Complex Systems **7** (1993) 445–467

Using the Functional Behavior of Neurons for Genetic Recombination in Neural Nets Training

Nachum Shamir*
David Saad†
Emanuel Marom
Faculty of Engineering, Tel Aviv University,
Ramat Aviv 69978, Israel

Abstract. We propose a new hybrid genetic back propagation training algorithm based on a unique functional matching recombination method. The method is used to evolve populations of neural networks and provides versatility in network architecture and activation functions. Net reorganization and reconstruction is carried out prior to genetic recombination using a functional behavior correlation measure to compare the functional role of the various neurons. Comparison is done by correlating the internal representations generated for a given training set. Net structure is dynamically changed during the evolutionary process, expanded by reorganization and reconstruction and trimmed by pruning unnecessary neurons. The ability to change net structure throughout generations allows the net population to fit itself to the requirements of dynamic adaptation, performance, and size considerations in the selection process, thus generating smaller and more efficient nets that are likely to have higher generalization capabilities. A functional behavior correlation measure is used extensively to explore and compare nets and neurons, and its ability is demonstrated by investigating the results of genetic recombination. The vitality of nets organized via the functional behavior correlation measure prior to genetic recombination is demonstrated by statistical results of computer simulations. The performance of the proposed method and its generalization capabilities are demonstrated using Parity, Symmetry and handwritten digit recognition training tasks.

*Current address: Department of Electrical Engineering, Technion–Israel Institute of Technology, Technion City, Haifa 32000, Israel.

†Current address: Department of Physics, University of Edinburgh, J. C. Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK.

1. Introduction

Constructive and destructive training algorithms for neural nets have significant importance because of their ability to construct minimal nets that are economical in terms of hardware and software and powerful in terms of generalization capabilities. This work presents a novel approach for combining genetic evolution, back propagation training, and various pruning methods to provide a powerful training algorithm capable of dynamically modifying net structure, functional composition, and weights, while adapting toward minimal net structure.

The basic concepts of construction algorithms are demonstrated by the Tiling [7] and Upstart [8] algorithms that create a feedforward network of binary neurons and a single output neuron. The more advanced Cascade Correlation [9, 10] algorithm uses continuous neurons with no limitation on the number of output neurons. In these algorithms, a combined training algorithm and construction operator provide the capability of building the net gradually. Neurons and layers are added to the net as needed and convergence is guaranteed regardless of the initial net structure.

Such "forward progressing" algorithms suffer from significant drawbacks in their ability to produce minimal nets. It has been suggested [1, 4, 5, 6] that the most efficient and promising way to produce a minimal net is to train a sufficiently large net that is pruned both during and after training. A "forward progressing" algorithm is likely to produce nets that have more neurons and layers than actually needed; but due to the nature of the construction operator, it is difficult to remove redundant units.

Extensive use of genetic algorithms for neural net training and construction has been shown to improve existing training techniques and overcome certain limitations such as local minima traps, network paralysis, and others. Genetic algorithms are based on various aspects of natural selection and genetics. Taking an optimization problem and encoding its solution proposals into a population of artificial "chromosomes," one makes use of selection and reproduction operators similar to natural ones, and evolves a population of solution proposals. Using effective natural selection criteria and efficient reproduction methods, the population is enhanced with each generation. The result is a powerful stochastic optimization technique.

The application of genetic algorithms for neural net training is done in a variety of ways. In common applications, demonstrated successfully by the GENITOR and GENITORII algorithms [12, 13, 14], network weights are encoded into artificial "chromosomes" represented by binary strings. Conventional training techniques (restricted to a predefined network structure) are then applied. A more global approach that encodes net connectivity was demonstrated by Whitley et al.[14], in which an evolutionary process was used to create the interconnections map, and conventional training methods were used to train the proposed nets. These training algorithms are confined to a predefined network architecture where the number of layers and neurons do not change during training.

An example of a genetic algorithm that searches for an appropriate network architecture was presented by Kitano [15]. Recognizing the "scalability" problem of methods based on direct encoding of the interconnections map, Kitano proposed the encoding of a graph generation grammar that encodes the network construction rules into the artificial chromosomes. The encoded grammar was significantly shorter than direct encoding methods, and provided the means for dealing with dynamically changing configurations. It is important to note that every offspring net created by this algorithm was trained from a random initial state. Besides connectivity patterns, no weight values were transferred to the offspring nets and an expensive (in terms of computational complexity) back propagation training was used. In a procedure similar to the introduction of priors a particular configuration was chosen from a certain class of possible solutions. The exact solution, that is, the explicit weight matrix, is defined by a complementary training process (such as gradient descent) that refines the solution within the given class of solutions.

We propose a new functional matching recombination method for use with genetic back propagation hybrids, where matching is done by comparing the functional role of neuron pairs correlated by their corresponding internal representations. The representation of neuron functionality by a vector of internal representations for the entire training set is called the *functional behavior of neurons* [1, 2] and is also used for observing the results of genetic recombination. The proposed recombination method takes into account the fact that neurons performing equivalent tasks may be located at different positions in the hidden layers and therefore encoded at different locations in the genetic "chromosomes" (this is often called the *problem of hidden neuron location permutation* [20, 21]). By rearranging and reconstructing the parent nets prior to genetic recombination, the recombination efficiency is significantly enhanced and vital internal structures are preserved. In addition, since parent nets are reconstructed prior to genetic recombination, the population may include a variety of network structures.

The main contribution of the proposed method is its ability to handle heterogeneous populations of nets having different sizes and structures. Its ability to transfer network infrastructures and their corresponding weight values from parents to the offspring nets is also recognized. This enables a smoother evolution of the population between different classes of solutions (mainly configurations and activation functions), thereby creating offspring nets with enhanced initial performance that require fewer retraining cycles. By adding a pruning phase, the hybrid algorithm adaptively changes the structure of the nets in the population. These changes are controlled by a balance between expansion and pruning processes.

2. Genetic algorithms

Genetic algorithms are stochastic optimization methods that imitate natural processes by applying "evolution" to a population of solutions proposed for

a given optimization task. Each proposal in the population is given a fitness value representing its performance when applied to a specific task. During evolution, pairs of individual solutions are chosen from the population (at random), and together they produce an offspring representing a new solution proposal. The offspring is given a fitness value with which it competes for a place in the population. Similar to biological natural selection, less fit individuals generally do not survive and are removed from the population. The fitness of the entire population is enhanced with each generation until the most fit individuals reach the global optimum. Several variations of genetic operators have been suggested and to clarify the types of operators used in this work, we provide the following brief review.

Each individual "chromosome" is represented by a string of encoded features and parameters. Then the reproduction stage mentioned above is carried out in the following manner.

