The Role of Metaheuristic Algorithm in weight Training and Architecture evolving of Feedforward Neural Networks

Foad Najafi

Department of Electrical and Computer Engineering, University of Hawaii, Manoa

Abstract:

In this paper the role of metaheuristic algorithm for training and finding the best architecture is investigated. Metaheuristic algorithm consist of two mine group: Evolutionary algorithm (EA) and swarm intelligence-based algorithm (SI). The representation of problem and the methods of solving it is reviewed.

Keywords: *Neural Network, Neuroevolutionary, Metaheuristic, Evolutionary Algorithm, PSO*

I. Introduction:

Neural network and metaheuristic algorithm are two technique of machine learning. Each of them is employed for different purposes. NN is used for classification, regression, etc., however, a metaheuristic algorithm is used to find the optima in a huge search space. To use a neural network, first, it should be trained. In the process of training, the weight of each connection is obtained so that the total error (real output minus predicted amount) became minimum. That's where stochastic search space come in to help find the best set of weights. Therefore, finding weights of a neural network can be interpreted as finding the optima of a vast search space. The focus of this paper is on the use of metaheuristic algorithm on training and evolving structure of feed-forward neural networks.

A. Feedforward neural network (FNN)

A feedforward neural network is one of the main types of neural network. This structure consist of an input layer, output layer and one or more hidden layer in between and the layers are connected together through connection weights. The process of training a neural network actually is finding the best set connection weights so that the error(real output- predicted output) become minimum [1].

The most commonly used technique that is based on a deterministic method for training Neural Network is back-propagation (BP). Which is a based on gradient descent method. The weight of each node is adjusted gradually until it finds the minimum error. Since this method is based on the gradient, its search space is local. Therefore, the chance of being trapped in local minima is high.

One way to avoid this situation is to train ANN multiple time with different initial weights. However, this can take a lot of time. Another method is adding noise to the weight when weight are updating or using *Momentum*, which increases the learning rate gradually thorough time[2]

B. Metaheuristic Algorithm

With the increase of computational power, meta-heuristic optimization technique became so common. Some of the most well-known technique are Genetic Algorithm (GA), Ant Colony Optimization (ACO), and particle swarm optimization (PSO). The reason why these algorithm are so popular can have different reasons. First, they are simple. i.e., they are usually inspired by simple natural process or animal behavior. Their simplicity allows scientist to simulate natural phenomenon into codes and find new optimization one. Second, they are flexible. The meta-heuristic algorithms are not dependent on the type of problem. Once an algorithm is coded for one problem, it can easily be used for different problems with minor changes because the algorithm consider the problem like a black box. The only important information about a problem is their input and output. Therefore, the most important part of a problem is how to represent the problem. Third, they don't need gradient to solve a problem. This quality makes them quite attractive for problems that gradient cannot be calculated or it is hard to calculate. Finally, they have the ability to skip local optima. The search space of a real problem is usually unknown and complex with many local optima. These algorithms are able to search the space thoroughly [3]. The global search ability of metaheuristic algorithm let them skip the local minima. Therefore, it makes it an ideal method for training and evolving ANN [4] which its search space is un-known with lots of local minima. Metaheuristic algorithm can be divided into two groups. Single-solution and population-based one. in [5], [6] Tabu search (TS) which is a Single-solution algorithm is used to train NN and [7], [8] Proposed using simulated annealing (SA).

The population-based algorithm can be divided into two main section: first, evolutionary algorithms (EA) that starts with a population and evolves it. EAs are derived from natural selection of Darwinian Theory. In this theory, just the best group of a population can survive and reproduce. Another population-metaheuristic algorithm is swarm intelligence (SI) based algorithms. SI originated from the individual and social behavior of animals (such as ant, bee, wolf, ETC..) in a group that had a common target such as finding food.

Use of metaheuristic algorithm for weight training and architecture design of feedforward artificial neural networks.

Designing of a neural network is usually is the work of a human who is an expert with the method of trial and error with the training method of back propagation. However, with the help metaheuristic, we can train network with higher rate of generalization due to its universal search space. Furthermore, we can find the optimal architecture of a neural network much faster that trial and error.

