their connections contribute to the calculation of the complexity term of the fitness function.

The approach presented above has been implemented in two prototypes of evolutionary based neural systems AGWIN [101, 104] and ECO [139]. Genetic search for optimal neural networks presented in this section not only optimizes the network topology but performs also the feature selection for a given problem. There is no doubt that genetic algorithms can be used to efficiently solve the problem of network optimization considering not only static aspects of network architecture but also dynamic ones. The preferred method is hybrid, i.e. the minimization of the cost function is performed using the gradient-based methods but optimization of the fitness function is done using the genetic algorithm. Other methods described below use global minimization primarily to find better solutions of the training problem.

12 PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) [149, 150] is inspired by evolutionary computations and artificial life, but it is quite different from genetic algorithms. *Swarm* is a population of interacting points, called "particles", flowing through space of adaptive parameters, each with some 'velocity' (this is simply a $\Delta\mathbf{X}$ change), trying to optimize some objective function through collaborative search process. Each particle (corresponding to chromosome in GA) is a vector $\mathbf{X}_i$ in the space of adaptive parameters, initialized randomly and given a random velocity. For each position of the particle a fitness value is calculated and the information about the current as well as the best parameter values $\mathbf{B}_i$ achieved by each particle are stored. Parameters corresponding to the best result for the whole swarm $\mathbf{C}$ and the best results for individual particles are used to influence the trajectories of evolving particles. This is done by adding to the current position $\mathbf{X}_i$ of the particle i the velocity vector:

$$V_{id} = U_i \cdot V_{id} + c_1 \cdot r_1(\mathbf{B}_{id} - \mathbf{X}_{id}) + c_2 \cdot r_2(\mathbf{C}_d - \mathbf{X}_{id}) \tag{34}$$

where d enumerate adaptive parameters, r_1, r_2 are random numbers in $[0,1]$ interval, and U_i are inertia parameters, similar to the momentum in standard backpropagation algorithms. Inertia may be decreased to change the exploration from more global to more local, playing similar role as the temperature in simulated annealing. This simple update equation allows to mix the influence of the globally best solution and locally best solution on the trajectories of swarm particles. A local version of this model replaces $\mathbf{C}$ by a vector of the best parameters in a local topographical neighborhood.

Although little is known about the performance of the method and there are no proofs that it really converges preliminary comparison with genetic methods or even standard backpropagation methods shows that it is faster and capable of finding better solutions [150]. The quality of the swarm solution in case of the MLP training results to large degree from optimization of individual slopes of the sigmoidal functions. This algorithm is very simple to implement and may be extended along the lines of adaptive simulated annealing.

13 OPTIMIZATION OF NETWORK ARCHITECTURES VIA GLOBAL MINIMIZATION OF THE COST FUNCTION

Finding optimal neural architectures is a special case of a general optimization problem. Genetic connectionist methods are used primarily for optimization of neural architectures. Can one use other global optimization methods to find

optimal architectures?

In principle starting from a model that is sufficiently complex and using a good global method of minimization of the error function an optimal network with non-zero connections, and thus appropriate architecture, may be found. In practice such strategy will work only for very large training sets and will be quite costly from the computational point of view. For small or medium datasets this strategy is used with regularization terms added to the error function [30], for example with quadratic weight decay terms:

$$E(W) = E_0(W) + \frac{\lambda}{2}\sum_{i,j} W_{ij}^2 \tag{35}$$

where $E_0(W)$ is the standard quadratic error measure. Various complexity regularization methods, such as the minimum description length, information-theoretic criteria, Optimal Brain Damage or Optimal Brain Surgeon procedure (cf. [1, 2, 4]) may be used to select neural model most suitable for a given data (for a survey of the network pruning methods see [140]). The quadratic weight decay term corresponds to the assumption of the Gaussian distribution of the weights:

$$P(W|M) \propto \prod_{ij} e^{-\alpha_1 W_{ij}^2} \tag{36}$$

Taking sum of absolute values instead of squares leads to the Laplacian regularization [141]. Another useful weight decay term is:

$$E(W) = E_0(W) + \frac{\lambda}{2}\sum_{i,j} \frac{(W_{ij}/W_0)^2}{1+(W_{ij}/W_0)^2} \tag{37}$$

where W_0 is an adaptive parameter and the summation runs over all weights and biases.

Networks created using complexity regularization may be large, with many neurons and small weights, but the network output is smooth, leading to a good generalization. In Bayesian approach to neural computing the λ 'hyperparameter' is automatically adjusted to the complexity of the data [30, 142]. Autoclass, one of the most successful approaches to unsupervised classification, is based on the Bayesian techniques estimating the likelihood of different models and then using these likelihoods to combine these models in a committee for final prediction [143]. This approach is computationally quite costly, requiring optimization of many trial models, but the results are frequently very good.

Regularization theory is described in many textboox [1, 2]. One simple reason why regularization helps MLP networks to improve generalization by avoiding overfitting of the data is based on the following argument: for normalized input vectors $\mathbf{X}$ but arbitrary weight vectors $\mathbf{W}$ the range of the sigmoid argument lies in the $[-|\mathbf{W}|,+|\mathbf{W}|]$ interval. A unipolar sigmoid has a maximum curvature around ± 2.4, therefore smaller weights of the norm mean that the network operates in an almost linear regime. Regularization methods force the weights to become small and thus the network approximation to the training data becomes more smooth.

