Feature Selection and Function Approximation Using Adaptive Algorithms

Varun Kumar Ojha

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VŠB–Technical University of Ostrava

Supervisors
Prof. Dr. Ajith Abraham and Prof. RNDr. Václav Snášel, CSc.

This dissertation is submitted for the degree of
Doctor of Philosophy

June 2016

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

Varun Kumar Ojha
June 2016

Acknowledgements

I express my deepest gratitude to my supervisors Prof. Dr. Ajith Abraham and Prof. Dr. Vaclav Snášel for their immense support, advice, and encouragements, without which, it is unthinkable for me to accomplish this thesis at all.

I would like to acknowledge the financial support by the IPROCOM Marie Curie Initial Training Network, funded by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007–2013/, under REA grant agreement number 316555.

I would like to thank Prof. Dr. Chuan-Yu (Charley) Wu, Department of Chemical and Process Engineering, University of Surrey, Guildford, United Kingdom, for his inspirations, and collaboration for the understanding pharmaceutical industrial part that helped me in computational intelligence modeling of the industrial problems.

I would like to thank Dr. Deepak Amban Mishra, a friend of mine, for his motivations during the period of my thesis writing. I am thankful to my colleagues in the Faculty of Electronic, Electrical and Computer Science for their technical interactions and advises, whenever, I was in need. I am also thankful to the committee members of the examination board.

Finally, I would like to thank my family, friends, and teachers who have encouraged me during the period of my research and thesis writing.

Abstract

Multiobjective heterogeneous flexible neural tree (HFNT) and multiobjective hierarchical fuzzy inference tree (HFIT) are two novel adaptive algorithms, which were proposed for the feature selection and function approximation after comprehensive literature reviews of the neural network and fuzzy inference system paradigms, respectively. The proposed algorithms were designed as a tree-like model, and the best tree-structure was selected from a topological space by applying a multiobjective evolutionary algorithm that simultaneously minimized both approximation error and tree complexity. Further, the parameter vector of the selected tree, from the Pareto front, was tuned by using a metaheuristic algorithm. For HFNT, the dynamics of natural selection was exploited to introduce functional heterogeneity in the HFNT nodes, and a diversity index was introduced for creating diverse HFNTs during its tree optimization phase. Subsequently, an evolutionary ensemble of HFNTs was proposed for making use of the final population. On the other hand, the HFIT nodes were low-dimensional type-1 or type-2 fuzzy inference systems, and the tree-like model was a hierarchical arrangement of such nodes. The performance of both HFNT and HFIT on benchmark datasets was better than the performance of the algorithms in the literature. Additionally, both HFNT and HFIT was used for the predictive modeling of the industrial problems, in which the feature selection was a crucial challenge in addition to the prediction. High approximation ability with the simple model generation is the vital contribution of the proposed algorithms for predictive modeling of complex problems.

Keywords: feedforward neural network; fuzzy inference system; multiobjective; meta-heuristics; ensemble learning; feature selection.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

A^i	type-1 fuzzy Set
B^i	type-1 consequent function
$\mathcal{F}$	space of all possible optimization problems
c_f	$c_f : Y \times \hat{Y} \longrightarrow \mathbb{R}_{\geq 0}$ is a cost function
cr	crossover rate
c_t^y	value of a cost function c_f after t iterations
E	approximation error (MSE or RMSE)
E_n	approximation error on training set
E_t	approximation error on test set
g	gradient of error with respect to vector $\mathbf{w}$
H	Hessian matrix
$k(\mathbf{w})$	function that returns size of a vector
N	total number of input-output pairs into a training set
∇^2	second-order partial derivative
p	total input feature
pc	crossover probability
p^i	$0 < p^i \leq p$ number of input for a rule
pm	mutation probability

$\mathcal{P}(\cdot)$	power set of the set "$\cdot$"
P	predictability score of the input features
q	total output feature
$\mathbf{R}_i$	collection or a set of rules
R^i	fuzzy type-1 or type-2 rule
$\mathbb{R}^n$	n-dimensional real space
r_n	correlation coefficient on training set
R	selection rate of the input features
r_t	correlation coefficient on test set
S	problem search space $S \subset \mathbb{R}^n$
$\tilde{A}^i$	type-2 fuzzy Set
$\tilde{B}^i$	type-2 consequent function
$\mathbf{w}$	n-dimensional solution vector $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$ of a model
W_A	solution archive, where $W_A \subset W$
W_h	population matrix $W_h = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$ of weight vectors
W_g	population of trees (either HFNT or HFIT depending on the context)
$\parallel \mathbf{w} \parallel$	euclidean norm of vector $\mathbf{w}$
$\mathbf{w}^*$	optimal solution of a problem
$\mathbf{x}_i$	p-dimensional input vector $\mathbf{x}_i = \langle x_{i1}, x_{i2}, \ldots, x_{ip} \rangle$
X	input data $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$
$\mathbf{y}_i$	q-dimensional desired output vector $\mathbf{y}_i = \langle y_{i1}, y_{i2}, \ldots, y_{iq} \rangle$
Y	desired output data $Y = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N)$
$\hat{\mathbf{y}}_i$	q-dimensional models' output vector $\hat{\mathbf{y}}_i = \langle \hat{y}_{i1}, \hat{y}_{i2}, \ldots, \hat{y}_{iq} \rangle$
$\hat{Y}$	model's outputs $\hat{Y} = (\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_N)$

Z	$Z = \{z_1, z_2, \ldots z_p\}$ is a set of p input features
$\mathbb{Z}^+$	positive integers
$\mathbf{Z}$	$\mathbf{Z} = \mathcal{P}(Z) - \phi$ is a power set of Z (excluding null set ϕ)
$\mathbf{Z}^s$	$\mathbf{Z}^s \subset \mathbf{Z}$ is a set of selected feature sets
Z^*	optimum set of input features

Greek Symbols

α	momentum factor
δ	deferential weight factor in DE algorithm
$\Delta\mathbf{w}^t$	weight-change computed at the t-th learning iteration
$\emptyset$	empty set
ϵ	termination criteria for an algorithm
η	learning rate
γ	multiplicative factors for MFs center
λ	hyperparameter to control two objectives
μ_A	type-1 membership function
$\bar{\mu}_{\tilde{A}}$	type-2 upper membership function
$\mu_{\tilde{A}}$	type-2 membership function
$\underline{\mu}_{\tilde{A}}$	type-2 lower membership function
ω_j	j-th class/category of multi-class classification problem
$\varphi(\cdot)$	activation function that limits the output of a neural node
σ	width of a membership function

Acronyms / Abbreviations

ACO	Ant Colony Optimization
ANFIS	Adaptive Network Based Fuzzy Inference System

ANN	Artificial Neural Network
BP	Backpropagation
CG	Conjugate Gradient
DB	Data Base
DE	Differential Evolution
DyEFuNN	Dynamic Evolving Fuzzy Neural Network
EA	Evolutionary Algorithm
EFS	Evolving Fuzzy Systems
EFuNN	Evolving Fuzzy Neural Network
EP	Evolutionary Programming
FALCON	Fuzzy Adaptive Learning Control Network
FCV	Fold Cross-Validation
FIS	Fuzzy Inference System
FNN	Feedforward Neural Network
FNT	Flexible Neural Tree
FRBS	Fuzzy Rule Base System
GA	Genetic Algorithm
GP	Genetic Programming
GPR	Gaussian Process Regression
HFIS	Hierarchical Fuzzy Inference System
HFIT	Hierarchical Fuzzy Inference Tree
HFNT	Heterogeneous Flexible Neural Tree
HFNT^{M}	Heterogeneous Flexible Neural Tree (Multiobjective)
IT2FIS	Interval Type-2 Fuzzy Inference System

IT2FNN	Interval Type-2 Fuzzy Neural Network
IT2FS	Interval Type-2 Fuzzy Set
KB	Knowledge Base
LMF	Lower Membership Function
MCC	Microcrystalline Cellulose
McIT2FIS	Meta-Cognitive Interval Type-2 Neuro-Fuzzy Inference System
MF	Membership Function
MIF	Most Insignificant Features
MLP	Multilayer Perceptron
MOGP	Multiobjective Genetic Programming
MRIT2NFS	Mutually Recurrent Interval Type-2 Neural Fuzzy System
MSE	Mean Squared Error
MSF	Most Significant Features
NEAT	NeuroEvolution of Augmenting Topologies
NFL	No Free Lunch Theorem
NN	Neural Network
NSGA	Nondominated Sorting Genetic Algorithm
PLGA	Poly (Lactic-co-Glycolic Acid)
PSO	Particle Swarm Optimization
RB	Rule Base
REP	Reduced Error Pruning
RIT2FNS-WB	Reduced IT2NFS-Weighted Bound-set
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network

SaFIN	Self-Adaptive Fuzzy Inference Network
SA	Simulated Annealing
SEIT2FNN	Self-Evolving Interval Type-2 Fuzzy Neural Network
SIT2FNN	Simplified IT2FNN
SLP	Single Layer Perceptron
SONFIN	Self Constructing Neural Fuzzy Inference Network
SVR	Support Vector Regressor
T1FIS	Type-1 Fuzzy Inference System
T1FS	Type-1 Fuzzy Set
T1HFIT$^{\mathrm{M}}$	Type-1 HFIT (Multiobjective)
T1HFIT$^{\mathrm{S}}$	Type-1 HFIT (Single objective)
T2FIS	Type-2 Fuzzy Inference System
T2FS	Type-2 Fuzzy Set
T2HFIT$^{\mathrm{M}}$	Type-2 HFIT (Multiobjective)
T2HFIT$^{\mathrm{S}}$	Type-2 HFIT (Single objective)
TSCIT2FNN	Compensatory IT2FNN
TSF	Total Selected Features
TSK	Takagi–Sugeno–Kang FIS
UMF	Upper Membership Function

Chapter 1

Introduction

Developing algorithms for understanding data and discovering meaningful knowledge from them has been a several decades quest, and to do so, two tasks usually performed are feature selection and function approximation. Feature selection plays a crucial role in analyzing the most significant variables in a dataset, and the function approximation is involved in finding the meaningful relationship among the variables of a dataset. Therefore, the goal is to develop the algorithms that perform these two tasks efficiently and simultaneously.

