Genetic Algorithm for Neural Networks: A Comprehensive Study

Muskaan Chaudhary[1] Himani Singh[2]
[1,2]Department of Computer Engineering
[1,2]Gujarat Technological University, Gujarat, India

Abstract— Genetic algorithms are used as a computer-based search method to find a correct or estimated solution to optimization and search problems. As modeling methods and problem solvers, genetic algorithms are on the rise in neural networks owing to their simplicity and ability to optimize in complex multimodal search spaces. This paper explains the basic genetic algorithm and recounts its history in the literature of artificial neural network.

Keywords: Artificial Neural Network, Genetic Algorithm, ANN, GA-NN

I. INTRODUCTION

Parallel distributed processing (PDP) is getting extremely popular in various fields ranging from psychology and cognitive science to signal processing and pattern recognition. Through PDP, the single powerful processor of the traditional von Neumann computer is replaced with a network of simple interconnected processing units. The computational power of the network unfolds from the collective activity of these units operating in parallel [1]. PDP structures, based on features shared with collections of actual biological neurons, are also called neural networks. Among these similarities, the predominant ones are the ability to learn mappings from a set of inputs to a set of outputs based on training examples, and the ability to generalize beyond the particular examples learned. These beneficial abilities have caused a growth in neural network research activity in the past few years [2]. New learning algorithms have been developed, mathematical foundations have been deeper and wider, and network models have been trained and implemented in a number of domains to solve difficult problems. Still, there is one aspect of current neural network research which, if left unaddressed, may hinder the progress in the future. This complex aspect is the problem of network design [2].

A. Artificial Neural Network

An Artificial Neuron can be interpreted as an engineering approach to biological neurons, which includes multiple inputs and a single output. Artificial Neural Network is made up of a vast number of basic processing components that are layered and intertwined with one another.

An example of a distributed, concurrent, data processing system is a neural network. This requires a huge number of modules of computation or nodes with basic computational capability. The network size is often found to be the same as the number of nodes. The nodes are connected with each other in order to generate a fully connected or nearly fully connected network. The network conducts simultaneous data processing based on a paradigm of constraint satisfaction that has been shown to contribute to shared computing capability. It is possible to interpret the functioning of neural networks as an iterative process. There's a special attribute connected with each node. Depending on the other node value, the value of each node is updated for each iteration. It is usually a matrix-vector multiplication update operation. This algorithm for the update represents each node affecting the other network node. Several nonlinear operations, like clipping or clamping, are often accompanied by the multiplication of the matrix-vector. We can consider an associative memory system as one of the many uses of a neural network. For this particular system, if a collection of patterns, identified as 'library elements', has been stored in the network, by providing the network with noisy or inconsistent information on the pattern in consideration, the user will recollect a preferred pattern or library element. For instance, if the library elements contain a set of two-dimensional images, then in order to return the entire image, a portion of one image can be entered into the neural network. A neural-network-centered perceptual memory is robust and fault-tolerant [3]. However, the neural network allows each node to have a distinct processor. This can cause considerable problems when integrating the network in the large systems essential for practical applications.

B. Genetic Algorithm

GA is a type of parallel iterative algorithm with some learning capacity based on the principle of "survival of the fittest" and "natural selection", which repeats evaluation, selection, crossover and mutation after initialization until the stopping condition is satisfied.

Naturally, GA is concurrent and it exhibits tacit parallelism, which does not test and optimize a single solution but simultaneously analyzes and modifies a series of solutions [4]. While selection operator can pick some "good" solutions as seeds with random initialization, crossover operator can produce new solutions that ideally preserve good parental characteristics, and mutation operator can improve diversity and provide an opportunity to escape from local optima.

GAs have been extensively researched and successfully implemented in many fields because of their effective and stable performance. However, for these problems without clearly undefined types of objective functions, the GA should be effectively provided with an estimated fitness value to direct the evolutionary search.