An alternative approach, frequently used in connection with genetic algorithms, is to use smaller networks, trying different architectures or different classification models. Although regularization may smooth the error function landscape to some degree the optimal solution may still be hard to find by gradient methods, requiring either very large networks or smaller networks with large weights. For example the experiments with the P^*_{SCG} method [51] showed

that even ± 10 may not be sufficient. We have recently proposed [7, 8] another form of regularization, which assumes that these weights are distributed around $a > 0$ and $b < 0$. Such distribution is enforced by the following penalty term:

$$E(W) = E_0(W) + \frac{\lambda_1}{2}\sum_{i,j} W_{ij}^2 + \frac{\lambda_2}{2}\sum_{i,j} W_{ij}^2 (W_{ij}-a)^2 (W_{ij}+b)^2 \tag{38}$$

The second term with λ_1 leads to a large number of zero weights, i.e. elimination of irrelevant features, and the third term vanishes for weights equal 0, a or $-b$. Similarly as in the weight pruning technique case in the backpropagation algorithm these terms lead to the additional change of weights:

$$\Delta W_{ij} = \lambda_1 W_{ij} + \lambda_2 W_{ij}(W_{ij}-a)(W_{ij}+b)(3W_{ij}^2 + 2W_{ij}(b-a) - ab) \tag{39}$$

where λ_1 and λ_2 scale the relative importance of auxiliary conditions. This form of error function has two advantages: independent parameters control enforcing of 0 and $a,\ b$ weights, and an interpretation of this function from the Bayesian point of view [142] is straightforward. It defines our prior knowledge about the probability distribution $P(W|M)$ of the weights in our model M. A network trained on classification tasks should give crisp logical decision "yes", "no" or "irrelevant", therefore *a priori* conditional probability [142] is:

$$P(W|M) = Z(\alpha)^{-1} e^{-\alpha E_a(W|M)} \propto \left(\prod_{ij} e^{-\alpha_1 W_{ij}^2}\right)\left(\prod_{ij} e^{-\alpha_2 (W_{ij}-1)^2}\right)\left(\prod_{ij} e^{-\alpha_2 (W_{ij}+1)^2}\right) \tag{40}$$

Prior knowledge about the problem may also be inserted directly into the network structure, defining initial conditions modified further in view of the incoming data. The network is trained starting with relatively large values of regularization parameters. The training error will initially be large since MLP has many irrelevant connections. Optimal value of $a,\ b$ parameters are found iteratively, starting from $a = b = 1$ values:

$$\begin{aligned} a &= +\sum_{ij} W_{ij}^3 (W_{ij}+b)^2 \Big/ \sum_{ij} W_{ij}^2 (W_{ij}+b)^2 \\ b &= -\sum_{ij} W_{ij}^3 (W_{ij}-a)^2 \Big/ \sum_{ij} W_{ij}^2 (W_{ij}-a)^2 \end{aligned} \tag{41}$$

The penalty function (38) encourages some connections to become small and they are finally deleted, therefore this approach already includes partial optimization of architecture. Using it together with global minimization methods should lead to both optimal architecture and the global minimum for the error function. Only a few experiments on the real world datasets were done so far.

Initialization methods for optimization of neural architectures

Good initialization of network structure and adaptive parameters may bring the neural model close enough to the global minimum to make the local minimization techniques sufficient for finding an optimum solution. Recently we have proposed several initialization methods based on clusterization [7] and statistical discriminant analysis [8]. The method works for single or more hidden layers and could be used with various parameters for initial clusterization (or simply to provide multistart parameters (one may also add some random numbers to the proposed initial weights) and

to set up different structures of networks. What is of primary importance is that – as already has been mentioned – in small networks with small number of parameters it is very hard to find globally optimal set of weights and therefore if multistart gradient methods are used a good initialization is needed. The multistart gradient method is relatively fast comparing to most global minimization methods and with a proper starting point may be an alternative to global minimization.

The methods presented in [7, 8, 9] are easily extended to any architecture containing in the first hidden layer sufficient number of the hidden neurons to account for all the clusters; parameters of other layers should be set up in such a way that the second hidden layer is treated as output nodes and further layer just pass information. In short our minimal architecture is embedded in more complex architecture, and all extra connections have small random weights, while the output from the embedded three-layered network is passed to the final output through the extra hidden layers. In this way the same initialization by prototypes may be used in more complex architectures. Since the initial network should be similar to the network with globally optimal architecture and parameters multistart local optimization of such networks may be an inexpensive, but interesting alternative to global minimization. The weights may be arbitrarily large (the norm of the weight simply changes the slope of the sigmoidal function), and the resulting networks are quite small. The results of optimization through initialization have not yet been compared with those obtained by optimization using global minimization techniques.

14 REMARKS ON NEW GLOBAL OPTIMIZATION METHODS

Some proposals for trying different combination of various global optimization methods were already described in the text. Here a few more methods worth trying in neural networks context are described. Quite a lot of work should still be done in this area before we will understand the weak and the strong points of different methods in real-life applications. Books on global optimization [17] contain descriptions of more methods that could potentially be useful for optimization of architectures or error functions in neural networks. Hybrid gradient-global methods are especially worth developing.

Some modifications of the existing methods that have not yet been used for neural networks are listed below:

1. From the formal point of view genetic algorithms define specific prescription allowing to make changes of adaptive parameters, and therefore they may be combined with the extension of Monte Carlo approach proposed by Dittes [23].