1.1 Problem statement

Feature selection and function approximation are usually viewed within the supervised learning paradigm. Two computational intelligence models, feedforward neural network (FNN) and fuzzy inference system (FIS), perform these tasks efficiently when trained/optimized 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), a model $f(\mathbf{x}, \mathbf{w})$ produces output $\hat{Y} = (\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_N)$, where $f : X \times Y \longrightarrow \hat{Y}$, $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$ is the n-dimensional solution vector, and $\hat{\mathbf{y}}_i = \langle \hat{y}_{i1}, \hat{y}_{i2}, \ldots, \hat{y}_{iq} \rangle$ is a q-dimensional model's output, which is then compared with desired output $\mathbf{y}_i$, for all $i = 1$ to N, by using some error/distance/cost function c_f. For the problems whose desired outputs are continuous variables, *mean squared error* is one of the commonly used cost function, which is expressed as:

$$c_f(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{q} \left(y_{ij} - \hat{y}_{ij} \right)^2 , \tag{1.1}$$

where y_{ij} are the desired outputs and $\hat{y}_{ij}$ are the model's outputs, and their differences are summed over N data pairs.

Thus, a model $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{w})$, parameterized by p-dimensional input vector $\mathbf{x}$ and an n-dimensional real-valued vector $\mathbf{w}$, under a supervised learning training tries to find a casual relationship between input X and output Y by searching optimum parameter vector $\mathbf{w}^*$ and by minimizing the cost function $c_f : Y \times \hat{Y} \longrightarrow \mathbb{R}_{\geq 0}$. Such process is called function approximation, which may be defined as:

Definition 1.1 (Function approximation) *Given the dataset (X, Y), the goal in function approximation is to find a function $f(\mathbf{x}, \mathbf{w}^*)$ such that $c_f(\mathbf{y}, f(\mathbf{x}, \mathbf{w}^*)) \leq c_f(\mathbf{y}, f(\mathbf{x}, \mathbf{w}))$ holds for any $\mathbf{w} \in \mathbb{R}^n$.*

Let an input vector $\mathbf{x}$ have p input feature. Then, for a set of input features Z, the feature selection may be defined as:

Definition 1.2 (Feature selection) *Given an input feature set $Z = \{z_1, z_2, \ldots, z_p\}$ and the power set $\mathbf{Z} = \mathcal{P}(Z) - \emptyset$, a feature selection method finds an optimal set Z^*, for which a model gives the lowest approximation error, where $Z^* \in \mathbf{Z}$ and $|Z^*| \leq |Z|$.*

The following section describes some preliminary concepts of computational intelligence methods and tools for the understanding of the proposed algorithms: multiobjective heterogeneous flexible neural tree (HFNT) and multiobjective hierarchical fuzzy inference tree (FHIT). Both of the proposed algorithms were developed after a comprehensive literature review of FNN and FIS paradigms.

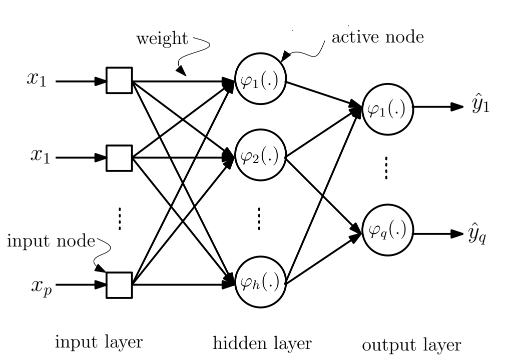
1.2 Preliminaries

1.2.1 Feedforward neural network

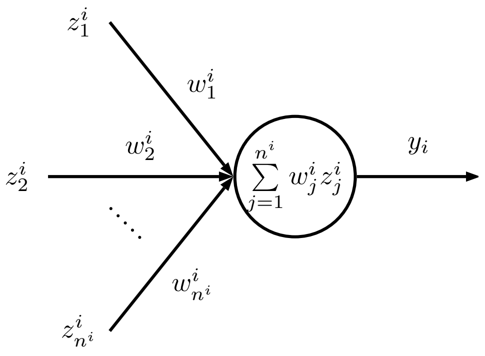
Feedforward neural networks (FNNs) are the computational models that consist of many neurons (*node*), which are connected using synaptic links (*weights*) and are arranged in layer-by-layer basis. Thus, FNNs have a specific structural configuration (*architecture*), in which the nodes at a layer have forward connections from the nodes of its previous layer (Fig. 1.1(a)). A node of a FNN is capable of processing information coming through the connection weights (Fig. 1.1(b)). Mathematically, the output y_i (excitation) of a node (node indicated as i) is computed as:

$$y_i = \varphi_i \left(\sum_{j=1}^{n^i} w_j^i z_j^i + b^i \right), \tag{1.2}$$

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. An *activation function* limits the amplitude of the output at a node into a certain range.



(a) Three-layer feedforward neural network



(b) Node of the network

Fig. 1.1 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.

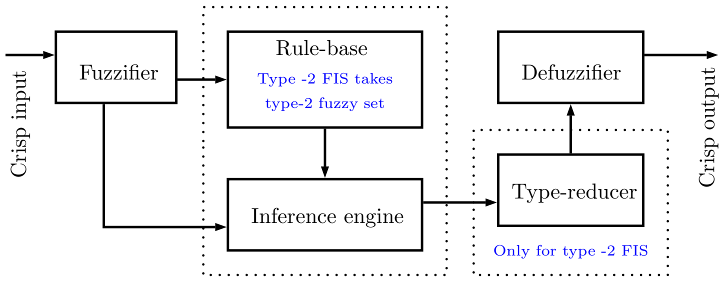


Fig. 1.2 Typical fuzzy inference system.

1.2.2 Fuzzy inference system

Fuzzy inference system (FIS) has a vast range of application domain since it perform human-like reasoning by modeling ambiguous, uncertain, incomplete, inaccurate, and noisy information. Therefore, it is preferred for prediction modeling in several industrial applications, where often noisy data are available for the modeling. A FIS is composed of a fuzzifier that fuzzifies input information, an inference engine that infers information from a rule-base, and a defuzzifier that returns crisp outcome (Fig. 1.2). In simple words, a FIS infer the information from a rule-base. Hence, a FIS can be described as a set of rules, where the rules are in IF-THEN form, i.e., in the antecedent and the consequent form. Takagi–Sugeno–Kang (TSK) model is a widely used FIS model that embraced IF-THEN form in which the antecedent part consists of membership functions (MFs), and the consequent part consists of a linear polynomial function [1]. A detail discussion of FIS is available in Chapter 4.

1.2.3 Metaheuristic algorithms

Metaheuristics are nature-inspired algorithms that optimizes a function $f(\mathbf{x}, \mathbf{w})$ by searching an optimum parameter vector $\mathbf{w}^* \in \mathbb{R}^n$ for the function through a vast search space.

For this purpose, the metaheuristic algorithms efficiently combines two technique: exploitation and exploration. By exploitation, it creates a new solution from the already discovered solutions, and by exploration, it creates a new solution by searching through the new areas of the search space [2]. The concept of the exploration and the exploitation are contradictory to each other, and a good search algorithm must find trade-offs between these two concepts. Hence, an effective algorithm should, intelligently, combine these two strategies.

A general procedure of the metaheuristics is illustrated in Algorithm 1.1, where index t indicates the iteration and $\mathrm{MH}_{\mathrm{Operator}}$ indicates a metaheuristic operator applied over the solution vector $\mathbf{w}$ of the population $W_h = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$. The definition of the metaheuristic operator ($\mathrm{MH}_{\mathrm{Operator}}$) is varied from algorithm to algorithm.

There are several types of metaheuristic algorithms. Each metaheuristic algorithm is inspired by a form of natural heuristics. For example, artificial bee colony [3], bacteria foraging optimization [4], and particle swarm optimization [5] are inspired from the foraging behavior of swarm. On the other hand, differential evolution [6] and genetic algorithm [7] are inspired by evolutionary process of natural selection. However, the metaheuristic algorithms follows a common procedure as laid down in Algorithm 1.1, but they differ in their respective design of metaheuristic operators, that is, the heuristics.

Algorithm 1.1 Basic metaheuristic framework for optimization.

```
1:  procedure METAHEURISTICS($W_h, \epsilon$)
2:      Initialize $W_h^0$
3:      Fittest solution $\mathbf{w}^* = \min(W_h^0)$
4:      repeat
5:          $W_h^{t+1} := \mathrm{MH}_{\mathrm{Operator}}(W_h^t)$
6:          $\hat{\mathbf{w}} = \min(W_h^{t+1})$
7:          if $\hat{\mathbf{w}} < \mathbf{w}^*$ then
8:              $\mathbf{w}^* = \hat{\mathbf{w}}$
9:          end if
10:     until Stopping criteria $\epsilon$ satisfied
11:     return $\mathbf{w}^*$
12: end procedure
```

1.2.4 Multiobjective optimization

Multiobjective optimization procedure involved in optimizing two or more objective functions, simultaneously. It is an efficient method for evaluating Pareto-optimal solutions for the multiobjective problems. To generalize, the function/model (e.g., FNN or FIS) optimization need to be addressed from the multiobjective point of view because merely optimizing the training error can not provide generalization. Hence, a multiobjective optimization problem was studied in this research, which was of the following form:

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 the 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, that is, no solution can dominate any other solution. In other words, a nondominated solution is the 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 1.3 *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 1.4 *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.*

A multiobjective algorithm must provide a homogeneous distribution of a population along Pareto-front and improve solutions along successive generations [8]. Hence, three basic operators can be used [8]. 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 [9]. It should be noted that W_g is the population, $\mathbf{c}$ is the set/vector of objective functions, $W_A \subset W$ is the archive. So, a basic framework of multiobjective can be drawn as per Algorithm 1.2.

