Complex Systems **6** (1992) 459–477

A Distributed Genetic Algorithm for Neural Network Design and Training

S. Oliker
M. Furst
Dept. of Electrical Engineering–Systems,
Faculty of Engineering, Tel Aviv University,
Ramat Aviv 69978, Israel

O. Maimon
Dept. of Industrial Engineering,
Faculty of Engineering, Tel Aviv University,
Ramat Aviv 69978, Israel

Abstract. A new approach for designing and training neural networks is developed using a distributed genetic algorithm. A search for the optimal architecture and weights of a neural network comprising binary, linear threshold units is performed. For each individual unit, we look for the optimal set of connections and associated weights under the restriction of a feedforward network structure. This is accomplished with the modified genetic algorithm, using an objective function—fitness—that considers, primarily, the overall network error; and, secondarily, using the unit's possible connections and weights that are preferable for continuity of the convergence process. Examples are given showing the potential of the proposed approach.

1. Introduction

Interest in neural networks has increased tremendously in recent years. Much of this renewed interest has been motivated by the development of supervised learning algorithms that were applied for different purposes, especially for classification tasks. However, none of the existing algorithms offers an analytic solution that shows which neural network should be designed for a given problem [6]. The most common algorithm for training neural networks is backpropagation [14]. In many applications that used backpropagation, an appropriate neural network was found after a reasonable number of iterations [14, 6]. The main drawback of the backpropagation algorithm is that there is nothing in the algorithm that prevents it from becoming trapped in local minima; evidently, in many problems, such as adder and parity [14],

the neural network that is obtained by backpropagation does not give the correct solution.

Several studies used genetic algorithms (GAs) while training neural networks [17, 13, 11, 5, 8, 9]. They chose GAs because they were found to be effective in a wide variety of search and optimization problems that otherwise were trapped in local minima [7, 3]. GAs were included in the training of neural networks in several ways by different studies. Whitley and Hansen [17] and Montana and Davis [13] presented training processes in which GAs were applied for determination of the weights between connected nodes in neural networks. Both studies yield better results when compared to backpropagation, including cases that are often trapped in local minima. Miller and Tod [11], Harp, Samad, and Guha [5], and Kitano [8] presented attempts to produce a process using GAs in which connections between nodes could be constructed or deleted during the training period. However, the network performance was evaluated by means of the conventional backpropagation algorithm. Therefore, the combination of backpropagation and GAs yielded reasonable results only for limited cases.

Using the backpropagation method in training neural networks limits the potential of GAs to avoid entrapment in local minima. Koza [9] suggested a process of neural-network design using GAs for both the structure and the weights, and tested it for an XOR problem. These attempts were practical only for networks with a limited number of units because the search space increased exponentially [8].

A different approach for building a feedforward layered network was presented by Mézard and Nadal, using the tiling algorithm for any given Boolean function [10]. The architecture is not fixed in advance, but is generated by a growth process that adds layers (and units inside a layer) until convergence. Alpaydin's Grow-and-Learn (GAL) algorithm [1] presents a process that also guarantees an error-free training data set, by adding and removing "exemplar units" in order to get the correct classification. The network size obtained in these algorithms might grow more than necessary, and the generalization thus obtained might be unsatisfactory (as Baum and Haussler [2] show).

In the present study, we introduce a distributed genetic algorithm for training neural networks with GAs in which the search for the optimal neural network is done separately for every unit that is a single neuron in the neural network. This search method is one of the major differences between our work and other studies that combine both methodologies. The search space is drastically reduced in comparison to the previous works that consider the entire neural network. The search is done over the whole unit's possible connections and weights in every given state of the network during the process. This gives our approach an advantage over the gradient descent methods—robustness to entrapment in local minima—and therefore expands the range of solvable problems. This method yields a neural-network training algorithm that performs dynamic modification of a network structure and its weights, and at the same time eliminates the need to predetermine the network structure (the number of internal layers and number of units in each

layer, for example). The algorithm iteratively alters connection weights, and either generates or eliminates connections in order to reduce the error of a given network. Nevertheless, when no alteration is found to reduce the network's error in an iteration, each unit is examined to see whether replacing it by optional unit inputs would help improve the convergence process to the global minimum. For this purpose the algorithm defines for each unit a group of strings used as optional unit inputs, which is called "the unit genotype population." This group of strings is revised with the GA operators at the end of each iteration.

2. The Algorithm Description

2.1 Problem Statement

Let us define $\mathcal{N}$ as a network with a feedforward structure, $\mathcal{N} = \{V, C, W, B\}$, where V, C, W, and B are sets of units, connections, weights, and bias values, respectively. V is the union of three disjoint sets V_N, V_H, and V_M, where V_N includes N units in the input layer, V_H includes H hidden units, and V_M includes M units in the output layer. The training data set is composed of P elements. Each element p is a pair of vectors $\{\vec{X}^p, \vec{D}^p\}$, where $\vec{X}^p = (x_0^p, \ldots, x_{N-1}^p)$ is the input vector, and $\vec{D}^p = (d_0^p, \ldots, d_{M-1}^p)$ is the output vector whose values are defined as $d_i^p = \pm 1$ for $i = 0, \ldots, M-1$.