2. Selecting up to K contributions to the error function from randomly selected training vector and its $K-1$ nearest neighbors is a modification of the Dittes Monte Carlo procedure [23]. It may be used in any SA scheme.

3. An interesting possibility is the combination of simulated annealing with genetic algorithms, called GA-SA [24]. GA are used to speed up the SA in this case. Parallelized recombinative simulated annealing (PRSA) has been described by Goldberg [144].

4. Parallel multi-simulated annealing procedure (a 'population' of SA runs created during a single run) may be used for initialization of the gradient descent searches around promising values found after a fixed number of function evaluations. The list of hyperboxes containing the local minima found by gradient procedure should be kept to avoid repetitions.

5. SA or GA may be combined with "rough quantization" approach, i.e. at the beginning only a few bits per weight are allowed and the changes are relatively large since they have at least the size corresponding to the flipping of the least significant bit. For example, a sign plus a single bit per weight gives the possibility to have 3 values, $0, \pm 1$ and in our experience this may already be sufficient to get quite good results [41]. Annealing is than equivalent to increasing the resolution of parameters around the most promissing minima.

6. The Alopex algorithm may also be combined with the "rough quantization" approach described above.

7. Numerical gradient techniques may be used in GA or SA optimization but so far have never been used for neural optimization.

15 FINAL REMARKS AND COMPARISON OF SOME RESULTS

In our survey we have included many global optimization algorithms, but more methods have been described in the mathematical literature. Not all of them are suitable for optimization of neural systems. Cutting plane, successive approximation and successive partition methods are generally applied to concave minimization [145], but it is not clear how to apply these methods in context of neural systems, therefore they were omitted here. A few global optimization methods have already been tried in the context of neural networks, usually with very good results.

Several challenging test problems for global optimization algorithms are known. In some of the test problems the number of local minima grows exponentially large. Ingber and Rosen [38] have compared SA with genetic algorithms for functions containing as many as 10^{50} local minima. However, since non-linear optimization problems differ significantly what works well for some problems may not work at all for other problems. Since there is little or no experience for most of these methods in context of neural systems it is hard to say which one is preferable. To compare different minimization methods and methods that optimize network architecture a series of benchmarks for neural systems would be useful. Simple benchmarks that allow to observe scaling properties of algorithms for problems of different size include:

1. Parity problem for growing number of bits, which may be solved efficiently if a specific architecture is evolved.

2. Noisy XOR problem, in which the output $y = x_1.XOR.x_2$, and the rest of inputs are given a zero-mean noise. The network should evolve to the simplest XOR two-input structure.

3. Except for redundant inputs in the noisy XOR problem linearly dependent inputs and non-linear dependencies among inputs may be introduced for test purposes.

4. Hypercube classification problem allowing to test the scaling properties of various classifiers [146].

Unfortunately no systematic comparison of these methods is available. Ingberg [24, 38] gives some comparison of ASA with GA on a suit of test problems that are commonly used to test genetic approaches. In all cases ASA has outperformed GA. Another comparison was made with GENECOP program of Michalewicz on a set of "Colville problems", for which ASA again performed better. Comparing ASA with other simulated annealing approaches also showed the superiority of this approach. However, since ASA has many parameters and GA have also many parameters it is too soon to conclude that ASA is a better method. Tabu search has been applied to various problems [43]. Some

other interesting methods used for optimization include dynamic hill climbing (DHC), which seems to give similar convergence as the simulated quenching approach [24]. GA compared to the multistart gradient methods are rarely more efficient [147, 148]. RasID algorithm has been used so far only for one real problem (gasoline blending problem [48]) outperforming slightly backpropagation with momentum and adaptive learning rates.

The NOVEL results were compared [20] with a number of other minimization methods in application to the two-spiral and a few other problems. This comparison included: simulated annealing (SIMANN from Netlib library, but not with Adaptive SA); two genetic algorithms GENOCOP (due to Michalewicz [11]) and LICE (due to Sprave [151]); GRAD-MS using multiple random initial points followed by gradient descent; TN-MS, truncated Newton method with multistarts, and Cascade Correlation constructive neural algorithm with random initialization. Best results obtained with no more than 20 hours of computations on Sun SS 20/71 workstation were given [20] for 3 to 6 hidden units, or 18-42 adaptive parameters. In all cases NOVEL algorithm achieved the best results, with 80-100% correct results on the test set, followed closely by the SIMANN approach. We may conclude that application of ASA to the same problem should do better since ASA is a significant improvement over standard simulated annealing. TN-MS achieved third-best results, although for 6 hidden neurons test results were below 90%. Cascade Correlation came fourth, with about 75% correct answers for 6-hidden unit test case, but only 20% for 3 hidden unit, compared to about 80% achieved by NOVEL and SIMANN. Genetic algorithms achieved the worst results, below 60% in all cases, being unable to find good solutions. NOVEL has also been tried on Sonar, Vovel, 10-parity and NetTalk datasets from the UCI repository [13], using different number of hidden units, achieving very good results on the test sets, and falling behind TN-MS only in one case.

From these few comparisons scattered in the literature one can conclude that genetic algorithms, most frequently combined with neural systems, are usually not the best solution to the global minimization problem and to the optimization of neural architectures. There is still little experience with other methods and many variants of global minimization have not yet been tried. Application of global minimization techniques to neural systems will undoubtedly be quite important in near future and we do hope that our survey will motivate more active research on this topic.