- Two parents are chosen at random from the population, using a nonuniform probability distribution. Higher probability is given to more fit individuals, thus adding *selective pressure* [11, 13] to the evolutionary process.
- A new offspring is created by combining attributes (selected at random with equal probability) from each of the two parents.
- Mutations are applied to the offspring by making small random changes to the encoded features.
- According to the principle of natural selection, the new offspring must compete for a place in the population. A fitness value is given to the new offspring, which is then compared to the least fit individual in the population, resulting in the survival of the more fit individual and the elimination of the other.

The parameters of this artificial evolutionary process must be selected carefully. The nonuniform distribution underlying the selection of parents must be carefully chosen to avoid exaggerated preference of highly fit individuals which may cause a premature population convergence and loss of diversity. On the other hand, if more fit individuals are not granted a statistical preference, the convergence rate is slowed and processing time is wasted. The rate of mutation must also be carefully considered. High mutation rates may slow the evolutionary process by creating a mass of low-fitness individuals that can destroy the evolution process by transforming it into a random search. An extremely low mutation rate, on the other hand, may result in the loss of population diversity and prevent convergence in the vicinity of the global optimum.

When genetic algorithms are used for evolving populations of neural networks, major encoding difficulties are encountered. Neurons that perform equivalent tasks within different networks are located at different positions in

the hidden layers and therefore encoded at different locations in the "chromosomes" of a selected pair of parent nets. Recombining the two parent "chromosomes," which are arbitrarily ordered, may produce meaningless random results as well as untrainable offspring nets. This obstacle is often called *the problem of hidden neuron location permutation* and is known to have a disastrous effect on genetic algorithm training processes [20, 21]. The damage is usually caused by the destruction of vital infrastructures of both parent nets during recombination, since weight values belonging to different infrastructures are mixed. To minimize the damage during recombination, one must evaluate the infrastructure functional similarity in both parent nets and encode the connectivity patterns and weight values of similar infrastructures at similar locations in the parent "chromosomes." Since neuron activation functions are nonlinear, it is impossible to evaluate the similarity of neurons and network infrastructures by comparing network connectivity or weight values. In the next section, we present a measure for evaluating infrastructure functional similarities.

3. Functional behavior of neurons

The "functional behavior" of a neuron describes how that neuron and its corresponding subnet respond when various input vectors are presented to the net. The net is fully forward connected and has no feedback connections; weight values are real and neuron activation functions are continuous. Each subnet is a part of the net starting at an input layer and ending at a single hidden or output neuron that is the output neuron of that subnet. The subnet contains all relevant neurons in previous layers and all interconnections leading to those neurons. Every subnet represents a real function $f: \Re^n \to \Re$ on the set of input vectors where n is the number of input neurons. (Note that a binary representation is a special case of the general continuous representation.) This function is the subnet response function. The output of the subnet ending at neuron i is represented by

$$s^i = f(v_1, \ldots, v_n) \tag{1}$$

where $v_1, \ldots, v_n$ are the components of the input vector and s^i is the output value of neuron i. The neuron's functional behavior is defined as the response of the corresponding subnet to the set of input vectors

$$B^i = (s_1^i, \ldots, s_p^i) \tag{2}$$

where p is the number of input vectors in the data set and s_j^i represents the output value of neuron i when input vector j is presented to the net.

To compare different neurons and functional behaviors, the latter is normalized with respect to its overall norm $E^i = \sum_{j=1}^{p} (s_j^i)^2$, that is,

$$\tilde{B}^i = \left(\frac{s_1^i}{\sqrt{E^i}}, \ldots, \frac{s_p^i}{\sqrt{E^i}} \right). \tag{3}$$

This normalized representation simplifies both graphic and numerical functional comparisons. The degree of matching between a pair of neurons i_1 and i_2 is given by the correlation of their corresponding normalized functional behaviors:

$$\text{match}(i_1, i_2) = \tilde{B}^{i_1} \cdot \tilde{B}^{i_2} = \frac{1}{\sqrt{E^{i_1} E^{i_2}}} \sum_{j=1}^{p} s_j^{i_1} s_j^{i_2}. \tag{4}$$

This normalized matching factor lies in the interval $[-1, 1]$. Its magnitude represents the functional similarity of the corresponding subnets, where the negative sign indicates an opposite response. For linearly dependent functional behavior vectors, the matching factor is either 1 or -1:

$$\text{match}(\mathrm{i}_1, \mathrm{i}_2)_{B^{i_1} = \alpha B^{i_2}} = \tilde{B}^{i_1} \cdot \tilde{B}^{i_2} = \frac{1}{\sqrt{\alpha^2 E^{i_2} E^{i_2}}} \sum_{j=1}^{p} \alpha s_j^{i_2} s_j^{i_2} = \text{sign}(\alpha). \tag{5}$$

where α denotes the linear dependence.

4. Genetic recombination operator

This section proposes a new genetic recombination operator based on the functional behavior correlation measure. The main objectives of the proposed operator are to increase neural net recombination efficiency by rearranging the internal structure of parent nets, providing the capability to recombine parent nets of different sizes and thus making the handling of heterogeneous net populations possible. The two parent nets undergo internal rearrangement and reconstruction, creating a pair of target nets that have identical architecture and well-matched internal order.

The mutual reordering process and reconstruction of the parent nets is performed simultaneously on the two nets, layer by layer.

1. **Input layer**: The functional role of every input neuron is uniquely defined by its location in the input layer, and all input neurons are copied to the same locations in the target nets.

2. **Hidden layer**: There is no dependency between the functional role of hidden neurons and their locations in the hidden layers. Therefore one must identify related neurons in both nets and place them in the same location in the hidden layers of the corresponding target nets. Neurons are copied to target nets together with their entire set of internal parameters, input connections, and weight values.
   Hidden layer processing is done in the following manner:

   - All neurons from the currently processed hidden layer of the first net are copied to the corresponding target net.
   - For each neuron in the first net, the neuron with the highest functional similarity is identified in the second net and copied to the

equivalent location in the second target net, parallel to the original neuron in the first net. The functional similarity of neurons is evaluated by equation (4), the functional behavior correlation measure.

- If the matching factor (4) is negative, invert the activation function and the outgoing connections of the second neuron. When the activation function is antisymmetric, one simply inverts the incoming connections instead of the activation function.
- When all neurons from the first net have been processed, repeat the process for the second net by identifying and copying neurons with the highest functional similarity from the first net.
- During the reordering process and reconstruction, neurons are duplicated and may therefore appear more than once in the hidden layers of the target nets. The functional performance of the target nets is restored by compensating the outgoing connections of the duplicated neurons. The compensation is carried out by dividing the amplitudes of the corresponding output connections by the number of duplications.

The number of neurons in the hidden layer created in the target nets is the sum of hidden layer sizes of the two original nets.