II. GENETIC ALGORITHMS AND NEURAL NETS

The GA aims for global solutions and does not need to be distinguishable from the objective function. Since error surfaces can be very complex with many local optima for even simple problems, the GA appears to be best suited for this form of search. The genetic algorithm searches for points from one population to another, concentrating so far on the best solution field, while simultaneously sampling the total space of the parameter. A Genetic Adaptive Neural Network Training (GANNT) algorithm was designed by Dorsey et al. and showed that the GA also functions well to optimize the NN. The GANNT algorithm is distinct from other genetic search algorithms because, instead of binary representations

of weights, it uses real values [5]. For NN training, this algorithm has been shown to outperform SA.

A. Cellular Growth Evaluation

In the shape of a grammar tree, every cell carries a copy of the genetic code. Each cell also has a pointer in the grammar tree that points to a node [6]. Each node is an instruction from the program [7]. A single ancestor cell with ties to input cells and output cells begins with growth [7]. The parent cell splits into two cells in a sequential division, denoted by S, so that the first child inherits all of the parent's input connections and the second child inherits all of the parent's output connections; the first child is also bound to the second child by a single connection. In fig.1., the second child is put under the first child during the Sequential Divide. A branch point is also a S node, with the top child cell moving its pointer below S to the left branch node and the bottom child moving its pointer below S to the right branch node. The parent cell divides into two cells that inherit all of the parent's input and output relations in a Parallel divide, denoted by P [8]. The two child cells are positioned side by side in fig.1. during a Parallel Division [9]. With the left hand child cell shifting its pointer to the left branch node below P and the right hand child moving its pointer to the right branch node below P, a P node is also a branch point [9]. The E, which is the end or closure sign, is the next symbol encountered in fig.1. After reading the E symbol, a cell terminates growth. The secret unit's threshold is raised by the program-symbol A. In this case, the link register has not been reset and has its original default setting such that it points to the leftmost fan-in relation. Fig.1 provides an example of a simple grammar tree creating an XOR network. The program symbol denoted '-' sets the weight of the input link pointed out by the link register to 1.

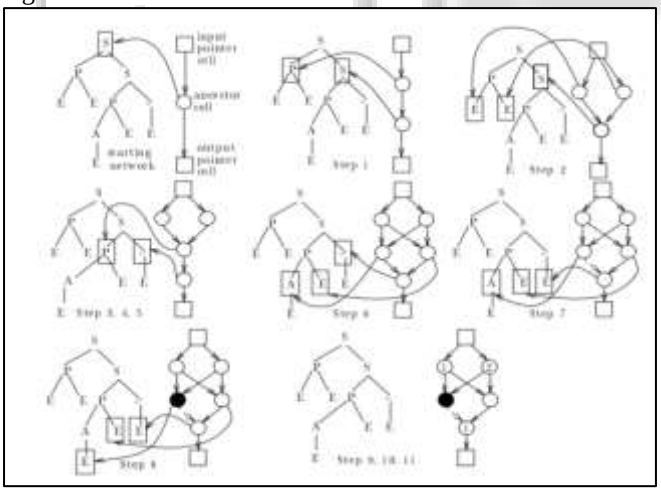

Fig. 1: The cellular development process

Cellular decoding uses a special recursive program-symbol denoted R, in order to reuse subcomponents of the neural network [10]. A counter that regulates the number of recursive leaps that can be made is connected with R. The cell moves its interpreting head back to the root cell of the grammar tree when R is met by a cell [10]. Each time the recursive leap is made, the corresponding counter decrements. The cell does not reset the pointer when the counter equals 0, but either moves forward in the grammar tree, or gives up its processing head and terminates growth. Gruau and Whitley (1993) give an example of how to generalize the XOR net solution to cover all parity problems by putting a R symbol in the grammar tree's leftmost leaf node in Fig.1 [11]. After the genetic algorithm has created a family of recursively formed networks that deal with lower order cases (3 to 6 inputs) on parity and symmetry issues, recursive network encoding reflects a general relationship and automatically generalizes to deal with arbitrarily wide problems.