REFERENCES

[1] C. Bishop, *Neural networks for pattern recognition*. Clarendon Press, Oxford, 1995.

[2] S. Haykin, *Neural networks: a comprehensive foundations*. MacMillian, 1994.

[3] J. Zurada, *Introduction to artificial neural systems.* West Publishing Company, St Paul, 1992.

[4] B.D. Ripley, *Pattern recognition and neural networks.* Cambridge University Press, 1996.

[5] R. Rojas, *Neural networks. A systematic introduction.* Springer, 1996.

[6] M.H. Hassoun, *Fundamentals of artificial neural networks.* MIT Press, 1995.

[7] W. Duch, R. Adamczak and N. Jankowski, Initialization and optimization of multilayer perceptrons, In *Third Conference on Neural Networks and Their Applications*, pp. 99-104, Kule, Poland, Oct. 1997; Initialization of adaptive parameters in density networks, ibid, pp. 105-110.

[8] W. Duch and R. Adamczak, Statistical methods for construction of neural networks. In *International Congress on Neural Information Processing*, pp. xxx-yyy, Kitakyushu, Japan, Oct. 1998.

[9] W. Duch, K. Grudziński and G.H.F. Diercksen, Minimal distance neural methods. In *World Congress of Computational Intelligence*, pp. 1299-1304, Anchorage, Alaska, IJCNN'98 Proceedings, May 1998.

[10] J. Schmidhuber and S. Hochreiter, Guessing can outperform many long time lag algorithms. *Technical Note* IDSIA-19-96, 1996.

[11] Z. Michalewicz, *Genetic algorithms+data structures=evolution programs*, 3rd ed, Springer, Berlin, 1996.

[12] S. Kirkpatrick, C.D. Gellat Jr. and M.P. Vecchi, Optimization by simulated annealing, *Science*, 220: 671-680, 1983.

[13] C.J. Mertz and P.M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/pub/machine-learning-databases.

[14] W. Schiffman, M. Joost and R. Werner, Comparison of optimized backpropagation algorithms, In *Proc. of the European Symposium on Artificial Neural Networks*, pp. 97-104, Brussels, 1993.

[15] W. Duch, R. Adamczak and K. Grąbczewski, Extraction of logical rules from backpropagation networks. *Neural Processing Letters* 7: 1-9, 1998.

[16] W. Duch and N. Jankowski, *New neural transfer functions*, Applied Mathematics and Computer Science, 7: 639-658, 1997.

[17] R. Horst and P.M. Pardalos (eds), *Handbook of global optimization*, Kluwer, Dodrecht 1995; J.D. Pinter, *Global optimization in action*, Kluwer, Dodrecht 1996.

[18] E. Barnard, Optimization for training neural nets. *Transactions on Neural Networks*, 3(2):232–240, 1992.

[19] P. P. van der Smagt, Minimization methods for training feed-forward networks. *Neural Networks*, 7(1): 1-11, 1994.

[20] Y. Shang and B.W. Wah, *Global optimization for neural network training*, IEEE Computer, 29: 45-54, 1996.

[21] S.E. Fahlman and C. Lebiere, The Cascade-Correlation learning architecture, In *Advances in Neural Information Processing Systems*, vol. 2, Morgan Kaufmann, pp. 524-532, 1990.

[22] W. Duch, R. Adamczak, K. Grąbczewski and G. Żal, Hybrid neural-global minimization method of logical rule extraction. *Journal of Advanced Computational Intelligence*, 1998 (in print).

[23] F-M. Dittes, *Optimization of rugged landscapes: a new general purpose Monte Carlo approach*, *Physical Review Letters*, 76: 4651-4655, 1996.

[24] L. Ingberg, Simulated annealing: Practice versus theory, *J. Math. Computer Modeling*, 18: 29-57, 1993; Adaptive simulated annealing (ASA): Lessons learned, *J. Control and Cybernetics*, 25: 33-54, 1996.

[25] L. Ingber, Very fast simulated re-annealing. *Mathematical Computer Modeling* 12: 967-973, 1989.

[26] L. Herault, Rescaled simulated annealing. In *World Congress of Computational Intelligence*, pp. 1239-1244, IJCNN'98 Proceedings, Anchorage, Alaska, May 1998.

[27] J. Engel, Teaching feed-forward neural networks by simulated annealing. *Complex Systems* 2:641-648, 1988.

[28] D.H. Ackley, G.E. Hinton, T.J. Sejnowski, A learning algorithm for Boltzmann machines, *Cognitive Science*, 9: 147-169, 1985.

[29] J.L. McClelland and D.E Rumelhart, *Explorations in parallel distributed processing: computational models of cognition and perception.* The MIT Press, Cambridge, MA 1986.

[30] R.M. Neal, *Bayesian learning in neural networks.* Lecture Notes in Statistics vol. 118, Springer, 1996

[31] C. Peterson, J.R. Anderson, Neural networks and NP-complete optimization problems: a performance study on the graph bisection problem, *Complex Systems*, 2:59-89, 1988.

[32] A. L. Yuille and J. J. Kosowsky, Statistical physics algorithms that converge. *Neural Computation*, 6(3):341-356, 1994.

[33] Z. He, C. Wu, J. Wang and C. Zhu, A new vector quantization algorithm based on simulated annealing. In *Proc. of 1994 Int. Symp. on Speech, Image Processing and Neural Networks*, Vol. 2:654-657, 1994.