3. **Output layer**: The functional role of each output neuron is uniquely defined by its location in the output layer. All output neurons are copied together with their entire set of internal parameters, input connections, and weight values, to the same locations in the target nets.

After the two parent nets are reordered and reconstructed, and the target nets have been created, an offspring net is created. Each building block of the recombination process consists of a neuron, its internal parameters (for example, an activation function), a set of input connections, and the corresponding weight values. It therefore represents a specific function performed by that neuron (on the space of net inputs as well as output of neurons in previous layers). The function is kept intact throughout recombination and transferred to the offspring net. The offspring net is created by parsing the two reconstructed nets simultaneously, selecting at random which of them will supply the next building block, which is then copied to the same location in the offspring net. Mutations are applied by adding random noise to the offspring weights. The offspring net is retrained and submitted to pruning [1, 4, 5, 6] and retraining cycles.

5. The hybrid training system

We will now describe an entire training algorithm utilizing genetic algorithms, back propagation, pruning, and the recombination operator defined in section 4. Training is divided into two stages:

1. initial population generation; and
2. genetic population evolution.

Initial population generation is done by training, pruning, and retraining nets seeded with random initial weights. The method used for training is back propagation with momentum ($\alpha = 0.4$) and weight decay ($\varepsilon = 0.005$). Back propagation training continues until all output polarities are correct or the epoch limit of 100 epochs is reached, whichever comes first. Network pruning is done using three different methods:

1. neuron merging using functional behavior correlation to find and merge matchable pairs of neurons [1];
2. neuron pruning by removing irrelevant neurons (see, for example, [5]); and
3. interconnections pruning by measuring the interconnection relevance and removing irrelevant interconnections (see, for example, [6]).

Functional matching is evaluated by (4) for all pairs of neurons, and those pairs having matching magnitudes greater than or equal to 0.75 are marked as merging candidates. This parameter has been determined experimentally [1].

The relevance of each neuron k is evaluated by calculating the influence of its removal on net output

$$R_k = \sum_{i=1}^{q} \sum_{j=1}^{p} (O_j^i - \overline{O^{(k)^i}_j})^2 \tag{6}$$

where O_j^i represents the net output bit i when input vector j is fed to the net. The term $\overline{O^{(k)^i}_j}$ is the net output generated for the same data when neuron k is removed. The constant p is the number of vectors in the data set, and q is the number of output neurons. In our simulations we used the relevance measure defined at equation (6), but it should be noted that the amount of computation can be reduced by using approximations such as those described in [5, 6, 17].

The relative relevance of neuron k is defined by:

$$\rho_k = \frac{R_k}{\frac{1}{N}\sum_{i=1}^{N} R_i} \tag{7}$$

where N is the total number of neurons. The relative relevance of interconnections is calculated similarly. In all experiments, we used a relative neuron relevance pruning threshold of 0.3 and a relative interconnections relevance pruning threshold of 0.2. The three pruning methods are used together: the merging and pruning candidates are first marked, and then the nets are pruned and retrained.

The fitness of each individual net is determined by counting the number of correct outputs generated for the set of training or testing vectors. When

the fitness of two nets is equal, priority is given to the smaller net. The number of initially trained nets must be larger than the target population size. This ensures populations of nets with enhanced initial performance exceeding that of random net selection.

Once the initial population is created, the evolutionary process may begin. For each evolutionary step, two individual nets are chosen at random from the population such that fitter individuals have a greater chance of being selected (this is called *selective pressure* [11, 13]). The selected pair of individuals recombines and creates a new offspring as described in section 4. The new offspring is trained, pruned, and retrained. Its fitness is calculated and compared to the most similar net in the population, which is replaced if the offspring is found to be more fit. Such a method of selection [2] shows better performance and preserves the diversity of an evolving population of neural networks.

6. Experimental results

The performance of the proposed hybrid training system is illustrated by four examples. In the first example, the influence of functional reorganization on genetic recombination efficiency is examined by comparing the recombination performance with and without reorganization, using the Parity-6 data set. In the second example, the generalization enhancement properties of the proposed system are demonstrated using the Symmetry-16 data set, whose performance is compared to that of a computationally equivalent non-genetic training program. The third is a unique experiment involving heterogeneous populations of nets, each implementing a variety of decision functions. In the last experiment, nets are trained to identify handwritten digits.

6.1 The influence of functional reorganization on genetic recombination efficiency

This experiment was designed to isolate and investigate the effects of hidden neuron functional reorganization on the efficiency of genetic recombination. Two types of recombination methods were compared. In the first, hidden neurons were not reorganized prior to recombination; in the second, hidden neurons were reorganized as described in section 4. All other recombination and retraining parameters, including net growth factors, were identical in the two experiments.

The population included ten nets. All nets were fully forward connected and had an initial structure of 6:6:6:1. The nets were exhaustively trained by the Parity-6 data set, subjected to pruning (by the three methods discussed in section 5), and retrained. The initial population evolved for 2000 generations. Population evolution was performed according to the guidelines described in section 5, using all three pruning methods and functionally matched recombination. It is important to note that all nets in the population had perfect performance throughout the generations, since all nets

in the initial population had perfect performance and these were replaced by offspring nets that had equivalent or better scores. Both recombination experiments were performed using the same population of nets. For the first pair of tests we used the nets of the initial population, for the second pair we used the nets of the population after 1000 generations of evolution, and for the last pair of tests we used the nets of the population after 2000 generations were performed.

Recombination efficiency was measured by the number of retraining epochs needed to achieve perfect performance of an offspring. An adequate recombination method was one that efficiently transferred vital infrastructures from parent to offspring nets, giving high initial performance to the offspring and reducing the amount of retraining. Note that during recombination tests offspring were not subjected to pruning and did not participate in the evolution of the population.

For each histogram in Figures 1(a) and 1(b), 900 recombinations were made, each time selecting a pair of nets from the population (with uniform probability) and retraining the newly created offspring. The results are displayed as a histogram of the required number of retraining epochs needed to achieve perfect training, where retraining was limited to 100 epochs. The peaks shown in the histograms at 100 epochs represent the amount of untrainable offspring created by the corresponding recombination method, while the peaks at zero epochs represent offspring created with perfect initial performance. The experiments were performed three times: for non-evolved, for 1000 generation-old, and for 2000 generation-old populations, exploring the effect of population aging upon the performance of the corresponding recombination method. The results of recombination without functional reorganization are displayed in Figure 1(a), while the results for the functionally matched recombination method are displayed in Figure 1(b). For the initial non-evolved population, the two experiments presented in Figure 1(a) and Figure 1(b) show a minor advantage for the functionally matched recombination method: the number of untrainable nets is 137 (out of 900), while 153 of the nets were untrainable for the recombination without functional reorganization. The convergence properties of the functionally matched recombination method are clearly better for the 1000 and the 2000 generation-old populations.