1.2.5 Ensemble learning

The generalized solution to a problem may be obtained by using ensemble, which allows us to combine or fuse two or more model's outputs into a single output (decision), i.e., a collective decision is given importance over the individual decision. To construct an

Algorithm 1.2 General multiobjective metaheuristics framework.

```
 1: procedure MULTIOBJECTIVE MH $(W_g, W_A, \mathbf{c}, \epsilon)$
 2:     Initialize $W_g^0$.
 3:     Evaluate $W_g^0$ over objectives $\mathbf{c}$.
 4:     Initialize archive $W_A$.
 5:     repeat
 6:         $W_g^{t+1} = \text{MO-MH}_{operator}(W_g^t)$.
 7:         Evaluate $W_g^{t+1}$ over objectives $\mathbf{c}$.
 8:         Update archive $W_A$.
 9:     until  Stopping criteria $\epsilon$satisfied
10:     $W_g^* = W_A$.
11:     return $W_g^*$.
12: end procedure
```

ensemble system, one must create M many diverse candidates, which may be created as one-by-one basis or may be taken from the final population of a metaheuristic optimization. Let the collection of M be a bag of ensemble candidates and size of bag be denoted as $|M|$ [10]. Then, the decisions for the classification problems may be combined by using weighted majority voting, and for the regression problems may be combined by using weighted arithmetic mean [11]. To compute the combined vote of the candidates $m_1, m_2, \ldots, m_{|M|}$ of the bag M, where each candidate m_i has the decision/class ω_1 or ω_2 or $\ldots$ or ω_C, an indicator function $\mathbb{I}(\cdot)$ can be used that returns 1 if "$(\cdot)$" is true or returns 0 if "$(\cdot)$" is false. Thus, the ensemble decision $\hat{y}$ using weighted majority voting is computed as:

$$\hat{y} = \arg\max_{j=1}^{C} \sum_{i=1}^{|M|} w_i \mathbb{I}\left(m_i = \omega_j\right), \tag{1.3}$$

where w_i is the weight associated with the i-th candidate in the bag M. Equation (1.3) says that the ensemble decision $\hat{y}$ is set to the class ω_j if the total weighted vote received by ω_j is higher than the total vote received by the other classes. Similarly, the ensemble decision for the regression problem is computed using weighted arithmetic mean as:

$$\hat{y} = \sum_{i=1}^{|M|} w_i \, m_i, \tag{1.4}$$

where w_i and m_j are the weight and the output of the i-th candidate bag M. The weights w_i in (1.3) and in (1.4) may be computed according to the fitness of the models, or may be computed by using metaheuristic algorithms.

1.3 Objectives

The following objectives were framed in this research for the purpose of feature selection and the function approximation:

1) To investigate available metaheuristics based design of FNN and FIS algorithms.
2) To investigate multiobjective optimization methods to adapt the topology and learning parameters, which could lead to a low approximation error and a less complex model.
3) To develop effective data mining algorithms based on the adaptive data structures for the feature selection and the function approximation.
4) To validate the developed algorithms on benchmark datasets and real-world problems.

1.4 Steps towards objectives

1.4.1 Study of FNN and FIS based learning methods

Development of novel algorithm for feature selection and function approximation was primary objective of this research. Hence, the available machine learning tools and techniques were investigated, first. For example, feature selection methods such as backward feature elimination, correlation-based feature selection, classifier-based feature selection, and wrapper feature selection were studied [12]. Similarly, function approximation methods such as linear regression, Gaussian process regression, multilayer perceptron, reduced error pruning tree, and sequential minimal optimization regression were studied. Additionally, ensemble methods such as evolutionary weighted ensemble, quality weighted output regression, mean output regression, random subspace, bagging, etc., were investigated [13, 14]. Moreover, neuro-fuzzy system and metaheuristic based type-1 and type-2 FIS design were explored [15–18]. Also, the comprehensive literature reviews, focused specifically on metaheuristic-based design and structure optimization for both FNN and FIS paradigms, were explored. The purpose of the literature review was to look for FNN and FIS based models to identify scope for the novel design of algorithms.

1.4.2 Proposal of heterogeneous flexible neural tree

The flexible neural tree (FNT) is an algorithm for the feature selection and function approximation proposed by Chen et al. [19, 20]. FNT perform feature selection and function approximation simultaneously. FNT has neural nodes (FNN-like nodes) and has feedforward connection. Since FNT design is a tree-like structure and evolves using evolutionary algorithms, its size (complexity) can be very large if the tree is created by minimizing only approximation error. Hence, a multiobjective optimization framework for FNTs tree structure optimization was proposed in this research. For this purpose, nondominated sorting genetic programming was applied for optimizing approximation error, model complexity, and models diversity, simultaneously. Additionally, adaptation in node was introduced to offer heterogeneity. Thus, the proposed algorithm was named heterogeneous FNT (HFNT). Finally, HFNT was evaluated on the benchmark problems and an industrial problem (a predictive modeling of pharmaceutical granules).

1.4.3 Proposal of hierarchical fuzzy inference tree

FIS is an efficient way to model inaccurate, imprecise, incomplete, and noisy data [21, 22]. Therefore, FIS was studied and the role of metaheuristic-based optimization of type-1 and type-2 FIS for solving data mining task was investigated. The basic need of FIS design is its optimization of rule base (which can be represented as a structure, i.e., a connection based model), and the optimization of the rule base parameters. In the past, neural network and hierarchical based FIS design were proposed by researchers. In this research, a tree-like structure was proposed for FIS design that was a hierarchical arrangement of low-dimensional FISs and resembled FNN-like connection. For this purpose, the node of the tree was designed as type-1 and type-2 FIS. Subsequently, multiobjective evolution algorithm was applied to create a hierarchical fuzzy inference tree (HFIT). The performance of the proposed HFIT was evaluated over six examples including a real-wolrd application (drug dissolution rate prediction).

1.5 Organization of the thesis

In this thesis, two novel algorithms HFIT and HFIT were proposed after the literature review of metaheuristic based design of FNN and FIS. Hence, first, in Chapter 2, state-of-the-art review of FNN optimization is discussed, which provides the ground for identifying the current challenges and future research direction. Subsequently, the proposed multiobjective HFNT is explained in Chapter 3. Chapter 4 provides a detailed state-of-the-art review of metaheuristic-based design of FIS, which is followed by a detail description of the proposed multiobjective HFIT algorithm in Chapter 5. Finally, Chapter 6 offers concise conclusions and future research directions derived from this research.

Chapter 2

Metaheuristic design of feedforward neural network

Feedforward neural network (FNN) is a widely used computational model for solving function approximation and pattern recognition problems, where optimization is the key to its success. FNN optimization is not limited to minimizing only its cost function; rather, improving FNNs generalization ability is its true optimization. To achieve generalization, researchers have significantly contributed to FNNs optimization that includes optimization of its weights, network architecture, activation nodes, learning parameters, learning environment, etc. Limitations of gradient-based FNN optimization approach, which is local minima problem, is to some extent solved by metaheuristics that are good at delivering solutions closer to a global optimum solution (if one exists). Metaheuristics facilitates to formulate all the FNN components for the optimization. Thus, it enhanced the chances of achieving generality. Metaheuristics addressed FNN optimization in several innovative ways such as EPNet, neuroevolution of augmenting topologies, flexible neural tree, cooperative coevolution neural network, multiobjective optimization to FNN, ensemble, etc. This chapter provides a comprehensive state-of-the-art study of FNN.

2.1 Introduction

History of artificial neural network (ANN) goes back to 1943 when McCulloch and Pitts [23] proposed a computational model that imitates the human brain. Feedforward neural networks (FNNs) are the special type of ANN models that are capable of learning and recognizing, and can solve a large range of complex problems: pattern recognition [24], clustering and classification [25], function approximation [26], prediction [27], forecasting [28], control [29], signal processing [30], speech processing [31], etc. In addition, the structural representation of FNN makes it appealing because it allows us to perceive a computational model (a function) in a structural/network form. Moreover, it is the structure of the FNN that makes it a universal function approximator. Thus, it has the capabilities of approximating any continuous function [32].

The structure of a FNN consists of several neurons (processing units) arranged in layer-by-layer basis, and the neurons in a layer have forward connections (weights) from the neurons at its previous layer. Fundamentally, FNN optimization/learning/training is met by searching an appropriate network structure (a function) and the weights (the parameters of the function) for solving a given problem [33]. Finding a suitable network structure consists of searching the appropriate neurons, the number of neurons, and arrangements of the neurons, etc. Finding the weights indicates the optimization a real vector that represents the weights of the network. Therefore, learning is an essential and distinguished aspect of FNN. The literature contains numerous algorithms, techniques, and procedure for a FNN optimization. Earlier in FNN research only gradient-based optimization techniques were popular choices; however, gradually due to the limitations of gradient-based algorithms, the necessity of metaheuristic based optimization techniques were recognized.

Metaheuristics formulates FNN components, such as weights, structure, nodes, etc., into an optimization problem. Metaheuristics uses various heuristics for finding a near-optimum solution. Additionally, multiobjective metaheuristic framework deal with multiple objectives simultaneously, and the existence of multiple objectives in FNN optimization is evident since the minimization of FNNs 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 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 chapter provides a comprehensive literature review to address various aspects of FNN optimization. Hence, the chapter is arranged into the following main parts.

1) Discussion on the importance of FNN as a function approximator and its preliminary concepts (Section 2.2):

    a) Introduction to the factors that influences the FNN optimization (Section 2.2.2).

    b) Introduction to the conventional FNN optimization algorithms (Section 2.2.3).

2) Discussion on the metaheuristic framework for optimizing FNN (Section 2.3):

  a) Significance of the metaheuristics methods for FNN optimization (Section 2.3.1).
  b) Role of hybrid metaheuristics in the FNN optimization (Section 2.3.2).

3) Discussion on the role of multiobjective metaheuristics in FNN optimization to achieve generalization (Section 2.4).

4) Discussion on the ensemble techniques (Section 2.5).

5) Discussion on the challenges and future research directions (Section 2.6).

2.2 Feedforward neural network

The intelligence of human brain is due to its massively parallel network system, in other words, the architecture of the brain. Similarly, a proper design of 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.