The evolutionary algorithm could when there is not a there is no exact mathematical definition. By today, in the case of the neural network, there is no mathematical theory that can help us analyze to find the best structure of the neural network. However, with the help of metaheuristic, near optimal structure of the neural network can be found. Therefore, the role of metaheuristic in

designing NN is: 1) weight training. 2) Architecture design. There are other usage for metaheuristic on case of NN. However, in this paper the focus is on weight training and evolve of architecture.

1. Evolutionary Algorithm

Evolutionary algorithms are a type of metaheuristic algorithm that are inspired by natural evolution. The main type of EA's are evolution strategy (ES) [9], [10] evolutionary programming [11], [12] and genetic algorithm (GA) [13], [14]. These algorithms are useful when dealing with a large, complex problem with lots of local minima. Because of their global search abilities, they are less likely to be trapped in local minima while other method that are based on gradient information might easily stuck in local minima

a. Evolutionary Neural Network:

The evolution in neural network takes place in so many different dimensions such as weight training, topology design, learning rule design. Weight training would be a process that weight of connections is changed adaptively until it converges to the global optimum. In the case of evolution of topology, different possible architectures are the search space and the role of EA is to find the best architecture adaptively. The evolution of learning rule is defined as a process that learns the best learning methods. Though, EA can be used in every stage. From steps of defining the architecture to the training of Neural network.

A. The evolution of connection weights

Weight training is formulated as a minimization problem. Where fitness function is the error function and the inputs are weight of each connection in the neural network. Most of the training algorithm are based on gradient descent such as Backpropagation and conjugate gradient [15]–[17] and there are numerous successful ANN of this type [18], [19]. However, BP has some disadvantages due to its dependence to gradient descent. It cannot work in spaces where the gradient is not known and it is easily trapped in local minima because of its local search ability. On the other hand, EA doesn't need gradient information and its global search space is ideal for not being trapped in local minima [20], [21].

1. Representation of the problem:

Like any other optimization, solving weight training problem consists of two part. First, representation of the problem, i.e., whether the presentation should be binary or real value. Second, search operators such as mutation, crossover, and selection algorithm. Different combination of these two parts results in the different algorithm.

2. Binary representation

This representation is usually used in the Genetic algorithm. The weight of the network is encoded as a binary string of a certain length [20], [21]. The accuracy of this presentation directly related to the length of the string. However, increasing the number of bits results in slower computation. Therefore, there is a trade-off between accuracy and speed of convergence of ANN. By putting the binary string of each weight in a line, we can have a chromosome of binary bits and we can perform mutation and crossover on the chromosome population to find the best set of weights with the

lowest error. The advantages of binary representation are simplicity and generality and are easy to perform crossover and mutation. It is also faster than other representation because weights are directly coded into bits. However, this presentation does not have a good result when applying crossover. This problem which is also known as permutation problem [22], [23] happens when two different bit string with totally different structure are selected for crossover because of their fitness function are the same. When this happens, the result of crossover would be quite irrelevant and the algorithm might be stuck in local minima.

3. Real-number representation

Instead to binary representation [24]–[26] proposed using real numbers to represent weights directly. Since the weight of each weight is defined by a real number, each individual sample would be a real vector. Therefore, traditional mutation and crossover can't be applied directly. In [24] Montana and Davis introduced some genetic operator for mutation and crossover on this kind of representation. The idea behind the design of this networks was to preserve useful features that are produced through search steps.

Another way to evolve real presentation vector is to use an Evolutionary algorithm that directly works with real numbers such as evolutionary strategy (ES) and evolutionary programming (EP) [25], [27], [28] . These two algorithms are just based on mutation. Therefore, we won't see the undesired effect of permutation problem on the next generation. The mutation operator is mostly Gaussian mutation. However, in [29] the Cauchy mutation is used too.

4. Comparison between B.P and evolutionary algorithm

As it was mentioned before, the revolutionary algorithm is attractive because they can search the space globally while the search space could have lots of dimensions, be multi-modal and even non-differentiable space. However, it doesn't need gradient information. Therefore, it is ideal for search spaces that gradient information is not available. One other main advantage of using EA on ANN is that it could be easily used on different type of networks with minor changes, i.e., an algorithm that is used for training feedforward neural network, can be used to train recurrent neural network [30].