As populations grow older, superior infrastructures gradually dominate the "chromosomes." Recombination without functional reorganization fails to preserve and transfer these superior structures to the created offspring, as is evident by the fact that the histograms have similar shapes (Figure 1(a)). We find that vital infrastructures are destroyed during recombination, retraining success is reduced, and the number of untrainable offspring is increased. The functionally matched recombination method, on the other hand, preserves those infrastructures as the population evolves. The number of nets that require less retraining to achieve the desired performance is increased, as is the number of nets with perfect initial performance (represented by the peaks at zero epochs in Figure 1(b)).

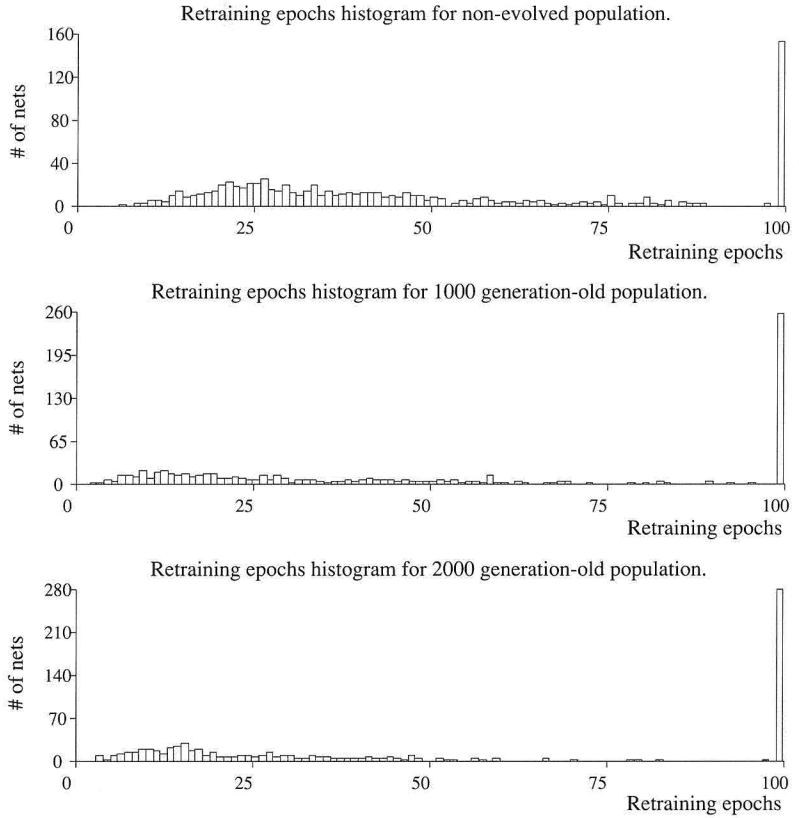


Figure 1: Recombination performance benchmark. Recombination performance was tested for the initial, 1000 generation-old, and 2000 generation-old populations of ten nets, perfectly trained by the Parity-6 data set. Each of the three tests included 900 independent recombinations and retrainings. The histograms show the distribution of the number of retraining epochs required to restore perfect performance of the generated offspring. The peak at 100 epochs represents those nets that could not be retrained, while the peak at zero epochs (if any) represents nets that were produced with perfect initial performance. (a) Performance benchmark for recombination without functional reorganization.

6.2 Generalization enhancement

Generalization capability is the most important property of neural networks, and efforts are being made to improve it. Learning from a set of examples (called the training vectors), nets are capable of responding correctly to new inputs that were not presented previously during training. However, not all

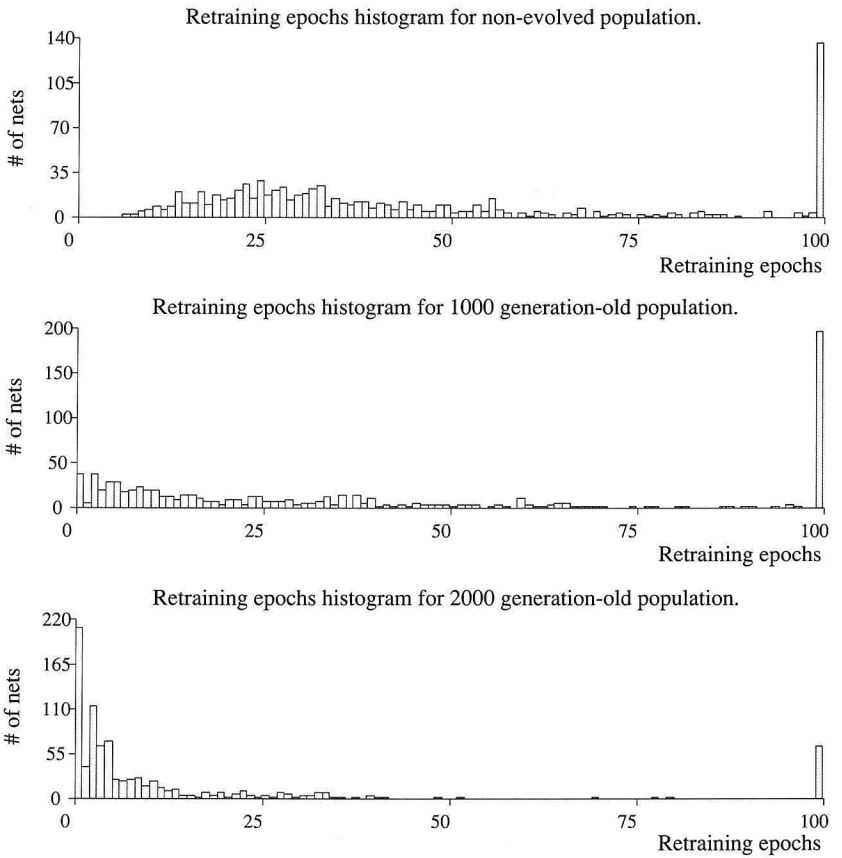


Figure 1: Recombination performance benchmark (*continued*). (b) Performance benchmark for functionally matched recombination.

trained nets have good generalization capabilities and additional steps—such as pruning, weight decay, and so forth [19]—must be taken to improve generalization. Generalization capability is a function of the capacity of the trained net configuration and the complexity of the task itself [22, 23]. Training a powerful net to perform a simple task results in perfect training but poor generalization, while training an over-simplified net to perform a complicated task may result in an incomplete training. Many generalization failures are due to overfitting or "tuning to the noise," a situation where nets learn the noise as well as the data. Pruning and other generalization enhancement techniques help alleviate these problems by reducing net capacity, usually by minimizing the number of free parameters in the net.