Single layer perceptron (SLP), consists of an input and an output layer, is the simplest form of ANN model [34]. However, SLPs are incapable of solving nonlinearly separable patterns [35]. Hence, a multilayer perceptron (MLP) was proposed in [36], which addressed the limitations of SLPs by including one or more hidden layers in between an input layer and an output layer. A trained MLP is then capable of solving nonlinearly separable patterns [37]. In-fact, MLPs (in general FNNs) are capable of addressing a large class of problem in the domain of pattern recognition and prediction. Moreover, MLP is considered as a *universal approximator* [32]. Cybenko [38] referring to Kolmogorov's theorem[1] showed that a 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 [32]. Hence, several researchers praised FNN for its universal approximation ability [40–43].

Other ANN models such as radial basis function [44] and support vector machine [45] 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 [46], Kohenen's self-organizing map [47], and learning-vector-quantization [47] are two-layer FNN architectures 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, in which connections between nodes may form cycle, is known as a recurrent neural network (RNN) or a feedback network model. RNNs are

[1]Kolmogorov's theorem: "All continuous functions of n variables have an exact representation in terms of finite superposition and compositions of a small number of functions of one variable [39]."

good at performing sequence recognition/reproduction or temporal association/prediction tasks. RNNs such as Hopfield network [48] and Boltzmann machine [49] are good at the application for memory storage and remembering input–output relations. Moreover, Hopfield network was designed for solving nonlinear dynamic systems, where stability of dynamic system are studied under the neurodynamic paradigm. A collection of RNN models such as temporal RNN [50], echo state RNN [51], liquid state machine [52] and backpropagation de-correlation [53] 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 FNNs were pointed-out, the discussion in this chapter is, however, limited to only FNNs.

2.2.1 Components of FNNs

Essentially, the FNN in Fig. 1.1(a) is a phenotype/structural representation of a function $f(\mathbf{x}, \mathbf{w})$, which is parameterized by p-dimensional input vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_p \rangle$ and 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 to a given problem. Therefore, two tasks involved in solving a problem using FNN are to discover appropriate function $f(\mathbf{x}, \mathbf{w})$ (i.e., architecture optimization) and to discover appropriate weight vector $\mathbf{w}$ (i.e., weight optimization) using some *learning algorithm*. Network architecture optimization includes the searching of appropriate activation functions $\varphi(\cdot)$ at the nodes, the number of nodes, number of layers, arrangements of the nodes, etc. Therefore, several components of 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 used to be the connection weights.

2.2.2 Factors influencing FNN optimization

Learning environment

A FNN is trained by supplying a training data (X, Y) as described in Section 1.1. Hence, the *supervised learning* of a FNN is met by the minimization/reduction of the error/distance function value, in an iterative manner. One very commonly known supervised learning algorithm is Delta rule or Widrow-Hoff rule [54], in which the n-dimensional weight vector $\mathbf{w}$ of a FNN is optimized as:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \Delta \mathbf{w}^t, \tag{2.1}$$

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:

$$\Delta \mathbf{w}_i^t = \eta^t e_i^t \mathbf{x}_i^t, \tag{2.2}$$

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 for i-th training input $\mathbf{x}_i^t$ presented to a FNN at the t-th iteration. 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 the FNNs output at t-th iteration.

Contrary to the mentioned supervised learning paradigm, there are two other learning forms for the spacial cases of FNNs: 1) the *unsupervised learning*—for the unlabeled training data [55]; and 2) the *reinforcement learning*—for the training data with insufficient input–output relations [56]. Focus of this chapter is, however, on supervised learning paradigms only.

Error function

Supervised learning, essentially, is the minimization the difference between the desired output $\mathbf{y}_i$ and the model's output $\hat{\mathbf{y}}_i = f(\mathbf{x}_i, \mathbf{w})$ by comparing the difference using a cost function $c_f : Y \times \hat{Y} \longrightarrow \mathbb{R}_{\geq 0}$ as described in Section 1.1.

Usually, in regression problems, the *mean squared error* (1.1) is used, or some other similar function such as sum of squared error, root of mean square error, mean absolute error, correlation coefficient, etc. [57] are used. However, the cost function *mean squared error* (1.1) or any similar squared error-based cost function is inconsistent for solving classification problems [58]. In-stead, the *percentage of good classification*, which has consistent behavior can be used [58], but, 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 cost functions. A detailed list of the cost function for classification problems is available in [59]. In this chapter, FNN optimization is discussed only in the context of the cost function (1.1).

Local minima problem

Let $c_f : S \longrightarrow \mathbb{R}_{\geq 0}$, where $S \subset \mathbb{R}^n$ is nonempty and compact [60]. Therefore, the following may be defined.

Definition 2.1 (Global minima) *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 2.2 (Local minima) *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 use the cost function (1.1) or any similar function for FNN optimization has the tendency to fall in local minima [61]. 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 (models with $h-1$ hidden units) can be a local or saddle point of a larger FNN model (models with h hidden units) [62].

However, there are some ways to avoid or eliminate local minima in FNN optimization [63, 64]. 1) If the weights and the training patterns are assigned randomly to a three-layer FNN that contains h many neurons at the hidden layer, then a gradient decent algorithm can avoid trapping into local minima [65]. 2) If linearly-separable training data and pyramidal network structure are taken, then the error surface will be local minima free [66]. 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.

However, these 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, these methods do not necessarily guarantee to converge to global minima, and the conditions such as a large number of hidden neurons and linearly separable training patterns are unlikely conditions for the real-world problems [67].

Generalization

The generalization is a crucial aspect of FNN optimization, which indicates the ability of a FNN to offer general solutions rather than performing best for a particular case. To achieve generalization, FNNs need to avoid both *underfitting* and *overfitting* during training, which is associated with high statistical *bias* and high statistical *variance* [68]. Therefore, one has to address trade-offs between bias and variance. In addition, for a good generalization, the number of training pattern should be sufficiently larger than the total number of connections in FNN [69].

The standard methods for 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 [70, 71], 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. Early stopping scheme suggests stopping of training at the point (epoch) from that onward the cost function value computed on cross-validation set starts to rise [61, 72–74]. Similarly, adding noise (jitters) into the training pattern improves FNNs *generalization* ability, and removing insignificant weights from a trained FNN improves its *fault tolerance* ability [75].

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 because the model with lower network complexity possesses higher generalization ability than the models with higher network complexity [76]. Moreover, network with lower weight magnitude possesses better generalization ability [76].

2.2.3 Conventional optimization approaches

Finding suitable algorithms for FNNs optimization has always been a difficult task. FNN optimization using conventional gradient-based algorithms is viewed as an unconstrained optimization problem [33]. The cost function c_f has to be optimized to satisfy Definition 2.1. Therefore, the gradient of error g^t at the t-th iteration is computed as:

$$g^t = \frac{\partial c_f}{\partial \mathbf{w}^t}, \tag{2.3}$$

where g^t is a *first-order partial derivative* of 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 an iteration t are updated as:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \Delta\mathbf{w}^t, \tag{2.4}$$

where weight-change $\Delta\mathbf{w}^t$ is equal to $-\eta^t g^t$ and η^t is the learning rate. Weights updated using (2.3) and (2.4) is known as the steepest decent approach.

Now, instead of using first-order partial derivative, *second-order partial derivative* (∇^2) of the cost function c_f can be used as:

$$H^t = \nabla^2 c_f = \frac{\partial^2 c_f}{\partial \mathbf{w}}, \tag{2.5}$$

where H^t is *Hessian* matrix at the t-th iteration. Hence, 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:

$$\Delta\mathbf{w}^t = -{H^t}^{-1} g^t, \tag{2.6}$$

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's method* or Newton update [33]. In the past, several algorithms were proposed using (2.3) and (2.6). Some of them are summarized in this chapter as follows.

The *backpropagation* (BP) algorithm is a first-order gradient descent technique used for optimizing FNN [37]. In BP training, error computed at the output layer is propagated backward to hidden layers. BP algorithm has two phases of computation: *forward computation* and *backward computation*, where at t-th iteration, the l-th layer weight-change $\Delta\mathbf{w}^t$ is computed as:

$$\Delta\mathbf{w}_l^t = \alpha^t\mathbf{w}_l^{t-1} + \eta^t g^t \mathbf{z}_{l-1}, \tag{2.7}$$

where $\mathbf{z}$ 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 a gradient descent technique. The momentum factor α^t let BP training to be biased with previous iteration weights that help convergence rate to be faster. BP is sensitive to these parameters [37]. If learning rate is too small, learning will become slow, and if 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 of minima, and a low momentum factor may avoid local minima, but learning will be slow. The classical BP algorithms are slow and have a tendency to fall in local minima [66].

Since the basic version of BP is sensitivity towards learning rate and momentum factor [37], several improvements suggested by researchers: 1) a fast BP algorithm called *Quickpro* was proposed in [77]; 2) a *delta-bar* technique and an *acceleration* technique was suggested for tuning BP learning rate η in [78] and in [79], respectively; and 3) a variant of BP, called resilient propagator (*Rprop*) was proposed in [80].

In Rprop, if the gradient direction in iteration t remains unchanged from its previous iteration $t-1$, then 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 [80]. However, it is not faster than Quickpro, but still faster than BP [81].

Contrary to BP, a second-order minimization method called *conjugate gradient* (CG) can be used for weights optimization [82]. 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, minimization performed in one step will not be undone by any further steps. Similar to CG, many other variants of derivative-based conventional methods are used for weights optimization: *Quasi Newtons* [83], *Gauss-Newton*, or *Levenberg-Marquardt* [84] are among them. Quasi Newtons uses a second-order partial derivative of error (2.5), and it computes its weight search direction by using Broyden-Fletcher-Goldfarb-Shanno method [85]. In Gauss-Newton method, FNN optimization is framed as a nonlinear least square optimization problem, which suggests to use sum of squared error (1.1) [84]. Many researchers suggested that Levenberg-Marquardt method outperform BP, CG, and Quasi-Newton method [86]. Several other methods were proposed for FNN are based on Kalman-filter [87, 88] and recursive least squares method [89].