To accomplish an appropriate neural network for a given task, we start with a network whose numbers of units in the input and output layers are driven by the problem; the initial neural network has an arbitrary number of hidden units with the constraint of a feedforward structure. The purpose of the distributed genetic algorithm presented here is to design a neural network that will yield a minimum error between the given output vectors and those derived by the neural network.

For each training element p, E^p is the error that is obtained by

$$E^p = \frac{1}{2} \sum_{m=0}^{M-1} |d_m^p - a_{N+H+m}^p|, \tag{1}$$

where a_i^p is the output of unit v_i obtained by the NN for input vector $\vec{X}^p$. The output of a unit v_i is defined as

$$a_i^p = \begin{cases} x_i^p & \text{if } v_i \in V_N \\ \operatorname{sgn}\left(\sum_{j=0}^{N+H-1} w_{ij} a_j^p + b_i \right) & \text{otherwise} \end{cases} \tag{2}$$

where w_{ij} is the weight of the connection from unit v_j to unit v_i, and b_i is the bias of unit v_i. The neural network is modified after each iteration according to convergence criteria that will be discussed in section 2.3. The modifications are applied not only to the connection weights (as is usually done in neural network learning algorithms), but also to the neural network

Step 1.	Set arbitrary initial state of network and units genotype populations
Step 2.	Apply the training data set and calculate the network output error and fitness value
Step 3	Calculate fitness values for all the genotypes
Step 4.	Terminate when achieving desired network error or exceeding a maximum number of iterations
Step 5.	Modify the network by using the best genotypes
Step 6.	Create a new generation of genotypes for each unit with the use of GA operators
Step 7.	Modify inputs to isolated units
Step 8.	Repeat by going to Step 2

Figure 1: The neural network distributed genetic algorithm.

structure (by adding and deleting connections between units), which is less common.

The network $\mathcal{N}$ is modified in an iterative process to reduce the current error E, which is the overall neural network mean error for the entire training data set:

$$E = \frac{1}{P} \sum_{p=0}^{P-1} E^p \tag{3}$$

2.2 The neural network distributed genetic algorithm

The distributed genetic algorithm operates separately on each of the hidden and output-layer units. The algorithm specifies a population of genotypes for each of these units. For unit v_i there are K genotypes, where each genotype g_i^k $(k = 1, ..., K)$ includes connections with weights and a bias value. Every genotype maintains the feedforward structure of the network.

The iterative process is composed of eight primary steps, which are summarized in figure 1. A concise version of the algorithm appears in Appendix A.

The algorithm is performed by applying the following steps for a given problem:

Step 1: The algorithm sets the number of the input and output units, an arbitrary number of hidden units, and an initial feedforward structure. K genotypes are assigned randomly for each hidden and output unit.

Step 2: The whole training data set is applied to the neural network by setting the input layer to $\vec{X}^p$, calculating the output of the neural network,

and comparing it to the set $\vec{D}^p$. Two terms are then derived: a fitness value equal to the network error as defined in (3), and an additional term that is determined according to heuristic convergence criteria that are discussed in section 2.3.2.

Step 3: Step 2 is repeated for every genotype of the hidden and output-layer units. The complexity is significantly reduced because of the binary output of the units (for more details, see section 2.3.1).

Step 4: The algorithm stops if the desired network error is obtained or the number of iterations exceeds a maximum number.

Step 5: For every unit, the genotype with the minimum fitness value is chosen as a candidate to replace the current unit. If among all the candidates there is one that yields a minimum error, it replaces the current unit in the NN. If among the candidates there are several units that yield the same minimum error, the replacement of the different units is done sequentially. The replacement is stopped if one of the candidates causes a structural change in the neural network (that is, the addition or deletion of connections).

Step 6: In the reconstructed neural network, new genotypes in the population are assigned to each unit in the hidden layer that has a path to a unit in the output layer, and to units in the output layer. A new population of K genotypes is reproduced from the population of K current genotypes by the genetic algorithm [3] as follows.

1. A pair of genotypes is picked from the current population by the ranking selection method. By using ranking selection in the algorithm the selective pressure and population diversity are better controlled, and premature convergence caused by genotypes with very high fitness ratios is avoided [3, 4, 18]. The ranking is performed according to the fitness value, which allows the definition of a linear distribution function for the genotype selection procedure.

2. A new pair of genotypes are produced by a uniform crossover operation that is applied to the selected pair [3, 15]. The different connections in the reproduced genotypes get their weights with equal probability from one of the parent genotypes.

3. A mutation operation is applied to the new pair of genotypes. This operation includes random changes of weights and bias values, and generation of new possible connections at low probability.

Step 7: Since the algorithm makes dynamic modifications in the neural network structure, some units might become isolated, with no route to the output-layer units and no effect on the neural network output error. After a number of iterations, the connections to such units are modified randomly.

Step 8: Perform the next iteration by going to Step 2.