This experiment demonstrates the generalization capabilities of the proposed algorithm and compares its performance with that of a computation-

ally equivalent non-genetic training algorithm. Genetic evolution was carried out by the algorithm described in section 5, while non-genetic training was done using randomly created nets instead of genetically created offspring. We refer to the second method as selective back propagation. The randomly created nets had an initial structure of 16:14:1 (fully forward connected), were trained, pruned, and retrained, and had to compete for a place in the population. In both cases, the initial population included the same 20 nets that were pruned and retrained by the selected training data set, and the same training and pruning parameters were used.

The task selected for the demonstration consisted of training a net to perform a Symmetry test for binary strings, returning $+1$ if half the string is a mirror reflection of the other half, and -1 otherwise. The input strings were 16 bits long, spanning an input space of 2^{16} vectors. From this set, three disjoint subsets were created at random, each containing 200 vectors. The first was used as a training set, the second for resolving the fitness of newly created and retrained offspring, while the last set was used as a measure of the generalization ability of the nets. Note that the last set was used only for performance testing, and did not influence the training.

The results shown in Figure 2 give the average fitness and generalization scores for all the nets in the two populations, the first having evolved by genetic evolution and the other by selective back propagation. The average fitness and generalization scores are displayed as a function of the generation/iteration count. The average training scores of both populations are approximately 100% at all times and therefore not displayed. The superiority of genetic evolution is demonstrated by the improvement of both fitness and generalization scores throughout generations, while only minor improvement is found for selective back propagation. After 1000 generations, genetic evolution achieved a population training score average of 99.5%, a fitness score average of 91.5%, and a generalization score average of 90%. The best net achieved training, fitness, and generalization scores of 100%.

After 1000 iterations the selective back propagation algorithm achieved a training score average of 99.5%, a fitness score average of 83.5%, and a generalization score average of 83%, while the best net achieved a training score of 98.5%, and fitness and generalization scores of 96.5%. The increase in both fitness and generalization scores and the results obtained by comparison with the selective back propagation method demonstrates the superiority of genetic evolution and the success of its implementation for evolving neural net populations.

6.3 Activation function adaptation

We now demonstrate the versatility of the proposed hybrid algorithm by showcasing its ability to automatically choose an appropriate neuron activation function. Populations of heterogeneous nets were evolved, each composed of a mixture of neurons where the activation function was randomly selected from four different types of functions:

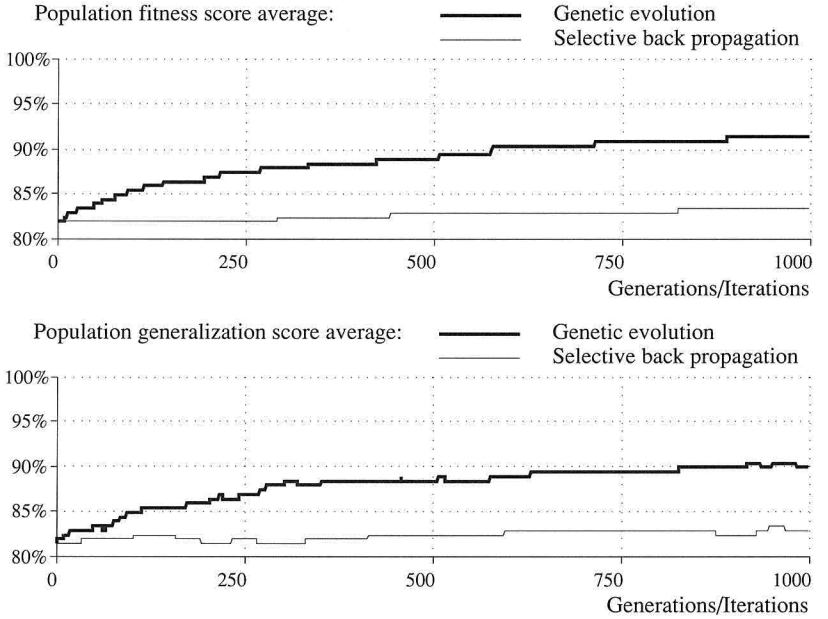
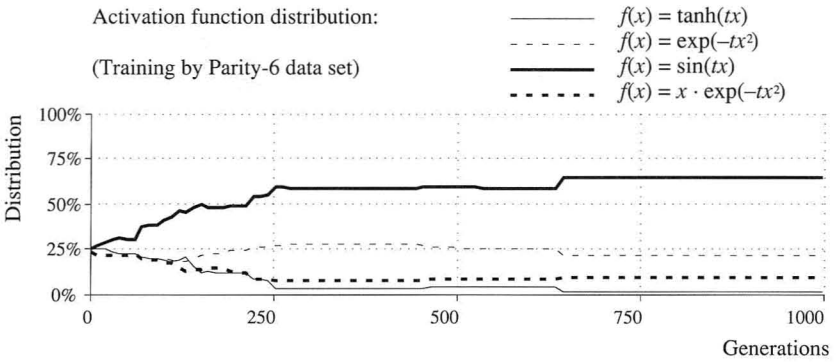


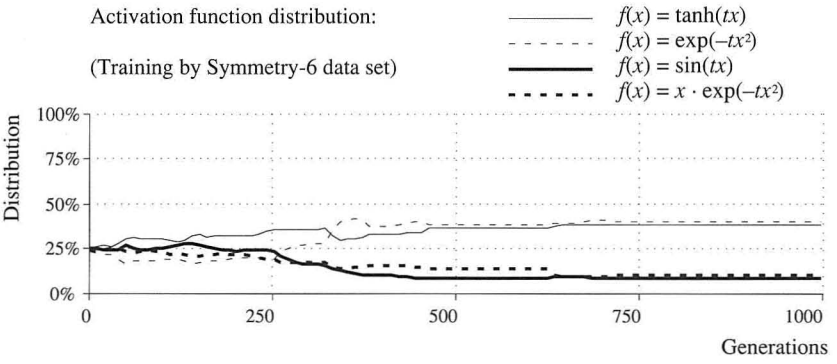
Figure 2: Generalization benchmark. The average fitness and generalization scores of all 20 nets in the population are displayed as a function of the generation count for both a genetically evolving population of nets and a population evolving by selective back propagation. Nets were trained by the Symmetry-16 data set, where 200 vectors were used for training, 200 for resolving fitness of results, and 200 for testing the generalization capability of the nets. The average training scores of both populations was approximately 99.5% at all times and therefore not displayed. The superiority of genetic evolution is demonstrated by the marked improvement of both fitness and generalization scores throughout the generations, while only minor improvement is evident for the selective back propagation method.

- $f(x) = \tanh(tx)$, most commonly used for neural net training.
- $f(x) = \exp(-tx^2)$, commonly used for classification tasks.
- $f(x) = x\exp(-tx^2)$, provides higher versatility than the $\exp(-tx^2)$ function.
- $f(x) = \sin(tx)$, which separates any given set of points on the real axis using a single parameter.