However, these algorithms have the tendency to fall in local minima, and they are only used for optimizing weights. Hence, generalization may not be achieved by using only these methods. Therefore, metaheuristics have a role to play in FNN optimization.

2.3 Metaheuristic approach

So far, gradient-based optimization algorithms were discussed, which are local search algorithms. They are good at exploiting the obtained solutions to find new solutions, but 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 already discovered solution [2]. Exploitation and exploration are two contradictory strategies, and a good search algorithm must find a trade-off between these two. Metaheuristic is the procedure that uses nature-inspired heuristics to combine these two strategies [90]. Hence, metaheuristics are alternative to the conventional approaches for optimizing FNNs.

Unlike traditional methods, which require the objective function to be continuous and differential, metaheuristics have the ability to address complex, nonlinear, and non-differentiable problems. However, 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 theorem* (NFL) 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, they compared the sum over all cost function c_f of $P\left(c_t^y \mid c_f, t, a_1\right)$ to the sum over all cost function c_f of $P\left(c_t^y \mid c_f, t, a_2\right)$, and found that the performance $P\left(c_t^y \mid c_f, t, a_i\right)$ is independent of algorithm a_i when averaged over all cost functions, where

$P(c_f \mid c_f, t, a_i)$ was the probability of getting a sequence of cost value c_t^y of an algorithm a_i for t iterations over a cost function c_f.

Theorem 2.1 *For any pair of algorithms* a_1 *and* a_2

$$\sum_{c_f} P\left(c_t^y \mid c_f, t, a_1\right) = \sum_{c_f} P\left(c_t^y \mid c_f, t, a_2\right)$$

Theorem 2.1 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 is if it is specialized to the structure of the specific problem under consideration" [93]. This is the reason why in the past researchers were inclined to create and improvise algorithms for optimizing FNN.

2.3.1 Metaheuristic formulation of FNN components

Metaheuristics are stochastic/non-deterministic algorithms. Hence, they do not guarantee a global optimal solution. However, metaheuristics can offer a near-optimal solution. Moreover, metaheuristics efficiently solve a wide range of complex continuous optimization problems. Especially, when 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 (1.1) can be minimized. However, the goodness (performance) of FNN cost function depends not only on finding optimum weights, but finding optimum architecture, activation function, parameter setting of learning algorithm, and training environment are equally important. To apply metaheuristic algorithms for optimizing FNN, using some strategy, the FNN components (*phenotype*) need to be formulated into a vector (*genotype*) form.

Usually, the values of FNN components such as weights, architecture, activation function, learning rule, etc., are considered arbitrarily. Then, a learning algorithm is applied to search weights, but the values of other components are kept fixed to their initial setting. However, metaheuristics allow us sto optimize each component simultaneously or by a combination strategy (Fig. 2.1). Using a *Venn diagram*, Fig. 2.1 illustrates the spectrum metaheuristics optimization of FNN components: weights, architecture, activation function, and learning rule's parameters. In Fig. 2.1, area "a1" indicates optimization of weights; area "a2" indicates optimization of weights and architecture; area "a3" indicates optimization of weights, architecture, and activation function; and the areas "a4" to "a8" indicate all other possible combinations. Examining Fig. 2.1, 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.

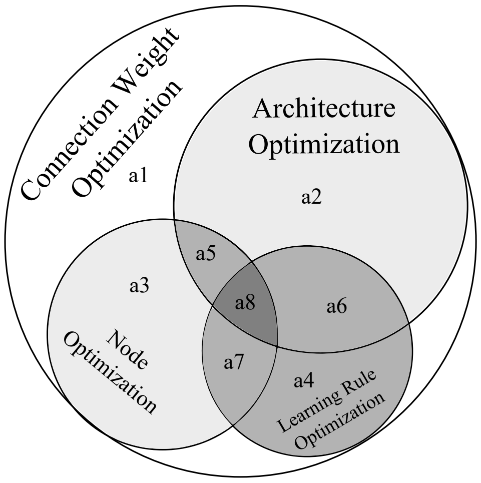

Connection Weight Optimization
Architecture Optimization
Node Optimization
Learning Rule Optimization
a1
a2
a3
a4
a5
a6
a7
a8

Fig. 2.1 Spectrum of metaheuristic design for FNN.

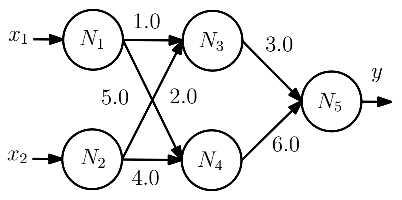
Each FNN component can be separately optimized as one-by-one basis. Therefore, first, weights can be optimized; second, architecture; third, activation function; and so on. Another way is to optimize all or a combination of FNN components, simultaneously. Therefore, weights and architecture; or weights and activation-functions; or weights, architecture, and activation functions; and so on, can be optimized, simultaneously. 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 in the literature can be applied on the designed vector to obtained an optimum FNN.

Now, two types of metaheuristics are available for FNN optimization [94]: *single-solution*-based and *population*-based metaheuristics. In single-solution-based metaheuristics, a genotype $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$ is used. Whereas, in a population-based metaheuristic, 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 are used.

Yao and Liu [95] identified evolution at various components of FNN that fall into the spectrum of metaheuristic design of FNN shown in Fig. 2.1. This section will describe *how researchers applied metaheuristics for evolving FNN*. The evolution (terms optimization, adaptation, and evolution are used in similar context) in FNN components is described here one-by-one, as follows.

Weight optimization

FNN weights optimization is the most common and widely studied approach, in which FNN weights are mapped onto an n-dimensional weight vector $\mathbf{w}$, where n is the total number of weights in a network. The vector $\mathbf{w} \in \mathbb{R}^n$ is a genotype representation of a phenotype (FNN structure). The individual weights $w_i \in \mathbf{w}$ may be encoded using real



(a)

$$c = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \mathbf{0} & 0 & 1 & 1 & 0 \\ 2 & 0 & \mathbf{0} & 1 & 1 & 0 \\ 3 & 0 & 0 & \mathbf{0} & 0 & 1 \\ 4 & 0 & 0 & 0 & \mathbf{0} & 1 \\ 5 & 0 & 0 & 0 & 0 & \mathbf{0} \end{array}$$

(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)

Fig. 2.2 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.

value, l-bits binary coding, l-bits Gray coding, 32-bit IEEE floating point coding, etc. Fig. 2.2 is an example of phenotype to genotype mapping, where a phenotype (Fig. 2.2(a)) that has the connectivity matrix c (Fig. 2.2(b)) is encoded into three different weight vectors (Fig. 2.2(c)).

FNN weights optimization using metaheuristic is practiced from early 80s, which is evident from Engle [96] work on FNN weight vector optimization using simulated annealing (SA) algorithm. SA simulates annealing process of metallurgy industry and uses Monte Carlo method [97] to generate states that led to an optimal solution [98]. To optimize weight vector using SA, first the phenotype was mapped onto a real-valued weight vector (Fig. 2.2(c)) and to compute fitness of FNN, a *reverse mapping* form genotype (weight vector **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 [99–101].

Similar to Engle [96] approach of phenotype to genotype mapping and vice versa, in [102], FNN weight optimization was performed using tabu search (TS), which is a metaheuristic inspired by the human behavior of tabooing objects [103]. In [104], 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 [105, 106]. However, SA and TS are single solution based algorithms, which has a limited scope of exploring search space to obtain global optimal solution. In contrast, evolutionary algorithms (EA) and bio-inspired or swarm-based metaheuristics are population-based algorithms that use multiple agents to explore a search space. Hence, they have better exploration ability than SA, TS, BP, CG, and other single solution based algorithms [7, 5].

EA framework offers an exploration of a vast search space and guarantees to find a near-optimal solution. Since EAs does not depend on gradient information, it solves a large range of complex, nonlinear, nondifferentiable, multimodal optimization problems. EAs are inspired by the "natural selection" and use metaheuristic operators such as *selection*, *crossover*, and *mutation* to find a near-optimal solution [7]. For optimizing weights, EA uses two categories of vector representation: binary-valued vector, and real-valued vector. In Fig. 2.2(c), three types of weight vector representation is illustrated: 1) real-valued coded chromosome; 2) binary coded chromosome; and 3) binary Gray coded chromosome.

EAs paradigm covers several algorithms: genetic algorithm (GA) [7]; genetic programming (GP) [107], evolutionary programming (EP) [108], evolutionary strategies (ES) [109], etc. GA uses genetic operators such as selection, crossover, and mutation to search optimum genetic vector from a search space; whereas, only mutation operator are used in ES to evolve a real vector solution. On the other hand, GP searches an optimum program structure from a topological search space of computer programs, and EP are used for evolving parameters of a computer program whose structure is kept fixed.

Goldberg and Holland [7] gave the idea of FNN training using GA. However, Whitley and Hanson [110] were the first to propose "GENITOR," a GA based FNN training procedure that used binary-coded chromosome (Fig. 2.2(c)) for optimizing weights. Many others followed the idea of GENITOR with some additional improvements such as connectivity optimization and reduced search space introduction [111, 112]. On the other hand, in [113], binary Gray coding (Fig. 2.2(c)) was used for optimizing weights, where at first, GA was used for finding initial weights that were further optimized by using BP and vice versa.

Binary bit-string representation of weights leads to a precision problem, that is, *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 real-coded chromosome Fig. 2.2(c) directly [114–118].

Traditional evolutionary algorithms operators are applied on the binary chromosome. Thus, operators such as bias-mutation, unbias-mutation, node-mutation, weight-crossover, and gradient-operator, etc., were defined for operating on real-valued chromosome [114]. On the other hand, a matrix-based representation of weights, where a column-wise and a row-wise crossover operators can also be defined [119]. Moreover, an evolutionary inspired algorithm called differential evolution (DE) [6] that imitates mutation and crossover operator to solve complex continuous optimization problems was found to be performing efficiently for real-valued weight vector optimization [120, 121]

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 simulate mutation and crossover operators for real-valued weight vector [5]. Swarm-based paradigm includes several algorithms that are inspired by the various forms of natural heuristics such as the foraging behavior of a swarm of animals, birds, insects, etc. Like EAs, swarm-based metaheuristics are population-based metaheuristics, which works on a collection of several potential solutions.

