Metaheuristic design of feedforward neural networks: a review of two decades of research

Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research

Varun Kumar Ojha*[1], Ajith Abraham[2], and Václav Snášel[1]

[1]*Dept. of Computer Science, VŠB-Technical University of Ostrava, Ostrava, Czech Republic*
[2]*Machine Intelligence Research Labs (MIR Labs), Auburn, WA, USA*

Abstract

Over the past two decades, the feedforward neural network (FNN) optimization has been a key interest among the researchers and practitioners of multiple disciplines. The FNN optimization is often viewed from the various perspectives: the optimization of weights, network architecture, activation nodes, learning parameters, learning environment, etc. Researchers adopted such different viewpoints mainly to improve the FNN's generalization ability. The gradient-descent algorithm such as backpropagation has been widely applied to optimize the FNNs. Its success is evident from the FNN's application to numerous real-world problems. However, due to the limitations of the gradient-based optimization methods, the metaheuristic algorithms including the evolutionary algorithms, swarm intelligence, etc., are still being widely explored by the researchers aiming to obtain generalized FNN for a given problem. This article attempts to summarize a broad spectrum of FNN optimization methodologies including conventional and metaheuristic approaches. This article also tries to connect various research directions emerged out of the FNN optimization practices, such as evolving neural network (NN), cooperative coevolution NN, complex-valued NN, deep learning, extreme learning machine, quantum NN, etc. Additionally, it provides interesting research challenges for future research to cope-up with the present information processing era.

Keywords: Feedforward neural network; metaheuristics; nature-inspired algorithms; multiobjective; ensemble.

*Corresponding Author

Engineering Applications of Artificial Intelligence 60 (2017) 97–116

1 Introduction

Back in 1943 McCulloch and Pitts [1] proposed a computational model inspired by the human brain, which initiated the research on artificial neural network (ANN). ANNs are capable of learning and recognizing and can solve a broad range of complex problems. Feedforward neural networks (FNNs) are the special type of ANN models. The structural representation of an FNN makes it appealing because it allows perceiving a computational model (a function) in a structural/network form. Moreover, it is the structure of an FNN that makes it a universal function approximator, which has the capabilities of approximating any continuous function [2]. Therefore, a wide range of problems is solved by the FNNs, such as pattern recognition [3], clustering and classification [4], function approximation [5], control [6], bioinformatics [7], signal processing [8], speech processing [9], etc.

The structure of an FNN consists of several neurons (processing units) arranged in layer-by-layer basis and the neurons in a layer have connections (weights) from the neurons at its previous layer. Fundamentally, an FNN optimization/learning/training is met by searching an appropriate network structure (a function) and the weights (the parameters of the function) [10]. Finding a suitable network structure includes the determination of the appropriate neurons (i.e., activation functions), the number of neurons, and the arrangements of neurons, etc. Similarly, finding the weights indicates the optimization of a vector representing the weights of an FNN. Therefore, learning is an essential and distinguished aspect of the FNNs.

Numerous algorithms, techniques, and procedures were proposed in the past for the FNNs optimization. Earlier, in FNN research, only the gradient-based optimization techniques were the popular choices. However, gradually because of the limitations of gradient-based algorithms, the necessity of metaheuristic-based optimization methods were recognized.

Metaheuristics formulate the FNN components, such as weights, structure, nodes, etc., into an optimization problem. Metaheuristics implement various heuristics for finding a near-optimum solution. Additionally, a multiobjective metaheuristic deals with the multiple objectives simultaneously. The existence of multiple objectives in the FNNs optimization is evident since the minimization of FNN's approximation error is desirable at one hand, and the generalization and model's simplification is at the other.

In a metaheuristic or multiobjective metaheuristic treatment to an FNN, an initial population of FNNs is guided towards a final population, where usually the best FNN is selected. However, selecting only the best FNN from a population may not always produce a general solution. Therefore, to achieve a general solution without any significant additional cost, an ensemble of many candidates chosen from a metaheuristic final population is recommended.

This article provides a comprehensive literature review to address the various aspects of the FNN optimization, such as:

1. The importance of an FNN as a function approximator and its preliminary concepts

(Section 2), including the introduction to the factors influencing FNN optimization (Section 2.2) and introduction to the conventional optimization algorithms (Section 2.3).

2. The role of metaheuristics and hybrid metaheuristics in FNNs optimization (Section 3).

3. The role of multiobjective metaheuristics (Section 4) and the ensemble methods (Section 5).

4. The current challenges and future research directions (Section 6).

2 Feedforward neural networks

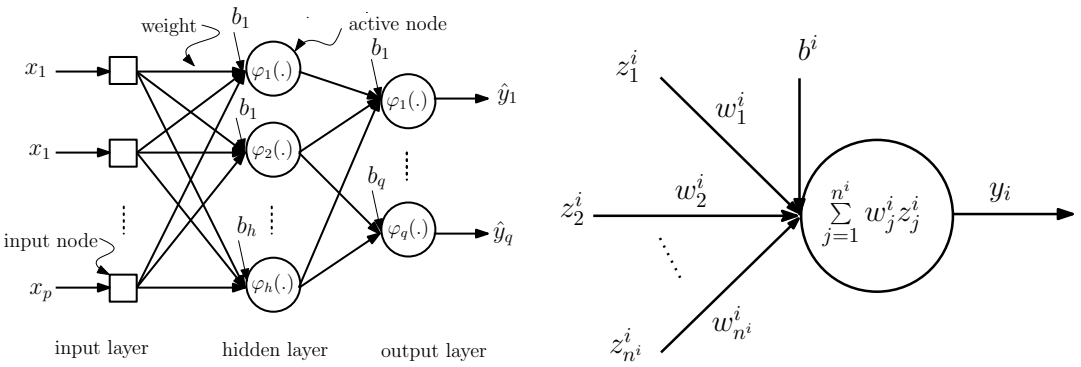
The intelligence of human brain is due to its massively parallel neurons network system. In other words, the architecture of the brain. Similarly, a proper design of an ANN offers a significant improvement to a learning system. The components, such as nodes, weights, and layers are responsible for the developments of various ANN models.

A single layer perceptron (SLP) consists of an input and an output layer, and it is the simplest form of ANN model [11, 12]. However, SLPs are incapable of solving nonlinearly separable patterns [13]. Hence, a multilayer perceptron (MLP) was proposed, which addressed the limitations of SLPs by including one or more hidden layers in between an input and an output layer [14]. Initially, the backpropagation (BP) algorithm was used for the MLP training [15]. A trained MLP is then found capable of solving nonlinearly separable patterns [15]. In fact, MLPs (in general FNNs) are capable of addressing a large class of problem pertaining to pattern recognition and prediction. Moreover, an FNN is considered as a **universal approximator** [2]. Cybenko [16] referring to Kolmogorov's theorem[1] showed that an FNN with only a single internal hidden layer—containing a finite number of neurons with any continuous sigmoidal nonlinear activation function—can approximate any continuous function. Also, the FNN structure (architecture) is itself capable enough to be a universal approximator [2, 18]. Hence, several researchers praised FNN for its universal approximation ability [19–22].

Many other ANN models, like radial basis function [23] and support vector machine [24] are a special class of three-layer FNNs. They are capable of solving regression and classification problems using supervised learning methods. In contrast, adaptive resonance theory [25], Kohenen's self-organizing map [26], and learning-vector-quantization [26] are two-layer FNNs that are capable of solving pattern recognition and data compression problems using unsupervised learning methods.

Additionally, the ANN architecture with feedback connections, in other words, a network where connections between the nodes may form cycles is known as a **recurrent neural net-**

[1]Kolmogorov's theorem: "All continuous functions of n variables have an exact representation in terms of finite superpositions and compositions of a small number of functions of one variable [17]."



(a) Three-layer feedforward neural network (b) Node of the network

Figure 1: Three-layer feedforward neural network (a), where input layer has p input nodes, hidden layer has h activation functions, and output layer has q nodes.

work (RNN) or feedback network model. The RNNs are good at performing sequence recognition/reproduction or temporal association/prediction tasks. RNNs such as Hopfield network [27] and Boltzmann machine [28] are good at the application for memory storage and remembering input–output relations. Moreover, Hopfield network was designed for solving nonlinear dynamic systems, where the stability of a dynamic system is studied under the neurodynamic paradigm [27].

A collection of RNN models, such as temporal RNN [29], echo state RNN [30], liquid state machine [31] and backpropagation de-correlation [32] forms a paradigm called reservoir computing, which addresses several engineering applications including nonlinear signal processing and control. Although some other ANN models that are capable of doing a similar task that of the FNNs were pointed out in this Section, the discussion in this article is; however, limited to only FNNs.

2.1 Components of FNNs

FNNs are the computational models that consist of many neurons (*node*), which are connected using synaptic links (*weights*) and are arranged in layer-by-layer basis. Thus, the FNNs have a specific structural configuration (*architecture*) in which the nodes at a layer have forward connections from the nodes at its previous layer (Fig. 1(a)). A node of an FNN is capable of processing information coming through the connection weights (Fig. 1(b)). Mathematically, the output y_i (excitation) of a node (node indicated as i) is computed as:

$$y_i = \varphi_i \left(\sum_{j=1}^{n^i} w_j^i z_j^i + b^i \right), \tag{1}$$

where n^i is the total incoming connections, z^i is the input, w^i is the weight, b^i is the bias, and $\varphi_i(\cdot)$ is the *activation function* at the i-th node to limits the amplitude of the output the node

into a certain range.

Fig. 1(a) is a structural representation of an FNN, i.e., a phenotype of a function $f(\mathbf{x}, \mathbf{w})$, which is parameterized by a p-dimensional input vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_p \rangle$ and an n-dimensional real-valued weight vector $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$. The function $f(\mathbf{x}, \mathbf{w})$ is a solution of a problem. Therefore, two tasks involved in solving a problem using an FNN are: to discover an appropriate function $f(\mathbf{x}, \mathbf{w})$ (i.e., the architecture optimization) and to discover an appropriate weight vector $\mathbf{w}$ (i.e., the weights optimization) using some *learning algorithm*.

The architecture optimization indicates the search for the appropriate activation functions at the nodes, the number of nodes, number of layers, the arrangements of the nodes, etc. Therefore, several components of an FNN optimization are: the connection **weights**; the **architecture** (number of layers in a network, the number of nodes at the hidden layers, the arrangement of the connections between nodes); the **nodes** (activation functions at the nodes); the **learning algorithms** (algorithms training parameters); and the **learning environment**. However, traditionally, the only component that was optimized was the weights of the connections by keeping other components fixed to the initial choice.

2.2 Influencing factors in FNN optimization

2.2.1 Learning environments

An FNN is trained by supplying the training data (X, Y) of N input–output pairs, i.e., $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ and $Y = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N)$. Each input $\mathbf{x}_i = \langle x_{i1}, x_{i2}, \ldots, x_{ip} \rangle$ is a p-dimensional vector, and it has a corresponding q-dimensional desired output vector $\mathbf{y}_i = \langle y_{i1}, y_{i2}, \ldots, y_{iq} \rangle$. For the training data (X, Y), an FNN produces an output $\hat{Y} = (\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_N)$, where a vector $\hat{\mathbf{y}}_i = \langle \hat{y}_{i1}, \hat{y}_{i2}, \ldots, \hat{y}_{iq} \rangle$ is a q-dimensional FNNs output, which is then compared with the desired output $\mathbf{y}_i$, for all $i = 1$ to N by using some error/distance/cost function. The minimization/reduction of the error/distance function, in an iterative manner, is referred as a **supervised learning**. One very commonly known supervised learning algorithm is Delta rule or Widrow-Hoff rule [33,34] in which the n-dimensional weight vector $\mathbf{w}$ of an FNN is optimized as:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \Delta\mathbf{w}^t, \tag{2}$$

where $\Delta \mathrm{w}^t$ is weight change (an additive term) at t-th iteration. The weight change $\Delta \mathrm{w}^t$ is computed as:

$$\Delta\mathbf{w}_i^t = \eta^t e_i^t \mathbf{x}_i^t, \tag{3}$$

where η^t is a learning rate, which controls the magnitude of weight change at t-th iteration and e_i^t is the error at t-th learning iteration corresponding to i-th training input $\mathbf{x}_i^t$ presented to an FNN. The error e_i^t at the t-th iteration may be computed as: $e_i^t = \sum_{j=1}^{q} (y_{ij}^t - \hat{y}_{ij}^t)^2$, where y_{ij}^t and $\hat{y}_{ij}^t$ are the desired output and FNN's output at t-th iteration respectively.

Contrary to the supervised learning paradigm, there are two other learning forms for the spacial cases of FNNs: 1) the *unsupervised learning*—for the unlabeled training data [35], and 2) the *reinforcement learning*—for the training data with insufficient input–output relations [36]. The focus of this article is, however, on supervised learning paradigms only.

2.2.2 Error functions

A supervised learning, essentially, is the minimization of the difference/distance between the desired output $\mathbf{y}_i$ and the model's output $\hat{\mathbf{y}}_i = f(\mathbf{x}, \mathbf{w})$ by comparing the difference/distance using a cost function $c_f : Y \times \hat{Y} \longrightarrow \mathbb{R}_{\geq 0}$. For this propose, several cost function can be designed. For instance, in regression problems, *mean squared error* is one of the commonly used cost function, which is written as:

$$c_f(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{q} (y_{ij} - \hat{y}_{ij})^2 , \tag{4}$$

where y_{ij} are the desired response and $\hat{y}_{ij}$ are the FNN's responses, and their differences were summed over N data pairs. Some other functions like sum of squared error, root of mean square error, mean absolute error, correlation coefficient, etc., can be used for evaluating the FNN's predictability [37].

The cost function (4) or any similar squared-error-based cost function is inconsistent for solving classification problems [38]. Instead, the *percentage of good classification*, which has consistent behavior, can be used [38]. However, the percentage of good classification is satisfactory until no preference was given to a particular class. Therefore, *accuracy* and *miss-classification rate* are used as the cost functions. A detailed list of the cost function for evaluating the classification problems is available in [39–41].

In this article, cost function mentioned for FNN optimization is discussed in a general sense, which can be thought as the equivalent to any other user-defined cost function. Another factor related to fitness of an FNN is to compare the cost functions of two or more FNN models [42,43]. Some researchers also argue to statistically compare the predicted outputs of two or more FNN models to establish the differences in their performances [44].

2.2.3 Local minima problem

Let $c_f : S \longrightarrow \mathbb{R}_{\geq 0}$, where $S \subset \mathbb{R}^n$ is nonempty and compact (for detailed information about topological compactness, see [45]). Therefore, the following may be defined:

Definition 1. *A point* $\mathbf{w}^* \in S$ *is called* ***global minima*** *if* $c_f(\mathbf{w}^*) \leq c_f(\mathbf{w})$ *for any* $\mathbf{w} \in S$ *holds.*

Definition 2. *A point* $\mathbf{w}^* \in S$ *is called* ***local minima*** *if there exists* $\epsilon > 0$*, and an* ϵ*-neighborhood* $B_\epsilon(\mathbf{w}^*, \epsilon)$ *around* $\mathbf{w}^*$ *such that* $c_f(\mathbf{w}^*) \leq c_f(\mathbf{w})$ *for any* $\mathbf{w} \in S \cap B_\epsilon(\mathbf{w}^*, \epsilon)$ *holds.*

Learning algorithms when to using the cost function (4) or any similar function for FNN optimization has the tendency to fall in local minima [46]. Moreover, the geometrical structure (parameter space) of a three-layer perceptron may fall to local minima and plateaus during its optimization. It indicates that the critical point corresponding to global minima of a smaller FNN model (model with $h-1$ hidden units) can be a local or saddle point of a larger FNN model (model with h hidden units) [47]. However, there are some ways to avoid or eliminate local minima in FNN optimization [48, 49]:

1) If the weights and training patterns are assigned randomly to a three-layer FNN that contains h neurons at the hidden layer, then a gradient-descent algorithm can avoid trapping into local minima [50].

2) If linearly-separable training data and pyramidal network structure are taken, then the error surface will be local minima free [51].

3) If there are N many noncoincident input patterns to learn and three-layered FNN with $N-1$ sigmoid hidden neurons and one dummy hidden neuron is used, then the corresponding error surface will be local minima free.

4) If the training algorithms can be improved as similar to as the global descent learning algorithm proposed by Cetin et al. [52] to replace gradient-descent algorithms, then it can avoid local minima.

These four methods depend on the number of hidden neurons, the number of training samples, the number of output neurons, and a condition that says the number of hidden neurons should not be less than the number of training samples. Moreover, it does not necessarily guarantee to converge to global minima and to set preconditions for the number of hidden neurons and linearly separable training patterns are unlikely conditions for the real-world problems [53].

2.2.4 Generalization

The generalization is a crucial aspect of an FNN optimization, where it is an ability to offer the general solutions rather than performing best for the particular cases. To achieve generalization, FNNs need to avoid both *underfitting* and *overfitting* during training, which is associated with high statistical *bias* and high statistical *variance* [54]. Therefore, one has to address trade-offs between bias and variance. Also, for a good generalization, the number of training pattern should be sufficiently larger than the total number of connections in FNN [55].

The standard methods to achieving generalization are determining an optimum number of free parameters (i.e., equivalent to find an optimum network architecture), *early stopping* of training algorithms, *adding regularization term* with the cost function [56,57], and *adding noise* to the training data.

In *early stopping*, a dataset is divided into three sets: a training set, cross-validation set, and test set. The early stopping scheme suggests stopping of training at the point (epoch) from which onward the cost function value computed on cross-validation set starts to rise [46,58–60]. Similarly, adding noise (jitters) into the training pattern improves FNN's *generalization* ability and removing insignificant weights from a trained FNN improves its *fault tolerance* ability [61]. Moreover, generalization is related to sparsity and stability of a learning algorithm [62].

Now, if the approximation error of two FNN models trained on the same training data is close/similar, then the model with simple network structure (lower number of free parameters) should be selected as the best model. It is because the model with lower network complexity possesses higher generalization ability than the models with higher network complexity [63]. Moreover, the network with lower weight magnitude possesses better generalization ability [63].