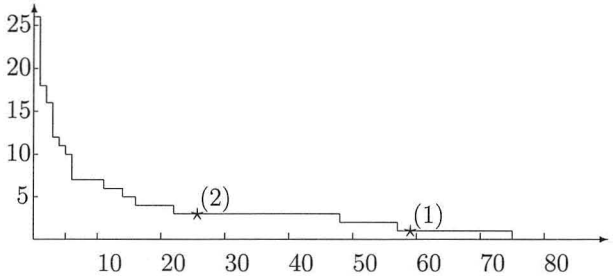

Figure 2: Error versus changes in the network.

This iterative training process dynamically modifies the neural network in order to find a solution. Figure 2 illustrates this behavior for the two-bit adder problem. The network's error is displayed whenever the network is altered. Changes occur both with and without error reduction (as a result of modifications of the network toward a preferable state). This representative example indicates a local minimum entrapment when considering the output error as the only criteria.

Another illustration describes the learning process behavior for a simple, small neural network of the "parity four" problem. (In this problem there are four input units and one output unit. For this example we chose two hidden units.) Six different states of the learning process are shown in figure 3. In the initial state (a) unit 6 is isolated. Following one iteration (b) the neural network was modified so that the connection from hidden unit 5 to output unit 7 was eliminated, thus two hidden units became isolated. This connection was regenerated, however, in a later iteration of the learning procedure (c). The connection between the two hidden units 5 and 6 was generated only following the random modifications of the input connections to the isolated unit 6 (d). Evidently, without this modification the neural network did not converge to a final state. In the final state (f) none of the units were isolated; in comparison to the initial state, two connections had been added, one from hidden unit 6 to output unit 7, and the second between hidden unit 6 and hidden unit 5. Note that the connection between units 6 and 5 was generated only after the connection between these two units in the opposite direction was eliminated (d and e).

2.3 Evaluation of genotype fitness

The fitness value is defined by heuristic considerations, and it consists of two main components. The first relates to the network error for the genotype, and the second expresses the NN convergence ability. The genotype with the smallest fitness value will be favored.

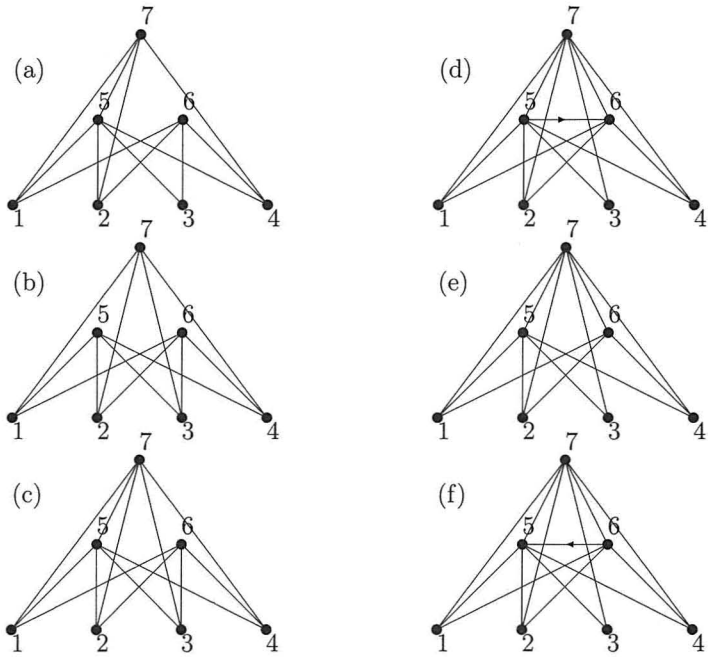

Figure 3: An example of modifications in network structure in the learning phase of a parity-four problem; (a) represents the initial random network, (b–f) the developments on the way to the final network.

2.3.1 Calculation of network error for the genotype

In every iteration one wants to choose the best genotype of all hidden and output units; therefore, the neural network error has to be calculated $(H + M)K + 1$ times. Instead of calculating the network error for each genotype replacement, we take advantage of the fact that the output value of a hidden unit or an output unit can be either $+1$ or -1 (2). As a result, the genotype output can either yield the unit output or its inverse. The genotype error for a given training vector p is thus equal either to E^p (1), or to $\tilde{E}_i^p$—which is calculated as E^p, except that unit v_i output a_i^p is replaced by $-a_i^p$. Therefore, in the beginning of every iteration we obtained only E^p and $H + M$ values for $\tilde{E}_i^p$, and the error of every genotype was then assigned as E^p or $\tilde{E}_i^p$.

2.3.2 Evaluation of genotype influence on network convergence

The network error is not the only criterion for network modifications, as several genotypes might have the same error E; in that case our purpose is to find a genotype that is most likely to decrease the neural network error in succeeding iterations. For every element p in the training data set, a procedure for determining the preferred genotype is applied. The procedure is based on heuristic rules (whose detailed consideration follows) that eventually define a value $f(p)$. The procedure is repeated for every element in the training data set, and the genotype is selected according to the minimum $\sum_{E^p>0} f(p)$.