The particle swarm optimization (PSO) algorithm, which is inspired by the foraging behavior of swarm, is a population-based algorithm that explores a search space to optimize the objective function of given problems [5]. 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 [5]. Moreover, each particle bears two properties: location and velocity. 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 [5].

It was found that PSO guides a population of FNN weight vectors towards an optimum population [122, 123]. Hence, many researchers resorted to working on swarm based metaheuristics for FNN optimization. A *cooperative PSO*, which suggests to split solution vector into n parts, where each part are optimized by a swarm with m particles [124]. Thus, $n \times m$ combinations for constructing n-dimensional composite vector is allowed, in which, each swarm contributed to the fitness of a solution. Such cooperation between swarms led to a better performance than that of the basic version of PSO for optimizing FNNs. Similarly, a *multiphase PSO* was proposed, in which particle position was updated only when improvement in location was found; otherwise, location was copied as-it-is into the next generation [125]. A *cultural cooperative particle swarm optimization* (CCPSO) approach, in which a collection of multiple swarms that interact by exchanging information was proposed [126]. 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 [127]. Apart from PSO, there are numerous metaheuristic algorithms, among which some significant metaheuristics are discussed here that were applied for FNN optimization.

Ant colony optimization (ACO) is inspired by the foraging behavior of ants that is their (ant's) ability to explore the area around their nest (colony) for searching food source and their ability to choose the shortest path to food-source by communicating among each other's using pheromone secretion [128]. This behavior of ants led to the development of ACO algorithm [129]. The continuous version of ACO [130] was efficiently applied to optimize FNN weight vector [131, 132]. Artificial bee colony (ABC) is, on the other hand, a metaheuristic inspired by foraging behavior of honeybees, where three kinds of honeybees such as employed-bee, onlooker-bee and scout-bee are responsible for searching food-source [3].

Considering the efficiency of harmony search (HS) algorithm—that has a slow convergence rate, but guarantees a near-optimum solution [133]—many researchers applied HS for optimizing weight vector of FNN [134, 135]. Moreover, efficiency of HS comes from

$$S \rightarrow \begin{bmatrix} A & B \\ C & D \end{bmatrix}.$$

$$A \rightarrow \begin{bmatrix} z & u \\ z & z \end{bmatrix}, B \rightarrow \begin{bmatrix} z & z \\ c & z \end{bmatrix}, C \rightarrow \begin{bmatrix} z & z \\ z & z \end{bmatrix}, D \rightarrow \begin{bmatrix} z & z \\ z & z \end{bmatrix}.$$

$$u \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \; z \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, c \rightarrow \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix},$$

$$a \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \; i \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

(a)

$$S \rightarrow \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b)

(i) ⟨0110 110 01 1⟩

(ii) ⟨**0**0110 0**0**110 000**0**1 000**0**1 0000**0**⟩

(c)

Fig. 2.3 Mapping of phenotype (Fig. 2.2 to genotype (for architecture). (a) Indirect encoding schemes for architecture (Fig. 2.2(a)), where S is start symbol, A, B, C, and D are variables, and a, c, i, and u are terminals. (b) Complete connectivity derived from rules operation shown in Fig. 2.3(a). (c) Direct encoding to a vector of connectivity matrix (Fig. 2.2(b)): i) upper right triangle; ii) complete connectivity.

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 [133, 136].

In the past, many other swarm-based metaheuristics were used to optimize weight vector of FNN. In [137] a study elaborately explains the use of the following metaheuristics for optimizing FNN: SA, TS, GA, ACO, scatter search, greedy randomized search procedure, memetic algorithm, variable neighborhood search, and estimations of distribution algorithm. Many researchers studied the performance of metaheuristic algorithms for the training of FNN and reported that the metaheuristic approaches outperform all the conventional methods by a huge margin [138–140].

Architecture optimization

The basic architecture optimization approach is a *cascade correlation learning*, which iteratively add nodes to hidden layer to construct optimum architecture [77]. In other words, a *constructive* (add node iteratively) and *destructive* (prune nodes iteratively) method [141]. However, the constructive and the destructive methods for optimizing architecture are no different from manual *trial-and-error* method. Therefore, genetic representation of FNN architecture as mentioned in Figs. 2.3(a), 2.3(b), 2.3(c) can be used for architecture optimization, which is equivalent to searching optimum architecture from a compact space of FNN topology [142, 143].

Let us discuss the genetic representation of architecture in detail. A direct encoding scheme (Fig. 2.3(c)) was proposed in [144, 145], where authors used an adjacency matrix (Fig. 2.2(b)) to represent connections between nodes, where between any two node 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 network's structural information can be encoded into genotype, then it will avoid chromosome length problem [146], and the encoded network structural information can be accessed using rule-based recursive equation [147]. 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) [148].

Indirect 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. 2.3(a)) allow us to get access to complete structural information (Fig. 2.3(b)). Hence, a rule based encoding scheme allows a better FNN architecture optimization than a direct encoding scheme [149].

Unlike weight optimization that has only limited ways of genetic representation, FNN architecture optimization is an interesting area of research because of various ways to represent architecture into genotype. It is evident from a fractal configured FNN representation in [150], where authors define each node using parameters, namely, edge code, input coefficient, and output coefficient. Similarly, in [151], GA was applied to evolve each layer separately, and in [152], 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. 2.2(c)) and architecture vector $\mathbf{a} = \langle a_1, \ldots a_m \rangle$ (binary vector as Fig. 2.3(c)) into a combined genotype. Hence, a single solution vector of the form: $\langle w_1, \ldots, w_n, a_1, \ldots, a_m \rangle$ is obtained [153], which can be optimized by using metaheuristics.

Many researchers improvised the algorithms itself to optimize architecture. Such examples are as follows: in [154], 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) that was applied for optimizing the architecture of FNN by adding or deleting hidden node. Similarly, in [155], a hybrid Taguchi-genetic algorithm was proposed for optimizing FNN architecture and weights, where authors used a genetic representation of the weights, but they selected structure using constructive method (by adding hidden nodes one-by-one). A *multidimensional PSO* approach was proposed in [156] 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. Whereas, GP can optimize a phenotype itself, where genetic representation is not required [157]. Therefore, EP and GP can be directly applied to a population FNN architecture to evolve an optimum FNN architecture [115, 158, 159].

The design of FNN architecture is responsible for processing high-dimensional data. Hence, *deep network* paradigm offers a study of massive and deep structure of the neural network that can process complex problems related to speech processing, natural language processing, signal processing, etc. [160, 161]. Such a variant of FNN is *convolutional neural networks* (ConvNets), which is designed to process data that has multiple arrays such as a color image composed of three 2D arrays [160, 161]. ConvNets has three-dimensional arrangements of neural nodes. Hence, it efficiently receives 3D inputs and process them to produce 3D outputs.

Input layer optimization

Input layer optimization resembles feature reduction, which is traditional performed separately by dimensionality reduction methods [162]. 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 FNNs performance [163, 164]. 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 [163]. 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 FNN performance [165].

Binary PSO [166], which optimizes a discrete optimization problem was employed for selecting the input features which were binary coded [167, 168]. In [167], 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 a FNN in a hybrid manner [169].

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 [170], an adaptive selection of input examples was performed by employing genetic selection, where two point and one points crossover operations created new example patterns. For the crossover operations, the parents examples were drawn from the original input set. In addition, 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 [171], 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 [170, 171].

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 [172, 173]; 2) by optimizing the arguments of activation function [174]; and 3) by placing a complete model at the nodes of a network [175, 176].

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 [177]. In [172], evolution in FNN node were offered by selecting sigmoid and Gaussian function adaptively at the nodes [172]. In [178], an input dependent FNN that had a combined chromosome representation (Fig. 2.4) 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 [179], 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 FNNs by recombination and mutation operations and selects a 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 above-mentioned methods can improve FNN performance to some extent [180–182].

The third aspect of node optimization is to design a node as a model itself. Such modification lead to a variety of neural network paradigms such as *polynomial neural network* [175, 183, 184], where the nodes are designed as a polynomial function based on inputs to the nodes. Similarly, the nodes of a *GMDH neural network* is designed as an Ivakhnenko polynomial [185]; the nodes of a *complex value neural network* or *multivalued neural network* is designed with a complex value activation functions [176]; 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 [186]; the nodes of *fuzzy neural network* paradigm is designed using the concepts of fuzzy theory [187]; etc. In all such methods, metaheuristics have a significant role in the optimization, especially, the underlying parameters of the models.

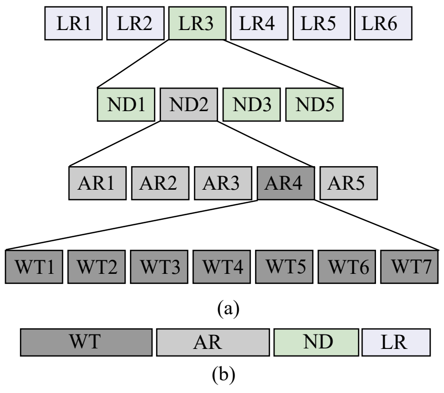


Fig. 2.4 Meta-learning scheme (a), where LR is learning parameter, ND is activation function, AR is architecture, and WT is weights [191]. Combined chromosome structure (b).

Learning algorithm optimization

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 the BP by applying some metaheuristics [113]. To optimize the parameters of a FNN learning algorithm, its parameters (e.g., BP parameters) and learning rules are encoded onto a genotype [146, 188]. However, formulating BP parameter such as learning rule, which is a dynamic concept, into a static chromosome is disadvantageous [189]. Hence, a genetic coding of learning rule for optimizing FNNs was used [189]. Moreover, assuming that each node in a network uses same learning rule, evolution in learning rule was proposed in [190], where weights optimization related to a particular node depended only on the input/output at that node.