2.3 Conventional optimization approaches

Finding a suitable algorithm for the FNNs optimization has always been a difficult task. The FNN optimization using conventional gradient based algorithms is viewed as an unconstrained optimization problem [10,64]. The cost function c_f has to be optimized to satisfy Definition 1. Therefore, the gradient of error g^t at t-th iteration is computed as:

$$g^t = \frac{\partial c_f}{\partial \mathbf{w}^t}, \tag{5}$$

where g^t is a *first-order partial derivative* of the cost function c_f with respect to weight vector $\mathbf{w}$. Hence, a gradient-descent approach starts with an initial guess $\mathbf{w}_0$ and generates a sequence of weight vector $\mathbf{w}_1, \mathbf{w}_2, \ldots$ such that c_f reduces in each iteration. The connection weights at iteration t are updated as:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \Delta\mathbf{w}^t, \tag{6}$$

where the weight change $\Delta\mathbf{w}^t$ is equal to $-\eta^t g^t$, and η^t is the learning rate. The weights updated using (5) and (6) is known as the steepest decent approach. Now, instead of using a first-order partial derivative, a *second-order partial derivative* (∇^2) of cost function c_f can be used as:

$$H^t = \nabla^2 c_f = \frac{\partial^2 c_f}{\partial \mathbf{w}}, \tag{7}$$

where H^t is *Hessian* matrix at the t-th iteration [65]. Hence, the weight change $\Delta\mathbf{w}^t$ using second–order Taylor's series expansion of cost function c_f around point $\mathbf{w}^t$ is computed as:

$$\Delta\mathbf{w}^t = -H^{t-1} g^t, \tag{8}$$

where H^{t-1} is the inverse of Hessian matrix H^t and the weight change $\Delta\mathbf{w}^t$ is known as the *Newton method* or Newton update [10]. In the past, several algorithms were proposed using (5) and (8). Some of them are summarized as follows:

Backpropagation (BP) is a **first-order gradient-descent** algorithm for the FNNs optimization [14, 15]. In BP, the error computed at the output layer is propagated backward to the hidden layers. BP algorithm has two phases of computation: *forward computation* and *backward computation*, where at t-th iteration, the weight change $\Delta\mathbf{w}^t$ for l-th layer is computed as:

$$\Delta\mathbf{w}_l^t = \alpha^t \mathbf{w}_l^{t-1} + \eta^t g^t \mathbf{y}_{l-1}, \tag{9}$$

where $\mathbf{y}$ is inputs/excitation from previous layer $l-1$, η^t is learning rate and α^t is momentum factor.

The choice of learning rate η^t and momentum factor α^t are critical to gradient-descent technique. The momentum factor α^t allows BP training to be biased with previous iteration weights that help convergence rate to be faster. BP is sensitive to these parameters [15]. If the learning rate is too small, learning will become slow, and if the learning rate is too large, learning will be zigzag and algorithm may not converge to required degree of satisfaction. Additionally, a high momentum factor leads to a high risk of overshooting minima and a low momentum factor may avoid local minima, but learning will be slow. The classical BP algorithm is slow and has a tendency to fall in local minima [51].

Since the basic version of BP is sensitivity towards learning rate and momentum factor [15], several improvements were suggested by researchers: 1) a fast BP algorithm, called *Quickpro* was proposed in [66, 67]; 2) a *delta-bar* technique and an *acceleration* technique was suggested for tuning BP learning rate η in [68] and in [69] respectively; and 3) a variant of BP, called resilient propagator (*Rprop*) was proposed in [70].

In the *Rprop*, if the gradient direction in iteration n remains unchanged from its previous iteration $t-1$, then the weight change will occur in larger magnitude, else in smaller. In simple words, if gradient sign remains unchanged from previous iterations, the magnitude of learning rate η will be large, otherwise small. The proposed *Rprop* improves determinism of convergence to global minima [70]. However, it is not faster than the *Quickpro*, but still faster than BP [71].

Contrary to BP, a **second-order minimization method**, called *conjugate gradient* (CG) can be used for weights optimization [72–74]. The CG does not proceed down with a gradient; instead, it moves in the direction that is conjugate to the direction of the previous step. In other words, the gradient corresponding to the current step stays perpendicular to the direction of all the previous steps, and each step is at least as good as its previous step. Such series of steps are non-interfering. Hence, the minimization performed in one step will not be undone by any further steps. Several variants of the CG were proposed in the past [75].

Similar to the CG, many other variants of derivative-based conventional methods are used for weights optimization: *Quasi-Newton* [65], *Gauss-Newton* [76], or *Levenberg-Marquardt* [77]. Quasi-Newton uses a second-order partial derivative (7) of error (4), and it computes its weight search direction by using Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [78]. In Gauss-Newton method, the FNNs optimization is framed as a nonlinear least square optimization

problem, which suggests to using the sum of squared error (4) [77]. Many researchers suggested that the Levenberg-Marquardt (LM) method outperforms BP, CG, and Quasi-Newton methods [79, 80]. Several other methods were proposed for the FNNs optimization are based on *Kalman-filter* [81, 82] and *recursive least squares method* [83].

2.4 Comments on conventional approaches

The gradient-descent based conventional algorithms operate on a single solution (a weight vector) during the optimization. Thus, these algorithms are computationally faster than the algorithms that use two or more solution vectors during the optimization and select the best solution vector at the end of optimization iterations. Moreover, the gradient-decent methods such as BP [15], Online BFGS [84] can be applied for the stochastic as well as batch mode training of the FNNs.

The basic advantages of the stochastic/online training of an FNN are its ability to address redundancy in training pattern, the inclusion of training data that are currently not in training set (i.e., the possibility of dynamic learning), and faster training than that of batch mode [85]. Whereas, most of the other second-order gradient-descent methods and metaheuristic algorithms can only use batch mode training. However, a batch mode (offline) training of an FNN can at least guarantee a local minima under a simple condition compared to a stochastic/online training, and for a larger dataset, batch mode training can be faster than stochastic training [86].

On the contrary to the advantages of the conventional methods, they have several limitations. For example, they have the tendency to fall in local minima, and they are only used for optimizing the FNN weights. Primarily, the gradient-descent algorithms depend on the error function, e.g., mean square error or sum of squared error. For instance, the least square methods like the Gauss-Newton and LM works only if the cost function is the sum of squared error. The Newton method has to compute a Hessian matrix (7), which has to be positive definite and computing the Hessian matrix (7) can be hard and expensive. Similarly, the Quasi-Newton and CG methods need to use a line-search method that sometimes can be expensive.

Moreover, the FNN generalization, as mentioned in Section 2.2.4, needs the reduction in the number of weights (less complex network architecture). Hence, the application of conventional algorithms is limited compared to the metaheuristic algorithms such as the genetic algorithm (GA) that can be directly applied to an FNN for its automatic structure determination and complexity reduction [87, 88]. Similarly, a metaheuristic algorithm can formulate an FNN such that the insignificant weights of the network can be eliminated to improve the FNN generalization ability. Moreover, metaheuristic algorithms can evolve an FNN as a whole by optimizing its components simultaneously.

3 Metaheuristic approaches

So far, only the gradient-descent based algorithms were discussed, which are local search algorithms. They are good at exploiting the obtained solutions to find new solutions. However, to find a global optimum solution, any optimization algorithm must use two techniques: 1) *exploration*—to search new and unknown areas in a search space, and 2) *exploitation*—to take advantage of the already discovered solution [89]. The exploration and exploitation are two contradictory strategies and a good search algorithm must find a trade-off between these two. Metaheuristic is the procedure that implements nature-inspired heuristics to combine these two strategies [90]. Hence, metaheuristic approaches are alternative to the conventional approaches for optimizing the FNNs.

Unlike the conventional methods, which require the objective function to be continuous and differential, the metaheuristic algorithms have the ability to address complex, nonlinear, and non-differentiable problems. However, the optimization algorithms are often biased towards a specific class of problems, that is, "there is no such universal optimizer which may solve all class of problem," which is evident from no free lunch theorem [91].

Wolpert and Macready [91] introduced *no free lunch* (NFL) theorem to answer the question, "whether a general purpose optimization algorithm exists." Moreover, Wolpert [92] introduced NFL for optimization algorithm to answer the question, "How does the set of problems $F_1 \subset \mathcal{F}$ for which algorithm a_1 performs better than algorithm a_2 compares to the set $F_2 \subset \mathcal{F}$ for which the reverse is true." Here, $\mathcal{F}$ is space of all possible problems. To answer this question, Wolpert proposed NFL theorem, which says that "the average performance of any pair of algorithms across all possible problems is identical" [91].

Therefore, a straightforward interpretation of NFL is as follows. "A general purpose universal optimization strategy is impossible, and the only way one strategy can outperform another if it is specialized to the structure of the specific problem under consideration" [93]. Schumacher et al. [94] argue that the NFL theorem [91] holds only for the set of problems which are closed under permutation (c.u.p). Therefore, indeed the performance of one algorithm can be improved over another for the problems that are not c.u.p and most of the real-world problems are not c.u.p [95]. Such is the reason why in the past researchers were inclined to create and improvise algorithms for optimizing the FNNs.

3.1 Metaheuristic algorithms

Since a large variety of metaheuristic algorithms are available, it is difficult to classify metaheuristic algorithms precisely into different classes. Though, intuitively, three basic categorize can be done:

3.1.1 Single solution based algorithms

A single solution based metaheuristic algorithm operates on a single solution (candidate) and applies some heuristic inspired by the nature or some other phenomena on the current solution. For example, algorithms like simulated annealing (SA) [96], tabu search (TS) [97], variable neighborhood search (VNS) [98], greedy randomized adaptive search (GRAP) [99], etc., improves a single solution by searching around its neighborhood and continue to improve the solution until a satisfactory solution is obtained. For instance, the heuristics of some algorithms are as follows:

SA is a probabilistic approach that imitates the cooling strategy (annealing process) of a metallurgy industry. It uses Monte Carlo method [100] to determine the acceptance probability of a newly generated solution in the neighborhood of the current solution. Hence, for a given search space, SA should guide a solution towards a global optimal solution [96, 101]. Similarly, TS is inspired by the human behavior of tabooing objects [97]. In other words, TS discourages (tabu) the acceptance of the solutions that are already explored in the past. Hence, it improves upon SA by introducing some additional restriction on the acceptance of the new solutions.

Since a single solution based algorithm exploits the current solution, it also is known as the *local search algorithm*. The focus of this article is to illustrate the application of the metaheuristic algorithms for the FNNs optimization. Hence, for the detailed description of the mentioned algorithms, the readers may explore the respective references.

3.1.2 Population based algorithms

Population based algorithms operate on the multiple solutions (candidates) and apply the heuristics inspired by nature, biological evolution, biology, or some other forms. In contrast to the single solution based algorithms, the population based algorithms have a high exploration (global search) ability. The following are the population based algorithms:

Evolutionary algorithms (EA) Genetic algorithms (GA) [87, 88], evolutionary programming (EP) [102], evolutionary strategy (ES) [103], genetic programming (GP) [104], differential evolution [105], etc., are the algorithms inspired by the dynamics of natural selection and use the operators, such as *selection*, *crossover*, and *mutation* to find a near-optimal solution [88]. EA framework offers an exploration of a vast search space and guarantees to find a near-optimal solution. Since EAs do not depend on gradient information, they solve a large range of complex, nonlinear, nondifferentiable, multimodal optimization problems. Also, EAs give a wider scope in the FNN optimization since EAs can optimize both discreet and continuous optimization problem, and the FNN components can be formulated into both ways.

The differences between EAs can be briefly stated as follows: GA uses genetic operators, such as selection, crossover, and mutation to search optimum genetic vector from a search space [88];

whereas, only the mutation operator are used in ES to evolve a real vector solution [103]. On the other hand, GP searches an optimum program structure from a topological search space of computer programs [104] and EP are used for evolving parameters of a computer program whose structure is kept fixed [102].

Swarm intelligence (SI) SI algorithms are inspired by the collective and self-organized behavior of the swarm (insects, birds, fish, etc.). Particle swarm algorithm (PSO) [106], ant colony optimization (ACO) [107], artificial bee colony (ABC) [108], bacterial foraging optimization (BFO) [109], etc., are some widely used SI algorithms. The basic principle of SI algorithms is as follows. First, a swarm (collection of solutions) are randomly generated. Then, the heuristic inspired by the swarm behavior modifies the current solution. For example, in PSO, ACO, ABC, and BFO, the heuristics are inspired by the foraging behavior of birds, ants, bees, and bacteria respectively. In these algorithms, an FNN component is formulated as a solution for the optimization.

In PSO, a swarm, as a whole, is like a flock of birds (particles, which are the weight vectors) that collectively foraging (explore search space) for food (best weight vector) and is likely to move close to an optimum food-source [106, 110]. Moreover, each particle bears two properties: location and velocity. The location of a particle is a solution vector (weight vector $\mathbf{w}$), and velocity η is a vector equal to the size of the location vector. Each particle determines its movement using knowledge of its best locations, global best location, and random perturbations [106, 111].

In ACO, the artificial ants explore the area around their nest (colony) for searching a food source. ACO takes advantage of ants ability to choose the shortest path to a food source by communicating among each other's using a pheromone secretion [112]. This behavior of ants led to the development of ACO algorithm [107].

Similarly, in ABC, three kinds of honey bees, such as employed bee, onlooker bee, and scout bee are responsible for searching food source [108]. Each employed bee memorizes a food source, i.e., a solution (weight vector). Then, each onlooker bee examines the nectar amount (fitness of solution) of a food source memorized by the employed bees and depending on nectar amount; they send scout bees for searching new food source. Hence, they iteratively construct the solution. The readers are encouraged to explore the detail description the algorithms in their respective references.

Other metaheuristics The population based metaheuristic algorithms metaphor is exploited to device several algorithms. There are algorithms inspired by the behavior of animals, birds, and insects, such as gray wolf optimization (GWO) [113], flower pollination (FP) [114], cuckoo search (CS) [115], firefly (FF) [116], etc.

Similarly, there are algorithms inspired by some phenomenon observed in the physics and

chemistry, such as harmony search (HS) [117], central force optimization (CFO) [118], gravitational search optimization (GSO) [119], etc. A detailed list and classification of metaheuristic algorithms are provided in [120].

The growing number of metaheuristic algorithms has drawn researchers to examine the metaphor, the novelty, and the significant differences among the metaheuristic algorithms [121, 122]. In [122], the author provided an insight of the metaheuristic developed over the time, starting from SA to TS, EA, ACO, HS, etc. The author claimed that most of the metaheuristic are similar in nature and do not provide a groundbreaking method in optimization. Despite the criticisms, the author acknowledged the quality of metaheuristic research has been produced and can be produced.

3.1.3 Hybrid and memetic algorithms

An effective combination of various metaheuristic algorithms may offer a better solution than that of a single algorithm. A paradigm of hybrid algorithms, called *memetic algorithm* gave a methodological concept for combining two or more metaheuristics (global and local) to explore a search space efficiently and to find a global optimal solution [123].

The conventional algorithms have the local minima problem because they lack global search ability, but they are fast and good in local search. On the other hand, the metaheuristics are good in global search, but they suffer premature convergence [124, 125]. Therefore, a combination of these two may offer a better solution in FNN optimization than that of using any one of them alone (Fig. 2). To reach a global optimum, a hybrid strategy can be used. Fig. 2 shows an impact of hybridization of metaheuristics on the FNNs optimization. The hybrid algorithms can be categorized in two paradigms:

1) The combination of conventional and metaheuristic algorithms—to take advantage of local search and global search algorithms.

2) The combination of two or more metaheuristic algorithms—to make use of different heuristics or a combined influence of two or more heuristics of global search algorithms.

Under the definition of memetic algorithms, researchers combine EAs with conventional algorithms [126]. For example, the effectiveness of global (GA) and local search (BP) combination is explained in [127, 128]. Similarly, a hybrid PSO and BP algorithms for optimizing FNN were found useful in obtaining better approximation than using one of them alone [129].

A convergence scenario similar to Fig. 2 was illustrated in [130], where ABC was applied for searching initial weights and LM was applied for optimizing the already discovered weights. An example of effectively combining two metaheuristic GA and PSO is illustrated in [131], where both GA and PSO optimized the same population. A detailed description of the hybrid metaheuristic algorithms for the FNN optimization is described in the following Section.

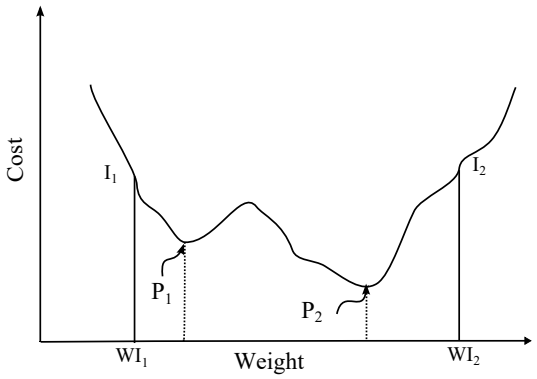


Figure 2: Metaheuristic may be used for finding initial weights WI_2 and conventional algorithms may be for finding global optima P_2 and vice versa [132].

3.2 Metaheuristic formulation of the FNN components

Metaheuristics are stochastic/non-deterministic algorithms. Hence, they do not guarantee a global optimal solution, but they can offer a near-optimal (satisfactory) solution. Moreover, metaheuristics efficiently solve a wide range of complex continuous optimization problems; especially when the problems have incomplete and imprecise information.

The basic form of FNN optimization is the act of searching its weights (free parameters of FNN) such that the cost function (4) or similar function can be minimized. However, the goodness (performance) of FNN cost function depends not only on finding optimum weights, but finding the optimum architecture, activation function, parameter setting of learning algorithm, and training environment are equally important. To apply metaheuristic algorithms for optimizing an FNN, its components (*phenotype*) need to be formulated into a vector (*genotype*) form.

Usually, the FNN components, such as weights, architecture, activation function, learning rule, etc., are considered arbitrarily. Then, a learning algorithm is applied to search weights while the other components are kept fixed to their initial setting. The metaheuristics, on the other hand, allow us to optimization each component simultaneously or a combination of components efficiently (Fig. 3).