Let us define an *effective unit* v_i as a unit whose $\tilde{E}_i^p < E^p$. Such a unit might have a genotype that yields an opposite output, which certainly will affect and reduce the error E^p. That genotype will replace its unit if it reduces the overall error E accumulated for the entire learning data set. On the other hand, when the unit is not effective no change (genotype) can improve the present network state. Therefore, we define the first and fundamental heuristic assumption as follows.

Assumption 1. *Increasing the number of effective units in the network improves the convergence process.*

We will add several other assumptions that will enable intermediate states with more effective units, or with other benefit criteria. The subsequent assumptions are expressed in terms of the fitness value that the genetic algorithm accumulates for the entire learning data set, and are verified by the results of the simulations. The proposed algorithm treats effective and non-effective unit's genotypes differently.

Heuristic rules for effective units. Let us define an *inverse connection* as a connection from hidden unit v_j to unit v_i that yields an inverse output of v_i (a_i^p) by substituting $-a_j^p$ for a_j^p in equation (2). If this substitution does not affect v_i output, the connection is considered to be a non-inverse connection.

A *disturbing connection* is a connection between two effective hidden units v_i and v_j, which is a non-inverse connection from v_j to v_i. Such a connection prevents generation of new connections under the feedforward restriction. Our goal in the algorithm is to allow as many structural changes of the neural network as needed, therefore we will prefer a network with a minimum number of disturbing connections to effective units. We can therefore state the following assumption.

Assumption 2. *A genotype of an effective unit v_i with a smaller number of disturbing connections is preferred.*

If such a genotype is not found, we consider the next assumption.

Assumption 3. *Among genotypes having the same number of disturbing connections, the one whose disturbing connections have the lowest weights (w_{ij}) is preferred.*

In the case that effective unit v_i does not have disturbing connections, its genotypes are evaluated according to the *desired-inverse connection.* A desired-inverse connection is a connection from a non-effective hidden unit to an effective unit that is neither an inverse nor an allowed (but nonexistent) connection, according to the feedforward constraint. A non-effective unit might become an effective unit in succeeding iterations if its connection to another effective unit becomes an inverse connection.

According to Assumption 1, we prefer neural networks with the maximum number of effective units; therefore, we shall prefer a genotype from the effective unit pool which has a maximum number of inverse connections from hidden non-effective units, or, in other words, has minimum desired-inverse connections. Thus, we state the next assumption.

Assumption 4. *A genotype of an effective unit v_i with a smaller number of desired-inverse connections is preferred.*

We illustrate the latter assumption by means of the two-bit adder problem. The network architecture in state (1) of figure 2 is displayed in figure 4. In that stage, according to Assumption 4, the process modifies the network in order to establish a preferable state toward convergence (without gaining a direct error reduction). In Figure 4, unit 6 is an effective unit while unit 5 is not; the connection from unit 5 to unit 6 is non-inverse, such that unit 6 has one desired-inverse connection. The algorithm prefers a unit 6 genotype that decreases the number of desired-inverse connections for that unit. In this genotype, the weights of the connections from units 1, 2, and 5 to unit 6 were modified; as a result, the connection between units 5 and 6 becomes an inverse connection. Unit 5 is thereby turned into an effective unit, enabling continuation of the convergence.

In order to differentiate among genotypes of an effective unit that have no disturbing connections, but have the same number of desired-inverse connections, we estimate how far each genotype output is from being inverted.

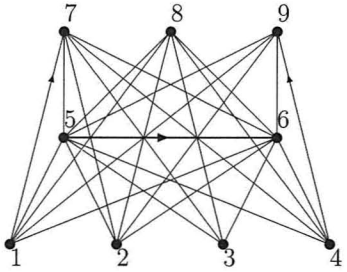

Figure 4: First example of an intermediate network stage.

Among the genotypes that obtained the same error, we shall prefer the one that is the closest to being inverted, since only that genotype whose output is the inverse of the original output has the potential to reduce the error.

To measure how far the output is from being inverted, two parameters must first be defined. The first parameter is β_i^{kp}, which determines how far genotype k is from actually reducing the error E^p (that is, changing unit v_i output from a_i^p to $-a_i^p$). This parameter is defined as

$$\beta_i^{kp} = \left| \sum_{j=0}^{N+H-1} w_{ij}^k a_j^p + b_i^k \right|, \tag{4}$$

where w_{ij}^k are the weights and b_i^k is the bias of genotype k of unit v_i. The second parameter, λ_{ij}^{kp}, evaluates the distance of a desirable-inverse connection from becoming an inverse connection. The second parameter is defined as

$$\lambda_{ij}^{kp} = \left| \sum_{n=0}^{N+H-1} w_{in}^k a_n^p + b_i^k + 2w_{ij}^k(-a_j^p) \right| \tag{5}$$

A genotype with smaller values of β and λ is preferred.