There are some studies that state that EA is slower than the BP [31], [32]. However, they did not clarify which type of type EA they compared with BP. The authors in [26], [27], [33], [34] stated that evolutionary training algorithm is much faster than back-propagation based algorithm. One thing that might make the comparison hard is the accuracy, i.e., a 6-bit string chromosome genetic algorithm might be faster than BP. However, if we use an 8-bit representation, it could be slower. Despite numerous papers than state evolutionary training is much faster than BP training, Kitano [35] argues that GA-BP, a technique that combines GA with BP, at the best condition, is equal to GA, therefore it does not need to use GA training. However, there are many papers that report an excellent result that by using hybrid evolutionary and gradient descent algorithm [36]–[39]. The paradox between Kitano's work and other works might be related to the different structure of EA and BP that has been discussed, i.e., did Kitano compared a slow GA training method with a fast BP method? The answer to this question doesn't have a clear answer. As the no-free-lunch-theorem [40] states, the best method is always depending on the problem.

5. Hybrid training

The EA are good in global search. However, they are not as good in local search. This means that when they get near the global optima, they spend more time in comparison with the algorithm that has search locally. Therefore, the idea of combining the global search ability of EA with local search ability of another algorithm can have faster and more accurate results. The local search could be BP [35], [41] or another local search algorithm such as simulated annealing [42]. The method is that GA starts with a random set of weights and train them up to a near optimal condition. After that, the pre-trained weights are delivered to a local algorithm to find the global optima. Hybrid training has been used successfully in different applications [22], [36], [37]. By using a global search first, we skip the local minima and by using local search algorithm, we can find the global optima faster and more accurate.

II. The evolution of the architecture

In section one, it is assumed that the structure of the ANN is fixed and only the weight of connections is evolved. However, even if the best set of weight for a given architecture is found. It doesn't guaranty that we have the best-trained network since we have no information about the architecture. The design of architecture is one of the most important parts of finding the best solution. However, up to now, there is no strict mathematical analysis that can help us find the best structure. Right now, the design of the structure is done by and expert human that tries to find the best structure with the excruciating method of trial and fail.

Overfitting and having a simple structure are two ends of structure design spectrums, i.e., a small and limited structure is limited in doing tasks such as classification or prediction. However, a big and complex structure may easily overfit. Overfitting is a problem that happens when the network is too big. When an ANN is overfitted, it has a very small error on training data. However, when it is tested with test data, the error is out of acceptable criteria.

The constructive and destructive algorithm [43]–[45] tried to automate the structure design. In the constructive approach, starts with a minimal network and adds, layer, nodes, and connection whenever it is necessary. While the destructive architecture starts with a complex structure and removes layers, node and connections whenever are needed. However, these structures are easier to be trapped in local minima and they just represent just a limited number of structures [46].

Like any other optimization problem, the possible architecture of networks shares the search space and each point in this discrete structure are a representation of a specific structure. To solve this optimization problem, some fundamental constraint should be considered such as the minimum number of layers and nodes, minimum training error, etc., about architecture. This space has some qualities that are mentioned in [47]:

- 1: this surface infinite since the number layers and node are not limited
- 2: the surface is discrete therefore, it is not differentiable
- 3: the surface is deceptive because of the permutation problem
- 4: the surface is multi-modal since similar structures may have similar performance.

Simulation of topology has two dimensions. First: the genotype presentation. Second: the EA used to evolve the topology.

There is two different view about encoding architecture. One encode every parameter in the architecture which is called direct encoding. In this way, every connection, node, and layer are encoded into genotype. This representation can have the most accurate answer. However, evolution could be so slow due to a large number of parameters. This representation can be useful for small networks. On the other hand, we can have representation that just has general information about the network such as the number of layers, the node in each layer, Etc., this representation is called indirect representation and good for large neural networks. The typical pseudo code for the evolution of architecture is shown below:

1. Decode each architecture into a genotype. If the encoding is indirect, further rules and constraints should be implemented.
2. Trained each architecture with the training data
3. Compute the fitness (error function) of each topology
4. Select parents to produce offspring according to their fitness
5. Apply search operator such as mutation and crossover to produce offspring.