The *Venn diagram* in Fig. 3 illustrates the spectrum of FNN components optimization: weights, architecture, activation function, and learning rule's parameters. In Fig. 3, area "a1" indicates the optimization of weights; area "a2" indicates the optimization of weights and architecture; area "a3" indicates the optimization of weights, architecture, and activation function; and all other possible combinations. Examining Fig. 3, one can say that the strength and complexity of optimization increases from area denoted "a1" to "a8," where "a1" is the simplest approach and "a8" is the most sophisticated approach.

Each FNN component can be separately optimized on a one-by-one basis. Therefore, firstly, the weights can be optimized by keeping a fixed architecture; secondly, the architecture can

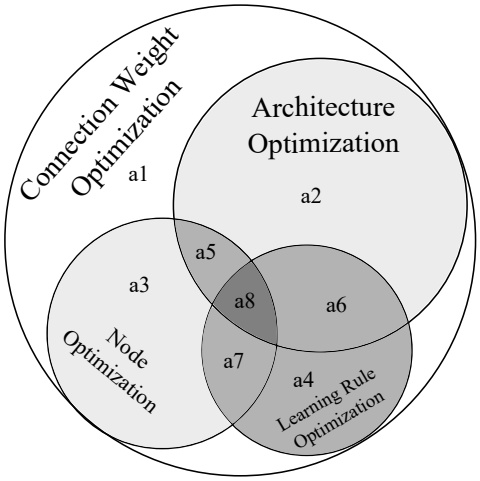


Figure 3: Spectrum of metaheuristic design of FNN

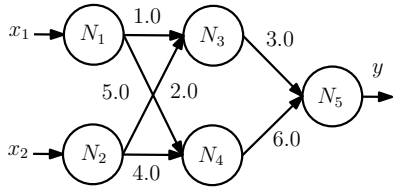
be optimized keeping weights fixed; thirdly, the activation function can be optimized keeping architecture and/or weights fixed; and so on. Another way is to optimize all or a combination of FNN components simultaneously. Therefore, weights and architecture can be optimized, simultaneously; or weights and activation functions, simultaneously; or weights, architecture, and activation functions simultaneously; and so on. In the simultaneous optimization of all or a combination of components, a vectored representation of all components, or a combination of components can be optimized respectively. Once a vectored representation (genotype) is designed, then one of the available metaheuristic algorithms in the literature can be applied to optimize the designed vector to obtained an optimum FNN.

Now, there are *single-solution* based and *population* based metaheuristics [133]. In a single-solution based metaheuristic algorithm, a genotype $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$ is used. Whereas, in a population-based metaheuristic algorithm, a collection of many genotypes are used. In other words, a population matrix $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$ of m weight vectors is used.

Yao and Liu [134] identified evolution at various components of FNN that fall into the spectrum of metaheuristic design of FNN shown in Fig. 3. This Section will describe *how researchers applied metaheuristics for evolving FNN*. The evolution in FNN components is described here one-by-one, as follows. Here, the word optimization, adaptation, and evolution are used in the similar context.

3.2.1 Weight optimization

FNN weight optimization is the most common and widely studied approach, in which the weights are mapped onto an n-dimensional weight vector $\mathbf{w}$, where n is the total number of weights in a network. The vector $\mathbf{w}$ is a genotype representation of a phenotype (FNN structure), where the weight $\mathbf{w} \in \mathbb{R}^n$. The weights w_i, an element of vector $\mathbf{w}$, may be encoded in the following ways: by assigning a real value, l-bits binary coding, l-bits gray coding, IEEE



(a)

$c =$

	1	2	3	4	5
1	**0**	0	1	1	0
2	0	**0**	1	1	0
3	0	0	**0**	0	1
4	0	0	0	**0**	1
5	0	0	0	0	**0**

(b)

(i) $\mathbf{w} = \langle 1.0, 5.0, 2.0, 4.0, 3.0, 6.0 \rangle$

(ii) $\mathbf{w} = \langle 0001, 0101, 0010, 0100, 0011, 0110 \rangle$

(iii) $\mathbf{w} = \langle 0001, 0111, 0011, 0110, 0010, 0101 \rangle$

(c)

Figure 4: Mapping of phenotype to genotype. (a) Phenotype of a three-layer FNN. (b) Adjacency matrix. (c) Weights encoding: i) real value; ii) 4-bits binary; iii) 4-bits gray encoding.

floating point coding, etc. Fig. 4 is an example of phenotype to genotype mapping, where a phenotype shown in Fig. 4(a) that has the connectivity matrix c as per Fig. 4(b) is encoded into three different weight vectors shown in Fig. 4(c).

FNN weights optimization using metaheuristic is practiced from **early 80's** when even the term metaheuristic was not used. Engle's [135] work on FNN weight vector optimization using SA was the first evidence of metaheuristic application. To optimize weight vector using SA, first, the phenotype was mapped onto a real-valued weight vector (Fig. 4(c)), and to compute fitness of the FNN, a *reverse mapping* from genotype (weight vector $\mathbf{w}$) to phenotype (FNN) was used. Such process was continued until a satisfactory solution was found. SA based FNN weight optimization was found to be performing better in comparison to conventional approaches [136–138].

Similar to Engle's [135] approach of phenotype to genotype mapping and vice versa, in [139], the FNN weights optimization was performed using TS. In [140], an improvised TS, called reactive tabu search was used for optimizing weights. Several studies show that TS when used for optimizing FNN weights, outperformed BP and SA algorithm [141, 142]. However, SA and TS are single solution based algorithms, which has a limited scope of exploring search space to obtain a global optimal solution. In contrast, the EAs, SI, or other bio-inspired metaheuristics are population-based algorithms that operate on multiple agents to explore a search space. Hence, they have a better exploration ability than SA, TS, BP, CG, and other single solution based algorithms [88, 110].

For optimizing the FNN weights, EAs use two **types of vector representation**: real-valued and binary valued vector representation. In Fig. 4(c), following weight vector representation are illustrated: 1) real-valued coded chromosome, 2) binary coded chromosome, and 3) binary gray-coded chromosome.

Goldberg and Holland [88] gave the idea of FNN training using GA. However, Whitley and Hanson [143] were the first to propose "GENITOR," a GA based FNN training procedure that used binary-coded chromosome (Fig. 4(c)) for optimizing the weights. Many others followed the idea of GENITOR with some additional improvements such as connectivity optimization and reduced search space introduction [144, 145]. On the other hand, in [146], a binary gray

coding (Fig. 4(c)) scheme was used for optimizing the weights, where at first, GA was used for finding initial weights that were further optimized by using BP and vice versa.

The binary bit-string representation of the weights leads to a **precision problem**, i.e., *how many bits would be sufficient for representing weights* and *what would be the total length of a chromosome*. Moreover, the binary representation is computationally expensive because, in each training iteration, a binary to real-valued mapping and vice versa is required. Hence, it is advantageous to use the real-coded chromosome (Fig. 4(c)) directly [147–151].

Traditional **EA operators** are applied on the binary chromosome. Thus, operators, such as bias-mutation, unbiased-mutation, node-mutation, weight-crossover, and gradient-operator, etc., were defined, for operating on the real-valued chromosome [147]. On the other hand, a matrix-based representation of weights, where a column-wise and a row-wise crossover operators were also defined [152]. The GA-based real coded weights optimization outperforms BP and its variants for solving real-world applications [153–156]. Moreover, an evolutionary inspired algorithm, called differential evolution (DE) [105,157] that imitates mutation and crossover operator to solve complex continuous optimization problems was found to be performing efficiently for real-valued weight vector optimization [158–160].

Similar to DE, **swarm-based or bio-inspired based metaheuristics** directly apply heuristics inspired by nature on a real-valued vector. Hence, they are advantageous in comparison to an EA-based algorithm that needs to simulated mutation and crossover operators for real-valued weight vector [110]. It was found that **PSO** guides a population of the FNN weight vectors towards an optimum population [161,162]. Hence, many researchers resorted to working on swarm based metaheuristics for the FNN optimization.

A *cooperative PSO*, which suggests to splitting a solution vector into n parts, where each part optimized by a swarm of m particles [163,164]. Thus, an $n \times m$ combinations are constructs an n-dimensional composite vector, where each m swarm contributes to the fitness of a solution. Such cooperation between swarms led to a better performance than that of the basic version of PSO. Similarly, a *multi-phase PSO* was proposed in which particle position was updated only when improvement in location was found; otherwise, the location was copied as-it-is into the next generation [165].

A *cultural cooperative particle swarm optimization* (CCPSO) approach in which a collection of multiple swarms that interact by exchanging information was proposed by Lin et al. [166]. The CCPSO performed better than BP and GA when it was applied for optimizing a fuzzy neural network. Similarly, a hierarchical particle swarm optimization was used to design a beta basis function neural network [167].

Apart from PSO, there are numerous metaheuristic algorithms among which, some significant metaheuristics were discussed here that were applied for FNN optimization. The continuous version of **ACO** [168] was efficiently applied to optimize the FNN weight vector [169]. ACO trained FNN was found efficient in solving real-life applications, such as scheduling, prediction,

image recognition, etc. [170, 171].

ABC was efficiently applied on weight vector for optimizing the FNNs [172–174]. Similarly, considering the efficiency of **HS** algorithm—that has a slow convergence rate, but guarantees a near-optimum solution [117]—many researchers applied HS for optimizing weight vector of the FNNs [175, 176]. Moreover, the efficiency of HS comes from using m many harmonies (weight vectors), and iteratively improvising each harmony by computing new harmony (new solution vectors) using heuristic inspired by music pitch modification [117, 177, 178].

In the past, many **other forms of metaheuristics** were also used for optimizing the FNNs. For example, the application of FF, CS, GSO, BFO, and CFO algorithms for the FNN weights optimization is available in [179], [180], [181], [182, 183], [184] respectively.

Moreover, a comparative study showed that FF algorithm performed better than that of BP, GA, and ABC for weight vector optimization [185]. In [186] a detailed study explains the application of the local and global metaheuristic algorithm for FNN optimization. For example, local search algorithms like SA, TS, GRAP, VNS [98], estimations of distribution algorithm [187] and global search algorithms like GA, ACO, and memetic algorithm, were examined thoroughly in [186]. Additionally, many researchers studied the performance of metaheuristic algorithms for the training of the FNN and reported that the metaheuristic approaches outperform all the conventional methods by a huge margin [188–190].

The **memetic algorithm** supports the hybridization of two or more global metaheuristics for the FNN optimization, which is evident from the following examples. A hybrid GA and PSO approach for optimizing the FNN were proposed in [131], where both GA and PSO were suggested to run over the same population—randomly generated population W of m individuals (the same individual was treated as a chromosome in GA and a particle in PSO). In each generation of GA and PSO, the fitness of each individual was computed. Then, the best performing individuals (top-half) were marked as elites. The elite individuals were copied to next generation and half of the copied elites were optimized using PSO and the remaining half using GA through tournament selection and crossover operation.

Similarly, a PSO and SA based **hybrid algorithm** for optimizing FNN were proposed in [191], where, in each iteration, each PSO particle was governed by SA metropolis criteria [100] that determined global best particle for PSO algorithm. There are several other hybrid algorithm examples available in the literature: a hybrid PSO and GA [192]; hybrid GA and DE [193]; hybrid PSO and GSO [194]; and hybrid PSO and optimal foraging theory [195].

3.2.2 Architecture plus weight optimization

The basic architecture optimization approach is a *cascade correlation learning*, which iteratively adds nodes to hidden layer to construct optimum architecture [66]. Moreover, a *constructive* (add node iteratively) and *destructive* (prune nodes iteratively) method [196]. However, the constructive and the destructive methods for optimizing architecture are no different from the

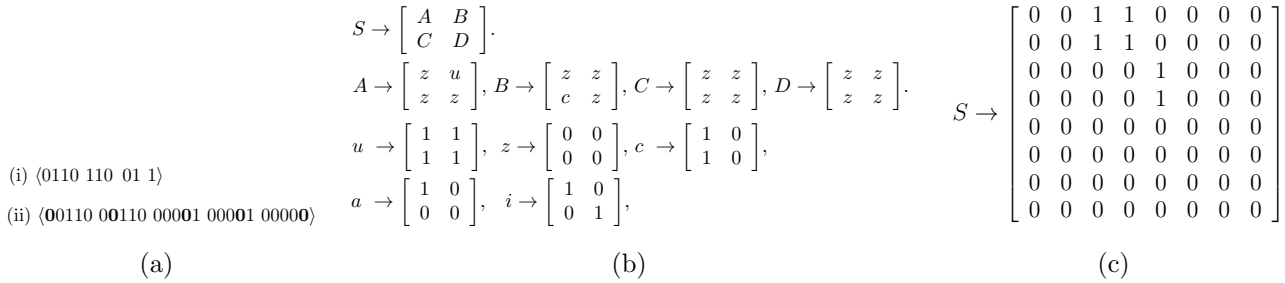


Figure 5: Mapping of phenotype (Fig. 4 to genotype (for architecture). (a) Direct encoding to a vector of connectivity matrix (Fig. 4(b)): i) upper right triangle; ii) complete connectivity. (b) Indirect encoding schemes for architecture (Fig. 4(a)), where S is a start symbol, A, B, C, and D are the variables, and a, c, i, and u is the terminal. (c) Complete connectivity derived from rules operation shown in Fig. 5(b).

manual *trial-and-error* method. Therefore, genetic representation of the FNN architecture as mentioned in Figs. 5(a), 5(b), 5(c) can be used for architecture optimization, which is equivalent to searching optimum architecture from a compact space of FNN topology [197, 198].

Let us discuss the genetic representation of architecture in detail. A **direct encoding scheme** (Fig. 5(a)) was proposed in [199, 200], where authors used an adjacency matrix (Fig. 4(b)) to represent connections between nodes, where between any two nodes i and j, a presence of connection is indicated by "1", and absence of connection is indicated by "0". Hence, they were able to encode complete structural information into a chromosome. However, it is disadvantageous because chromosome length increases with network size. Therefore, if only the network's structural information can be encoded into genotype, then, it will avoid chromosome length problem [201]. Additionally, the encoded network structural information can be accessed using rule-based recursive equation [202]. Moreover, the represented parametric/structural information into the chromosome can indirectly provide access to the rest of the structural details from a predefined archive (parametric information) [203].

The **indirect encoding scheme** reduces chromosome length, where parametric information, such as the number of hidden layers, the number of nodes at hidden layers, the number of connection, etc., makes an archive s. The production rule (Fig. 5(b)) allow us to get access to complete structural information (Fig. 5(c)). Hence, a rule based encoding scheme allows a better FNN architecture optimization than a direct encoding scheme [204].

Unlike the weight optimization that has only limited ways of genetic representation, the FNN architecture optimization is an interesting area of research as there are various ways to represent architecture into genotype. It is evident from a fractal configured FNN representation in [205], where authors define each node using parameters, namely, edge code, input coefficient, and output coefficient. Similarly, in [206], GA was applied to evolve each layer separately and in [207], a grammar encoding and colonial competitive algorithm were proposed.

Another approach to the genetic representation of architecture is to encode weights $\mathbf{w}$ (real vector: Fig. 4(c)) and architecture vector $\mathbf{a} = \langle a_1, \ldots a_m \rangle$ (binary vector as Fig. 5(a))

into a combined genotype. Hence, a single solution vector $s = \langle w_1, \ldots, w_n, a_1, \ldots, a_m \rangle$ is obtained [208], which can be optimized by using metaheuristics.

Many researchers improvised the algorithms itself to optimize architecture. Such examples are as follows: in [209], a **PSO-PSO** method was proposed, in which a PSO (inner PSO block) was applied for optimizing weights that were nested under another PSO (outer PSO block) which was applied for optimizing the architecture of FNN by adding or deleting hidden node. Similarly, in [210, 211], a hybrid Taguchi-genetic algorithm was proposed for optimizing the FNN architecture and weights, where authors used a genetic representation of the weights, but they select structure using constructive method (by adding hidden nodes one-by-one). A multidimensional PSO approach was proposed in [212] for constructing FNN automatically by using an architectural (topological) space. Moreover, the individuals in the swarm population were designed in such a way that it optimized both position (weights) and dimension (architecture) of an individual in each iteration. Thus, optimized FNN weights and architecture simultaneously.

So far, only genetic representation was discussed for evolving architecture. However, **GP** can optimize a phenotype itself, where genetic representation is not required [213]. Therefore, EP and GP can be directly applied to a population FNN architecture to evolve an optimum FNN architecture [148, 214, 215].

The design of the FNN architecture is responsible for processing high-dimensional data. Hence, **deep learning** paradigm offers study massive and deep structure of the neural network that can process complex problems related to speech processing, natural language processing, signal processing, etc., [216, 217]. Such a variant of the FNN is *convolutional neural networks* (ConvNets), which is designed to process data from the multiple arrays form such as a color image composed of three 2D arrays [216, 217]. The ConvNets has a three-dimensional arrangement of neural nodes. Hence, it efficiently receives 3D inputs and processes them to produce 3D outputs [218].

In contrast to deep network paradigm, an **extreme learning machine** (ELM) based hierarchical learning framework (H-ELM) proposed in [219] claimed a faster learning than deep learning by ELM [220] based auto-encoding. The proposed H-ELM framework worked in two phases: unsupervised hierarchical feature representation and 2) supervised feature classification [219].

3.2.3 Input layer optimization

Input layer optimization resembles feature reduction, which is traditionally performed separately by dimensionality reduction methods [221]. However, reducing input dimension by optimizing input layer, i.e., by feeding a subset of input features at the input layer than by feeding the whole set of input features enhances FNN's performance [222, 223]. Therefore, FNN has a functional dependency on the problem at hand.

EAs select a subset of input features for which FNN perform better than that of the complete feature set [222]. For this purpose, a genetic representation of input features is required in which the available features are placed on a genetic strip, and the presence of a feature is marked as "1" and the absence of a feature is marked as "0." Such mechanism of input layer optimization was found advantageous in improving NN performance [224].

Moreover, binary PSO [225], which is a **discrete optimization method** was employed for selecting the input features which were binary coded [226, 227]. In [226], a modified version of binary PSO was proposed where EA like mutation operator was applied to mutated binary vectors. Similarly, ACO, which traditional solves discrete optimization problem was applied to select input features and training of an FNN in a hybrid manner [228].