Heuristic rules for non-effective units. Hidden units that are non-effective do not have the potential to reduce the error in the current iteration for a given input. We therefore shall define heuristic rules that will allow as much generation of new connections as possible, thereby increasing the number of effective units in succeeding iterations. The new connections thus generated can contribute to the reduction of the neural network error. Existing connections between hidden units limit the generation of new connections because of the feedforward structure restriction. Let us define an *internal connection* as a connection between hidden units, which allows us to state the following assumption.

Assumption 5. *A genotype with a smaller number of internal connections is preferred.*

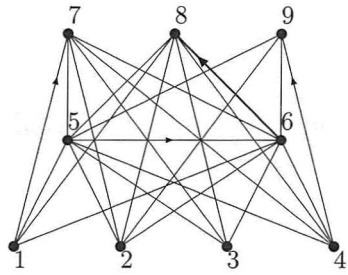

Figure 5: Second example of an intermediate network stage.

If all the genotypes have the same number of internal connections we shall prefer those with the lowest weights (see Assumption 3).

Assumption 6. *Among genotypes having the same number of internal connections, the one with the lowest weight values of internal connections is preferred.*

An output unit is considered to be non-effective when it has the correct output; inverting such a unit increases the error. Any inverse connection from hidden unit v_j might convert correct output. Therefore, we have the next assumption:

Assumption 7. *A non-effective output unit genotype with a smaller number of inverse connections is preferred.*

We again exemplify the latter assumption by means of the two-bit adder problem. The network architecture in state (2) of figure 2 is described in figure 5. There the output of unit 8 is correct and the output of unit 9 is not. Unit 8 in this state is a non-effective output unit. The connection from unit 6 to unit 8 is inverse, and the algorithm prefers a genotype of unit 8 that hass a non-inverse connection from unit 6. Replacing unit 8 changes the weigths from units 2, 3, and 6, as well as the bias; and for one of the patterns with erroneous network output the connection from unit 6 becomes non-inverse. As a consequence any change in unit 6 would not increase the error, and, indeed, the subsequent iteration allows a change in unit 6.

To differentiate among the genotypes of output unit v_i that have the same number of inverse connections, we define a parameter τ_{ij}^{kp}. This parameter evaluates how far the inverse connection is from being non-inverse:

$$\tau_{ij}^{kp} = \left| \sum_{n=0}^{N+H-1} w_{in}^k a_n^p + b_i^k + 2w_{ij}^k(-a_j^p) \right| \tag{6}$$

A genotype with smaller values of τ is preferred.

The overall fitness. Based on the above assumptions, the algorithm updates the genotype fitness value for every training data set. For a given element p in the training data set we define a fitness value for every hidden unit v_i, as follows:

$$f_i^p = \begin{cases} K_1 E^p + K_2 N_i^p + K_3 \eta_i^p & \text{if } v_i \in Y^p \\ K_1 E_i^p + K_2 S_i^p + K_3 w_{S_i^p} & \text{otherwise} \end{cases} \tag{7}$$

where

$$\eta_i^p = \begin{cases} w_{zi} & \text{if } Z_i^p > 0 \\ \min\limits_j \lambda_{ij}^p, \beta_i^p & \text{otherwise} \end{cases}$$

$$N_i^p = \begin{cases} Z_i^p & \text{if } Z_i^p > 0 \\ R_i^p & \text{otherwise} \end{cases}$$

Y^p represents the effective units set, S_i^p the number of internal connections to unit v_i, and $w_{S_i^p}$ the minimal weight of internal connection to unit v_i; $w_{S_i^p} = \min_{w_{ij} \in W} w_{ij}$, $v_i, v_j \in V_H$. Z_i^p represents the number of disturbing connections, and R_i^p the number of the desired-inverse connections.

For an output unit the fitness is defined as

$$f_i^p = \begin{cases} K_1 E^p & \text{if } v_i \in Y^p \\ K_1 E_i^p + K_2 I_i^p + K_3 t_i^p & \text{otherwise} \end{cases} \tag{8}$$

where I_i is the unit v_i inverse connections set, and $t_i^p = \min_j \tau_{ij}^p$. $K_1 \gg K_2 \gg K_3$ are positive constants. The genotype that obtains the minimum of the $\sum_{E^p > 0} f(p)$ is selected.

3. Simulation results

Initial results are presented here for the algorithm that we have proposed. The algorithm was tested in parity, symmetry, and two-bit adder problems. Such problems are used as benchmarks for various neural network training methods [14].

Parity. Extensively discussed in [12] and [14], parity is the most commonly used problem for comparison of neural network training methods. For a binary input vector containing values of $+1$ and -1, the output required is $+1$ if the $+1$ number at the input vector is even, and -1 otherwise. This problem is difficult to solve using a neural network because the output required is different for the closest input vectors (which differ only by a single bit). For a network containing a hidden layer, the minimum number of units in the hidden layer is identical to the number of the input-layer units. Only when additional connections are allowed—for example, between the input and output layers—can the number of hidden units be reduced.