Using a type of Automated Feature Description like that used in Genetic Programming is another way to reuse development code [9]. Subtrees are generated such that in the main grammar tree, the main tree will leap to a subtree, run the subtree, and return to the associated program-symbol. Subtrees therefore act like subroutines of the software. To develop a method to regulate the gait of a 6-legged robot, Gruau used Automatic Feature Description. The use of Automated Feature Description results in a neural network that is simpler, more modular and well organized [11]

III. GANN STRATEGY

A GA–NN strategy is suggested for optimization problems without especially known forms of objective functions, because of the good approximation ability of NN as well as the robust evolutionary searching performance of Genetic algorithms [12].

The searching mechanism of the strategy can be discussed briefly as follows. First of all, NNs are created with certain training algorithms based on a collection of training samples, as estimated models of the practical problem. Afterwards, the Genetic Algorithm is utilized to explore good solutions among the solution space. Once a new solution is generated by the Genetic Algorithm, the NN is used to learn its fitness value for the Genetic Algorithm to carry on its searching process. Until the Genetic Algorithm's stopping criterion is reached, the strategy will generate the best solution resulting from the Genetic Algorithm and its success will be calculated by thorough assessment based on the actual problem. It is proposed to use several NNs to provide statistical predicted performance for the Genetic Algorithm in order to increase the accuracy and robustness of the results. In problems where the form of objective function is known, but it is hard to evaluate the performance, NN can still be built to quickly provide performance assessment in order to increase the efficiency of genetic search.

The objective values of solutions can be predicted precisely by the NN with a certain extent of precision because of its good approximation performance, so as to enhance the searching efficiency of Genetic Algorithm without utilizing much computational time for evaluation purpose. The Genetic Algorithm, on the other hand, due to its effective and robust search performance, can guarantee promising solutions.

A. Implementation of GANN strategy in neural network design

The above strategy can be applied to some engineering optimization problems without explicitly known types of objective functions but with some available samples. In this

section, the strategy is applied in detail for the neural network design.

Sometimes, samples are possible to be obtained for some engineering problem whose objective function can't be known easily. In order to get some samples, the following simulation assumes that the objective function is known in the stage of NN design but is unknown in the stage of genetic search process. A total of 50 feasible samples are generated randomly based on the constraints and objective function in the limited region. Later, based on the samples, the BPM algorithm designs a three-layer neural network (M = 3, n1 = 4, n2 = 7, n3 = 1) with k = 0.05, b = 0.4 and the total training step 10000. One NN resulting from a random initial condition is depicted as follows (here, only the weights are given without providing the detail structure) [12].

$$(w_{ij}^{1,2})\,|_{\text{i=0-4,j=1-7}} = ,$$

$$\begin{bmatrix} 1.154 & 2.621 & 6.228 & 0.935 & 1.168 & 1.341 & 2.809 \\ 0.578 & 2.083 & 5.285 & -1.099 & 0.532 & -4.489 & 5.472 \\ 0.405 & 0.729 & 0.756 & 0.72 & 0.389 & -0.302 & -0.117 \\ 0.012 & -0.082 & -0.481 & 0.159 & 0.062 & 0.219 & 0 \\ 0.042 & 0.03 & 0.037 & 0.037 & 0.032 & 0.035 & 0 \end{bmatrix}$$

$(w_{i,1}^{2,3})\,|_{\text{i=0-4}}$ = [3.317, -0.302, 1.429, 5.094, -1.773, 0.334, -3.579, 3.912]

IV. GENETIC ALGORITHMS FOR CONSTRUCTION OF NEURAL NETWORKS

Some of the early attempts to encode architectures of the neural network assume that the amount of hidden units has been limited; the genetic algorithm could then be used to decide increasing combinations of weights or hidden units within a finite set of architectures generate enhanced computational conduct. Typically, these specifically coded network architectures have been trained by back propagation. A traditional indicator of fitness is the training period. Whitley, Starkweather and Bogart (1990) demonstrate that it is possible to use the genetic algorithm to identify network topologies that reliably show enhanced learning speeds over the traditional fully linked forward feed network [13]. They also analyze how to produce selective pressure on smaller networks and minimize training time by using weights that have already been designed for larger fully connected networks by initializing the reduced networks [14].