Input layer optimization which is related to input feature reduction can also be thought as **training data optimization**. Training data optimization is helpful, particularly when data is insufficient or noisy. In [229], an adaptive selection of input examples was performed by employing genetic selection, where two-point and one-point crossover operations created new example patterns. For the crossover operations, the parent's examples were drawn from the original input set. Also, mutation operators were also applied for generating new child example. Hence, the efficiency of FNN was improved when trained over the modified new examples.

Additionally, in [230], an **input example generation** methods was proposed in which the input space was divided into many regions, and k-nearest neighbor method was applied to determine/generate a new virtual example, mainly for the sparse region of the input space. Hence, both the above methods of input example generation sought to enrich knowledge space for the FNN learning [229, 230].

3.2.4 Node optimization

Primarily, node optimization can be addressed in three ways: 1) by choosing activation functions at the FNN active nodes from a set of activation functions [156, 231]; 2) by optimizing the arguments of activation function [232]; and 3) by placing a complete model at the nodes of a network [233, 234].

It was found that FNN performed better when it has **non-homogeneous nodes** (different activation function at different nodes) than that of the homogeneous nodes [235]. In [231], evolution in FNN nodes were offered by selecting sigmoid and Gaussian function adaptively at the nodes [231]. Moreover, adaptation in both nodes and architecture using EAs, where the design of nodes was inspired by locus flight system and tailflip of crayfish [236], can further improve FNN performance [237]. For this purpose, in [238], an input dependent FNN that had a combined chromosome representation (Fig. 6) was proposed, where a real-coded GA for simultaneous optimization of weights, activation functions, and architecture was used.

On the other hand, to optimize nodes, a **family competitive EA** was proposed in [239], where three operators, such as decrease-Gaussian-mutation, Cauchy-mutation, and self-adaptive-

mutation were defined. Moreover, family-competition is a process that generates a pool of L many FNNs by recombination and mutation operations and selects an FNN from that pool. The family-competition with different mutation operator is repeated until the best FNN is found. Many others found that the adaptation in FNN nodes by one of the methods mentioned above can improve FNN performance to some extent [240–243].

The third aspect of node optimization is to design a node as a model itself. Such modification leads to variate of neural network paradigms such as *polynomial neural network* [233,244], where the nodes are designed to as a polynomial function based on inputs to the nodes. Similarly, the nodes of a *GMDH neural network* is designed as an Ivakhnenko polynomial [245]; the nodes of a *complex value neural network* or *multivalued neural network* is designed with a complex value activation functions [234]; the node of *spiking neural networks* has specific behavior, in which a node signal is propagated to another node only if the intrinsic quality of neural activation value is above a defined threshold [246]; the nodes of *fuzzy neural network* paradigm is designed using the concepts of fuzzy theory [247]; the node and the architecture of the *Quantum neural network* are inspired by the quantum computing [248–251]. In all such methods, metaheuristics have a significant role in the optimization.

3.2.5 Learning algorithm optimization

The initial thought of learning algorithm optimization is the optimization of its parameters. For example, the optimization the learning rate and the mutation factor parameters of BP by applying some metaheuristics [146]. To optimize the parameters of an FNN learning algorithm, its parameters (e.g., BP parameters) and learning rules are encoded onto a genotype [201, 252]. However, formulating BP parameter such as learning rule, which is a dynamic concept, into a static chromosome is disadvantageous [253]. Hence, a genetic coding for four components (current weight, activation function of the incoming node and outgoing nodes, input) local to weight in an FNN can encode [253]. Moreover, assuming that each node in a network uses same learning rule, an evolution in learning was proposed in [254], where weights optimization related to a particular node depended only on the input/output at that node. Evolution in learning rule can be described as [255]:

$$\Delta \mathbf{w}^t = \sum_{k=1}^{n} \sum_{i_1,i_2,\ldots,i_k=1}^{n} \left(\theta_{i_1,i_2,\ldots,i_k} \prod_{j=1}^{k} w_{ij}^{t-1} \right) \tag{10}$$

where t is the iteration, $\Delta \mathbf{w}^t$ is weight change, $w_1, w_2, \ldots, w_n$ are weights associated with a node, θ_i is real-valued coefficient that is determined by using EAs. However, learning rule (10) is impractical because of it required huge computation time. Hence, bio-inspired algorithms may be employed for determining the coefficient in (10).

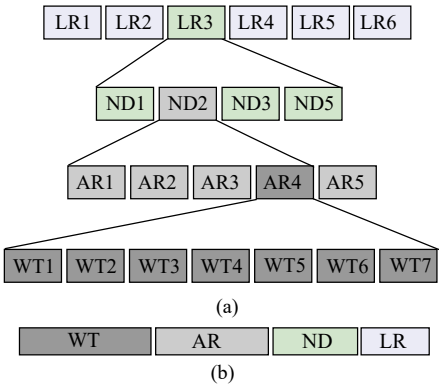


Figure 6: Meta-learning scheme (a), where LR is learning parameter, ND is activation function, AR is architecture, and WT is weight [256]. Combined chromosome structure (b)

3.2.6 Combination of FNN components optimization

Fig. 3 is an impressive representation of the most of FNN optimization combinations, where the genetic representation of the combination of FNN components can be represented in Fig. 6, which refers to a hierarchy of combination, called meta-learning scheme. In the meta-learning scheme, top down or bottom up optimization approach, which means, weights to learning rule and learning rules to weight optimization can be adopted [256]. However, it resembles one-by-one learning scheme. Hence, the advantageous approach is to represent each component of FNN side-by-side onto a genetic vector for optimization, which indicates the confluence of all components of FNN as indicated by area "a8" in Fig. 3.

Yao [132, 255] summarized all such form of adaptation in **evolutionary artificial neural network** (EANN), which is a special class of artificial neural network, where in addition to learning; evolution is another fundamental form of adaptation. Infact a paradigm, called *Neuroevolution* that accommodates adaptive learning all or some components of FNN in some intuitive ways by applying EAs. For examples, generalized acquisition of recurrent links (GNARL) [257], evolutionary programming net (EPNet) [258], neuroevolution of augmenting topologies (NEAT) [259], hypercube-based neuroevolution of augmenting topologies (HyperNEAT) [260], evolutionary acquisition of neural topologies (EANT2) [261], and heterogeneous flexible neural tree (HFNT) [262] optimizes both FNN structure and parameters (weights) using some direct or indirect encoding methods. Moreover, several other paradigms and methods proposed in the past for the simultaneous optimization of FNN components are described as follows.

A **structured genetic algorithm** was proposed in [263], which simultaneously optimized both architectures and weights. It was found that the simultaneous optimization of both weight and architecture lead to a better generalization [56, 197, 264, 265]. Considering permutation[2]

[2]Permutation problem occurs when using traditional crossover operator, where a population has traditional

problem in a GA, EP-based mutation mechanism for evolving FNN architecture was proposed in [258] is known as EPNet.

A **neuroevolution of augmenting topologies** (NEAT) introduced in [259] was a GA-based evolution of an FNN phenotype as a whole, in which a special mutation and crossover operator were defined for manipulating nodes and connections of FNN. Specifically, the linear network information FNN weights, nodes, and connection information were encoded using genetic encoding. The proposed NEAT was evaluated over several applications, and its performance was found outperforming static FNN topology.

A **virtual subpopulation** approach was proposed in [266] for the optimization of FNN using EAs. Later, while indicating a permutation problem, crossover operator as a combinatorial optimization problem was proposed in which each hidden node was considered as a subnetwork and a complete network was evolved using the evolution of several subnetworks [267]. Additionally, GA-based and SA-based crossover operators were applied to generate an offspring (new individual subnetwork). To maintain diversity in population, two mutation operators such as BP-mutation and random-mutation were proposed. In BP-mutation, few iterations of BP algorithm were applied to update weights of the subnetwork, and in random mutation, weights of subnetwork were randomly replaced with new weights. Hence, a *coevolution* of FNN weights and architecture was proposed that evolved FNN with the *cooperation* of the individuals of a subnetwork population.

A **cooperative coevolution neural network** process—inspired by virtual subpopulation approach [266]— was proposed in [268], which was a symbiotic, adaptive neuroevolution (SANE) algorithm for constructing FNN in a dynamic environment. Unlike conventional evolutionary approach, which uses a population of FNNs, SANE uses a population of nodes, where each node establishes connections with the other nodes to form a complete network.

Two reasons of better performance of SANE over conventional and stand-alone metaheuristics were suggested. First, since SANE consider the nodes as functional components of the FNN, it accurately searches and evaluates nodes as genetic building blocks. Second, since a node alone cannot perform well and evolutionary process evolves different types of nodes, SANE was able maintains diversity in the population. Later, the concept of SANE was extended, in which the selection of several individuals from a population of hidden nodes was combined in a various permutation in order to form several complete networks, i.e., evolution in hidden nodes led to an evolution of the complete network [269].

A concept of **sparse neural trees**, in which GP for evolving network structure and GA for parameter optimization was proposed in [270]. Similarly, a *flexible neural tree* (FNT) concept, where GP was used for the adaptation in network structure and SA for the optimization of the parameters (including parameters of activation function) was proposed in [271, 272]. FNT is a tree-like model where adaptation in all components of is equally important. Moreover,

genetic representation of FNN architecture.

its components adaptation may take many forms (Fig. 3). Hence, a beta basis function—which has several controlling parameters, such as shape, size, and center—was used at non-leaf nodes of an FNT [273]. It was observed that embedding beta-basis function at FNT nodes has advantages over other two parametric activation function. A parallel evolution of FNT using MPI programming and GPU programming respectively were proposed in [274] and in [275].

A slightly different direction of FNN modification and improvement study can be seen as the study of **quantum neural network** (QNN). At the first place, the QNN as a *quantum perceptron* was proposed by Lewestein [276], where instead of classical weights, a unitary operator was used to map inputs to an output. The study in QNN encompasses the development of quantum weights, quantum neurons, quantum network, and quantum learning [249].

The design/algorithm of quantum network was thought as an algorithm that can find the control parameters for a coupled qubit system [277] as it appears in quantum computing. A comprehensive quantum inspired neural network is presented in [278], where two categories of inspiration were drawn: strongly and weakly quantum inspired FNN. In strongly inspired QNN, each pattern in a training set was considered as a particle which is processed by a number of FNNs in different universes. Such process was compared with the electrons or photons passing through many slits simultaneously. Whereas, in weakly inspired QNN, each training pattern (though as a particle) was in its own universe. Moreover, there were various QNN models proposed in the past by 1) Behrman et al. [279], 2) Chrisley [280], 3) Menneer [278], and Ventura [281]. A detailed description of these QNN models can be found in [282].

3.3 Comments on metaheuristics approaches

It is indeed can be concluded that metaheuristic algorithms have provided various dimension to the optimization of the FNNs. It has opened several ways such that a generalized FNN can be obtained. Especially the architecture simplification which is directly related to the generalization of an FNN can easily be achieved through the evolving FNN together with its other components. However, the primary disadvantage of using metaheuristic algorithms is the training time consumption. Since the metaheuristic algorithms use a population (many solution candidates) during optimization, the time consumption becomes directly proportional to the number of candidates in a population.

It can also be argued that both conventional and metaheuristic based FNN training take by far more training time than the *extreme learning machine* (ELM) [283]. ELM is a three-layered FNN architecture whose weights between input and hidden layer are randomly assigned and never updated. Additionally, the weights between hidden and output layer are updated in a single step using some least square estimation. Hence, extremely less time required for the learning of the FNN.

It is stated in NFL theorem [92] that it is difficult to find a metaheuristic algorithm that

solves all class of problems. Hence, a metaheuristic algorithm may find difficulty in optimizing the FNN that has been formulated for solving some specific problem (input patterns). Additionally, it is not theoretically possible to understand or determine that how fast a metaheuristics algorithm will converge or finds a satisfactory solution. The only way to determine a metaheuristic algorithm's convergence is by its empirical evaluations. Moreover, since each metaheuristic applies some specific heuristic, it is difficult to select one metaheuristic as the best metaheuristic at an instance for a problem. It is only possible to select a metaheuristic by empirically comparing the convergence speeds and trained FNN performances.

4 Multiobjective metaheuristic approaches

Multiobjective optimization procedure involves in optimizing two or more objectives functions, simultaneously. Multiobjective algorithms are efficient methods for evaluating Pareto-optimal solutions for multiobjective problems. Since optimizing training error cannot provide generalization alone, FNN optimization is viewed from the multiobjective perspective.

Let us first investigate: *why the multiobjective framework is needed for FNN optimization*, *what are the objective functions required for framing FNN as a multiobjective problem*, and *how the objective functions can be framed into multiobjective optimization*. Answers to these question lie in the following discussion.

First, a cost function (4) or any equivalent function is the foremost necessity for the supervising training of FNN.

Second, the generalization of FNN is an essential aspect of its optimization. One approach is to use validation error on cross-validation data because an FNN with low training error may not perform well on unseen (test) data unless FNN is generalized. Moreover, minimization of generalization error is essential than the minimization of training error.

Another approach is to add a regularization term to the training error to avoid overfitting. Additionally, minimizing network complexity leads to a better generalization [284]. Hence, generalization can be achieved by adding a complexity indicator term to training error, i.e., the generalization by minimizing training error and simplifying network complexity.

Third, reducing input-feature—when a problem is available with a huge input dimension (feature)—can lead to a better generalization. However, input dimension reduction and training error reduction are two contradictory objectives.

Finally, the conclusion is, the training error (4) or equivalent cost function c_f needs to be optimized with one or more additional objectives to achieve generalization, which is why multiobjective framework for optimizing FNN are used.

Let us say that training error (4) is c_{trn}, and an additional objective is c_{add}. So, a generalized error c_{gen} may be computed by adding an objective to training error as:

$$c_{gen} = c_{trn} + \lambda c_{add}, \tag{11}$$

where $\lambda > 0$ is a hyperparameter that controls the strength of additional objective c_{add}. The validation error term c_{cv}, regularization term c_{reg}, or network complexity c_{net} or a combination of all can be considered as an additional objective c_{add} in (11). The regularization term c_{reg} is the weight decay or norm of weight vector $\mathbf{w}$ as:

$$c_{reg} = \frac{1}{2}\sum_{i}^{n} w_i^2 = \frac{1}{2} \parallel \mathbf{w} \parallel^2 . \tag{12}$$

Similarly, a validation error c_{cv} is usually computed using (4) on a cross-validation data. On the other hand, the network complexity c_{net} is computed as:

$$c_{net} = \sum_{i}^{z}\sum_{j}^{z} c_{ij}, \tag{13}$$

where z is the number of nodes in the network c (Fig. 5(a)), or any user-defined function can also be used for evaluating network complexity, e.g., the number of nodes, the number of connections, etc.

However, the generalization objective of the form (11) is a scalarized objective that has two disadvantages [285]. First, determining an appropriate hyperparameter λ that controls the contradicting objectives. Hence, the generalization ability of the produced model by using (11) will be a mystery. Second, the objective (11) leads to a single best model that tells nothing about how contradicting objectives were handled. In other words, no single solution exists that may satisfy both objectives. Therefore, generalization error (11) need to be formulated into a multiobjective form: $\min\{c_{trn}, c_{reg}, c_{cv}, \ldots\}$, i.e., a multiobjective optimization needs to be performed as:

minimize $\{c_1(\mathbf{w}), c_2(\mathbf{w}), \ldots, c_m(\mathbf{w})\}$
subject to $\mathbf{w} \in S$,

where $m \geq 2$ is the number of objective functions $c_i : \mathbb{R}^n \rightarrow \mathbb{R}_{\geq 0}$. The vector of objective functions is denoted by $\mathbf{c} = \langle c_1(\mathbf{w}), c_2(\mathbf{w}), \ldots, c_m(\mathbf{w})\rangle$. The decision (variable) vectors $\mathbf{w} = \langle w_1, w_2, \ldots, w_n\rangle$ belong to the set $S \subset \mathbb{R}^n$, which is a subset of the decision variable space $\mathbb{R}^n$. The word "minimize" indicates the minimization all the objective functions simultaneously.

A nondominated solution is one in which no one objective function can be improved without a simultaneous detriment to at least one of the other objectives of the solution. The nondominated solution is also known as a Pareto-optimal solution.

Definition 3. *Pareto-dominance - A solution* $\mathbf{w}_1$ *is said to dominate a solution* $\mathbf{w}_2$ *if* $\forall i = 1, 2, \ldots, m$, $c_i(\mathbf{w}_1) \leq c_i(\mathbf{w}_2)$, *and there exists* $j \in \{1, 2, \ldots, m\}$ *such that* $c_j(\mathbf{w}_1) < c_j(\mathbf{w}_2)$.

Definition 4. *Pareto-optimal - A solution* $\mathbf{w}_1$ *is called Pareto-optimal if there does not exist any other solution that dominates it. A set Pareto-optimal solution is referred to as a Pareto-front.*

Now, a multiobjective algorithm must provide a homogeneous distribution of a population along Pareto front and improve solutions along successive generations [286]. Hence, three basic operators can be used [286]. 1) *Fitness assignment* to guide a population in the direction of Pareto-front using robust and efficient multiobjective selection method. 2) *Density estimation* to maintain solutions distributed over entire Pareto-front using operators that take account of the solution's proximity. 3) *Archiving* to prevent degradation in fitness during successive generations by maintaining an external population to preserve the best solutions and for periodic input to the main population. A detailed survey of multiobjective algorithms is available in [287, 288]. Now, based on the above discussion on the cost functions and generalization conditions, the multiobjective for FNN optimization can be categorized as *non-Pareto based multiobjective* approach and *Pareto-based multiobjective* approach.

4.1 Non-Pareto based multiobjective approaches

In non-Pareto based multiobjective approach, the objective functions are aggregated as mentioned in (11) or by some other means. For example, in [289], authors proposed to add a regularization term $c_0 - c_{reg}$ to training error $c_0 - c_{trn}$, where c_0 is the origin of the two objectives. To obtain an efficient solution, they designed a vector **vc** of **scalar objectives** by varying the hyperparameter λ from 0 to 1. Hence, training FNN for each scalar objective in vector **vc**, a Pareto set was obtained, and then, it was possible to select the best solution from the Pareto front. However, this was an expensive approach, which does not use any Pareto-based multiobjective algorithm to compute Pareto set; rather, an ellipsoid method [290] was applied to train FNN for each scalar objective $vc_i \in \mathbf{vc}$ sequentially.