A. The direct encoding

The direct encoding is divided into two methods. One method separates the architecture design from the weight training [48], [49]. The second approach evolves the architecture and weights simultaneously[50], [51]. In the first approach, each connection is encoded into a bit. 1 means that there is a connection and zero means there is no connection. Since each connection is connected to two nodes, with a binary rectangular with N dimension, we can represent a network with N nodes. For instance, if the value of $x_{3,5}$ is one, it means that there is a connection between node 3 and 5 in the network and if it is zero, it means that there is no connection between these two nodes. However, If the amount of $x_{i,j}$ is a real value, it represents the second structure and the weight and architecture can be evolved together. To make a binary (or real-valued) chromosome that represent the structure of each architecture, we can put the rows or column of the matrix in serial to make a chromosome. After creation of the chromosome we can apply mutation and crossover operator on it. We can easily apply design constraints on the matrix. For example, a feedforward neural network just has values on the top right triangle of the matrix since there is no connection a layer and its previous layer. However, for the recurrent representation we can have values in any part of matrix because a layer can be connected to its previous layer.

Since each connection can be added or removed from the architecture, the result can be very accurate. In the [21] Schaffer *et al.* showed that an ANN designed by EA had better generalization ability than an ANN that was designed by a human with BP method. However, if a network is too big, it is really time-consuming to find the best architecture with this method. Instead, we can use implement our knowledge into genotype to limit the possible architectures. This way of encoding is called indirect encoding. The problem of permutation still exists in this architecture. [52]

Suggested to not to use mutation to avoid permutation problem. However, Hancock in [23] argues that the permutation problem might not be so critical if we increase the population size.

B. The indirect encoding

In order to limit the possible topologies of architectures, we can implement predefined rules to restrict it. [53] Schaffarnly encoded some characteristic of architecture into the chromosome. The details can be either defined by our knowledge or it can be implemented through developmental rules. Therefore, we can have two type of indirect encodings.

1. Parametric representation

in this representation, genotype of architecture only contains information about the number of layers, number nodes in each layer and the number of connection between layers [53] This representation is useful when the ANN is considered to be so big or when we have enough information about the structure of ANN.

2. Developmental rule representation

In this structure, the rules are implemented into 2×2 matrixes. There are two types of matrixes: first, nonterminal in which the 4 elements of the 2×2 matrix are elements. Second, terminal matrixes in which the 4 elements are binary. The terminal matrix is only applied in the last step. To build a matrix that represents the structure of ANN, a number of the non-terminal matrices are applied first and in the last step, a terminal matrix is applied. For example, if 3 nonterminal and one terminal rule is used, the number of nodes would be $2 \times 2 \times 2 \times 2 = 16$. Which shows a neural network with 16 nodes. By combining different nonterminal matrixes, different architecture of ANN is defined. Developmental rules are used in [54] to construct chromosome representation. The destructive effect of crossover is decreased in this representation since it can preserve the structure that had been built already. However, due to the limited connection of this structure, it can't search the space fully.

3. Fractal representation

[55] and [56] proposed using fractals representation. They used real value representation.[56] Used simulated annealing for the evolution section.

II. Swarm intelligence based Algorithm (SI)

Another major group of metaheuristic algorithm is swarm intelligence based. One of the most important algorithm in this group is particle swarm optimization. From the time of the presentation of this algorithm, it has found a lot of usage. It also had been used for training of neural network widely. However, the abilities of this algorithm to find the structure is not yet fully investigated. The particle swarm optimization is a part of swarm intelligence technique. It was first invented by James Kennedy and Russell C. Eberhart [57] in 1995. This algorithm is inspired by the social and personal behavior of the fish or bird flock that when they are looking for the food (optima).

Particle Swarm Optimization (PSO) is a relatively recent heuristic search method whose mechanics are inspired by the swarming or collaborative behavior of biological populations. PSO

is similar to the Genetic Algorithm (GA) in the sense that these two evolutionary heuristics are population-based search methods. In other words, PSO and the GA move from a set of points (population) to another set of points in a single iteration with likely improvement using a combination of deterministic and probabilistic rules. The GA and its many versions have been popular in academia and the industry mainly because of its intuitiveness, ease of implementation, and the ability to effectively solve highly nonlinear, mixed integer optimization problems that are typical of complex engineering systems. The drawback of the GA is its expensive computational cost. This paper attempts to examine the claim that PSO has the same effectiveness (finding the true global optimal solution) as the GA but with significantly better computational efficiency (fewer function evaluations) by implementing statistical analysis and formal hypothesis testing. The performance comparison of the GA and PSO is implemented using a set of benchmark test problems as well as two space systems design optimization problems, namely, telescope array configuration and spacecraft reliability-based design [58].