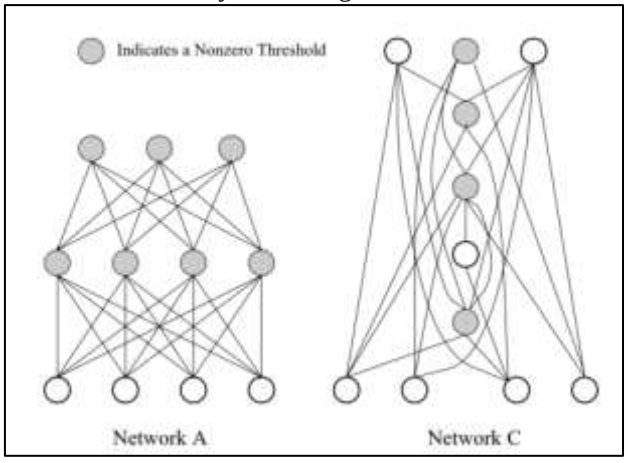

Fig. 2: Feed-forward networks

In Fig.2, an example of the attempt to decrease the network topology for a 2-bit adder is shown. By a genetic algorithm, Network C was evolved and trained to add between 8,000 and 9,000 training epochs on 50 of 50 experiments [15]. On 5 of the 50 studies, Network A failed to converge, and over half of the networks took more than 50,000 learning epochs to prepare. A Network B has been generated by adding direct connections to the Network A input-output nodes [16]. On 46 out of 50 exams, Network B mastered the training set out in between 10,000 and 50,000 training cycles. These early outcomes were promising, but the challenge in optimizing a network architecture explicitly is the high cost of each evaluation. If for each evaluation we need to run a backpropagation algorithm (or other quicker, enhanced method of gradient descent), the amount of assessments required to identify improved network architectures easily becomes computationally prohibitive. The cost of computation is usually so large that, even for the optimization of small topologies, genetic algorithms are inefficient. For instance, if an evaluation function on a complex topic requires one hour of computation time on modest-sized neural network architecture, then it takes one year to do just 9,000 architecture evaluations [17]. If the architecture is complicated, then 9,000 assessments are more likely to be insufficient for productive genetic search [18].

Also, proactive algorithms such as the Cascade Correlation Learning Architecture, which incrementally adds hidden units to the neural network as it learns, are one trend in neural networks that partially solves the architecture problem. The reason for comparison, however, is not just basic feedforward networks that are completely connected. The work of Angeline et al is another recent attempt to develop neural networks (1994). The GNARL method, or GeNeralized Acquisition of Recurrent Links, uses sorting and mutation to scan for space for potential recurring architectures of the neural network. GNARL tries to simultaneously learn weights and topology. This style of approach differs from positive algorithms such as Cascade Correlation in that a non-monotonic fashion investigates the domain of potential architectures.

V. OPTIMIZE THE NEURAL NETWORKS WEIGHTS BASED ON GA

The conventional weight training of the network normally uses the form of gradient descent; these algorithms easily get stuck to the local optimum and do not reach the global optimum. The development of relation weights is to implement a self-adapted and global training system [19].

Using GA to learn the link weights is one feature of using GA in neural networks, so we can substitute certain conventional learning algorithms to solve their defects [20]. The main issues are as follows:

1) Coding Strategy:

In the neural network, there are two methods of encoding the relation weights and the threshold. One is binary encoding, the other is encoding in real encoding. Binary encoding is represented by a fixed length of 0, 1 string for each weight. Real encoding is to communicate each weight with a real number; it overcomes binary encoding misuse, but the

operators, such as crossover and mutate operators, need to be re-designed.

2) Determine the fitness function:

If the network layout is set, typically the network with a significant error, the fitness is minimal using GA to develop the weights of neural networks. The fitness function, for instance, could be: F = C − E. Here, F is the individual's fitness, C is a large number, and E is the network's training error.

3) Evolution process:

Special operators may also be built to evaluate the global search operator of an algorithm, such as the range, crossover and mutation operator.

4) Train the network:

Since GA is good at searching for large-scale, complex, non-differentiable and multimodal spaces, the error function does not require gradient information. In the other hand, it is not appropriate to consider whether the error function can be differentiated, so that any penalty can be applied to the error function in order to increase the commonality of the network and reduce the complexity of the network.