[34] K. Valkealahti and A. Visa, Simulated annealing in feature weighting for classification with learning vector quantization. In *Proc. 9th Scandinavian Conference on Image Analysis*, 2: 965-971, 1995.

[35] H-S. Heon and S-L. Whan, LVQ combined with simulated annealing for optimal design of large-set reference models, *Neural Networks*, 9(2): 329-336, 1996.

[36] U. Kjærulff, Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2: 7-17, 1992.

[37] C. Luonan and K. Aihara, Chaotic simulated annealing by a neural network model with transient chaos, *Neural Networks*, 8(6): 915-930, 1995.

[38] L. Ingberg, B. Rosen, Genetic algorithms and very fast simulated reannealing: a comparison. *Journal of Mathematical and Computer Modelling*, 16: 87-100, 1992.

[39] K.P. Unnikrishnan, K.P. Venugopal, Alopex: a correlation-based learning algorithm for feedforward and recurrent neural networks, *Neural Computations*, 6: 469-490, 1994.

[40] S. Thrun *et al., The MONK's problems. A performance comparison of different learning algorithms.* Carnegi Mellon University, Technical Report CMU-CS-91-197, 1991.

[41] W. Duch, R. Adamczak and K. Grąbczewski, Extraction of logical rules from training data using backpropagation networks. In *The First Online Workshop on Soft Computing*, pp. 25-30, Aug. 1996; also available at http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/; W. Duch, R. Adamczak, K. Grąbczewski, Constrained backpropagation for feature selection and extraction of logical rules. In *First Polish Conference on Theory and Applications of Artificial Intelligence*, pp. 163-170, Łódź, Poland 1996.

[42] R. Battiti and G. Tecchiolli, The reactive tabu search, *ORSA Journal on Computing* 6: 126-140, 1994; Reactive search, a history-sensitive heuristics for MAX-SAT, *ACM Journal of Experimental Algorithmics*, 2, paper 2, 1997.

[43] D. Cvijović and J. Klinowski, Taboo search: an approach to the multiple minima problem, *Science*, 267: 664-666, 1995.

[44] R. Battiti and G. Tecchiolli, Training neural nets with the reactive tabu search, *Transactions on Neural Networks*, 6: 1185-1200, 1995.

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

[46] A.H.G. Rinnooy Kan and G.T. Timmer, A stochastic approach to global optimization, *American Journal of Mathematics and Management Sciences*, 4: 7-40, 1984.

[47] N. Baba, A new approach for finding the global minimum of error function of neural networks, *Neural Networks*, 2: 367-373, 1989.

[48] N. Baba, Y. Moogami, M.A. Kohzaki, Y. Shiraishi and Y. Yoshida, A hybrid algorithm for finding the global minimum of error function of neural networks and its applications, *Neural Networks* 7: 1253-1265, 1994.

[49] Qi-L. Liang, Z. Zhou and Z-M. Liu, A new approach to global minimum and its applications in blind equalization, In *Proc. of International Conference on Neural Networks*, pp. 2113-2117, 199?.

[50] R.A. Rosario *et al.*, A rapid multi-layer perceptron training algorithm, *Proc. of International Conference on Neural Networks*, pp. 824-829, Baltimore, Maryland, USA, 1992.

[51] B. Orsier, *Another hybrid algorithm for finding a global minimum of MLP error fucntions*. University of Geneva, Technical Report UNIGE-AI-95-6

[52] J. Chao, W. Ratanasuwan and S. Tsuji, A new global optimization method: "Rolling-Stone Scheme" and its applications to supervised learning of multi-layered perceptrons, In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks*, Vol.2, pp. 395-398, North-Holland, Amsterdam 1992.

[53] J.T.-H. Lo, *A new approach to global optimization and its application to neural networks*, In *Proc. of International Joint Conference on Neural Networks*, pp. 600-602, Baltimore 1992.

[54] More information on the P^*_{SCG} method and the implementation of the method in the SNNS simulator may be found at: http://cui.unige.ch/AI-group/staff/orsier.html

[55] Z. Tang and G.J. Koehler, Deterministic global optimal fnn training algorithms, *Neural Networks*, 7: 301-311, 1994.

[56] J. Hu, K. Hirasawa, J. Murata, RasID - random search method for neural network training. *Journal of Advanced Computational Intelligence*, 2(4): 134-141, 1998.

[57] N. Baba, T. Shoman and Y. Sawaragi, A modified convergence theorem for random optimization method. *Information Sciences*, 13: 159-166, 1977.

[58] L. Piela, J. Kostrowicki and H.A. Szeraga, The multiple-minima problem in conformational analysis of molecules. Deformation of the potential energy hypersurface by the diffusion equation method. *Journal of Physical Chemistry*, 93: 3339-3346, 1989.

[59] C. Simmerling and R. Elber, Hydrophobic "collapse" in a cyclic hexapeptide: computer simulations of CHDLFC and CAAAC in water, *Journal of American Chemical Society*, 116, 2534-2547, 1994.

[60] A. Roitberg and R. Elber, Modeling side chains in peptides and proteins: application of the locally enhanced sampling and the simulated annealing methods to find minimum energy conformations, *Journal of Chemical Physics*, 95: 9277-9287, 1991.

[61] K.A. Olszewski, L. Piela and H.A. Scheraga, Mean Field Theory as a tool for intramolecular conformal optimization. 1. Tests on terminally-blocked alanine and met-enkephalin, *Journal of Physical Chemistry*, 96: 4672-4676, 1992.