Problem	*Hidden units*	*Median*	*Success rate*
parity 4	4	148	0.98
parity 4	3	198	0.96
parity 4	2	460	0.72
parity 5	5	253	1.00
parity 5	4	203	0.98
parity 5	3	502	0.94
parity 6	6	2380	0.70
parity 7	7	3799	0.60
symmetry 4	2	160	0.92
symmetry 6	2	1788	0.70
2-bit binary adder	2	399	0.54

Table 1: Simulation results. The median is a measure of the number of iterations required for a success rate of 50%.

Symmetry. For this problem, a distinction is made if the input vector shows symmetry with respect to the center. The output required is $+1$ for a state of symmetry and -1 otherwise [14]. This problem, too, has been attempted by various training processes. The minimum solution for a network with a hidden layer is a network containing only two units in that layer.

Two-bit binary adder. In this case, the problem is to find a network that performs summing of two binary numbers with two bits each. This network contains four units in the input layer and three in the output layer, for which summation is required. Minimum implementation is obtained using a network containing only two hidden units. Two-bit adder problems are frequently trapped at local minima when the backpropagation algorithm is used, if hidden units are not added [14].

While simulating the above problems, the following parameters for the genetic algorithm were defined.

- Population size—40 genotypes per unit
- Selective pressure—1.6 (the top ranking genotype in the population is defined as being 1.6 times more likely to reproduce than the average individual in the population)
- Mutation rate—0.1

Convergence was checked in 50 trials for each problem, where every trial began with a random initial state of the neural network(that is, connections and their associate weights). The simulation results are summarized in table 1.

For comparison, in two problems we have tried to obtain convergence with the conventional backpropagation algorithm, without adding hidden units.

(Hidden units are usually added in order to avoid entrapment at local minima [14, 16].)

- Parity 4 with three hidden-layer units (with connections between output and input layer): No convergence was obtained in many trials, with various initial conditions and various values of gain and momentum [14].
- Two-bit binary adder: No convergence was achieved, and the back-propagation process was found to "reliably lead the system into local minima" [14] when searching for the solution with only two hidden units.

4. Convergence to a steady state

We can prove that the distributed iterative process we have presented, with some restrictions, is a converging process. The restrictions we set are the following:

- The algorithm is performed asynchronously (step 5 is applied only to a single unit).
- The stochastic modifications are avoided (step 7 is not applied).
- The genotype that can substitute its unit when it does not reduce the network error E should have the following properties: (1) for every element in the training data set, its outputs should be the same as the unit outputs; (2) it should not cancel an inverse connection to an effective unit; and (3) it should not add an inverse connection from an effective to a non-effective unit.

These restrictions enable us to prove that the training process converges to a steady state, but they also significantly slow the convergence process.

Let us define an "energy" function F.

$$F = \sum_{\substack{p=0 \\ E^p>0}}^{P-1} \Bigg(K_1 E^p + \sum_{V_i \in Y^p} (K_3 \tilde{N}_i^p + K_4 \eta_i^p) + \sum_{\substack{V_i \notin Y^p \\ V_i \in V_H}} (K_2 + K_3 S_i^p + K_4 w_{S_i^p}) + \sum_{\substack{V_i \notin Y^p \\ V_i \in V_M}} (K_3 I_i^p + K_4 t_i^p) \Bigg) \tag{9}$$

where

$$\tilde{N}_i^p = \begin{cases} K_5 Z_i^p & \text{if } Z_i^p > 0 \\ R_i^p & \text{otherwise} \end{cases}$$

and where $K_1 \gg K_2 \gg K_3 \gg K_4 \gg K_5 \gg 1$ are positive constants. We shall prove that F has a lower bound, and that it is decreased with every change in network structure and weights until it reaches a steady state.

If in a certain iteration $F = 0$, the obtained network represents the solution for the whole training data set. We shall show that a series of energy functions obtained in n iterations $F_1, F_2, \ldots, F_n$ satisfies $F_{i+1} \leq F_i$.

Theorem 1. *A network converges to a steady state when it is trained by the distributed genetic algorithm.*

Proof. To prove convergence to a steady state we shall show that $\lim_{n\to\infty} F_n \geq 0$. Since all the constants and terms for F_i are nonnegative, $F_i \geq 0$ for every i.

Every change in the network indicates that a certain genotype is selected to replace its unit inputs. If the current iteration n yields energy F_n, then selecting a genotype that replaces a unit v_i results in a modified neural network whose energy is F_{n+1}. We shall show that $F_{n+1} < F_n$. If the genotype error is smaller than the original error E, $F_{n+1} < F_n$. According to the definition, $K_1 \gg K_2 \gg K_3 \gg K_4 \gg K_5$; therefore, if the first term in equation (9) is greater in iteration n than in iteration $n+1$, the other terms are negligible. When the error of the selected genotype in iteration $n+1$ is equal to the error in iteration n, the second and third terms in equation (9) must be considered. A genotype was selected, thus there exists a unit v_i whose $\sum_{E^p>0} f_i(p)$ is smaller than that obtained in iteration n. However it is possible that for another unit v_j, $\sum_{E^p>0} f_j(p)$ is greater in iteration $n+1$. But in the restricted algorithm, it can happen only in two cases:

1. If v_j was not an effective unit in iteration n and became one in iteration $n+1$. However, even if $\sum_{E^p>0} f_j(p)$ is greater in iteration $n+1$, $F_{n+1} < F_n$ because the number of non-effective units in iteration $n+1$ was reduced relative to iteration n. The resultant energy as defined by equation (9) is lower, since K_2 multiplies the number of the non-effective units, and $K_2 >> K_3$.