A. Training the neural network with genetic algorithm

In Holland's 1975 book, Adaptation in Natural and Artificial Systems, the notion of training neural networks with genetic algorithms are found on the very basis. McInernery and Schraudolph (1990), Harp, Belew, Samad and Guha and Schafer, Caruana and Eshelman (1990), used genetic algorithm in order to set the learning and momentum rates for the feedforward neural networks [21]. The early efforts in this area were also contributed by Mühlenbein. This tuning was often performed in conjunction with other network changes, such as weight initialization or changing the topology of the network [15]. In addition, several researchers have also attempted to train feedforward neural networks to use genetic algorithms for decision [10]. Linked to this is the use of genetic search by Rogers(1990) and Wilson's work(1990) in optimizing Kanerva's(1988) sparse distributed memories, which learned predicates about input features to create new higher order inputs to a perceptron [2]. Rogers(1990) used genetic algorithms to optimize a sparse distributed memory's 'location addresses' (i.e. the layer mapping inputs to hidden units) [22]. The work done by Rogers was extended by Das and Whitley (1992) with the use of a genetic algorithm for "location address" optimization that actively gathers information about multiple local minima on the basis of relative global competitiveness. Each local optimum in this unique definition of search space illustrates a different and distinct data pattern that associates with some output or event of interest [22]. This enables to track multiple data patterns simultaneously, where each pattern corresponds to a distinct local optimum in the location address space. Two factors have hindered the application of genetic algorithms to simple weight training for neural networks. First, gradient methods have emerged that are highly effective for weight training in supervised learning applications, where input-output training examples are at hand and where the target network is a simple feed forward network. Second, the issue of training a feed forward Artificial Neural Network (ANN) may reflect an application which is inherently not a good match for genetic algorithms that depend heavily on recombination. Some researchers have used small populations and high mutation rates along with recombination while others do not use recombination (e.g. Porto and Fogel, 1990). First, we look at why optimizing the weights in a neural network may generate issues for algorithms that depend heavily on simple recombination schemes.

B. The Problem with ANN

The Competing Conventions problem is a significant reason why genetic algorithms may not provide a good approach to optimizing neural network weights. Nick Radcliffe (1990; 1991) has named it as the Permutations Problem [23]. The root of the issue is that there can be multiple equivalent symmetric solutions to a neural network optimization problem.

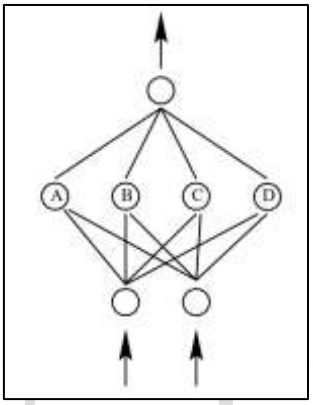

Fig. 3: A simple feed-forward neural network

Figure illustrates a simple feedforward network. Assume the vector,

$$w_{a,1}, w_{a,2}, w_{a,3}, w_{b,1}, w_{b,2}, w_{b,3}, w_{c,1}, w_{c,2}, w_{c,3}, w_{d,1}, w_{d,2}, w_{d,3}$$

The above equation is an arbitrary weight allocation of this neural network, where $w_{\alpha,I}$ moves through the hidden node α and i= 1, i= 2 is the input relation and i= 3 is the output connection. Note that there are 4! = 24 identical vectors for every vector of this type, describing a solution precisely the same [24]. In terms of neural network functionality and in terms of the resulting evaluation function, all permutations over the set of hidden unit indices, {a,b,c,d}, are identical vectors. This is because it has no impact on the network's functionality to rearrange the order of the secret units [24]. Therefore, there are H! symmetries and up to H! equivalent solutions, given H hidden units in a simple, completely connected feedforward network.

For a genetic algorithm that uses simple recombination, the concern this produces is as follows. If one does a simple crossover on a permutation such as [A B C D] and [D A C B], so certain components of the permutation are duplicated by the offspring and others are omitted [25]. Similarly, if various strings want to map the features of hidden nodes in different ways, then duplicating certain hidden units and omitting other hidden units would result in the recombination of these strings. In this case, it may be a drawback to use a population-based search type, because separate strings in the population cannot map functionality in the same way to the various hidden units [25].