[62] P.H. Winston, *Artificial intelligence, 3rd ed*, Addison Wesley, 1995.

[63] R.J. van Iwaarden, *An improved unconstrained global optimization algorithm*, PhD thesis in applied mathematics, University of Colorado, 1996.

[64] E. Hansen, *Global optimization using interval analysis*, Dekker, New York 1992.

[65] H. Ishibuchi, H. Tanaka and H. Okada, An architecture of neural networks with interval weights and its application to fuzzy regression analysis. *Fuzzy Sets and Systems* 57:27-40, 1993.

[66] L. Eshelman, editor, Proc. of the Sixth Intern. Conf. on Genetic Algorithms and Their Applications, Morgan Kaufmann, 1995.

[67] D.E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.

[68] R.F. Albrecht, C. R. Reeves and N. C. Steele (eds), *Artificial neural nets and genetic algorithms*, Springer Verlag, 1993.

[69] ICANNGA – *Proc. of the Intern. Conference on Artificial Neural Networks and Genetic Algorithms*, 1993, 1995, 1997.

[70] Y. Le Cun, *Modéles connexionistes de l'apprentissage.* PhD thesis, Paris, 1987.

[71] D.E. Rumelhart, G.E. Hinton and R.J. Wiliams, Learning internal representations by error propagation. *Parallel Distributed Processing.* MIT Press, vol. 1: 318-362, 1986.

[72] D.M. Skapura, *Building neural networks*, Addison-Wesley, 1996.

[73] D.B. Fogel, L.J. Fogel and V.M. Porto, Evolving neural networks, *Biological Cybernetics* 63: 487-493, 1990.

[74] J. Korczak and E. Dizdarevic, Genetic search for optimal neural network, In *Conf. On Neural Networks and Their Applications*, pp. 30-45, Kule, Poland, Oct. 1997.

[75] W. Schiffmann, M. Joost and R. Werner, Application of genetic algorithms to the construction of topologies for multilayer perceptrons, In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pp. 675-682, 1993.

[76] X. Yao, A Review of evolutionary neural networks, *International Journal of Intelligent Systems*, 8(4): 539-567, 1993.

[77] J.W. Boers and H. Kuiper, *Biological metaphors and the design of modular artificial neural networks.* Masters Thesis, Departments of Computer Sciences and Experimental and Theoretical Psychology. Leiden University, Netherlands, 1992.

[78] D.J. Chalmers, The Evolution of learning: an experiment in genetic connectionism, In *Proceedings of the 1990 Connectionist Models Summer School*, Morgan Kaufmann, pp. 81-90, 1990.

[79] S.A. Harp, T. Samad and A. Guha, Toward the genetic synthesis of neural networks, *Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 360-369, 1989.

[80] M. Mandischer, Representation and evolution of neural networks, In R.F. Albrecht, C.R. Reeves and U.C. Steele (eds), *Artificial Neural Nets and Genetic Algorithms*, pp. 643-649, 1993.

[81] V. Maniezzo, Genetic evolution of the topology and weight distribution of neural networks, *IEEE Transactions on Neural Networks*, 1(5): 39-53, 1994.

[82] G.G. Miller, P.M. Todd and S.U. Hedge, Designing neural networks using genetic algorithms, In J. Schaffer (ed.), *Proc. of the Third Intern. Conference on Genetic Algorithms and Their Applications*, Morgan Kaufmann, pp. 379-384, 1989.

[83] D.J. Montana, Neural network weight selection using genetic algorithms, In S. Goonatilake, S. Khebbal, editors, *Intelligent Hybrid Systems*, pp.85-104, Wiley, New York, 1995.

[84] R. Salustowicz, *A Genetic algorithm for the topological optimization of neural networks*, Diplomarbeit TU Berlin, 1995.

[85] D.J. Schaffer, D. Whitley and L. Eshelman, Combinations of genetic algorithms and neural networks: a survey of the state of the art, In *Proc. of the Conf. on Combination of Genetic Algorithms and Neural Networks*, pp. 1-37, 1992.

[86] B. Sendhoff and M. Kreutz, Evolutionary optimization of the structure of neural network using a recursive mapping as encoding, In *Proc. of the 3rd International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer Verlag, 1997.

[87] D. W. Pearson, N. C. Steele and R. F. Albrecht, Artificial neural nets and genetic algorithms. In *Proceedings of the International Conference*, Springer-Verlag, 1995.

[88] D.J. Schaffer, R.A. Caruana and J. Eshelmani, Using genetic search to exploit the emergent behavior of neural networks, In S. Forest, editor, *Emergent Computation*, North Holland, pp. 244-248, 1990.

[89] F.Z. Brill, D.E. Brown and W.N. Martin, Fast genetic selection of features for neural network classifiers, *Transactions on Neural Networks*, 3(2): 324-328, 1992.

[90] G.W. Game and C.D. James, The application of genetic algorithms to the optimal selection of parameter values in neural networks for attitude control systems, In *IEE Colloquium on 'High Accuracy Platform Control in Space'*, pp. 3/1-3/3, Digest No. 1993/148, IEE, London, 1993.

[91] D.L. Prados, New learning algorithm for training multilayer neural networks that uses genetic-algorithm techniques, *Electronics Letters*, 28(16): 1560-1561, 1992.