2. The number of disturbing connections Z_i^p to unit v_i is reduced and the number of desired-inverse connections to other units may be increased. In this case F is decreased, since $K_5 \gg 1$. ∎

Note that the number of iterations required to achieve convergence has no upper bound, since only some of the iterations yield changes in the network architecture or its weights. The process is composed of a series of changes, each of which reduces F until it stops at a global or local minimum.

5. Discussion and conclusions

We have presented a new approach to the design and training of neural networks, using a distributed genetic algorithm. Since most of the calculations are performed separately for each unit, the training process can efficiently use a parallel computer.

This process modifies network structure in order to find optimal parameters. For each unit, the best set of parameters is sought within a dynamic environment generated by other units that also vary. We conclude from the simulations that the presented approach provides robustness against entrapment at a local minimum, in contrast to gradient descent processes. This

approach eliminates the need to predetermine network structure elements, and achieves efficiency by using a search space orders of magnitude smaller than that used in standard methods.

The estimated search-space size of this algorithm indicates its practical significance for real-world problems previously solved using larger networks. For a network with n units (in which every unit can be connected to $n-1$ units and a bias, every connection can be one of $\mathcal{K}$ possible weights; and where the search space consists of all possible feedforward networks) the search space is of the order $\mathcal{K}^{(1/2)n(n+1)}$. In our algorithm the maximum local search space is $\mathcal{K}^n$ ($n-1$ connections and a bias); and for n units, $n\mathcal{K}^n$. This search space is for a given intermediate state of the network; for the entire process we must multiply it by the maximum possible number of intermediate states until convergence. This can be estimated for the restricted algorithm (as described in section 4), since the energy function F is bounded from below and is decreased with every change. In studying the energy function (9), we find that the changes with error reduction are primarily of order 2^n. The changes with no error reduction for a unit is maximum: $\mathcal{K}$ possible weight changes from each $(n-1)$ units and a bias, thus of the order $n\mathcal{K}$ and $n^2\mathcal{K}$ for all the units. Therefore, the search space for the restricted algorithm is of the order $2^n n^3 \mathcal{K}^n$. Clearly, this is significantly smaller then the search space of previous processes (for large n):

$$O(2^n n^3 \mathcal{K}^n) \ll O(\mathcal{K}^{\frac{1}{2}n^2}) \tag{10}$$

Future research should include testing the algorithm in practical problems, and implementing it on parallel hardware. From our earliest simulations, we have found this approach to be very promising.

Appendix A The algorithm

A concise version of the distributed genetic algorithm follows.

1. Set a random initial state of a feedforward network $\mathcal{N}$, and a set of genotypes $\mathbf{G}$.
2. Zero the overall network output error values and genotype fitness values.
   $E = 0$
   $f_i^k = 0 \quad \forall \quad 0 \leq k < K,\ N \leq i < N + H + M$
3. Set input vector $\vec{X}^p$ and calculate the unit outputs (equation (2)).
4. Calculate E^P (equation (1)) and update E (equation (3)).
5. Calculate $\tilde{E}_i^p$ for each of the hidden and output units.
6. Update f_i^k (which is assigned to each genotype), according to the flowchart in figure 6.

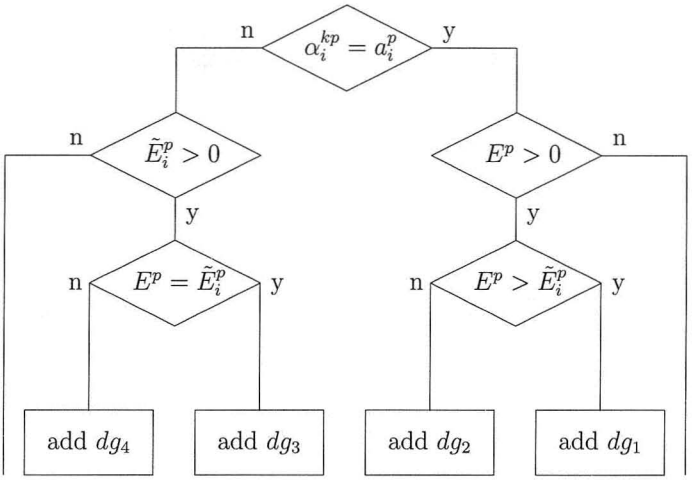

Figure 6: Fitness updating flow chart.