The two tasks selected for the demonstration were exhaustive training by both the Parity-6 and Symmetry-6 data sets. A population of 24 nets was randomly initialized and an activation function was chosen at random,



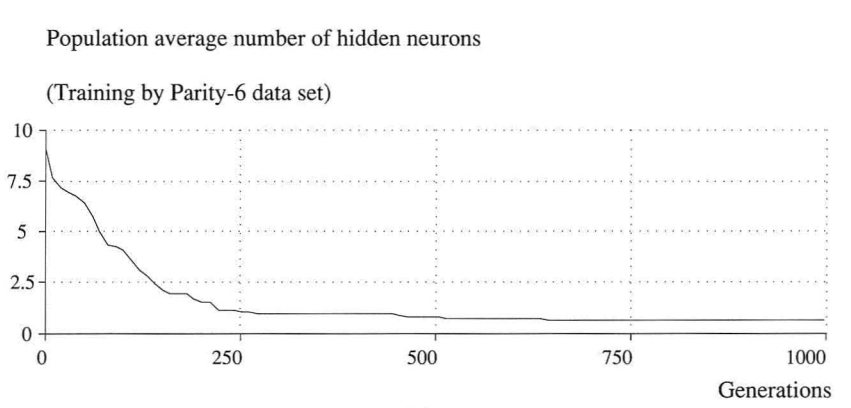
(a)



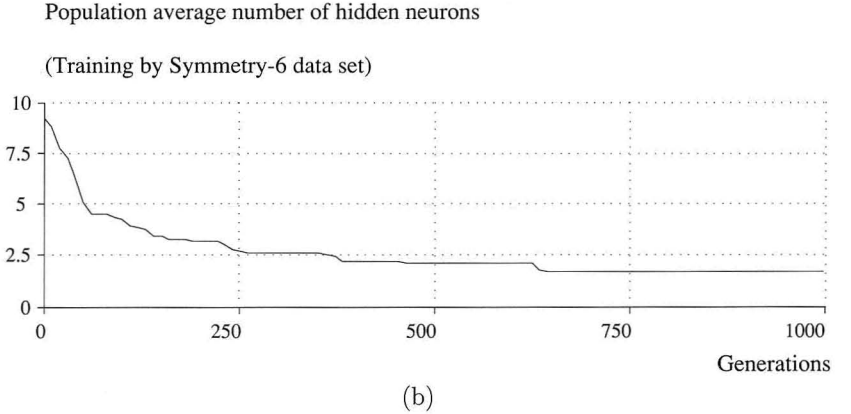
(b)

Figure 3: Evolution of activation function distribution. The distribution of neuron types in the population is displayed as a function of the generation count for the two exhaustive training experiments done by (a) Parity-6 and (b) Symmetry-6 data sets. In the Parity training exercise, neurons operating with the $\sin(tx)$ activation function dominated the population, comprising 65% of the hidden and output neurons in the entire population after 1000 generations. In the Symmetry training example, the population was dominated by neurons operating with the $\tanh(tx)$ and $\exp(-tx^2)$ activation functions, composing (after 1000 generations) 39% and 41% of the hidden and output neurons, respectively.

thereby forming nets composed of a mixture of neuron types. The selection process was evenly distributed among all participating neuron types, resulting in a uniform distribution of types in the initial population. Each net in the initial population was trained by back propagation and training was limited to 100 epochs. The choice of activation function is displayed in Figure 3,



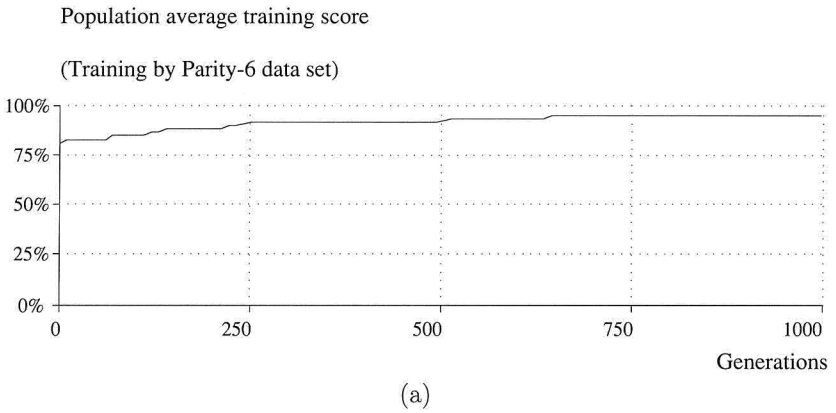
(a)



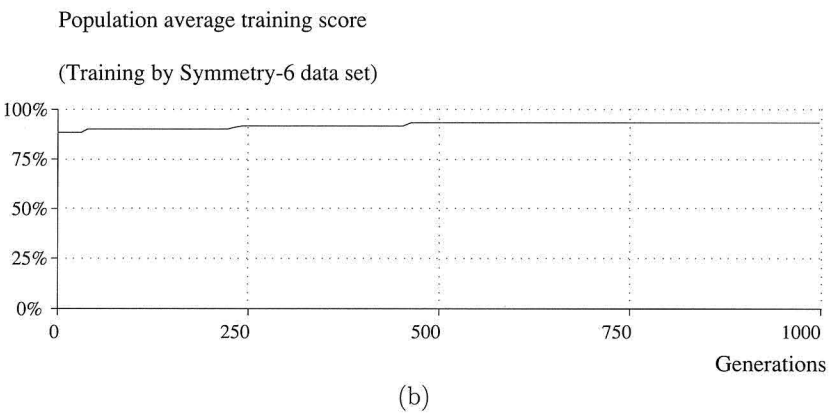
(b)

Figure 4: Average number of hidden neurons in the population. The average number of hidden neurons is displayed as a function of the generation count for the two exhaustive training experiments done by (a) Parity-6 and (b) Symmetry-6 data sets. In the Parity training example, the average was 0.67 hidden neurons after 1000 generations, indicating the existence of many perceptrons in the population. In the Symmetry training example, the average number of hidden neurons was 1.75 after 1000 generations.

the average number of hidden neurons in Figure 4, and the average training scores are shown in Figure 5. All distribution curves (Figure 3) start at 25% because of the uniform distribution of activation functions in the initial population. As the population of nets evolved, the distribution of neuron types, the average number of hidden neurons, and the average scores were observed for each generation. Since the hybrid system is capable of adding



(a)



(b)

Figure 5: Average training score. In each of the two experiments, average training score increased as the population evolved. In both cases, the best net performance was 100%, and the average scores were 95.3% for the Parity-6 training and 93.8% for the Symmetry-6 training after 1000 generations.

and removing neurons, the distribution of activation function types changed over time, automatically adapting toward optimal net structure and neuron composition.