Similarly, in [291], to achieve generalization, authors proposed a **sliding mode control BP** algorithm for the multiobjective treatment to FNN objectives c_{trn} and c_{reg}. The optimization trajectory of the 2D space of the objectives c_{trn} and c_{reg} was controlled by modifying BP weight update rules using two sliding surface control indicators each belongs to the mentioned objectives, respectively

Multiobjective treatment to FNN was also offered by using improvising metaheuristics itself such as a *predator-prey* algorithm was proposed in [292]. To get a generalized network, the predator-prey algorithm used a family of the randomly generated population of sparse neural networks, called *pray population* and an externally induced family of *predators population* whose job was to prune preys populations based on the objectives c_{trn} and c_{net} was also generated. Similarly, a hybrid multiobjective approach, where a geometrical measure based on singular-value-decomposition for estimating a necessary number of nodes in a network was proposed in [293].

Additionally, a micro-hybrid genetic algorithm was introduced to fine-tuning the network performance. A hybrid algorithm, which uses GA for evolving FNN and uses PSO, BP, and

LM for fine-tuning the evolved FNN was proposed in [294]. In the proposed hybrid algorithm, several objectives function such as training error c_{trn}, validation error c_{cv}, number of hidden layers c_{hid}, number of nodes c_{node}, and activation function c_{fun} were aggregated as:

$$c_{net} = \alpha c_{trn} + \beta c_{cv} + \gamma c_{hid} + \delta c_{node} + \theta c_{fun}, \tag{14}$$

where, α, β, γ, δ, and θ were controlling parameters. Hence, multiple objectives were optimized simultaneously.

As mentioned above in Section 4, the **aggregating objective** function has disadvantages in obtaining the best generalized solution. It is evident from (14) that determining hyper-parameters for controlling objective function is a challenging task. Therefore, Pareto-based multiobjective is an efficient choice for the multiobjective treatment of FNNs.

4.2 Pareto based multiobjective approaches

The advantages of applying Pareto-based learning is thoroughly explained and compared with a single and scalerized objective in [295]. For example, a **nondominated sorting genetic algorithm version II** (NSGA-II) [296] when used for optimizing objectives c_{trn} and c_{net} offers a Pareto set by optimizing both objectives simultaneously using a nondominated sorting method as defined in Definition 3. Hence, NSGA-II can be applied to obtained a regularized network by optimizing the objectives c_{trn} and c_{reg} [297].

Similarly, **Pareto differential evolution** (PDE) algorithm and its variant self-adaptive PDE algorithm was applied to optimize objectives c_{trn} and c_{net} simultaneously that offered a Pareto-set, from which the best solution was picked-up according to network complexity and approximation error examination [298, 299]. Simultaneous optimization of the objectives c_{trn} and c_{net} were also addressed using **multiobjective PSO** to generalize FNN performance [300].

For an image classification problem, authors in [301], pointed out two crucial points: the classification speed and the classification accuracy c_{acc}. The classification speed was then related to the network complexity (number of hidden neurons) c_{net}. The proposed trade-offs between classification speed and classification accuracy were addressed using NSGA-II.

Similarly, in [302], authors studied three methods for image classification problem: linear aggregating (LA), NSGA-II with deterministic selection (DM), and NSGA-II with tournament selection (LM). They proposed to optimize network complexity c_{net} and accuracy c_{acc}. Moreover, they combined regularization term c_{reg} with accuracy c_{acc} and proposed an adaptive strategic for designing network topology using reproduction operators for both hidden layer and input layer. The hidden layer operators were add-connection, delete-connection, add-node, and delete-node. The receptive (input) layer had the following operators: add-connection, delete-connection, add-node, and delete-node. Interestingly, they observed that DM and LM performed better than LA, i.e., Pareto-based multiobjective algorithms performed

better than that of the scalerized objectives. Such ability of the Pareto-based treatment to FNN to obtain general FNN was exploited by several researchers for solving many real-life applications [303–306].

Further, the **coevolution FNN** concept [269,307] was extended in [308] under the multiobjective framework, by using *subnetwork* and *network* concepts. A subnetwork was a collection of nodes, i.e., a subnetwork was considered as a hidden node for a network. Therefore, a network was a collection of subnetworks. So, a population P_1 of subnetwork, which was evolved separately using NSGA-II was used to construct a population P_2 of networks. Then, NSGA-II was again applied to evolve population P_2. Interestingly, authors defined separate objectives for population P_1 (subnetworks objectives) and P_2 (networks objectives) so that the functional diversity in both network and subnetwork can be maintained. Additionally, some metrics (objectives) for measuring network and subnetwork functional diversities were defined. The objective of subnetworks were *differences* (for maintaining functional diversity of subnetwork), *substitution* (to replace poor candidates by better candidates), and *complexity* (for counting the number of connection, nodes, and sum of all weights). Therefore, they coevolved overall network with the cooperation of subnetwork that evolves together with the whole network to get a general solution to a problem.

Apart from the discussed objective in this section, some interesting dimensions in multiobjective treatment to FNN can be noted in [309], in which authors proposed to apply NSGA-II for the simultaneous optimization of three objectives: *input-dimension*, training error, and network complexity. Hence, an optimized a network that performs well on the minimal set of input dimension was obtained. Similarly, in [41, 310], authors used a Pareto-based memetic algorithm approach for combining PDE and Rprop algorithms to minimize objective pairs *true classification rate* and *minimum sensitivity* (miss-classification rate), simultaneously.

As a result of metaheuristic or multiobjective metaheuristic treatment, a set of FNN network is obtained and selecting the best FNN from that set is a difficult task. Since selecting a single best FNN from may not offer a generalized solution and the residual error can still be remaining many problems when selection single best FNN [311], then an ensemble of a set of FNNs is recommended.

5 Ensemble of feedforward neural networks

Metaheuristics optimization of FNN leads to a final population that contains many solutions close to the best solution. Moreover, the solution in the final population are divers in the following sense: 1) parametric (each FNNs have different sets of weights); structural (each FNNs have different network configurations); and 3) training set (each FNNs are trained on different parts of a training set). Hence, a collective decision (ensemble) of l many diverse candidates selected from a final population may offer desired generalization [312]. The literature that

explains *how to construct diverse FNNs* and *how to combine decisions of diverse FNNs* are summarized as follows.

The very basic idea is to apply single-solution based algorithms on l many FNNs to get l many diverse solutions [46]. The decision of l many candidates which were created either by single solution-based metaheuristics, or by population-based metaheuristics, or by any other means are combined using the following methods [313, 314]: 1) majority voting method (for classification problems); 2) arithmetic mean (for regression problem); 3) rank-based linear combination; 4) linear combination by using *recursive least square* [315] (to minimize weighted least squares error so that redundant individuals are eliminated); 5) evolutionary weighted mean or majority voting (metaheuristic to determine impact of an FNN in ensemble); and 6) entropy-based method for combining FNNs in ensemble (assigning entropy to FNNs during the learning process) [316].

Since population-based metaheuristics lead to an optimized final population, it is advantageous to use the final population for making ensemble [312]. However, there are two fundamental problems with it [317]: 1) determining ensemble size, 2) how to maintain diversity in the population. Hence, a **negative correlation learning** (NCL) algorithm that optimized and combined individual FNNs in an ensemble during learning process was proposed in [317]. NCL optimized all individual FNNs simultaneously and interactively by adding a correlation penalty terms to the cost functions. Moreover, NCL produced negatively correlated and specialized FNNs by using co-operation among each FNNs of a population [318].

To determine the size of ensemble automatically, EA-based ensemble procedure was laid down in which NCL was applied during networks training. Moreover, different FNNs were allowed learn different parts of training data and the best (according to fitness) were selected for ensemble [319]. Additionally, a **constructive-cooperative-neural-network-ensemble** was proposed in [320] that determined ensemble size by focusing on accuracy and diversity during a constructive, cooperative procedure [321].

However, mere training fitness based selection of candidates for the ensemble is insufficient because it does not tell much about candidates role/influence in the ensemble. This problem was addressed in a **GA-based selective ensemble** method [322], which selects a subset of the population and determine the strength of selected candidates using GA. It was also shown that the ensemble of a subset of the population was found performing better than that of the whole population [322]. The effectiveness such GA-based selection was found efficient than the traditional ensemble methods: *bagging* [323] and *boosting* [324].

It is beneficial to partition/fracture training data and allows different FNN in the population to learn various parts of training data [323,324]. An evidence of such was examined in [325,326], where it was found that the ensemble of a few FNNs that was trained using *bootstrapping* performs better than that of an ensemble of a larger number of FNNs. Similarly, the efficiency of using distinct training sets for optimizing different FNNs was proved when a **class-switching**

ensembles approach proposed in [327] and were compared with bagging and boosting methods.

At one hand bootstrapping method allows FNN to learn different training samples. On the contrary, a **clustering-and-coevolution** approach for constructing neural network ensembles proposed in [328] partition the input space using a clustering method to reduced number of input nodes of FNNs. Hence, in the ensemble, diverse FNNs (different FNNs were specialized in various regions of input space) were created. Moreover, it reduced run time of learning the process by coevolving (divide-and-conquer method) different FNNs using cooperation between FNNs. Such method improves diversity and accuracy of an ensemble system [328].

Similarly, a method was suggested in [329] for generating diverse evolutionary FNNs using a fitness-sharing method—a fitness sharing method shares resources if the distance between the individuals is smaller than the predefined sharing radius. Specifically, authors proposed a speciation based evolutionary neural ensemble method for constructing ensemble by combining FNNs using a knowledge space method. On the other hand, a progressive interactive training scheme called a **sequential-neural-network-ensemble-learning** method, which trained FNNs one-by-one by interaction from a central buffer of FNNs was proposed in [330].

Both diversity and accuracy is a crucial aspect in construing ensemble of FNNs [313]. However, accuracy and diversity are contradictory to each other, so, a multiobjective approach may be applied to evolve FNN population by maintaining accuracy and diversity simultaneously [331]. For this purpose, **multiobjective regularized negative correlation learning** that maximized performance and maximized the negative correlation between individuals in population was found efficient [332].

6 Challenges and future scopes

The effectiveness of FNN training primarily depends on *data quality*, which is governed by the following *data quality assurance* parameters: accuracy, reliability, timeliness, relevance, completeness, currency, consistency, flexibility, and precision [333, 334]. Usually, *data cleaning* is a major step before modeling [335]. Therefore, training of the FNN remains always sensitive to the data cleaning process and it poses a significant challenge to adapt some mechanism in training process such the sensitivity towards data-clean may be reduced. Additionally, one problem related to data-driven modeling (FNN learning) is the data itself which can be insufficient, imbalanced, incomplete, high-dimensional, or abundant.

For the case **insufficient data**, usually the input hyperspace is exploited to generate virtual samples to fill the sparse area of the hyperspace, and by monitoring FNN performance on the virtually generated samples [336]. The second approach exploits the dynamics of EAs in conjunction with FNNs to obtain new samples [229]. However, this area is still much to explore, where some open questions such as how efficiently FNNs can be trained with virtually generate data to mitigate the insufficiency. On the other hand, research in the area of imbalance dataset

is continued to interest researcher [337].

The present era of data analysis is what we call *big data*, i.e., we need to deal not only with *high-dimensional data* but also with the *variety data* and **stream data** [338]. High-dimensional data, such as gene expression data, speech processing, natural language processing, social-network-data, etc., poses significant challenges. Such challenge is to some extent addressed by *deep learning* paradigms that allow the arrangement several units/layers of FNNs (or any other model) in a hierarchical manner to process and understand insights of such high-dimensional data [339, 340]. High-dimensional data can also be managed/reduced by encoding or decoding methods and using FNNs training [341]. Therefore, FNNs has a greater role in feature reduction.

In a **non-stationary environment**, such as stock-price market, weather forecasting, etc., data comes in the stream, i.e., data comes in sequential order, and traditionally, re-training based mechanics for dynamic learning (online learning) of FNN is the basic option [342]. However, it is still an open problem to design strategies for the dynamic training of FNN.

Apart from the crucial aspects that *how to manage non-stationary data*, the aspect that how to handle **multi-view (heterogeneity)** of data is an additional challenge. The quest of developing a model that can stand robust and efficient for the non-stationary data caused by the time-dependent process of data generation, and can accommodate new knowledge (newly generated data sample) is a significant topic in machine learning research [343]. On the other hand, integration of data or of the models for that matter for the heterogeneous data generated or gathered from different instruments and data-generation processes is a significant research problem [344, 345].

Moreover, present era, the **fourth industrial revolution**, is of *Internet of Things* (IoT) [346]. In IoT, sophisticated technologies such as *smartphone* and *smartwear* provide several forms of data, e.g., *human activity recognition* [347]. Additionally, it demands application to be simple. Hence, FNN models which when aims to such technologies needs to be less complex. Therefore, FNN architecture simplification or model's complexity reduction is a challenging task. Such problem can be addressed through the integration of FNN with statistical methods like the one usually done with *hidden Markov model* [348]. Therefore, such kind of modification to network architecture and specialized node design may lead to different paradigms of FNN that may solve various real-world complex problems.

7 Conclusions

Feedforward neural network (FNN) is used for solving a wide range of real-world problems, which is why researcher investigated many techniques/methods for optimizing and generalizing FNN. Specifically, metaheuristics allow us to innovate and improvise methods for optimizing FNN that in turn address its local minima and generalization problems.

Initially, only gradient based linear approximation and quadratic approximation methods

for optimizing FNNs were employed to train FNN. These conventional algorithms (backpropagation, Quickpro, Rprop, conjugate gradient, etc.) are local search algorithms that exploit current solution to generate a new solution; however, they lack in exploration ability, therefore, usually, finds local minima of an optimization problem.

Unlike conventional approaches, metaheuristics (e.g., genetic algorithm, particle swarm optimization, ant colony optimization, etc.) are good at both exploitation and exploration and can address simultaneous adaptation in each component of FNN. However, no single method can solve all kinds of problem. So, we need to improvise, adapt, and construct hybrid methods for optimizing FNN. Therefore, several dynamic designs of FNN are reported in the literature: EPNet (an adaptive method of FNN architecture optimization), neuro-evolution of augmenting topologies, flexible neural tree, cooperative coevolution neural network, etc., are among them. Hence, there is a wide spectrum of FNN optimization/adaptation is possible with metaheuristic treatment to FNNs (Fig. 3) in which the fundamental aspect is the formulation of FNN (phenotype) to vectored form (genotype) or any other form of mechanism for manipulation of FNN components.

Since there are many components to be manipulated by means of metaheuristic strategies and the availability of the fact that FNN generalization ability depends on the optimization its all the components, multiobjective treatment to FNN were used. The multiobjective-based training allows an FNN to evolve with handling two or more FNN-related objectives, such as approximation error, network complexity, input dimension, etc. Moreover, the generalization ability of system can be easily improved by combining decision of many candidates of the system. Hence, an ensemble of FNNs by making use of the metaheuristic final population was proposed and the two crucial aspect accuracy and diversity of an ensemble were taken care during propose of evolving FNNs.

It is evident from such aspects of FNN optimization that the future research will be able to bring the new paradigms of FNNs by applying or by the inspiration from the discussed methods in this article. Hence, that will overcome the data quality problem and will be handling new challenges of big data to cope-up with the new era information processing.

Acknowledgment

Authors would like to thank the all the anonymous reviewers for the technical comments, which enhanced the contents of the preliminary version of this paper. This work was supported by the IPROCOM Marie Curie Initial Training Network, funded through the People Programme (Marie Curie Actions) of the European Unions Seventh Framework Programme FP7/20072013/, under REA grant agreement number 316555.

References

[1] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biol.*, vol. 5, no. 4, pp. 115–133, 1943.

[2] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, Oct 1991.

[3] A. Jain, R. Duin, and J. Mao, "Statistical pattern recognition: a review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 4–37, Jan 2000.

[4] G. Zhang, "Neural networks for classification: a survey," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 30, no. 4, pp. 451–462, Nov 2000.

[5] R. Selmic and F. Lewis, "Neural-network approximation of piecewise continuous functions: application to friction compensation," *IEEE Trans. Neural Netw.*, vol. 13, no. 3, pp. 745–751, May 2002.

[6] H. Lam and F. Leung, "Design and stabilization of sampled-data neural-network-based control systems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 5, pp. 995–1005, Oct 2006.

[7] S. Mitra and Y. Hayashi, "Bioinformatics with soft computing," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 36, no. 5, pp. 616–635, Sep 2006.

[8] M. Niranjan and J. Principe, "The past, present, and future of neural networks for signal processing," *IEEE Signal Process. Mag.*, vol. 14, no. 6, pp. 28–48, Nov 1997.

[9] A. Gorin and R. Mammone, "Introduction to the special issue on neural networks for speech processing," *IEEE Trans. Speech Audio Process.*, vol. 2, no. 1, pp. 113–114, Jan 1994.

[10] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines.* Pearson Education Upper Saddle River, 2009, vol. 3.

[11] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386 – 408, Nov 1958.

[12] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial," *Comput.*, vol. 29, no. 3, pp. 31–44, Mar 1996.

[13] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry.* Cambridge, Mass.: MIT Press, 1988.

[14] P. J. Werbos, "Beyond regression: new tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, 1974.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, Oct 1986.

[16] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control, Signals, Syst.*, vol. 2, no. 4, pp. 303–314, Dec 1989.

[17] A. K. Kolmogorov, "On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition (in russian)," *Dokl. Akad. Nauk SSSR*, vol. 114, pp. 369–373, 1957.

[18] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Mar 1989.

[19] V. Kůrková, "Kolmogorov's theorem and multilayer neural networks," *Neural Netw.*, vol. 5, no. 3, pp. 501–506, 1992.

[20] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Netw.*, vol. 6, no. 6, pp. 861–867, Mar 1993.

[21] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Trans. Neural Netw.*, vol. 9, no. 1, pp. 224–229, Jan 1998.

[22] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, Jul 2006.

[23] D. Lowe and D. Broomhead, "Multivariable functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, pp. 321–355, 1988.

[24] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep 1995.