[59] Proposed cooperative PSO. In this algorithm vector of input is divided into a smaller vector and each part forms a separate swarm and looks for the optimum of its search space. The idea behind this configuration was to simulate crossover operator in genetic algorithm in which different individual share information with each other.

[60] Proposed using a PSO-based algorithm and apply it on several benchmark problems. The results were good but not as good as GA since the PSO method that was used was the simplest form of the PSO.

In [61] The opposition-based PSO is used. The opposition technique is a method of search in which the algorithm alongside the original particle, evaluate its opposite particle too. Between these two, the one that has higher fitness is elected for rest of iteration. The opposite here means the other side of the search criteria. For example, in a one-dimension with a span from 0 to 10, if 2 is selected, the opposite of 2 would be 8. The idea behind the opposition-based algorithm is to diversify the particle more.

[62] Used a different method of opposition. Instead of using opposition for all the particle in each iteration, it used it only for one particle at a time. [63] Used a hybrid PSO and GA to train ANN. First, a random set is selected and trained with PSO until the time it gets near the optima. Next, the pre-trained population by PSO is trained by GA to find the global optima. [64] uses a continuous version of Ant Colony optimization (ACO) algorithm. [65] proposes a hybrid PSO and artificial fish swarm algorithm (AFSA-PSO) and [66] used artificial bee colony (ABC) to train the neural network.

[67] Is one of the first works that trained an FFNN with PSO. Eberhard et Al. trained the network with two different methods. The first method (GBEST) which is the classic PSO was used and the other method is (LBEST) which divide the swarms into subgroups neighbors. Each group looks for the global optima individually. Therefore, change of trapping in local minima is lower for LBEST approach especially when the number of neighbors is minimum, i.e., $L=2$. However, LBEST is usually slower than GBEST.

[68]PSO algorithm has a good speed to get close to the optima. However, it is very slow near the optima because of its velocity. It proposed using a hybrid PSO-BP method that starts training with PSO and when it gets near the optima, it changes into BP method. The author argues that it is better than adaptive PSO (APSO) and BP algorithm in case of speed and convergence accuracy.

[69]propose using optimized particle swarm optimization (OPSO). There are some parameters in the PSO algorithm which are usually constant or changing linearly through time. The idea of OPSO is to use another PSO to optimize the free parameter of the PSO that trains weights of FNN, i.e., swarm on swarm on FNN.

[70]uses a hybrid PSO and simulated annealing (SI). However, this method is not like another hybrid that starts with PSO and when finishes the training with a local search algorithm. In this approach, SPO and SI are used in each step. First, a new spot of the swarm is calculated according to the PSO algorithm. Then, SI search operator is applied to the swarm. If the new spot is better, it will replace the old one. However, if the new spot is worse than original one, it may be replaced with lower probability. This method allows the algorithm to jump off the local minima because using SI allows the algorithm to risk the currently good position and leave it to find another good position. However, a classic PSO is more probable to be stuck in local minima in this situation.

One of the well-known problems of PSO is its slow convergence near optima.

To solve this, [71]proposed time-varying inertia weight. Inertia weight is big at the start with the start of iteration and decreases up to the end of the iteration. The idea behind this is to decrease momentum of swarm near optima which causes the particle to swing near optima. [72] Proposed time-varying acceleration coefficients. It decreases the cognitive coefficient, i.e., the personal behavior parameter and increases the social component through time. Big cognitive parameter at the start let the swarm search the space more freely by not being made to converge to GBEST. However, having smaller cognitive and bigger social parameter, lets the swarm converge to the GBEST faster.

To improve the search abilities of PSO several algorithms inspired from EA and incorporated mutation and into the PSO algorithm. [73] Incorporated GA mutation. It selects two swarms (parent). While r is uniformly random number, the offspring would be the sum of " r " multiplied by first parent and " $1-r$ " multiplied to the second parent.[74] Proposed adding gaussian mutation to the PSO. It adds a random number to the every dimension of every swarm. [75]the incorporated algorithm proposed in [17]-[20] into PSO all together to evolve and train FNN.