[92] M. Srinivas and L.M. Patnaik, Learning neural network weights using genetic algorithms - improving perfromance by search-space reduction, In *International Joint Conference on Neural Networks*, Vol. 2, pp. 187-192, 1991.

[93] D. Whitley, T. Starkweather and C. Bogart, Genetic algorithms and neural networks: optimizing connections and connectivity, *Parallel Computations*, 14(3): 347-361, 1990.

[94] B. Zhang and G. Veenker, Neural networks that teach themselves through genetic discovery of novel examples, In *Proceedings of the International Joint Conference on Neural Networks*, pp. 690-695, 1991.

[95] P. Arena, R. Caponetto, I. Fortuna and M. G. Xibilia, MLP optimal topology via genetic algorithms, In R.F. Albrecht, C.R. Reeves and N.C. Steele, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag, pp. 670-674, 1993.

[96] R.K. Belew, J. McInerney, and N.N. Schraudolph, Evolving network: using the genetic algorithm with connectionist learning, In C.G. Langton, C. Taylor, J.D. Farmer and S. Rasmussen, editors, *Artificial Life II*, Redwood City, CA:Addison-Wesley, 1991.

[97] H. Kwasnicka and P. Szerszon, NETGEN - evolutionary algorithms in designing artificial neural networks, In *Conf. On Neural Networks and Their Applications*, Kule, Poland, pp. 671- 676, 1997.

[98] P. Robbins, A. Soper and K. Rennolls, Use of genetic algorithms for optimal topology determination in back propagation neural networks, In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pp. 726-730, 1993.

[99] J.R. Koza and J.P. Rice, Genetic generation of both the weights and architecture for a neural network, In *Proc. of Intern. Joint Conference on Neural Networks*, 1991.

[100] S.G. Romaniuk and L.O. Hall, SC-net: a hybrid connectionist symbolic system, *Information Sciences*, 71: 223-268, 1993.

[101] J. Azevedo, *AGWIN - Manual d'utilisateur*, http://lsiit.u-strasbg.fr, Technical Report, LSIIT, Louis Pasteur University, Strasbourg, 1996.

[102] E. Blindauer, *Méthodes d'encodage génétique de réseaux neuronaux*, MSc thesis in Computer Science, Louis Pasteur University, Strasbourg, 1998.

[103] E. Dizdarevic, *Optimisation génétique de réseaux*, MSc thesis in computer science, Louis Pasteur University, Strasbourg, 1995.

[104] J. Korczak and E. Dizdarevic, *Genetic optimization of neural networks*, Technical Report, CRI, Louis Pasteur University, Strasbourg, 1994.

[105] The bibliography on evolutionary design of neural architectures: http://liinwww.ira.uka.de/bibliography/Neural/edna.html

[106] L. Marti, Genetically generated neural networks, I. Representational effects, In *Proc. of the Intern. Joint Conference on Neural Networks*, Vol. 4, pp. 537-542, 1992.

[107] D. Elizondo, *The recursive deterministic perceptron and topology reduction strategies for neural networks*, PhD thesis, Louis Pasteur University, Strasbourg, 1997.

[108] E. Fiesler, Comparative bibliography of ontogenic neural networks, In *Proceedings of the International Conference on Artificial Neural Networks*, Springer Verlag, pp. 793-796, 1994.

[109] T. Ash, *Dynamic node creation in backpropagation networks*, Technical Report, Institute of Cognitive Science, University of California, 1989.

[110] J.P. Nadal, Study of a growth algorithm for a feedforward neural network, *International Journal of Neural Systems*, pp. 55-59, 1989.

[111] M. Hagiwara, Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection, In *Proceedings of the International Joint Conference on Neural Network*, Edward Brothers, pp. 625-630, 1990.

[112] Y. Hirose, K. Yamashita, S. Hijaya, Back-Propagation algorithm which varied number of hidden units, *Neural Networks*, 4(1): 61-66, 1991.

[113] V. Honavar and L. Uhr, A Network of neuron-like units that learns to perceive by generation as well as reweighting of its links, In *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, pp. 472-484, 1988.

[114] M. Frean, The Upstart algorithm: a method for constructing and training feedforward neural networks, *Neural Computation*, 2: 198-209, 1990.

[115] B. Bonnlander and M.C. Mozer, Latticed RBF networks: an alternative to constructive methods, In *Advances in Neural Information Processing Systems*, vol. 5, Nature and Synthetic, 1993.

[116] Y. Chauvin, A Back-Propagation algorithm with optimal use of hidden units, In *Advances in Neural Information Processing Systems*, vol. 1, Morgan Kaufmann, pp. 519-526, 1989.

[117] N. Kadaba and N.E. Nygard, Improving the performance of genetic algorithms in automated discovery of parameters, In *Proceedings of the Seventh International Conference on Machine Learning*, Morgan Kaufmann, pp.140-148, 1990.

[118] N. Kadaba, N.E. Nygard and P.L. Juell, Integration of adaptive machine learning and knowledge-based systems for routing and scheduling applications, *Expert Systems and Applications*, pp. 15-27, 1991.

[119] D. Murray, Tuning neural networks with genetic algorithms, *AI Expert*, June 1994.

[120] J.D. Kelly and L. Davis, Hybridizing the genetic algorithms and the k-nearest neighbors classification algorithms, In *Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 377-383, 1991.

[121] E.J. Chang and R.P. Lippman, Using genetic algorithms to improve pattern classification performance, In *Advances in Neural Information Processing*, Vol. 3, Morgan Kaufmann, pp.797-803, 1991.