[25] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cogn. Sci.*, vol. 11, no. 1, pp. 23–63, Jan 1987.

[26] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, Jan 1982.

[27] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 79, no. 8, pp. 2554–2558, Apr 1982.

[28] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cogn. Sci.*, vol. 9, no. 1, pp. 147–169, 1985.

[29] P. F. Dominey, "Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning," *Biol. Cybern.*, vol. 73, no. 3, pp. 265–74, Aug 1995.

[30] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks-with an erratum note," German National Research Center for Information Technology, Bonn, Germany, Tech. Rep., 2001.

[31] T. Natschläger, W. Maass, and H. Markram, "The "liquid computer": A novel strategy for real-time computing on time series," *Special Issue on Foundations of Information Processing of TELEMATIK*, vol. 8, no. 1, p. 39–43, 2002.

[32] J. J. Steil, "Backpropagation-decorrelation: online recurrent learning with O(N) complexity," in *Proc. 2004 IEEE Int. Jt. Conf. Neural Netw.*, vol. 2, 2004, pp. 843–848.

[33] B. Wisrow, M. E. Hoff *et al.*, "Adaptive switching circuits," in *IRE WESCON Conv. Rec.*, vol. 4, Aug 1960, pp. 96–104.

[34] B. Widrow, "Adaptive sampled-data systems– a statistical theory of adaptation," in *IRE WESCON Conv. Rec.*, vol. 4, 1959, pp. 74–85.

[35] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cogn. Sci.*, vol. 9, no. 1, pp. 75–112, Jan 1985.

[36] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996.

[37] J. Pearce and S. Ferrier, "Evaluating the predictive performance of habitat models developed using logistic regression," *Ecol. Modell.*, vol. 133, no. 3, pp. 225–245, 2000.

[38] J. Twomey and A. Smith, "Performance measures, consistency, and power for artificial neural network models," *Math. Comput. Model.*, vol. 21, no. 1, pp. 243–258, 1995.

[39] M. J. Pencina, R. B. D Agostino, and R. S. Vasan, "Evaluating the added predictive ability of a new marker: from area under the roc curve to reclassification and beyond," *Stat. Med.*, vol. 27, no. 2, pp. 157–172, 2008.

[40] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inform. Process. Manage.*, vol. 45, no. 4, pp. 427–437, Jul 2009.

[41] J. Fernandez Caballero, F. Martinez, C. Hervas, and P. Gutierrez, "Sensitivity versus accuracy in multiclass problems using memetic pareto evolutionary neural networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 5, pp. 750–770, May 2010.

[42] J. Baranyi, C. Pin, and T. Ross, "Validating and comparing predictive models," *Int. J. Food Microbiol.*, vol. 48, no. 3, pp. 159–166, 1999.

[43] H. van der Voet, "Comparing the predictive accuracy of models using a simple randomization test," *Chemometr. Intell. Lab. Syst.*, vol. 25, no. 2, pp. 313–323, 1994.

[44] F. X. Diebold and R. S. Mariano, "Comparing predictive accuracy," *J. Bus. Econ. Stat.*, vol. 13, no. 3, pp. 253–263, 1995.

[45] D. A. Simovici and C. Djeraba, *Mathematical Tools for Data Mining.* Springer, 2008.

[46] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 10, pp. 993–1001, Oct 1990.

[47] K. Fukumizu and S.-I. Amari, "Local minima and plateaus in hierarchical structures of multilayer perceptrons," *Neural Netw.*, vol. 13, no. 3, pp. 317 – 327, May 2000.

[48] L. F. Wessels and E. Barnard, "Avoiding false local minima by proper initialization of connections," *IEEE Trans. Neural Netw.*, vol. 3, no. 6, pp. 899–905, Nov 1992.

[49] K.-A. Toh, "Deterministic global optimization for fnn training," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 33, no. 6, pp. 977–983, Dec 2003.

[50] T. Poston, C.-N. Lee, Y. Choie, and Y. Kwon, "Local minima and back propagation," in *Int. Jt. Conf. Neural Netw., 1991., IJCNN*, vol. 2, 1991, pp. 173–176.

[51] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 1, pp. 76–86, Jan 1992.

[52] B. C. Cetin, J. W. Burdick, and J. Barhen, "Global descent replaces gradient descent to avoid local minima problem in learning with artificial neural networks," in *IEEE Int. Conf. Neural Netw.*, 1993, pp. 836–842.

[53] D.-S. Huang, "The local minima-free condition of feedforward neural networks for outer-supervised learning," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 3, pp. 477–480, Jun 1998.

[54] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Comput.*, vol. 4, no. 1, pp. 1–58, Jan 1992.

[55] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proc. of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[56] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Comput.*, vol. 7, no. 2, pp. 219–269, Mar 1995.

[57] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, 1995.

[58] S.-i. Amari, N. Murata, K.-R. Muller, M. Finke, and H. H. Yang, "Asymptotic statistical theory of overtraining and cross-validation," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 985–996, Sep 1997.

[59] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria," *Neural Netw.*, vol. 11, no. 4, pp. 761–767, 1998.

[60] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approx.*, vol. 26, no. 2, pp. 289–315, Apr 2007.

[61] A. F. Murray and P. J. Edwards, "Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Trans. on Neural Netw.*, vol. 5, no. 5, pp. 792–802, 1994.

[62] O. Bousquet and A. Elisseeff, "Stability and generalization," *J. Mach. Learn. Res.*, vol. 2, pp. 499–526, 2002.

[63] R. Reed, R. J. Marks, and S. Oh, "Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 529–538, 1995.

[64] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, Apr 1987.

[65] O. T.-C. Chen and B. J. Sheu, "Optimization schemes for neural network training," in *1994 IEEE Int. Conf. Neural Netw., 1994. IEEE World Congr. Comput. Intell.*, vol. 2, 1994, pp. 817–822.

[66] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Francisco, CA, USA: Morgan Kaufmann, 1990, pp. 524–532.

[67] S. E. Fahlman, "An empirical study of learning speed in back-propagation networks," Carnegie Mellon University, Tech. Rep., 1988.

[68] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Netw.*, vol. 1, no. 4, pp. 295–307, 1988.

[69] F. M. Silva and L. B. Almeida, "Acceleration techniques for the backpropagation algorithm," in *Neural Networks*, ser. Lecture Notes in Computer Science. Springer, 1990, vol. 412, pp. 110–119.

[70] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The rprop algorithm," in *IEEE Int. Conf. Neural Netw., 1993., IJCNN*, 1993, pp. 586–591.

[71] W. Schiffmann, M. Joost, and R. Werner, "Optimization of the backpropagation algorithm for training multilayer perceptrons," University of Koblenz, Institute of Physics, Rheinau, Koblenz, Tech. Rep., 1994.

[72] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *J. Res. Nat. Bur. Stand.*, vol. 49, no. 6, pp. 409–436, Dec 1952.

[73] E. Barnard and R. A. Cole, "A neural-net training program based on conjugate-gradient optimization," Technical Repport CSE 89-014, Department of Computer Science, Oregon Graduate Institute of Science and Technology, Tech. Rep., 1989.

[74] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," *IEE Proc. G (Circuits, Devices, Syst.)*, vol. 139, no. 3, pp. 301–310, Jun 1992.

[75] Y.-H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," *SIAM J. Optim.*, vol. 10, no. 1, pp. 177–182, 1999.

[76] D. P. Bertsekas, *Nonlinear programming*, 2nd ed. Athena scientific Belmont, 1999.

[77] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *J. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, Jun 1963.

[78] R. Fletcher, *Practical Methods of Optimization*. John Wiley & Sons, 1987.

[79] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov 1994.

[80] G. Lera and M. Pinzolas, "Neighborhood based levenberg-marquardt algorithm for neural network training," *IEEE Trans. Neural Netw.*, vol. 13, no. 5, pp. 1200–1203, Sep 2002.

[81] S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Kalman filtering and neural networks*. Wiley Online Library, 2001.

[82] J. Sum, C.-S. Leung, G. H. Young, and W.-K. Kan, "On the kalman filtering method in neural network training and pruning," *IEEE Trans. Neural Netw.*, vol. 10, no. 1, pp. 161–166, 1999.

[83] M. R. Azimi-Sadjadi and R.-J. Liou, "Fast learning process of multilayer neural networks using recursive least squares method," *IEEE Trans. Signal Process.*, vol. 40, no. 2, pp. 446–450, 1992.

[84] N. N. Schraudolph, J. Yu, S. Günter *et al.*, "A stochastic quasi-newton method for online convex optimization," in *Proc. 11th Int. Conf. Artif. Intell. Stat.*, vol. 7, 2007, pp. 436–443.

[85] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Netw.*, vol. 16, no. 10, pp. 1429–1451, 2003.

[86] T. Nakama, "Theoretical analysis of batch and on-line training for gradient descent learning in neural networks," *Neurocomputing*, vol. 73, no. 1, pp. 151–159, 2009.

[87] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[88] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, no. 2, pp. 95–99, Oct 1988.

[89] J. G. March, "Exploration and exploitation in organizational learning," *Org. Sci.*, vol. 2, no. 1, pp. 71–87, Feb 1991.

[90] I. H. Osman and G. Laporte, "Metaheuristics: a bibliography," *Ann. Oper. Res.*, vol. 63, no. 5, pp. 511–623, Oct 1996.

[91] 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.

[92] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Comput.*, vol. 8, no. 7, pp. 1341–1390, Oct 1996.

[93] Y.-C. Ho and D. L. Pepyne, "Simple explanation of the no free lunch theorem of optimization," *Cybern.Syst. Anal.*, vol. 38, no. 2, pp. 292–298, Mar 2002.

[94] C. Schumacher, M. D. Vose, and L. D. Whitley, "The no free lunch and problem description length," in *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO-2001)*, 2001, pp. 565–570.

[95] C. Igel and M. Toussaint, "On classes of functions for which no free lunch results hold," *Inf. Process. Lett.*, vol. 86, no. 6, pp. 317 – 321, 2003.

[96] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Sci.*, vol. 220, no. 4598, pp. 671–680, May 1983.

[97] F. Glover, "Tabu search-part I," *INFORMS J. Comput.*, vol. 1, no. 3, 1989.

[98] N. Mladenović and P. Hansen, "Variable neighborhood search," *Comput. Oper. Res.*, vol. 24, no. 11, pp. 1097–1100, Nov 1997.

[99] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *J. Glob. Optim.*, vol. 6, no. 2, pp. 109–133, Jun 1995.

[100] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[101] V. Černỳ, "Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm," *J. Optim. Theory Appl.*, vol. 45, no. 1, pp. 41–51, Jan 1985.

[102] D. B. Fogel, *Evolutionary Computation: The Fossil Record.* Wiley-IEEE Press, 1998.

[103] H.-P. Schwefel, *Collective Phenomena in Evolutionary Systems.* Universität Dortmund. Abteilung Informatik, 1987.

[104] J. R. Koza, F. H. Bennett III, and O. Stiffelman, *Genetic Programming as a Darwinian Invention Machine.* Springer, 1999.

[105] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, Dec 1997.

[106] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Int. Symp. Micro Mach. Human Sci., 1995. MHS '95*, 1995, pp. 39–43.

[107] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern.*, vol. 26, no. 1, pp. 29 – 41, Feb 1996.

[108] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Computer Engineering Department, Erciyes University, Tech. Rep. TR06, 2005.

[109] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *IEEE Control Syst.*, vol. 22, no. 3, pp. 52–67, Jan 2002.

[110] J. F. Kennedy, J. Kennedy, and R. C. Eberhart, *Swarm Intelligence.* Morgan Kaufmann, 2001.

[111] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *The 1998 IEEE Int. Conf. Evol. Comput. Proc., IEEE World Congr. Comput. Intell.*, 1998, pp. 69–73.

[112] J.-L. Deneubourg, S. Aron, and S. Goss, "The self-organizing exploratory pattern of the argentine ant," *J. Insect Behav.*, vol. 3, pp. 159–169, Mar 1990.

[113] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, 2014.

[114] X.-S. Yang, "Flower pollination algorithm for global optimization," in *Unconventional computation and natural computation*. Springer, 2012, pp. 240–249.

[115] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *World Congr. on Nature and Biologically Inspired Comput., 2009. NaBIC 2009*, 2009, pp. 210–214.

[116] X.-S. Yang, "Firefly algorithm, stochastic test functions and design optimisation," *Int. J. Bio-Inspired Comput.*, vol. 2, no. 2, pp. 78–84, 2010.

[117] Z. W. Geem, J. H. Kim, and G. Loganathan, "A new heuristic optimization algorithm: harmony search," *SIMULATION*, vol. 76, no. 2, pp. 60–68, Feb 2001.

[118] R. A. Formato, "Central force optimization: a new metaheuristic with applications in applied electromagnetics," *Progress In Electromagnetics Research*, vol. 77, pp. 425–491, Jan 2007.

[119] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "Gsa: a gravitational search algorithm," *Inform. Sci.*, vol. 179, no. 13, pp. 2232 – 2248, Jun 2009.

[120] I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D. Fister, "A brief review of nature-inspired algorithms for optimization," *Elektrotehniški vestnik (English Ed.)*, vol. 80, no. 3, Jul 2013.

[121] D. Weyland, "A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology," *Int. J. Appl. Metaheuristic Comput.*, vol. 1, no. 2, pp. 50–60, 2010.

[122] K. Sörensen, "Metaheuristicsthe metaphor exposed," *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 3–18, 2015.

[123] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Caltech, Tech. Rep. Caltech Concurrent Computation Program, C3P Report, 1989.

[124] Y. Leung, Y. Gao, and Z.-B. Xu, "Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis," *IEEE Trans. on Neural Netw.*, vol. 8, no. 5, pp. 1165–1176, 1997.

[125] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Inf. process. lett.*, vol. 85, no. 6, pp. 317–325, 2003.

[126] J. Brownlee, *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[127] F. Yin, H. Mao, and L. Hua, "A hybrid of back propagation neural network and genetic algorithm for optimization of injection molding process parameters," *Mater. Des.*, vol. 32, no. 6, pp. 3457–3464, Jun 2011.

[128] M. Yaghini, M. M. Khoshraftar, and M. Fallahi, "A hybrid algorithm for artificial neural network training," *Eng. Appl. Artif. Intell.*, vol. 26, no. 1, pp. 293–301, Jan 2013.

[129] J.-R. Zhang, J. Zhang, T.-M. Lok, and M. R. Lyu, "A hybrid particle swarm optimization and back-propagation algorithm for feedforward neural network training," *Appl. Math. Comput.*, vol. 185, no. 2, pp. 1026 – 1037, Feb 2007.

[130] C. Ozturk and D. Karaboga, "Hybrid artificial bee colony algorithm for neural network training," in *IEEE Congr. Evol. Comput. (CEC), 2011*, 2011, pp. 84–88.

[131] C.-F. Juang, "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 2, pp. 997–1006, Apr 2004.

[132] X. Yao, "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539–567, 1993.

[133] I. Boussaid, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inform. Sci.*, vol. 237, pp. 82 – 117, Jul 2013.

[134] X. Yao and Y. Liu, "Towards designing artificial neural networks by evolution," *Appl. Math. Comput.*, vol. 91, no. 1, pp. 83 – 90, 1998.

[135] J. Engel, "Teaching feed-forward neural networks by simulated annealing," *Complex Syst.*, vol. 2, no. 6, pp. 641–648, 1988.

[136] Y. Shang and B. Wah, "Global optimization for neural network training," *Comput.*, vol. 29, no. 3, pp. 45–54, Mar 1996.

[137] R. S. Sexton, R. E. Dorsey, and J. D. Johnson, "Beyond back propagation: using simulated annealing for training neural networks," *J. Organ. End User Comput.*, vol. 11, no. 3, pp. 3–10, Jul 1999.

[138] D. Sarkar and J. M. Modak, "ANNSA: a hybrid artificial neural network/simulated annealing algorithm for optimal control problems," *Chem. Eng. Sci.*, vol. 58, no. 14, pp. 3131 – 3142, Jul 2003.

[139] D. Beyer and R. Ogier, "Tabu learning: a neural network search method for solving nonconvex optimization problems," in *Int. Jt. Conf. Neural Netw., 1991., IJCNN*, vol. 2, 1991, pp. 953–961.

[140] R. Battiti and G. Tecchiolli, "Training neural nets with the reactive tabu search," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1185–1200, 1995.

[141] 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, Apr 1998.

[142] J. Ye, J. Qiao, M. ai Li, and X. Ruan, "A tabu based neural network learning algorithm," *Neurocomputing*, vol. 70, no. 4, pp. 875 – 882, Jan 2007.

[143] D. Whitley, "The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 116–121.

[144] 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, Aug 1990.

[145] M. Srinivas and L. Patnaik, "Learning neural network weights using genetic algorithms-improving performance by search-space reduction," in *Int. Jt. Conf. Neural Netw., 1991. IJCNN*, 1991, pp. 2331–2336.

[146] R. K. Belew, J. Mcinerney, and N. N. Schraudolph, "Evolving networks: using the genetic algorithm with connectionist learning," University of California, San Diego, Tech. Rep. CS90–174, 1990.

[147] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Int. Jt. Conf. Artif. Intell.*, vol. 1, 1989, pp. 762–767.

[148] D. B. Fogel, L. J. Fogel, and V. Porto, "Evolving neural networks," *Biol. Cybern.*, vol. 63, no. 6, pp. 487–493, Oct 1990.

[149] J. Sietsma and R. J. Dow, "Creating artificial neural networks that generalize," *Neural Netw.*, vol. 4, no. 1, pp. 67–79, 1991.

[150] F. Menczer and D. Parisi, "Evidence of hyperplanes in the genetic learning of neural networks," *Biol. Cybern.*, vol. 66, no. 3, pp. 283–289, Jan 1992.

[151] R. Irani and R. Nasimi, "Evolving neural network using real coded genetic algorithm for permeability estimation of the reservoir," *Expert Syst. Appl.*, vol. 38, no. 8, pp. 9862–9866, Aug 2011.

[152] D. Kim, H. Kim, and D. Chung, "A modified genetic algorithm for fast training neural networks," in *Advances in Neural Networks ISNN 2005*, ser. Lecture Notes in Computer Science, J. Wang, X. Liao, and Z. Yi, Eds. Springer, 2005, vol. 3496, pp. 660–665.