In ESPNet [76], Yu et.al. Proposed a structure to evolve structure and weights simultaneously. The architecture of FNN is optimized using discrete PSO (DPSO). After finding the best structure, the weights are trained using PSO. When algorithm became near optimal and no future improvement is seen, the evolutionary strategy is applied on the trained network to find the global optima. In [77] where the authors used an Artificial Bee Colony (ABC) algorithm to evolve the design of an ANN with two different fitness functions.

VI. conclusion

In this paper the role of metaheuristic for evolving and training of neural network is reviewed. These algorithm showed that they are quite useful for training of neural network. They are faster and they find the global optima better than the famous back propagation. In the case of topology design, they can help human find the best architecture much faster. The majority of weight training with genetic algorithm had been done in 1990 and just some novel works has been done in the last decade.

The use of SI algorithm is still being investigated in the literature. Although there are many works that investigate the role of PSO for weight training just a few number of work address evolving architecture of neural network with particle swarm optimization.

Reference:

- [1] C. A. R. De Sousa, "An overview on weight initialization methods for feedforward neural networks," pp. 52–59, 2016.
- [2] J. A. Anderson, *An introduction to neural networks*. MIT press, 1995.
- [3] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, 2014.
- [4] M. Castellani and H. Rowlands, "Evolutionary Artificial Neural Network Design and Training for wood veneer classification," *Eng. Appl. Artif. Intell.*, vol. 22, no. 4, pp. 732–741, 2009.
- [5] R. Battiti and G. Tecchiolli, "Training neural nets with the reactive tabu search," *IEEE Trans. neural networks*, vol. 6, no. 5, pp. 1185–1200, 1995.
- [6] R. S. Sexton, B. Alidaee, R. E. Dorsey, and J. D. Johnson, "Global optimization for artificial neural networks: A tabu search application," *Eur. J. Oper. Res.*, vol. 106, no. 2, pp. 570–584, 1998.
- [7] N. K. Treadgold and T. D. Gedeon, "Simulated annealing and weight decay in adaptive learning: the SARPROP algorithm," *IEEE Trans. Neural Networks*, vol. 9, no. 4, pp. 662–668, 1998.
- [8] S. Chalup and F. Maire, "A study on hill climbing algorithms for neural network training," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999, vol. 3.
- [9] H.-P. Schwefel, *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981.
- [10] H.-P. Schwefel, "Evolution and optimum seeking. Sixth-generation computer technology series." Wiley, New York, 1995.
- [11] L. J. Fogel, *Artificial Intelligence Through Simulated Evolution*. [By] Lawrence J. Fogel... Alvin J. Owens... Michael J. Walsh. John Wiley & Sons, 1966.
- [12] D. B. Fogel, *System identification through simulated evolution: A machine learning approach to modeling*. Ginn Press, 1991.
- [13] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

- [14] D. E. Golberg, "Genetic algorithms in search, optimization, and machine learning," *Addion wesley*, vol. 1989, p. 102, 1989.
- [15] G. E. Hinton, "Connectionist learning procedures," *Artif. Intell.*, vol. 40, no. 1, pp. 185–234, 1989.
- [16] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation*, vol. 1. Basic Books, 1991.
- [17] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Process. Mag.*, vol. 10, no. 1, pp. 8–39, 1993.
- [18] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural networks*, vol. 3, no. 1, pp. 23–43, 1990.
- [19] S. S. Fels and G. E. Hinton, "Glove-talk: A neural network interface between a data-glove and a speech synthesizer," *IEEE Trans. Neural Networks*, vol. 4, no. 1, pp. 2–8, 1993.
- [20] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, 1990.
- [21] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," *Phys. D Nonlinear Phenom.*, vol. 42, no. 1–3, pp. 244–248, 1990.
- [22] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving networks: Using the genetic algorithm with connectionist learning," in *In*, 1990.
- [23] P. J. B. Hancock, "Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification," in *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, 1992, pp. 108–122.
- [24] D. J. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms.," in *IJCAI*, 1989, vol. 89, pp. 762–767.
- [25] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biol. Cybern.*, vol. 63, no. 6, pp. 487–493, 1990.
- [26] P. Bartlett and T. Downs, *Training a neural network with a genetic algorithm*. University of Queensland, 1990.
- [27] V. W. Porto, D. B. Fogel, and L. J. Fogel, "Alternative neural network training methods," *IEEE Expert Intell. Syst. Their Appl.*, vol. 10, no. 3, pp. 16–22, 1995.
- [28] S. Yao, C. J. Wei, and Z. Y. He, "Evolving wavelet neural networks for function approximation," *Electron. Lett.*, vol. 32, no. 4, p. 360, 1996.
- [29] X. Yao, G. Lin, and Y. Liu, "An analysis of evolutionary algorithms based on neighbourhood and step sizes," in *International Conference on Evolutionary Programming*, 1997, pp. 297–307.
- [30] M. Mandischer, "Evolving recurrent neural networks with non-binary encoding," in *Proceedings of the 2nd IEEE Conference on Evolutionary Computation (ICEC)*, 1995, vol. 2, pp. 584–589.
- [31] S. E. Fahlman, "Faster-learning variations on back-propagation: An empirical study," 1988.
- [32] E. M. Johansson, F. U. Dowla, and D. M. Goodman, "Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method," *Int. J. Neural Syst.*, vol. 2, no. 4, pp. 291–301, 1991.
- [33] D. L. Prados, "Training multilayered neural networks by replacing the least fit hidden neurons," in