For the Parity training example (Figure 3(a)), neurons operating with the $\sin(tx)$ activation function gradually dominated the population, comprising 65% of the hidden and output neurons after 1000 generations. The average size of the nets decreased (Figure 4(a)): after 1000 generations the average was 0.67 hidden neurons, indicating the existence of many "perceptrons" in the population.[1] The average training score increased over time (Figure 5(a)),

[1] A single node with a sinusoidal activation function represents a line that can separate any set of points in any given way using a single parameter; such a node possesses poor

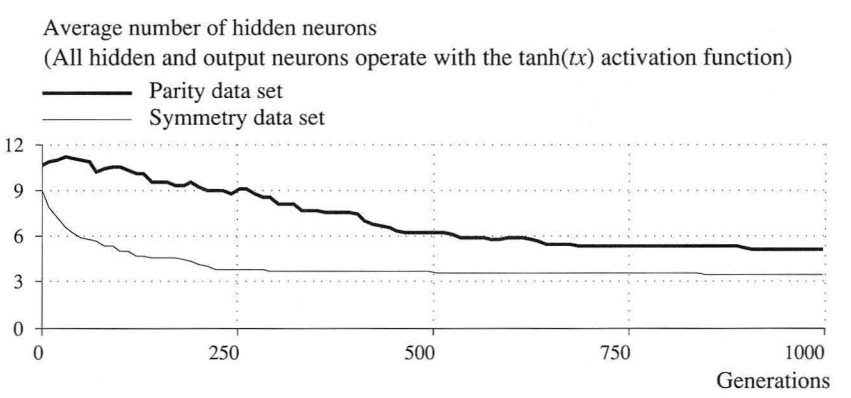


Figure 6: Reference training. Average number of hidden neurons is shown as a function of the generation count for two exhaustive training experiments done by Parity-6 and Symmetry-6 data sets. All nets in the population contained hidden and output neurons operating with the $\tanh(tx)$ activation function. The average number of hidden neurons after 1000 generations was 5.2 for the Parity training set and 3.6 for Symmetry.

reaching 95.3% after 1000 generations.

For the Symmetry training example (Figure 3(b)), the population was dominated by neurons operating with the $\tanh(tx)$ and $\exp(-tx^2)$ activation functions, consisting of (after 1000 generations) 39% and 41% of the hidden and output neurons, respectively. The average size of the nets again decreased (Figure 4(b)): after 1000 generations the average was 1.75 hidden neurons. The average training score increased with each generation (Figure 5(b)); reaching 93.8% after 1000 generations. In both experiments, the best net performance was 100%.

The size of the nets for both the Parity and Symmetry training experiments was significantly smaller than the sizes required for nets where all hidden and output neurons operate with the $\tanh(tx)$ activation function. The results displayed in Figure 6 show the average number of hidden neurons for all nets in a population utilizing the $\tanh(tx)$ activation function, exhaustively trained by the Parity-6 and Symmetry-6 data sets. The experiment was performed using the same parameters as in the previous experiments. The average number of hidden neurons after 1000 generations was 5.2 for the Parity training set and 3.6 for Symmetry.

The results of these experiments demonstrate the capability of the proposed algorithm to minimize net structure, choosing the most suitable architecture and activation function for a given task. The versatility of the

generalization capabilities. When training is done for generalization purposes, as it is in most cases, one should exclude sinusoidal decision functions to improve generalization capabilities.

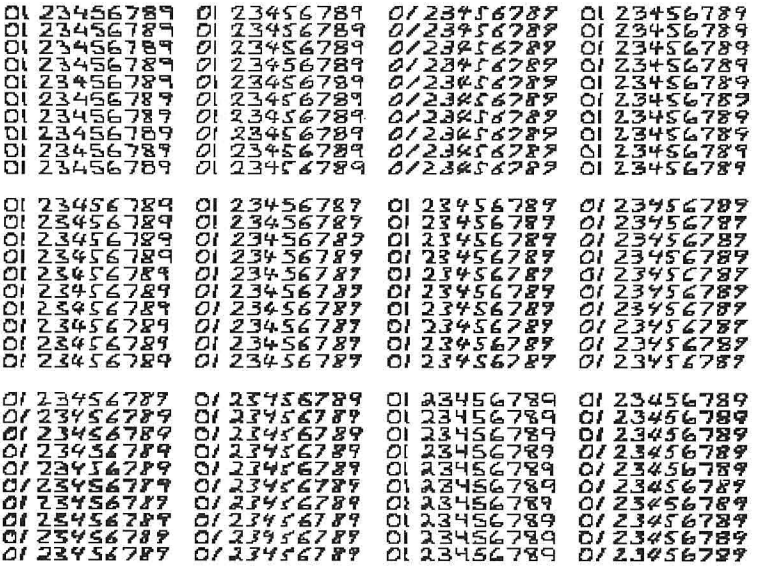


Figure 7: Handwritten digits. This set of 1200 digits was written by twelve people, ten times each. Due to computational limitations, data resolution was reduced from the original sampling resolution of 16×16 pixels to 8×8 pixels for each digit.

proposed algorithm is therefore increased beyond a simple construction and training algorithm, consisting of a method for adjusting internal neuron parameters (decision functions) throughout the process of training.

6.4 Handwritten digits recognition

The task of identifying handwritten digits provides a practical example of the proposed training system and highlights its ability to automatically adapt to diverse tasks. The data base includes 1200 digits written by twelve different people, ten times each (the data base was prepared by I. Guyon). Due to computational limitations, data resolution was reduced from the original sampling resolution of 16×16 pixels to a resolution of 8×8 pixels for each digit. The data set is displayed in Figure 7.

Training was divided into ten different runs, each dedicated to producing a single net capable of identifying a particular digit and rejecting all others. The original data set was split at random into two disjoint sets of 600 digits each. The first set was used for training and fitness evaluation and the second set was used to evaluate the quality of the final results by measuring the generalization ability of the fabricated nets. All trainings used the same parameters. The initial net structure was 64:6:1 (fully forward connected)

Digits identification results					
	Structure		Results		
Digit	Number of hidden neurons	Number of interconnections	Train Success	Test Success	Errors
0	0	8	100.0%	99.2%	5
1	0	7	100.0%	98.0%	12
2	1	24	100.0%	98.3%	10
3	2	24	100.0%	97.7%	14
4	0	15	100.0%	97.8%	13
5	1	19	100.0%	97.7%	14
6	1	19	100.0%	97.8%	13
7	1	15	100.0%	99.7%	2
8	1	32	100.0%	96.7%	20
9	3	150	100.0%	96.3%	22

Table 1: Result of digit-identification experiment. The structures and performance are presented for each of the 10 nets. All nets performed with a 100% success rate on the 600 training digits, and had extremely high test scores on the remaining 600 test digits. The resulting nets are very small; in particular, the nets for digits 0, 1, and 4 are perceptrons (with no hidden neurons).

where hidden and output neurons operated with a $\tanh(x)$ decision function. Back propagation initial training, retraining after genetic recombination, and retraining after pruning were limited to 100 epochs. The population size for each training was 10 nets. The initial population was created by choosing the best results from 50 random trainings, which we then let evolve for 200 generations.