[153] H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," in *Proc. 8th Nat. Conf. Artif. Intell. - Vol. 2*, vol. 2, 1990, pp. 789–795.

[154] R. S. Sexton, R. E. Dorsey, and J. D. Johnson, "Toward global optimization of neural networks: a comparison of the genetic algorithm and backpropagation," *Decision Support Syst.*, vol. 22, no. 2, pp. 171 – 185, Feb 1998.

[155] S. Ding, C. Su, and J. Yu, "An optimizing bp neural network algorithm based on genetic algorithm," *Artif. Intell. Rev.*, vol. 36, no. 2, pp. 153–162, Aug 2011.

[156] D. L. Tong and R. Mintram, "Genetic algorithm-neural network (GANN): a study of neural network activation functions and depth of genetic algorithm search applied to feature selection," *Int. J. Mach. Learn. Cybern.*, vol. 1, no. 1-4, pp. 75–87, Sep 2010.

[157] S. Das and P. N. Suganthan, "Differential evolution: a survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Oct 2011.

[158] S. Nolfi, D. Parisi, and J. L. Elman, "Learning and evolution in neural networks," *Adapt. Behav.*, vol. 3, no. 1, pp. 5–28, Jun 1994.

[159] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Process. Lett.*, vol. 17, no. 1, pp. 93–105, Feb 2003.

[160] A. Slowik, "Application of an adaptive differential evolution algorithm with multiple trial vectors to artificial neural network training," *IEEE Trans. Ind. Electron.*, vol. 58, no. 8, pp. 3160–3167, Aug 2011.

[161] A. Ismail and A. Engelbrecht, "Global optimization algorithms for training product unit neural networks," in *Proc. IEEE-INNS-ENNS Int. Jt. Conf. Neural Netw., 2000. IJCNN 2000,*, vol. 1, 2000, pp. 132–137.

[162] 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.

[163] F. Van den Bergh and A. Engelbrecht, "Training product unit networks using cooperative particle swarm optimisers," in *Proc. 2001. Int. Jt. Conf. Neural Netw.*, vol. 1, 2001, pp. 126–131.

[164] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 225–239, 2004.

[165] B. Al-kazemi and C. Mohan, "Training feedforward neural networks using multi-phase particle swarm optimization," in *Proc. 9th Int. Conf. Neural Inform. Process., 2002. ICONIP '02*, vol. 5, 2002, pp. 2615–2619.

[166] C.-J. Lin, C.-H. Chen, and C.-T. Lin, "A hybrid of cooperative particle swarm optimization and cultural algorithm for neural fuzzy networks and its prediction applications," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 39, no. 1, pp. 55–68, Dec 2009.

[167] H. Dhahri, A. Alimi, and A. Abraham, "Hierarchical particle swarm optimization for the design of beta basis function neural network," in *Intelligent Informatics*, ser. Advances in Intelligent Systems and Computing, A. Abraham and S. M. Thampi, Eds. Springer, 2013, vol. 182, pp. 193–205.

[168] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1155 – 1173, Mar 2008.

[169] K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training," *Neural Comput. Appl.*, vol. 16, no. 3, pp. 235–247, May 2007.

[170] R. Irani and R. Nasimi, "An evolving neural network using an ant colony algorithm for a permeability estimation of the reservoir," *Pet. Sci. Technol.*, vol. 30, no. 4, pp. 375–384, Feb 2012.

[171] N. Sharma, N. Arun, and V. Ravi, "An ant colony optimisation and nelder–mead simplex hybrid algorithm for training neural networks: an application to bankruptcy prediction in banks," *Int. J. Inform. Decision Sci.*, vol. 5, no. 2, pp. 188–203, 2013.

[172] D. Karaboga, B. Akay, and C. Ozturk, "Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks," in *Modeling Decisions for Artificial Intelligence*, ser. Lecture Notes in Computer Science, V. Torra, Y. Narukawa, and Y. Yoshida, Eds. Springer, 2007, vol. 4617, pp. 318–329.

[173] B. A. Garro, H. Sossa, and R. A. Vázquez, "Artificial neural network synthesis by means of artificial bee colony (ABC) algorithm," in *IEEE Congr. Evol. Comput. (CEC), 2011*, 2011, pp. 331–338.

[174] P. P. Sarangi, A. Sahu, and M. Panda, "Training a feed-forward neural network using artificial bee colony with back-propagation algorithm," in *Intelligent Computing, Networking, and Informatics*. Springer, 2014, pp. 511–519.

[175] A. Kattan, R. Abdullah, and R. Salam, "Harmony search based supervised training of artificial neural networks," in *2010 Int. Conf. Int. Syst., Model. and Simulation (ISMS),*, 2010, pp. 105–110.

[176] S. Kulluk, L. Ozbakir, and A. Baykasoglu, "Training neural networks with harmony search algorithms for classification problems," *Eng. Appl. Artif. Intell.*, vol. 25, no. 1, pp. 11 – 19, Feb 2012.

[177] M. Mahdavi, M. Fesanghary, and E. Damangir, "An improved harmony search algorithm for solving optimization problems," *Appl. Math. Comput.*, vol. 188, no. 2, pp. 1567 – 1579, May 2007.

[178] Q.-K. Pan, P. Suganthan, M. F. Tasgetiren, and J. Liang, "A self-adaptive global best harmony search algorithm for continuous optimization problems," *Appl. Math. Comput.*, vol. 216, no. 3, pp. 830 – 848, Apr 2010.

[179] M.-H. Horng, M.-C. Lee, R.-J. Liou, and Y.-X. Lee, "Firefly meta-heuristic algorithm for training the radial basis function network for data classification and disease diagnosis," in *Theory and New Applications of Swarm Intelligence*, R. Parpinelli and H. S. Lopes, Eds. InTech, 2012.

[180] R. A. Vázquez, "Training spiking neural models using cuckoo search algorithm," in *IEEE Congr. Evol. Comput. (CEC), 2011*, 2011, pp. 679–686.

[181] M. Ghalambaz, A. Noghrehabadi, M. Behrang, E. Assareh, A. Ghanbarzadeh, and N. Hedayat, "A hybrid neural network and gravitational search algorithm (HNNGSA) method to solve well known wessinger's equation," *World Acad. Sci. Eng. Technol.*, vol. 5, pp. 803–807, Jan 2011.

[182] M. Ulagammai, P. Venkatesh, P. Kannan, and N. P. Padhy, "Application of bacterial foraging technique trained artificial and wavelet neural networks in load forecasting," *Neurocomputing*, vol. 70, no. 16, pp. 2659 – 2667, Oct 2007.

[183] Y. Zhang, L. Wu, and S. Wang, "Bacterial foraging optimization based neural network for short-term load forecasting," *J. Comput. Inform. Syst.*, vol. 6, no. 7, pp. 2099–2105, Jan 2010.

[184] R. C. G. II, L. Wang, and M. Alam, "Training neural networks using central force optimization and particle swarm optimization: Insights and comparisons," *Expert Syst. Appl.*, vol. 39, no. 1, pp. 555 – 563, Dec 2012.

[185] S. Nandy, P. P. Sarkar, and A. Das, "Analysis of a nature inspired firefly algorithm based back-propagation neural network training," *Int. J. Comput. Appl.*, vol. 43, no. 2, pp. 8 – 16, Apr 2012.

[186] E. Alba and R. Marti, *Metaheuristic Procedures for Training Neural Networks*. Springer, 2006.

[187] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: a New Tool for Evolutionary Computation*. Springer, 2002, vol. 2.

[188] A. R. Carvalho, F. M. Ramos, and A. A. Chaves, "Metaheuristics for the feedforward artificial neural network (ann) architecture optimization problem," *Neural Comput. Appl.*, vol. 20, no. 8, pp. 1273–1284, Dec 2011.

[189] P. Kordík, J. Koutník, J. Drchal, O. Kovářík, M. Čepek, and M. Šnorek, "Meta-learning approach to neural network optimization," *Neural Netw.*, vol. 23, no. 4, pp. 568–582, May 2010.

[190] K. Khan and A. Sahai, "A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context," *Int. J. Intell. Syst. Appl.*, vol. 4, no. 7, p. 23, 2012.

[191] Y. Da and G. Xiurun, "An improved PSO-based ANN with simulated annealing technique," *Neurocomputing*, vol. 63, no. 0, pp. 527 – 533, Jan 2005.

[192] M. Ali Ahmadi, S. Zendehboudi, A. Lohi, A. Elkamel, and I. Chatzis, "Reservoir permeability prediction by neural networks combined with hybrid genetic algorithm and particle swarm optimization," *Geophys. Prospect.*, vol. 61, no. 3, pp. 582–598, May 2013.

[193] J. P. Donate, X. Li, G. G. Sánchez, and A. S. de Miguel, "Time series forecasting by evolving artificial neural networks with genetic algorithms, differential evolution and estimation of distribution algorithm," *Neural Comput. Appl.*, vol. 22, no. 1, pp. 11–20, Jan 2013.

[194] S. Mirjalili, S. Z. Mohd Hashim, and H. Moradian Sardroudi, "Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm," *Appl. Math. Comput.*, vol. 218, no. 22, pp. 11 125–11 137, Jul 2012.

[195] B. Niu, Y. Zhu, X. He, and H. Wu, "MCPSO: a multi-swarm cooperative particle swarm optimizer," *Appl. Math. Comput.*, vol. 185, no. 2, pp. 1050 – 1062, Feb 2007.

[196] M. Frean, "The UPSTART algorithm: a method for constructing and training feedforward neural networks," *Neural Comput.*, vol. 2, no. 2, pp. 198–209, Apr 1990.

[197] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 39–53, Jan 1994.

[198] W. Xi-Zhao, S. Qing-Yan, M. Qing, and Z. Jun-Hai, "Architecture selection for networks trained with extreme learning machine using localized generalization error model," *Neurocomputing*, vol. 102, pp. 3–9, Feb 2013.

[199] D. Whitley and T. Hanson, "Optimizing neural networks using faster, more accurate genetic search," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 391–396.

[200] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," *Physica D*, vol. 42, no. 1, pp. 244–248, Jun 1990.

[201] S. A. Harp, T. Samad, and A. Guha, "Towards the genetic synthesis of neural network," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 360–369.

[202] E. Mjolsness, D. H. Sharp, and B. K. Alpert, "Scaling, machine learning, and genetic neural nets," *Adv. Appl. Math.*, vol. 10, no. 2, pp. 137–163, Jun 1989.

[203] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Syst.*, vol. 4, no. 4, pp. 461–476, 1990.

[204] A. A. Siddiqi and S. M. Lucas, "A comparison of matrix rewriting versus direct encoding for evolving neural networks," in *The 1998 IEEE Int. Conf. Evol. Comput. Proc., 1998. IEEE World Congr. Comput. Intell.*, 1998, pp. 392–397.

[205] J. W. Merrill and R. F. Port, "Fractally configured neural networks," *Neural Netw.*, vol. 4, no. 1, pp. 53–60, 1991.

[206] H. C. Andersen and A. C. Tsoi, "A constructive algorithm for the training of a multilayer perceptron based on the genetic algorithm," *Complex Syst.*, vol. 7, no. 4, pp. 249–268, 1993.

[207] M. Tayefeh Mahmoudi, F. Taghiyareh, N. Forouzideh, and C. Lucas, "Evolving artificial neural network structure using grammar encoding and colonial competitive algorithm," *Neural Comput. Appl.*, vol. 22, no. 1, pp. 1–16, May 2013.

[208] T. Ludermir, A. Yamazaki, and C. Zanchettin, "An optimization methodology for neural network weights and architectures," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1452–1459, Nov 2006.

[209] M. Carvalho and T. Ludermir, "Particle swarm optimization of neural network architectures andweights," in *7th Int. Conf. Hybrid Intell. Syst., 2007. HIS 2007*, 2007, pp. 336–339.

[210] J.-T. Tsai, T.-K. Liu, and J.-H. Chou, "Hybrid taguchi-genetic algorithm for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 4, pp. 365–377, Aug 2004.

[211] J.-T. Tsai, J.-H. Chou, and T.-K. Liu, "Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 69–80, Jan 2006.

[212] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Evolutionary artificial neural networks by multi-dimensional particle swarm optimization," *Neural Netw.*, vol. 22, no. 10, pp. 1448 – 1462, Dec 2009.

[213] M. M. Khan, A. M. Ahmad, G. M. Khan, and J. F. Miller, "Fast learning neural networks using cartesian genetic programming," *Neurocomputing*, vol. 121, pp. 274 – 289, Dec 2013.

[214] J. R. Koza and J. P. Rice, "Genetic generation of both the weights and architecture for a neural network," in *Int. Jt. Conf. Neural Netw., 1991., IJCNN*, vol. 2, 1991, pp. 397–404.

[215] I. Tsoulos, D. Gavrilis, and E. Glavas, "Neural network construction and training using grammatical evolution," *Neurocomputing*, vol. 72, no. 1, pp. 269 – 277, Dec 2008.

[216] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[217] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[218] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2015.* IEEE, 2015, pp. 922–928.

[219] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 4, pp. 809–821, 2016.

[220] G.-B. Huang, "An insight into extreme learning machines: random neurons, random features and kernels," *Cognit. Comput.*, vol. 6, no. 3, pp. 376–390, 2014.

[221] I. K. Fodor, "A survey of dimension reduction techniques," Lawrence Livermore National Laboratory, Tech. Rep. UCRL-ID-148494, 2002.

[222] J. Fontanari and R. Meir, "Evolving a learning algorithm for the binary perceptron," *Network: Comput. Neural Syst.*, vol. 2, no. 4, pp. 353–359, 1991.

[223] Z. Guo and R. E. Uhrig, "Using genetic algorithms to select inputs for neural networks," in *Int. Workshop on Combinations of Genetic Algorithms and Neural Netw., 1992., COGANN-92*, 1992, pp. 223–234.

[224] S. Venkadesh, G. Hoogenboom, W. Potter, and R. McClendon, "A genetic algorithm to refine input data selection for air temperature prediction using artificial neural networks," *Appl. Soft Comput.*, vol. 13, no. 5, pp. 2253–2260, May 2013.

[225] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *IEEE Int. Conf. on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997*, vol. 5. IEEE, 1997, pp. 4104–4108.

[226] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines," *Expert Syst. Appl.*, vol. 35, no. 4, pp. 1817–1824, 2008.

[227] S. M. Vieira, L. F. Mendonça, G. J. Farinha, and J. M. Sousa, "Modified binary pso for feature selection using svm applied to mortality prediction of septic patients," *Appl. Soft. Comput.*, vol. 13, no. 8, pp. 3494–3504, 2013.

[228] R. K. Sivagaminathan and S. Ramakrishnan, "A hybrid approach for feature subset selection using neural networks and ant colony optimization," *Expert Syst. Appl.*, vol. 33, no. 1, pp. 49–60, 2007.

[229] B.-T. Zhang and G. Veenker, "Neural networks that teach themselves through genetic discovery of novel examples," in *Int. Jt. Conf. Neural Netw., 1991., IJCNN*, 1991, pp. 690–695.

[230] S. Cho and K. Cha, "Evolution of neural network training set through addition of virtual samples," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1996, pp. 685–688.

[231] Y. Liu and X. Yao, "Evolutionary design of artificial neural networks with different nodes," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1996, pp. 670–675.

[232] V. K. Ojha, A. Abraham, and V. Snášel, "Simultaneous optimization of neural network weights and active nodes using metaheuristics," in *14th Int. Conf. on Hybrid Intell. Syst. (HIS), 2014*, Dec 2014, pp. 248–253.

[233] S.-K. Oh and W. Pedrycz, "The design of self-organizing polynomial neural networks," *Inf. Sci.*, vol. 141, no. 3, pp. 237–258, 2002.

[234] A. Hirose, *Complex-valued neural networks.* Springer Science & Business Media, 2006.

[235] G. Mani, "Learning by gradient descent in function space," in *IEEE Int. Conf. Syst., Man, Cybern.*, 1990, pp. 242–247.

[236] J. P. Dumont, R. M. Robertson *et al.*, "Neuronal circuits: an evolutionary perspective," *Sci.*, vol. 233, no. 4766, pp. 849–853, Aug 1986.

[237] D. G. Stork, S. Walker, M. Burns, and B. Jackson, "Preadaptation in neural circuits," in *Int. Jt. Conf. Neural Netw.*, vol. 1, 1990, pp. 202 – 205.

[238] S. Ling, F. Leung, and H. Lam, "Input-dependent neural network trained by real-coded genetic algorithm and its industrial applications," *Soft Comput.*, vol. 11, no. 11, pp. 1033–1052, Sep 2007.

[239] J.-M. Yang and C.-Y. Kao, "A robust evolutionary algorithm for training neural networks," *Neural Comput. Appl.*, vol. 10, no. 3, pp. 214–230, Dec 2001.

[240] A. Alvarez, "A neural network with evolutionary neurons," *Neural Process. Lett.*, vol. 16, no. 1, pp. 43–52, Aug 2002.

[241] F. H. F. Leung, H. Lam, S. Ling, and P.-S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79–88, Jan 2003.

[242] M. F. Augusteijn and T. P. Harrington, "Evolving transfer functions for artificial neural networks," *Neural Comput. Appl.*, vol. 13, no. 1, pp. 38–46, Apr 2004.

[243] N. Nedjah, A. Abraham, and L. M. Mourelle, "Hybrid artificial neural network," *Neural Comput. Appl.*, vol. 16, no. 3, pp. 207–208, May 2007.

[244] A. Andoni, R. Panigrahy, G. Valiant, and L. Zhang, "Learning polynomials with neural networks," in *Proc. of the 31st Int. Conf. on Machine Learning (ICML-14)*, 2014, pp. 1908–1916.

[245] V. Puig, M. Witczak, F. Nejjari, J. Quevedo, and J. Korbicz, "A gmdh neural network-based approach to passive robust fault detection using a constraint satisfaction backward test," *Eng. Appl. Artif. Intell.*, vol. 20, no. 7, pp. 886–897, 2007.

[246] I. Sporea and A. Grüning, "Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 25, no. 2, pp. 473–509, 2013.