Southeastcon '92, Proceedings., IEEE, 1992, pp. 634–637.

- [34] D. L. Prados, “New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques,” *Electron. Lett.*, vol. 28, no. 16, pp. 1560–1561, 1992.
- [35] H. Kitano, “Empirical studies on the speed of convergence of neural network training using genetic algorithms,” *Proc. Eighth Natl. Conf.*, vol. 2, pp. 789–795, 1990.
- [36] A. P. Topchy and O. A. Lebedko, “Neural network training by means of cooperative evolutionary search,” *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 389, no. 1, pp. 240–241, 1997.
- [37] W. Kinnebrock, “Accelerating the standard backpropagation method using a genetic approach,” *Neurocomputing*, vol. 6, no. 5–6, pp. 583–588, 1994.
- [38] S.-L. Hung and H. Adeli, “A parallel genetic/neural network learning algorithm for MIMD shared memory machines,” *IEEE Trans. neural networks*, vol. 5, no. 6, pp. 900–909, 1994.
- [39] A. J. Skinner and J. Q. Broughton, “Neural networks in computational materials science: Training algorithms,” *Model. Simul. Mater. Sci. Eng.*, vol. 3, no. 3, p. 371, 1995.
- [40] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
- [41] S. Omatu and S. Deris, “Stabilization of inverted pendulum by the genetic algorithm,” in *Emerging Technologies and Factory Automation, 1996. EFTA'96. Proceedings., 1996 IEEE Conference on*, 1996, vol. 1, pp. 282–287.
- [42] X. Yao, “Optimization by genetic annealing,” in *Proceedings of the Second Australian Conference on Neural Networks*, 1991, pp. 94–97.
- [43] S. F. Christian and C. Lebiere, “The Cascade-Correlation Learning Architecture,” in *Advances in Neural Information Processing Systems 2*, 1990.
- [44] M. Frean, “The upstart algorithm: A method for constructing and training feedforward neural networks,” *Neural Comput.*, vol. 2, no. 2, pp. 198–209, 1990.
- [45] M. C. Mozer and P. Smolensky, “Skeletonization: A technique for trimming the fat from a network via relevance assessment,” 1989.
- [46] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.
- [47] G. F. Miller, P. M. Todd, and S. U. Hegde, “Designing neural networks using genetic algorithms,” in *Proceedings of the third international conference on Genetic algorithms*, 1989, pp. 379–384.
- [48] S. Olikar, M. Furst, and O. Maimon, *A distributed genetic algorithm for neural network design and training*. Publisher not identified, 1994.
- [49] W. Schiffmann, M. Joost, and R. Werner, “Synthesis and Performance Analysis of Multilayer Neural Network Architectures. University of Koblenz.” Institute of Physics, 1992.
- [50] F. Jian and X. Yugeng, “Neural network design based on evolutionary programming,” *Artif. Intell. Eng.*, vol. 11, no. 2, pp. 155–161, 1997.
- [51] C. Nikolopoulos and P. Fellrath, “A hybrid expert system for investment advising,” *Expert Syst.*, vol. 11, no. 4, pp. 245–250, 1994.