Combination of FNN components optimization

Fig. 2.1 is an impressive representation of the most of FNN optimization combinations, where the genetic representation of the combination of FNN components can be represented as Fig. 2.4, which refers to a hierarchy of combination called meta-learning scheme. In meta-learning scheme, top down or bottom up optimization approach, which means, weights to learning rule and learning rules to weight optimization, can be used [191]. However, meta-learning resembles one-by-one learning mechanism. Hence, the advantageous approach is to represent each component of FNN side-by-side onto a genetic vector for optimization, which indicate, the confluence of all components of FNN as indicated by area “a8” in Fig. 2.1. Yao [192, 193] 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. Moreover, several 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 [194], which simultaneously optimized both architectures and weights. It was found that the simultaneous optimization of both weight and architecture lead to a better generalization [70, 142, 195]. Considering permutation[2] problem in a GA, EP-based mutation mechanism for evolving FNN architecture was proposed in [196] is known as EPNet.

A *neuroevolution of augmenting topologies* (NEAT) introduced in [197] was a GA-based evolution of a 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 sub population* approach was proposed in [198] 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 [199]. Additionally, GA-based and SA-based crossover operators were applied to generate an offspring (new individual subnetwork), and 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 [198]— was proposed in [200], 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

[2]Permutation problem occurs when using traditional crossover operator, where a population has traditional genetic representation of FNN architecture.

a population of hidden nodes was combined in a various permutation to form several complete networks, i.e., evolution in hidden nodes led to an evolution of the complete network [201].

A concept of *sparse neural trees*, in which GP for evolving network structure and GA for parameter optimization was proposed in [202]. 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 [19, 203]. FNT is a tree-like model where adaptation in all components of is equally important. Moreover, its components adaptation may take many forms (Fig. 2.1). 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 [204]. 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 [205] and in [206].

2.3.2 Hybrid metaheuristics for FNN optimization

An effective combination of various metaheuristic algorithms and conventional 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 [207].

The conventional algorithms have 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 [208, 209]. Therefore, a combination of these two may offer a better solution in FNN optimization than that of using any one of them alone (Fig. 2.5). To reach a global optimum, a hybrid strategy may be used. Fig. 2.5 shows an impact of hybridization of metaheuristics on FNN optimization, which 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 used by global search algorithms.

Under the definition of memetic algorithms, researchers combine metaheuristics with conventional algorithms [210]. For example, the effectiveness of global (GA) and local search (BP) combination is explained in [211, 212]. Similarly, a hybrid PSO and BP

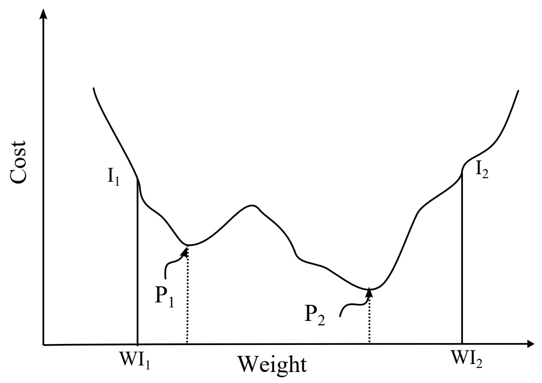


Fig. 2.5 Metaheuristic may be used for finding initial weights WI_2 and conventional algorithms may be for finding global optima P_2 and vice versa [192].

algorithms for optimizing FNN were found useful in obtaining better approximation than using one of the alone [123]. A convergence scenario similar to Fig. 2.5 was illustrated in [213], where ABC was applied for searching initial weights and the Levenberg-Marquardt algorithm was applied for optimizing the already discovered weights.

Hybridization of two or more global metaheuristics for FNN optimization are evident from the following examples. A hybrid GA and PSO approach for optimizing FNN was proposed in [214], where both GA and PSO were suggested to run over the same population—randomly generated population W of m individuals (the same individual was considered 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. Elite individuals were copied to next generation. Half of the copied elites were optimized using PSO and the remaining half using crossover operation and tournament selection as per GA. Similarly, a PSO and SA based hybrid algorithm for optimizing FNN were proposed in [215], where in each iteration, each PSO particle was governed by SA metropolis criteria [97] that determined global best particle for the PSO algorithm. There are Several other hybrid algorithm examples available in the literature: a hybrid PSO and GA [216]; hybrid GA and DE [217]; hybrid PSO and gravitational search algorithm [218]; and hybrid PSO and optimal foraging theory [219].

2.4 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 (1.1) or any equivalent function is the foremost necessity for the supervised 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 a 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 [220]. 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 (1.1) or equivalent cost function c_f is need to be optimized with one or more additional objectives to achieve generalization, which is why multiobjective framework for optimizing FNN are used. Let training error (1.1) be denoted by c_{trn} and an additional objective is denoted by c_{add}. Then, the generalized error c_{gen} may be computed by adding an objective to training error as:

$$c_{gen} = \lambda c_{trn} + (1 - \lambda) c_{add}, \tag{2.8}$$

where $0 \geq \lambda \leq 1$ 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 (2.8). The regularization term c_{reg} is the weight decay or norm of weight vector $\mathbf{w}$ as:

$$c_{reg} = \frac{1}{2} \sum_{i}^{n} w_i^2 = \frac{1}{2} \parallel \mathbf{w} \parallel^2 . \tag{2.9}$$

Similarly, a validation error c_{cv} is usually computed using (1.1) on a cross-validation data. On the other hand, the network complexity c_{net} is computed as:

$$c_{net} = \sum_{i}^{z} \sum_{j}^{z} c_{ij}, \tag{2.10}$$

where z is the number of nodes in the network c (Fig. 2.3(c)), or any user-defined function can be 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 (2.8) is a scalarized objective that has two disadvantages [221]. First, determining an appropriate hyperparameter λ that controls the contradicting objectives. Hence, the generalization ability of the produced model by using (2.8) will be a mystery. Second, the objective (2.8) 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 (2.8) need to be formulated into a multiobjective form: $\min\{c_{trn}, c_{reg}, c_{cv}, \ldots\}$, i.e., a multiobjective optimization need to be performed as described in Section 1.2.4.

Now, based on the above discussion on the cost functions and the generalization conditions, the multiobjective for FNN optimization can be categorized as *non Pareto-based multiobjective* approach and *Pareto-based multiobjective* approach.

2.4.1 Non-Pareto-based multiobjective approach

In nonPareto-based multiobjective approach, the objective functions are aggregated as mentioned in (2.8) or by some other similar means. For example, in [222], 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 [223] was applied to train FNN for each scalar objective $vc_i \in \mathbf{vc}$ sequentially. Similarly, in [224], 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 the 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 [225]. 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}. 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 [226]. 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 [227]. 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:

$$c_{net} = \alpha c_{trn} + \beta c_{cv} + \gamma c_{hid} + \delta c_{node} + \theta c_{fun}, \tag{2.11}$$

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

As mentioned above in Section 2.4, the aggregating objective function has disadvantages in obtaining the best generalized solution. It is evident from (2.11) that determining hyperparameters for controlling objective function is a challenging task. Therefore, Pareto-based multiobjective is an efficient choice for the multiobjective treatment of FNNs.

2.4.2 Pareto-based multiobjective approach

The advantages of applying Pareto-based learning is thoroughly explained and compared with single and scalerized objective in [228]. For example, a nondominated sorting genetic algorithm version II (NSGA-II) [229] 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 1.3. Also, NSGA-II can be applied to obtained a regularized network by optimizing the objectives c_{trn} and c_{reg} [230]. 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 [231, 232]. Simultaneous optimization of the objectives c_{trn} and c_{net} were also addressed using multiobjective PSO to generalize FNN performance [233].

For an image classification problem, authors in [234], 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 neuron) c_{net}. The proposed trade-offs between classification speed and classification accuracy were addressed using NSGA-II. Similarly, in [235], 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.

Further, the coevolution FNN concept [201, 236] was extended in [237] 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. Therefore, 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 the 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 [238], 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.

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 generalized solution and the residual error can still be remaining in solving many problems [239], then an ensemble of a set of FNNs is recommended.

2.5 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 diverse in the following sense: 1) parametric (each FNNs have different sets of weights); 2) 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 [240]. 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 FNNs to get l diverse solutions [61]. The decision of l 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 [11, 241]. 1) *Majority voting* method (for classification problems). 2) *Arithmetic mean* (for regression problem). 3) *Rank-based linear combination.* 4) *Recursive least square based linear combination* [242] (to minimize weighted least squares error so that redundant individuals are eliminated). 5) *Evolutionary weighted mean or majority voting* (metaheuristic to determine impact of a FNN in ensemble). 6) *Entropy-based method* for combining FNNs in ensemble (assigning entropy to FNNs during the learning process) [243].

Since population-based metaheuristics lead to an optimized final population, it is advantageous to use the final population for making ensemble [240]. However, there are two fundamental problems with it [244]: 1) determining ensemble size; and 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 [244]. 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 cooperation among each FNNs of a population [245]. 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 [246]. Additionally, a constructive-cooperative-neural-network-ensemble was proposed in [247] that determined ensemble size by focusing on accuracy and diversity during a constructive, cooperative procedure [248].

However, mere training fitness based selection of the 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 [249], which selects a subset of population, and determine the strength of selected candidates using GA. It was also shown that the ensemble of a subset of the population performed better than that of the whole population [249]. The effectiveness such GA-based selection was found efficient than the traditional ensemble methods: *bagging* [250] and *boosting* [251].

It is beneficial to partition/fracture training data and to allow different FNNs in the population to learn various parts of training data [250, 251]. An evidence of such was method was examined in [252, 253], where it was found that the ensemble of a small number of 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 [254] 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 [255] 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 different regions of input space) were created. Moreover, it reduced run time of learning process by coevolving (divide-and-conquer method) different FNNs using cooperation between FNNs. Such method improves diversity and accuracy of an ensemble system [255].

Both diversity and accuracy is a crucial aspect in construing ensemble of FNNs [11]. 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 [256]. For this purpose, multiobjective regularized negative correlation learning that maximized performance and maximized the negative correlation between individuals in population was found efficient [257].