Numerous solutions have been raised for the Competing Conventions Problems. Early on, in order to perform a type of intelligent crossover, Montana and

Davis(1989) tried to recognize functional aspects of hidden units during recombination [24]. Radcliffe (1991) also proposed a solution in which the hidden units are treated as a multiset: hidden units which have the same connectivity are considered to be the same, but the hidden units may have distinct connectivity. One can scan through the hidden units, during recombination, to learn which are identical and use this knowledge to direct crossover [23]. This concept and extensions have been applied by Hancock (1992) to understand how similar hidden units are; it is concluded that the permutation problem is not as bad as has often been indicated. More recently, Korning (1994) indicated that the conventional use of the standard quadratic error measurement, (target observed) 2, is part of the issue and recommends the use of other measurements of fitness. Korning also recommends 'killing of" any offspring that do not meet minimum fitness criteria, which might filter and remove incompatible parents' offspring. Overall, however, cases where genetic algorithms have been shown to produce results better than gradient-based approaches for supervised learning applications are very difficult to find [13]. A recent study emphasizes a theme originally put forward by Belew et al. (1990) [13]. Part of the conventional wisdom built around genetic algorithms is that a genetic algorithm is good at roughly characterizing a search space's structure and identifying regions of good average fitness, but not adept at exploiting the search space's local characteristics. One approach for using a genetic algorithm is to use it to find an initial set of good weights and turn over the search to a gradient based method. With this type of approach, Skinner and Broughton(1995) have reported good results and proposed that this approach is more effective than using gradient methods alone for complex problems involving large weight vectors.

C. Reason for Using Genetic Search

Searching for an architecture that performs best on some specified task according to some explicit performance requirements is an issue of network design. In turn, this approach can be viewed as searching for the surface identified by levels of trained network performance above the space of possible architectures of the neural network. The surface is infinitely large because the number of potential units and connections is unbounded. The surface is un-differentiable, since changes in the number of units or connections must be distinct and can have a discontinuous effect on the performance of the network. The mapping from network design to network output after learning is indirect, highly epistemic, and based on initial conditions (e.g. random starting weights), therefore the surface is complex and noisy [26]. Structurally similar networks can demonstrate very different capabilities for information processing, so the surface is misleading; conversely, Networks that are structurally different will display very similar capabilities, so the surface is multimodal [26]. To search for this large, epistemic, un-differentiable, complex, deceptive, noisy, multimodal surface, we pursue an automated process.

It is confirmed for enumerative search methods to get stuck in the combinatorial explosive space of network architectures. In the long run, random search methods are no better than enumerative methods, so useful designs are similarly unlikely to be found. Gradient descent search methods would also fail because they need a distinguishable surface with well-defined slopes and because they are imperfect at searching for complex, deceptive surfaces with several local minima [26]. For reasons mentioned earlier, heuristic knowledge-guided search by a human designer is likely to be inefficient, misdirected, expensive, and slow.

Holland's (1975) schema theorem proposed the general utility of genetic search for large, complicated, deceptive problem spaces. Thus, genetic algorithms may allow rapid, robust evolution of genotypes that define useful network architectures, in contrast to the above search techniques. Therefore, to automate neural network design, we suggest using genetic algorithms as the suitable evolutionary search technique.