K_1, K_2, and K_3 are constants for which the following holds: $K_1 \gg K_2 \gg K_3$. The superscript k denotes that the parameter is of genotype k.

$$dg_1 = K_1 E^p + K_2 N_i^{kp} + K_3 \eta_i^{kp} \tag{11}$$

where w_{zi}^{kp} is the minimal weight of a disturbing connection to unit v_i.

$$dg_2 = \begin{cases} K_1 E^p + K_3 \eta_i^{kp} & \text{if } \tilde{E}_i^p < E^p, \text{ and both the training data set vectors for which } E^p > 0 \text{ and } a_i^p \text{ are unchanged for a fixed number of cycles} \\ K_1 E^p + K_2 S_i^{kp} + K_3 w_{si}^{kp} & \text{otherwise, if } v_i \text{ is an hidden unit} \\ K_1 E^p + K_2 I_i^{kp} + K_3 t_i^{kp} & \text{otherwise} \end{cases} \tag{12}$$

where w_{si}^{kp} is the minimal weight of an internal connection to unit v_i.

$$dg_3 = \begin{cases} K_1 \tilde{E}^p & \text{if } \tilde{E}_i^p = E^p, \text{ and both the training data set vectors for which } E^p > 0 \text{ and } a_i^p \text{ are unchanged for a fixed number of cycles} \\ K_1 \tilde{E}^p + K_2 H & \text{otherwise} \end{cases} \tag{13}$$

$$dg_4 = K_1 \tilde{E}_i^p + K_2 (Q_i - I_i^k) + K_3 \delta_i^k \tag{14}$$

where Q_i is the number of posible connections to unit i that would not interfere with the feedforward structure of the network.

7. Repeat steps 3 to 6 for all training data set vector pairs P.

8. Find the genotype l with the lowest fitness value g_i^l in each genotype set of a non-isolated unit (where the superscript l denotes that the parameter is of genotype l)

$$f_i^l = \min_k f_i^k \tag{15}$$

and maintain $Z_i^l \leq Z_i$. This constraint can be relaxed if the network error calculated for that genotype is smaller than E.

For those genotypes that are found to have both the lowest element of network error and a better fitness than the existing unit inputs, the network is modified as follows:

$$w_{ij} = \begin{cases} w_{ij}^l & \forall \quad c_{ij}^l \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

$$b_i = b_i^l \tag{17}$$

If the network structure has been modified, no further modifications are executed during this cycle.

For an isolated unit v_i, random modifications of w_{ij} are done for each $c_{ij} \in C$, after a certain number of iterations.

If there exist $c_{ij} \in Q$ and $c_{ij} \notin C$ at a low probability, one of the connections is exchanged.

9. New populations of genotypes are produced with the three GA operators, according to the fitness values of f_i^k.

10. If $E > 0$, repeat steps 2 to 9 until the overall error obtained equals zero or the maximum number of iterations is reached.

References

[1] E. Alpaydin, "Grow-and-Learn: An Incremental Method for Category Learning," *Proceedings of the International Neural Network Conference*, (1990) 761–764.

[2] E. B. Baum and D. Haussler, "What Size Net Gives Valid Generalization," *Neural Computation*, **1** (1989) 151–160.

[3] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning* (Reading, MA, Addison-Wesley, 1989).

[4] D. E. Goldberg and K. Deb, "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms," TCGA Report No. 90007, The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscaloosa (1990).

[5] S. A. Harp, T. Samad, and A. Guha, "Towards the Genetic Synthesis of Neural Networks," *Proceedings of the Third International Conference on Genetic Algorithms*, (1989) 360–369.

[6] G. E. Hinton, "Connectionist Learning Procedures," *Artificial Intelligence*, **40** (1989) 185–233.

[7] J. H. Holland, *Adaptation in Natural and Artificial Systems* (Ann Arbor, University of Michigan Press, 1975).

[8] H. Kitano, "Designing Neural Networks Using Genetic Algorithms with Graph Generation System," *Complex Systems*, **4** (1990) 461–476.

[9] J. R. Koza, "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems," technical report STAN-CS-90-1314, Department of Computer Science, Stanford University (1990).

[10] M. Mézard and J. P. Nadal, "Learning in Feedforward Layered Networks: The Tilling Algorithm," *Journal of Physics A*, **22** (1989) 2191–2203.

[11] G. F. Miller and P. M. Todd, "Designing Neural Networks Using Genetic Algorithms," *The Third International Conference on Genetic Algorithms*, (1989) 379-384.

[12] M. L. Minsky and S. A. Papert, *Perceptrons* (Cambridge, MIT Press, 1969).

[13] J. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms" (BBN Systems and Technologies, Inc., 1989).

[14] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing* (Cambridge, MIT Press, 1986).

[15] G. Syswerda, "Uniform Crossover in Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms*, (1989) 2–9.

[16] G. Teasauro and B. Janssens, "Scaling Relationships in Back-propagation Learning," *Complex Systems*, **2** (1988) 39–44.

[17] D. Whitley and T. Hanson, "The Genitor Algorithms to Optimize Neural Networks," technical report CS-89-107, Department of Computer Science, Colorado State University (1989).

[18] D. Whitley, "The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best," *Proceedings of the Third International Conference on Genetic Algorithms*, (1989) 116–121.