[122] T. Drabe, W. Bressgott and E. Bartscht, Genetic task clustering for modular neural networks. In *Proc. of Int. Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing*, NICROSP 1996, pp. 339-347.

[123] Z. Guo, R.O. Uhrig, Use of genetic algorithms to select inputs for neural networks, In *Proc. on Combination of Genetic Algorithms and Neural Networks*, pp.223-234, 1992.

[124] S.A. Billings and G. L. Zheng, Radial Basis Function network configuration using genetic algorithms, *Neural Networks* 8(6): 877-890, 1995.

[125] B. Burdsall and C. Giraud-Carrier, GA-RBF: A Self-optimizing RBF network, In *Proceedings of the Third International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97)*, Springer-Verlag, pp. 348–351, 1997. URL: http://www.cs.bris.ac.uk/Tools/Reports/Bibtexs/1997-burdsall-0.bib

[126] M. Zamparelli, Genetically trained cellular neural networks, *Neural Networks* 10(6): 1143-1151, 1997.

[127] D. W. Opitz and J. W. Shavlik, Genetically refining topologies of knowledge-based neural networks, In *International Symposium on Integrating Knowledge and Neural Heuristics*, pp. 57-66, 1994.

[128] V.W. Porto, D.B. Fogel and L.J. Fogel, Alternative neural network training methods, *IEEE Expert*, pp. 16-21, June 1995.

[129] P.J. Angeline, G.M. Saunders and J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, *Transactions on Neural Networks*, 5: 54-65, 1994.

[130] X. Yao and Y. Lin A new evolutionary system for evolving artificial neural networks, *Transactions on Neural Networks*, 8(3): 694-713, 1997.

[131] S. Smolander and J. Lampinen, Determining the optimal structure for multilayer self-organizing map with genetic algorithm. In J. Parkkinen and A. Visa, editors, *Proc. of the 10th Scandinavian Conference on Image Analysis*, pp. 411-417, 1997.

[132] H. Mühlenbein, Limitations of multilayer perceptron networks - steps towards genetic neural networks, *Parallel Computing*, 14: 249-260, 1990.

[133] H. Kitano, Designing neural networks using genetic algorithms with graph generation system, *Complex Systems*, 4: 461-476, 1990.

[134] F. Gruau Genetic synthesis of modular neural networks, In S. Forrest, editor, *Genetic Algorithms: Proceedings of the 5th International Conference*, Morgan Kaufman, 1993.

[135] J.E. Baker, Reducing bias and inefficiency in the selection algorithm. In J.Grefenstette, editor, *Proc. of the Second Intern. Conference on Genetic Algorithms*, Los Altos, Morgan Kaufmann, pp. 14-21, 1987.

[136] D. Beasley, D.R. Bull, R. Martin, An overview of genetic algorithms: Part 1, Fundamentals; Part 2, Research topics. *University Computing*, 1:(2-4), 58-69; 170-181, Cardiff, 1993.

[137] T. Blickle, L. Thiele, *A comparison of selection schemes used in evolutionary algorithms*, Technical Report, ETH Zurich, 1997.

[138] F. Gruau, *Neural networks synthesis using cellular encoding and the genetic algorithm*, PhD thesis, LIP, Ecole Normale Superieure, Lyon, 1992.

[139] J. Korczak *et al.*, ECO. Research Report, LSIIT, Louis Pasteur University, Strasbourg, 1997.

[140] R. Reed, Pruning algorithms - a survey, *IEEE Transactions on Neural Networks* 4(5): 740-746, 1993.

[141] M. Ishikawa, Structural learning with forgetting, *Neural Networks* 9: 509-521, 1996.

[142] D.J. MacKay, A practical Bayesian framework for backpropagation networks, *Neural Computations* 4: 448-472, 1992.

[143] P. Cheesman and J. Stutz, Bayesian classification (AutoClass): theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, editors, *Advances in Knowledge discovery and Data Mining*, pp. 153-180, MIT Press 1996.

[144] D.E. Goldberg, *A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing, Complex Systems* 4: 445-460, 1990.

[145] R. Horst and H. Tuy, *Global optimization*, Springer Verlag, 1990.

[146] W. Duch, Scaling properties of neural classifiers. In *Third Conference on Neural Networks and Their Applications*, pp. 189-194, Kule, Poland, October 1997.

[147] M. Mitchell, J.H. Holland and S. Forrest, When will a genetic algorithm outperform hill climbing, *Advances in Neural Information Processing Systems*, Morgan Kaufmann Publishers, Vol. 6: 51-58, 1994.

[148] A. Juels and M. Wattenberg, Stochastic hillclimbing as a baseline method for evaluating genetic algorithm. *Advances in Neural Information Processing Systems*, MIT Press, Vol 8: 430-436, 1996.

[149] J. Kennedy and R.C. Eberhart, Particle swarm optimization, In *Proc. of the IEEE International Conf. on Neural Networks*, Piscataway, New Jersey, 1995, Vol. 4, pp. 1942-1948.

[150] R.C. Eberhart and Y. Shi, Particle swarm optimization, In *Proc. of the International Conf. on Neural Networks and the Brain*, Beijing, China, 1998, pp. PL5-PL13.

[151] J. Sprave, Linear neighborhood evolution strategies, In *Proc. of the 3-rd Annual Conf. on Evolutionary Programming*, San Diego, CA, World Scientific 1994.