VI. CONCLUSION

Researchers interested in combinations of genetic algorithms and neural networks face the task of showing how genetic algorithms can make a meaningful and competitive contribution to the field of neural networks. At present, it appears that it could be useful for supervised learning classification problems to use genetic algorithms and a collection of initial weights before implementing gradient-based approaches. The objective function may be set to balance the trade-off between the over-parameterization of the model that can over-fit the data and a parsimonious ANN that can provide a more stable solution using a global search algorithm such as the genetic algorithm. Combinations of genetic algorithms and neural networks are also expected to continue to have an effect on artificial life.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and J. L. Mcclelland, "A General framework for Parallel Distributed Processing," Parallel distributed processing: explorations in the microstructure of cognition. 1986.
[2] E. Hunt, M. Minsky, and S. Papert, "Perceptrons," Am. J. Psychol., 1971, doi: 10.2307/1420478.
[3] B. D. Brown and H. C. Card, "Stochastic neural computation I: Computational elements," IEEE Trans. Comput., 2001, doi: 10.1109/12.954505.
[4] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," IEEE Trans. Evol. Comput., 2002, doi: 10.1109/TEVC.2002.800884.
[5] R. S. Sexton, R. E. Dorsey, and J. D. Johnson, "Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing," Eur. J. Oper. Res., 1999, doi: 10.1016/S0377-2217(98)00114-3.
[6] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An Evolutionary Algorithm that Constructs Recurrent Neural Networks," IEEE Trans. Neural Networks, 1994, doi: 10.1109/72.265960.
[7] F. Z. Brill, D. E. Brown, and W. N. Martin, "Fast Genetic Selection of Features for Neural Network Classifiers," IEEE Trans. Neural Networks, 1992, doi: 10.1109/72.125874.

[8] R. C. Eberhart and R. W. Dobbins, "Designing neural network explanation facilities using genetic algorithms," 1991, doi: 10.1109/ijcnn.1991.170682.
[9] F. Gruau, "Genetic synthesis of Boolean neural networks with a cell rewriting developmental process," 1992, doi: 10.1109/COGANN.1992.273948.
[10] F. Gruau and D. Whitley, "Adding learning to the cellular development process: a comparative study," Evol. Comput., 1993.
[11] F. Gruau, "Automatic Definition of Modular Neural Networks," Adapt. Behav., 1994, doi: 10.1177/105971239400300202.
[12] L. Wang, "A hybrid genetic algorithm-neural network strategy for simulation optimization," Appl. Math. Comput., 2005, doi: 10.1016/j.amc.2005.01.024.
[13] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning," Artif. Life II, 1992.
[14] E. I. Chang and R. P. Lippmann, "Using Genetic Algorithms to Improve Pattern Classification Performance," NIPS 1990, 1990.
[15] H. Mühlenbein, "Limitations of multi-layer perceptron networks - steps towards genetic neural networks," Parallel Comput., 1990, doi: 10.1016/0167-8191(90)90079-O.
[16] H. Kitano, "A Simple Model of Neurogenesis and Cell Differentiation Based on Evolutionary Large-Scale Chaos," Artif. Life, 1994, doi: 10.1162/artl.1994.2.1.79.
[17] S. U. Miller, G.F.; Todd, P.M.; Hegde, "Designing Neural Networks using Genetic Algorithms," 1989.
[18] P. G. Korning, "Training neural networks by means of genetic algorithms working on very long chromosomes.," Int. J. Neural Syst., 1995, doi: 10.1142/S0129065795000226.
[19] E. Salajegheh and S. Gholizadeh, "Optimum design of structures by an improved genetic algorithm using neural networks," Adv. Eng. Softw., 2005, doi: 10.1016/j.advengsoft.2005.03.022.
[20] G. Chen and J. Yu, "Particle swarm optimization neural network and its application in soft-sensing modeling," 2005, doi: 10.1007/11539117_86.
[21] S. A. Harp, T. Samad, and A. Guha, "Designing application-specific neural networks using the genetic algorithm," Adv. neural Inf. Process. Syst. 2, 1990.
[22] R. Das and D. Whitley, "Genetic sparse distributed memory," 1992, doi: 10.1109/COGANN.1992.273945.
[23] N. J. Radcliffe, "Genetic set recombination and its application to neural network topology optimisation," Neural Comput. Appl., 1993, doi: 10.1007/BF01411376.
[24] D. J. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms," Proc. 11th Int. Jt. Conf. Artif. Intell. - Vol. 1, 1989.
[25] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," Parallel Comput., 1990, doi: 10.1016/0167-8191(90)90086-O.
[26] W. B. Dress, "DARWINIAN OPTIMIZATION OF SYNTHETIC NEURAL SYSTEMS.," 1987.