[247] R. Fullér, *Introduction to neuro-fuzzy systems*. Springer Science & Business Media, 2013, vol. 2.

[248] A. J. da Silva, T. B. Ludermir, and W. R. de Oliveira, "Quantum perceptron over a field and neural network architecture selection in a quantum computer," *Neural Netw.*, vol. 76, pp. 55–64, 2016.

[249] A. Narayanan and T. Menneer, "Quantum artificial neural network architectures and components," *Inf. Sci.*, vol. 128, no. 3, pp. 231–255, 2000.

[250] N. Kouda, N. Matsui, H. Nishimura, and F. Peper, "Qubit neural network and its learning efficiency," *Neural Comput. Appl.*, vol. 14, no. 2, pp. 114–121, 2005.

[251] P. Li, H. Xiao, F. Shang, X. Tong, X. Li, and M. Cao, "A hybrid quantum-inspired neural networks with sequence inputs," *Neurocomputing*, vol. 117, pp. 81–90, 2013.

[252] J. Baxter, "The evolution of learning algorithms for artificial neural networks," in *Complex Syst.–*, 1992, pp. 313–326.

[253] D. J. Chalmers, "The evolution of learning: an experiment in genetic connectionism," in *Proc. 1990 Connectionist Models Summer School*, 1990, pp. 81–90.

[254] H. B. Kim, S. H. Jung, T. G. Kim, and K. H. Park, "Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates," *Neurocomputing*, vol. 11, no. 1, pp. 101–106, May 1996.

[255] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep 1999.

[256] A. Abraham, "Meta learning evolutionary artificial neural networks," *Neurocomputing*, vol. 56, no. 0, pp. 1 – 38, Jan 2004.

[257] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, 1994.

[258] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.

[259] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun 2002.

[260] J. Gauci and K. Stanley, "Generating large-scale neural networks through discovering geometric regularities," in *Proc. 9th Ann. Conf. Genetic Evol. Comput.* ACM, 2007, pp. 997–1004.

[261] Y. Kassahun and G. Sommer, "Efficient reinforcement learning through evolutionary acquisition of neural topologies." in *Proc. 13th European Symp. on Artificial Neural Networks (ESANN 2005)*, 2005, pp. 259–266.

[262] V. K. Ojha, A. Abraham, and V. Snášel, "Ensemble of heterogeneous flexible neural trees using multiobjective genetic programming," *Appl. Soft Comput.*, vol. in press, 2016.

[263] D. Dasgupta and D. McGregor, "Designing application-specific neural networks using the structured genetic algorithm," in *Int. Workshop on Combinations of Genetic Algorithms and Neural Netw., 1992., COGANN-92*, 1992, pp. 87–96.

[264] H. Kitano, "Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms," *Physica D*, vol. 75, no. 1, pp. 225 – 238, Aug 1994.

[265] J. Arifovic and R. Genay, "Using genetic algorithms to select architecture of a feedforward artificial neural network," *Physica A*, vol. 289, pp. 574 – 594, Jan 2001.

[266] E. Salajegheh and S. Gholizadeh, "Optimum design of structures by an improved genetic algorithm using neural networks," *Adv. Eng. Softw.*, vol. 36, no. 1112, pp. 757 – 767, Nov 2005.

[267] N. García-Pedrajas, D. Ortiz-Boyer, and C. Hervás-Martínez, "An alternative approach for neural network evolution with a genetic algorithm: crossover by combinatorial optimization," *Neural Netw.*, vol. 19, no. 4, pp. 514–528, May 2006.

[268] D. E. Moriarty and R. Miikkulainen, "Forming neural networks through efficient and adaptive coevolution," *Evol. Comput.*, vol. 5, no. 4, pp. 373–399, Dec 1997.

[269] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez, "COVNET: a cooperative coevolutionary model for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 575–596, May 2003.

[270] B.-T. Zhang, P. Ohm, and H. Mühlenbein, "Evolutionary induction of sparse neural trees," *Evol. Comput.*, vol. 5, no. 2, pp. 213–236, Jun 1997.

[271] Y. Chen, B. Yang, and J. Dong, "Nonlinear system modelling via optimal design of neural trees," *Int. J. Neural Syst.*, vol. 14, no. 2, pp. 125–137, Apr 2004.

[272] Y. Chen, A. Abraham, and B. Yang, "Feature selection and classification using flexible neural tree," *Neurocomputing*, vol. 70, no. 1, pp. 305 – 313, Dec 2006.

[273] S. Bouaziz, A. M. Alimi, and A. Abraham, "Universal approximation propriety of flexible beta basis function neural tree," in *2014 Int. Jt. Conf. Neural Netw. (IJCNN)*, 2014, pp. 573–580.

[274] L. Peng, B. Yang, L. Zhang, and Y. Chen, "A parallel evolving algorithm for flexible neural tree," *Parallel Comput.*, vol. 37, no. 1011, pp. 653 – 666, Oct 2011.

[275] L. Wang, B. Yang, Y. Chen, X. Zhao, J. Chang, and H. Wang, "Modeling early-age hydration kinetics of portland cement using flexible neural tree," *Neural Comput. Appl.*, vol. 21, no. 5, pp. 877–889, Jul 2012.

[276] M. Lewenstein, "Quantum perceptrons," *J. Mod. Opt.*, vol. 41, no. 12, pp. 2491–2501, 1994.

[277] N. Gershenfeld and I. L. Chuang, "Quantum computing with molecules," *Sci. Am.*, vol. 278, no. 6, pp. 66–71, 1998.

[278] T. Menneer and A. Narayanan, "Quantum-inspired neural networks," Univ. of Exeter., Tech. Rep. Tech. Rep. R329, 1995.

[279] E. C. Behrman, J. Niemel, J. E. Steck, and S. R. Skinner, "A quantum dot neural network," in *Proc. of the 4th Workshop on Physics of Computation*, 1996, pp. 22–24.

[280] R. L. Chrisley, "Learning in non-superpositional quantum neurocomputers," *Brain, Mind and Physics. IOS Press, Amsterdam*, pp. 126–139, 1997.

[281] D. Ventura and T. Martinez, "An artificial neuron with quantum mechanical properties," in *Artificial Neural Nets and Genetic Algorithms*. Springer, 1998, pp. 482–485.

[282] T. G. Rudolph, "A heuristic review of quantum neural networks," Ph.D. dissertation, Imperial College London, 2011.

[283] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[284] Y. Jin, B. Sendhoff, and E. Körner, "Evolutionary multi-objective optimization for simultaneous generation of signal-type and symbol-type representations," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, vol. 3410. Springer, 2005, pp. 752–766.

[285] I. Das and J. E. Dennis, "A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems," *Struct. Optim.*, vol. 14, no. 1, pp. 63–69, 1997.

[286] A. Gaspar-Cunha and A. Vieira, "A multi-objective evolutionary algorithm using neural networks to approximate fitness evaluations," *Int. J. Comput., Syst., Signals*, vol. 6, no. 1, pp. 18–36, Jan 2005.

[287] C. A. C. Coello, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowl. Inform. Syst.*, vol. 1, no. 3, pp. 129–156, Aug 1999.

[288] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.

[289] R. de Albuquerque Teixeira, A. P. Braga, R. H. Takahashi, and R. R. Saldanha, "Improving generalization of MLPs with multi-objective optimization," *Neurocomputing*, vol. 35, no. 1, pp. 189 – 194, Nov 2000.

[290] R. G. Bland, D. Goldfarb, and M. J. Todd, "The ellipsoid method: A survey," *Oper. Res.*, vol. 29, no. 6, pp. 1039–1091, 1981.

[291] M. A. Costa, A. P. Braga, B. R. Menezes, R. A. Teixeira, and G. G. Parma, "Training neural networks with a multi-objective sliding mode control algorithm," *Neurocomputing*, vol. 51, pp. 467–473, Apr 2003.

[292] F. Pettersson, N. Chakraborti, and H. Saxén, "A genetic algorithms based multi-objective neural net applied to noisy blast furnace data," *Appl. Soft Comput.*, vol. 7, no. 1, pp. 387 – 397, Jan 2007.

[293] C.-K. Goh, E.-J. Teoh, and K. Chen Tan, "Hybrid multiobjective evolutionary design for artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 19, no. 9, pp. 1531–1548, Jul 2008.

[294] L. M. Almeida and T. B. Ludermir, "A multi-objective memetic and hybrid methodology for optimizing the parameters and performance of artificial neural networks," *Neurocomputing*, vol. 73, no. 7, pp. 1438 – 1450, Mar 2010.

[295] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Tran. on Syst., Man, and Cybern., Part C: Appl. Rev.*, vol. 38, no. 3, pp. 397–415, 2008.

[296] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Parallel Problem Solving from Nature PPSN VI*, ser. Lecture Notes in Computer Science, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H.-P. Schwefel, Eds. Springer, 2000, vol. 1917, pp. 849–858.

[297] Y. Jin, T. Okabe, and B. Sendhoff, "Neural network regularization and ensembling using multi-objective evolutionary algorithms," in *Congr. Evol. Comput., 2004. CEC2004*, vol. 1, 2004, pp. 1–8.

[298] H. A. Abbass, "Speeding up backpropagation using multiobjective evolutionary algorithms," *Neural Comput.*, vol. 15, no. 11, pp. 2705–2726, Nov 2003.

[299] H. Abbass, "The self-adaptive pareto differential evolution algorithm," in *Proc. 2002 Congr. Evol. Comput., 2002. CEC '02*, vol. 1, 2002, pp. 831–836.

[300] J. P. T. Yusiong and P. C. Naval Jr, "Training neural networks using multiobjective particle swarm optimization," in *Advances in Natural Computation*. Springer, 2006, pp. 879–888.

[301] S. Wiegand, C. Igel, and U. Handmann, "Evolutionary multi-objective optimisation of neural networks for face detection," *Int. J. Comput. Intell. Appl.*, vol. 4, no. 3, pp. 237–253, Sep 2004.

[302] S. Roth, A. Gepperth, and C. Igel, "Multi-objective neural network optimization for visual object detection," in *Multi-Objective Machine Learning*, ser. Studies in Computational Intelligence, Y. Jin, Ed. Springer, 2006, vol. 16, pp. 629–655.

[303] Y. Jin, B. Sendhoff, and E. Körner, "Evolutionary multi-objective optimization for simultaneous generation of signal-type and symbol-type representations," in *Evol. Multi-Criterion Optim.*, ser. Lecture Notes in Computer Science, vol. 3410, 2005, pp. 752–766.

[304] R. Furtuna, S. Curteanu, and F. Leon, "An elitist non-dominated sorting genetic algorithm enhanced with a neural network applied to the multi-objective optimization of a polysiloxane synthesis process," *Eng. Appl. Artif. Intell.*, vol. 24, no. 5, pp. 772 – 785, Aug 2011.

[305] A.-C. Zăvoianu, G. Bramerdorfer, E. Lughofer, S. Silber, W. Amrhein, and E. P. Klement, "Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives," *Eng. Appl. Artif. Intell.*, vol. 26, no. 8, pp. 1781 – 1794, 2013.

[306] Y. Karpat and T. Özel, "Multi-objective optimization for turning processes using neural network modeling and dynamic-neighborhood particle swarm optimization," *Int. J. Adv. Manuf. Technol.*, vol. 35, no. 3-4, pp. 234–247, Dec 2007.

[307] N. García-Pedrajas, C. Hervás-Martínez, and J. Muñoz Pérez, "SYMBIONT: a cooperative evolutionary model for evolving artificial neural networks for classification," in *Technologies for Constructing Intelligent Systems 2*, ser. Studies in Fuzziness and Soft Computing, B. Bouchon-Meunier, J. Gutiérrez-Ríos, L. Magdalena, and R. R. Yager, Eds. Physica-Verlag HD, 2002, vol. 90, pp. 341–354.

[308] N. García-Pedrajas, C. Hervás-Martínez, and J. Muñoz Pérez, "Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks)," *Neural Netw.*, vol. 15, no. 10, pp. 1259–1278, Dec 2002.

[309] O. Giustolisi and V. Simeone, "Optimal design of artificial neural networks by a multi-objective strategy: groundwater level predictions," *Hydrological Sci. J.*, vol. 51, no. 3, pp. 502–523, Jan 2006.

[310] M. Cruz-Ramírez, J. Snchez-Monedero, F. Fernndez-Navarro, J. Fernndez, and C. Hervs-Martnez, "Memetic pareto differential evolutionary artificial neural networks to determine growth multi-classes in predictive microbiology," *Evol. Intell.*, vol. 3, no. 3-4, pp. 187–199, Dec 2010.

[311] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce english text," *Complex Syst.*, vol. 1, no. 1, pp. 145–168, 1987.

[312] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 3, pp. 417–425, Jun 1998.

[313] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits Syst. Mag.*, vol. 6, no. 3, pp. 21–45, 2006.

[314] X. Yao and Y. Liu, "Ensemble structure of evolutionary artificial neural networks," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1996, pp. 659–664.

[315] M. H. Davis and R. B. Vinter, *Stochastic Modelling and Control*. London, UK: Chapman and Hall, 1985.

[316] Z. Zhao and Y. Zhang, "Design of ensemble neural network using entropy theory," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 838 – 845, Oct 2011.

[317] Y. Liu and X. Yao, "Ensemble learning via negative correlation," *Neural Netw.*, vol. 12, no. 10, pp. 1399–1404, Dec 1999.

[318] Z. Qin, Y. Liu, X. Heng, and X. Wang, "Negatively correlated neural network ensemble with multi-population particle swarm optimization," in *Advances in Neural Networks - ISNN 2005*, ser. Lecture Notes in Computer Science, J. Wang, X. Liao, and Z. Yi, Eds. Springer, 2005, vol. 3496, pp. 520–525.

[319] Y. Liu, X. Yao, and T. Higuchi, "Evolutionary ensembles with negative correlation learning," *IEEE Trans. Evol. Comput.*, vol. 4, no. 4, pp. 380–387, Nov 2000.

[320] M. M. Islam, X. Yao, and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 820–834, Jul 2003.

[321] X. Yao and M. M. Islam, "Evolving artificial neural network ensembles," *IEEE Comput. Intell. Mag.*, vol. 3, no. 1, pp. 31–42, Feb 2008.

[322] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: many could be better than all," *Artif. Intell.*, vol. 137, no. 1, pp. 239 – 263, May 2002.

[323] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug 1996.

[324] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, Jun 1990.

[325] B. Bakker and T. Heskes, "Clustering ensembles of neural network models," *Neural Netw.*, vol. 16, no. 2, pp. 261 – 269, Mar 2003.

[326] L. Chen, W. Xue, and N. Tokuda, "Classification of 2-dimensional array patterns: assembling many small neural networks is better than using a large one," *Neural Netw.*, vol. 23, no. 6, pp. 770 – 781, Aug 2010.

[327] G. M.-M. noz, A. Sánchez-Martínez, D. Hernández-Lobato, and A. Suárez, "Class-switching neural network ensembles," *Neurocomputing*, vol. 71, no. 13, pp. 2521 – 2528, Aug 2008.

[328] F. L. Minku and T. B. Ludermir, "Clustering and co-evolution to construct neural network ensembles: an experimental study," *Neural Netw.*, vol. 21, no. 9, pp. 1363 – 1379, Nov 2008.

[329] K.-J. Kim and S.-B. Cho, "Evolutionary ensemble of diverse artificial neural networks using speciation," *Neurocomputing*, vol. 71, no. 7, pp. 1604 – 1618, Mar 2008.

[330] M. Akhand, M. M. Islam, and K. Murase, "Progressive interactive training: a sequential neural network ensemble learning method," *Neurocomputing*, vol. 73, no. 1, pp. 260 – 273, Dec 2009.

[331] A. Chandra and X. Yao, "Ensemble learning using multi-objective evolutionary algorithms," *J. Math. Model. Algorithms*, vol. 5, no. 4, pp. 417–445, Dec 2006.

[332] H. Chen and X. Yao, "Multiobjective neural network ensembles based on regularized negative correlation learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 12, pp. 1738–1751, Feb 2010.

[333] Y. Wand and R. Y. Wang, "Anchoring data quality dimensions in ontological foundations," *Commun. ACM*, vol. 39, no. 11, pp. 86–95, Nov 1996.

[334] L. L. Pipino, Y. W. Lee, and R. Y. Wang, "Data quality assessment," *Commun. ACM*, vol. 45, no. 4, pp. 211–218, Apr 2002.

[335] M. A. Hernández and S. J. Stolfo, "Real-world data is dirty: Data cleansing and the merge/purge problem," *Data Min. Knowl. Discovery*, vol. 2, no. 1, pp. 9–37, 1998.

[336] S. Cho, M. Jang, and S. Chang, "Virtual sample generation using a population of networks," *Neural Process. Lett.*, vol. 5, no. 2, pp. 21–27, 1997.

[337] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi, "Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance," *Neural Netw.*, vol. 21, no. 2, pp. 427–436, 2008.

[338] P. Zikopoulos, C. Eaton *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[339] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov 2012.

[340] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul 2006.

[341] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[342] D. Saad, *On-line learning in neural networks*. Cambridge University Press, 2009, vol. 17.

[343] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: a survey," *IEEE Comput. Intell. Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[344] M. D. Ritchie, E. R. Holzinger, R. Li, S. A. Pendergrass, and D. Kim, "Methods of integrating data to uncover genotype-phenotype interactions," *Nature Rev. Genetics*, vol. 16, no. 2, pp. 85–97, 2015.

[345] P. Pavlidis, J. Weston, J. Cai, and W. N. Grundy, "Gene functional classification from heterogeneous data," in *Proc. of the 5th Ann. Int. Conf. on Comput. Biology*. ACM, 2001, pp. 249–255.

[346] P. Prisecaru, "Challenges of the fourth industrial revolution," *Knowledge Horizons. Economics*, vol. 8, no. 1, p. 57, 2016.

[347] E. Kim, S. Helal, and D. Cook, "Human activity recognition and pattern discovery," *IEEE Pervasive Comput.*, vol. 9, no. 1, pp. 48–53, 2010.

[348] E. Trentin and M. Gori, "A survey of hybrid ann/hmm models for automatic speech recognition," *Neurocomputing*, vol. 37, no. 1, pp. 91–126, 2001.