- [52] J. R. McDonnell and D. E. Waagen, "Evolving cascade-correlation networks for time-series forecasting," *Int. J. Artif. Intell. Tools*, vol. 3, no. 3, pp. 327–338, 1994.
- [53] S. A. Harp, T. Samad, and A. Guha, "Designing Application-Specific Neural Networks Using the Genetic Algorithm.," in *NIPS*, 1989, pp. 447–454.
- [54] F. Gruau, "Genetic synthesis of boolean neural networks with a cell rewriting developmental process," in *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, 1992, pp. 55–74.
- [55] J. W. L. Merrill and R. F. Port, "Fractally configured neural networks," *Neural Networks*, vol. 4, no. 1, pp. 53–60, 1991.
- [56] H. H. Szu and R. L. Hartley, "Nonconvex optimization by fast simulated annealing," *Proc. IEEE*, vol. 75, no. 11, pp. 1538–1540, 1987.
- [57] J. Kennedy, J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*. Morgan Kaufmann, 2001.
- [58] R. Hassan and B. Cohanin, "A comparison of particle swarm optimization and the genetic algorithm," *1st AIAA Multidiscip. Des. Optim. Spec. Conf.*, pp. 1–13, 2005.
- [59] F. van den Bergh, A. P. Engelbrecht, and A. P. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers," in *South African Computer Journal*, 2000.
- [60] M. Carvalho and T. B. Ludermir, "An analysis of PSO hybrid algorithms for feed-forward neural networks training," in *2006 Ninth Brazilian Symposium on Neural Networks (SBRN'06)*, 2006, pp. 6–11.
- [61] H. R. Tizhoosh, "Opposition-Based Learning: A New Scheme for Machine Intelligence.," in *CIMCA/IAWTIC*, 2005, pp. 695–701.
- [62] M. G. H. Omran, *Using opposition-based learning with particle swarm optimization and barebones differential evolution*. INTECH Open Access Publisher, 2009.
- [63] M. Yaghini, M. M. Khoshraftar, and M. Fallahi, "HIOPGA: a new hybrid metaheuristic algorithm to train feedforward Neural Networks for Prediction," in *Proceedings of the International Conference on Data Mining*, 2011, pp. 18–21.
- [64] C. Blum and K. Socha, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification," in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, 2005, p. 6--pp.
- [65] X. Chen, J. Wang, D. Sun, and J. Liang, "A novel hybrid Evolutionary Algorithm based on PSO and AFSA for feedforward neural network training," in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, 2008, pp. 1–5.
- [66] D. Karaboga, B. Akay, and C. Ozturk, "Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks," in *International Conference on Modeling Decisions for Artificial Intelligence*, 2007, pp. 318–329.
- [67] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *Proc. Sixth Int. Symp. Micro Mach. Hum. Sci.*, pp. 39–43, 1995.
- [68] J.-R. Zhang, J. Zhang, T.-M. Lok, and M. R. Lyu, "A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training," *Appl. Math. Comput.*, vol. 185, no. 2, pp. 1026–1037, 2007.

- [69] M. Meissner, M. Schmuker, and G. Schneider, "Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training," *BMC Bioinformatics*, vol. 7, p. 125, 2006.
- [70] Y. Da and G. Xiurun, "An improved PSO-based ANN with simulated annealing technique," *Neurocomputing*, vol. 63, no. SPEC. ISS., pp. 527–533, 2005.
- [71] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999, vol. 3.
- [72] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 240–255, 2004.
- [73] M. Lovbjerg, T. K. Rasmussen, and T. Krink, "Hybrid particle swarm optimiser with breeding and subpopulations," in *Proceedings of the genetic and evolutionary computation conference*, 2001, vol. 2001, pp. 469–476.
- [74] N. Higashi and H. Iba, "Particle swarm optimization with Gaussian mutation," in *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, 2003, pp. 72–79.
- [75] B. A. Garro, H. Sossa, and R. a. Vazquez, "Design of artificial neural networks using a modified Particle Swarm Optimization algorithm," *2009 Int. Jt. Conf. Neural Networks*, pp. 938–945, 2009.
- [76] J. Yu, S. Wang, and L. Xi, "Evolving artificial neural networks using an improved PSO and DPSO," *Neurocomputing*, vol. 71, no. 4–6, pp. 1054–1060, 2008.
- [77] B. A. Garro, H. Sossa, and R. A. Vázquez, "Artificial neural network synthesis by means of artificial bee colony (abc) algorithm," in *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011, pp. 331–338.