The results are summarized in Table 1, which describes the dimension, final training score (identical with the fitness score), and the test performance for the entire digit set of each of the ten nets. All nets performed with a 100% success rate on the 600 training vectors and had extremely high testing scores for the remaining 600 vectors. The resulting nets were very small. In particular, the nets trained to recognize digits 0, 1, and 4 are perceptrons (with no hidden neurons).

The results in Table 1 show the performance of each net trained to accept a single digit and reject all others. One identifies the actual digit by feeding the same data to each of the ten nets and selecting the net giving the highest output value. For the entire set of 1200 digits, the number of correct identifications was 1160, which is 96.7% of the entire data set. Generalization performance was calculated for the 600 digits that were not used for training, giving a success rate of 93.3%. One should note that no parameter optimization was carried out and no special net structure was created

prior to training. Thus, all structural refinements were automatically done by genetic training and evolution rules.

7. Conclusion

In this paper, we propose a new hybrid genetic back propagation training algorithm based on a unique functional matching recombination method. The method is used to evolve heterogeneous populations of neural networks and provides versatility in network architecture and activation functions. The performance of the proposed algorithm was tested on a wide variety of experiments, showing its ability to overcome the problems originating from location permutations of hidden neurons and to efficiently handle heterogeneous populations of nets having diverse structure and functional composition. Vital infrastructures are preserved throughout recombination and transferred to the next generation, thereby improving the quality and initial performance of the generated offspring.

We performed four experiments that demonstrate the utility of the proposed hybrid. The first experiment demonstrates the importance of functional reorganization of nets during genetic recombination, showing that vital infrastructures are transferred from parent to offspring nets only when functional reorganization is carried out. Functional reorganization was therefore found to have a critical influence on the success of genetic implementation and the ability to preserve and improve "genetic characteristics" throughout the evolutionary process.

In the second experiment, the generalization properties of the proposed hybrid algorithm were tested using the Parity-16 data set, and the results were compared to a computationally equivalent non-genetic training example, showing indisputably the advantage of genetic evolution by producing more efficient nets with higher generalization capabilities. The third experiment demonstrated the ability of the hybrid to handle a population of nets with heterogeneous functional composition, dynamically adapting both the structure and composition of the population. In the last experiment, we trained nets to identify handwritten digits where all structural refinements were automatically done by genetic training and evolution rules. These experiments demonstrate the efficiency and success of the implementation, highlighting the enormous contribution of genetic search to the success and adaptability of neural network training.

Acknowledgments

We thank J. Shamir for helpful discussions, advice, and support, and to I. Guyon for providing the data set for the handwritten digits.

References

[1] N. Shamir, D. Saad, and E. Marom, "Neural Net Pruning Based on Functional Behavior of Neurons," *International Journal of Neural Systems*, **4** (1993) 143–158.

[2] N. Shamir, D. Saad, and E. Marom, "Preserving the Diversity of a Genetically Evolving Population of Nets Using the Functional Behavior of Neurons," *Complex Systems* (in press, 1994).

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

[4] J. Sietsma and R. J. F. Dow, "Neural Net Pruning—Why and How?" pages 325–332 in *Proceedings of the IEEE International Conference on Neural Networks*, **1**, San Diego, CA (1988).

[5] M. C. Mozer and P. Smolensky, "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment," pages 107–115 in *Advances in Neural Information Processing*, **1**, edited by D. S. Touretzky (San Mateo, CA: Morgan Kaufmann, 1989).

[6] E. D. Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks," *IEEE Transactions On Neural Networks*, **1** (1990) 239–242.

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

[8] M. Frean, "The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks," *Neural Computation*, **2** (1990) 198–209.

[9] S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture," pages 524–532 in *Advances in Neural Information Processing Systems*, **2**, edited by D. S. Touretzky (San Mateo, CA: Morgan Kaufmann, 1990).

[10] S. E. Fahlman, "The Recurrent Cascade-Correlation Architecture," pages 190–196 in *Advances in Neural Information Processing Systems*, **3**, edited by R. P. Lippmann, J. E. Moody, and D. S. Touretzky (San Mateo, CA: Morgan Kaufmann, 1991).

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

[12] D. Whitley and T. Hanson, "The GENITOR Algorithm: Using Genetic Recombination to Optimize Neural Networks," Technical Report CS-89-107, Department of Computer Science, Colorado State University (1989).

[13] D. Whitley and T. Starkweather, "GENITORII: A Distributed Genetic Algorithm," *Journal of Experimental Theoretical Artificial Intelligence*, **2** (1990) 189–214.

[14] D. Whitley, T. Starkweather, and C. Bogart "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity," *Parallel Computing*, **14** (1990) 347–361.

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

[16] D. J. Montana and L. Davis, "Training Feedforward Networks Using Genetic Algorithms," pages 762–767 in *Eleventh International Joint Conference on Artificial Intelligence (Detroit 1989)*, edited by N. S. Sridharan (San Mateo, CA: Morgan Kaufmann, 1989).

[17] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," pages 598–605 in *Advances in Neural Information Processing Systems*, **2**, edited by D. S. Touretzky (San Mateo, CA: Morgan Kaufmann, 1990).

[18] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal Brain Surgeon and General Network Pruning," pages 293–299 in *IEEE International Conference on Neural Networks*, San Francisco (Piscataway, NJ: IEEE Press, 1993).

[19] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by Weight-Elimination with Application to Forecasting," pages 875–883 in *Advances in Neural Information Processing Systems*, **3**, edited by R. P. Lippmann, J. E. Moody, and D. S. Touretzky (San Mateo, CA: Morgan Kaufmann, 1991).

[20] N. Radcliffe, "Genetic Neural Networks on MIMD Machines," Ph.D. Thesis, University of Edinburgh (1990).

[21] N. J. Radcliffe, "Genetic Set Recombination and its Application to Neural Network Topology Optimization," *Neural Computing & Application*, **1** (1993) 67–90.

[22] V. N. Vapnik and A. Y. Chervonenkis, "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Theory of Probability and Its Applications*, **16** (1971) 264–280.

[23] V. N. Vapnik, "Estimation of Dependences Based on Empirical Data," (Berlin: Springer-Verlag, 1981).