2.6 Challenges and future scope

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 [258, 259]. Usually, *data cleaning* is a major step before modeling [260]. Therefore, training of the FNN remains always sensitive to the data-cleaning process and it posses significant challenge to adapt some mechanism in training process such that 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, imbalance, incomplete, high-dimensional, or abundant.

For 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 [261]. Second approach is exploit the dynamics of EAs in conjunction with FNNs to obtain new samples [170]. 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 is still with research community.

On the other hand, research in the area of imbalance dataset is continued to interest researcher [262].

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* [263]. 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 [264, 265]. High-dimensional data can also be managed/reduced by encoding or decoding methods and using FNNs training [266]. 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 (on-line learning) of FNN is the basic option [267]. However, it is still an open problem to design strategies for the dynamic training of FNN.

Moreover, present era, the fourth industrial revolution, is of *Internet of Things* (IoT) [268]. In IoT sophisticated technologies such as *smartphone* and *smartwear* provide several forms of data, e.g., *human activity recognition* [269]. 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* [270]. 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.

2.7 Summary

Feedforward neural network (FNN) is used for solving a wide range of real-world application problems, which is why researcher investigated many techniques/methods for optimizing and generalizing FNN. Specifically, metaheuristics allow 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. 2.1), 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 the 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 a FNN to evolve by simultaneously optimizing two or more FNN-related objectives: 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 aspects: accuracy and diversity of an ensemble were taken care during the FNNs training process.

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 chapter. Hence, that will overcome the data quality problem and will be handling new challenges of big data to cope-up with the new era of information processing.

Chapter 3

Multiobjective heterogeneous flexible neural trees

Machine learning algorithms are inherently multiobjective in nature, where approximation error minimization and model's complexity simplification are two conflicting objectives. A multiobjective genetic programming (MOGP) based framework was proposed for creating a heterogeneous flexible neural tree (HFNT$^{\mathrm{M}}$), which is a tree-like flexible FNN model. MOGP guided an initial HFNT$^{\mathrm{M}}$ population towards Pareto-optimal solutions, where the final population was used for making an ensemble system. A diversity index measure along with approximation error and complexity was introduced to maintain diversity among the candidates in the population. Hence, the ensemble was created by using accurate, structurally simple, and diverse candidates from MOGP final population. Differential evolution algorithm was applied to fine-tune the underlying parameters of selected candidates. A comprehensive test on benchmark and real-world (an industrial problem, where the data represented a dynamic environment of die filing process) datasets proved the efficiency of the proposed HFNT$^{\mathrm{M}}$ approach over other available prediction models. Moreover, heterogeneous creation of HFNT$^{\mathrm{M}}$ proved to be efficient in making ensemble system from the final population.

3.1 Introduction

Structure optimization of a FNN and its impact on FNNs generalization ability inspired the flexible neural tree (FNT) [19]. FNN components such as weights, structure, and activation function are the potential candidates for the optimization, which improves FNNs generalization ability to a high extent [196]. Moreover, the structure optimization held the responsible for generalization ability of a model. The significance of structure optimization is evident from the discussion provided in Chapter 2. FNT was an additional step into this series of efforts for the structure optimization [19]. Initially, FNT was evolved as a tree-like FNN model by using the probabilistic incremental program evolution (PIPE) [271]. The underlying parameter vector of the developed FNT (weights associated with the edges and arguments of the activation functions) was optimized by metaheuristic algorithms, which are nature-inspired parameter optimization algorithms [5]. Evolutionary process allowed FNT to select significant input features from an input feature set.

In the design of FNT, the non-leaf nodes are the computational node, which takes an activation function. Hence, rather than relying on a fixed activation function, if the selection of activation function at the computational nodes is allowed by evolutionary process itself. Then, it produces heterogeneous FNTs (HFNTs) with the heterogeneity in its structure, computational nodes, and input set, i.e., evolutionary process provides adaptation in structure, weights, activation functions, and input features. Therefore, an optimum HFNT is the one that offers the lowest approximation error with the simplest tree structure and the smallest input feature set. However, approximation error minimization and structure simplification are two conflicting objectives [228]. Hence, an evolutionary multiobjective approach may offer an optimal solution(s) by maintaining a balance between these objectives [272].

Moreover, in the proposed work, an evolutionary process guides a population of HFNTs towards Pareto-optimum solutions. Hence, the final population may contain several solutions that are close to the best solution. Therefore, an ensemble system was constructed by exploiting many candidates of the population (candidate, solution, and model are synonymous in this chapter). Such ensemble system takes advantage of many solutions including the best solution [240]. Diversity among the chosen candidates holds the key in making a good ensemble system [11]. Therefore, the solutions in a final population should fulfill the following objectives: low approximation error, structural simplicity, and high diversity. However, these objectives are conflicting to each others. A fast elitist nondominated sorting genetic algorithm (NSGA-II)-based multiobjective genetic programming (MOGP) was employed to guide a population of HFNTs [229]. The underlying parameters of selected models were further optimized by using differential evaluation (DE) algorithm [6]. Therefore, the key contributions of this chapter is as follows:

1) A heterogeneous flexible neural tree (HFNT) for function approximation and feature selection was proposed.

2) HFNT was studied under an NSGA-II-based multiobjective genetic programming framework. Hence, notation HFNT^{M} was used in this chapter.

3) Alongside approximation error and tree complexity, a diversity index was introduced to maintain diversity among the candidates in the population.

4) HFNT was found competitive with other algorithms when compared and cross-validated over classification, regression, and time-series datasets.

5) The proposed evolutionary weighted ensemble of HFNTs final population further improved its performance.

A detailed literature review provides an overview of FNT usage over the past few years (in Section 3.2). Conclusions derived from literature survey supports the HFNT$^{\text{M}}$ approach, where a Pareto-based multiobjective genetic programming was used for HFNT optimization (Section 3.3.1). Section 3.3.2 provides a detailed discussion on the basics of HFNT, MOGP for HFNT structure optimization and DE for HFNT parameter optimization. Efficiency of the above-mentioned hybrid and complex multiobjective HFNT algorithm (HFNT$^{\text{M}}$) was tested over various prediction problems using a comprehensive experimental set-up (Section 3.5). The experimental results (Section 3.6) supports the merits of proposed approach. Finally, a discussion of experimental outcomes and summary is provided in Section 3.8.

3.2 Background study and motivation

The literature survey describes the following points: basics of FNT, approaches to improve FNT, and its successful application to various real-life problems. Subsequently, the shortcomings of FNT are concluded.

FNT was first proposed by Chen et al. [19, 273], where a tree-like-structure was optimized by using PIPE. Then, its approximation ability was tested for time-series forecasting [20] and intrusion detection [274], where a variant of SA (called degraded ceiling) [275], and PSO [5], respectively, were used for FNT parameter optimization. Since FNT is capable of input feature selection, in [203], FNT was applied for selecting input features in several classification tasks, in which FNT structure was optimized by using GP [276], and the parameter optimization was accomplished by using memetic algorithm [277]. Additionally, they defined five different mutation operators, namely, changing one terminal node, all terminal nodes, growing a randomly selected sub-tree, pruning a randomly selected sub-tree, and pruning redundant terminals. Li et al. [278] proposed FNT-based construction of a decision trees whose nodes were conditionally replaced by neural node (activation node) to deal with continuous attributes when solving classification tasks. In other FNT based hybrid approaches, like in [279], GP was applied to evolve hierarchical radial-basis-function network model, and in [280] a multi-input-multi-output FNT model was evolved. Wu et al. [281] proposed to use grammar guided GP [282] for FNT structure optimization. Similarly, in [283], authors proposed to apply multi-expression programming (MEP) [284] for FNT tree-structure optimization and immune programming algorithm [285] for the parameter vector optimization. To improve classification accuracy of FNT, Yang et al. [286] proposed a hybridization of FNT with a further-division-of-partition-space method. In [287], authors illustrated crossover and mutation operators for evolving FNT using GP and optimized the tree parameters using PSO algorithm.

A model is considered efficient if it has generalization ability. It is known that a consensus decision is better than an individual decision. Hence, an ensemble of FNTs may lead to a better-generalized performance than a single FNT. To address this, in [288] author authors proposed to make an ensemble of FNTs to predict the chaotic behavior of stock market indices. Similarly, in [289, 290], the proposed FNTs ensemble predicted the breast cancer and network traffic better than individual FNT. In [291], protein dissolution prediction was easier using ensemble than the individual FNT.

In [292], authors proposed to use multi-agent system [293] based FNT (MAS-FNT) algorithm, which used GEP and PSO for the structure and parameter optimization, respectively. The MAS-FNT algorithm relied on the division of the main population into sub-population, where each sub-population offered local solutions and the best local solution was picked-up by analyzing tree complexity and accuracy.

Chen et al. [20, 203] referred the arbitrary choice of activation function at non-leaf nodes. However, they were restricted to use only Gaussian functions. A performance analysis of various activation function is available in [294]. Bouaziz et al. [295, 296] proposed to use beta-basis function at non-leaf nodes of an FNT. Since beta-basis function has several controlling parameters such as shape, size, and center, they claimed that the beta-basis function has advantages over other two parametric activation functions. Similarly, many other forms of neural tree formation such as balanced neural tree [297], generalized neural tree [298], and convex objective function neural tree [299], were focused on the improvement of neural nodes.

FNT was chosen over the conventional neural network based models for various real-world applications related to prediction modeling, pattern recognition, feature selection, etc. Some examples of such application are cement-decomposing-furnace production-process modeling [300], gene regulatory network reconstruction and time-series prediction from gene expression profiling [301], stock-index modeling [302], anomaly detection in peer-to-peer traffic [303], intrusion detection [304], gesture recognition [305], risk prediction in grid computing [306], etc.

The following conclusions can be drawn from the literature survey. First, FNT was successfully used in various real-world applications with better performance than other existing function approximation algorithms. However, it was mostly used in time-series analysis. Second, the lowest approximation error obtained by an individual FNT during an evolutionary phase was considered as the best structure that propagated to the parameter optimization phase. Hence, there was no consideration as far as structural simplicity and generalization ability are concerned. Third, the computational nodes of the FNT were fixed initially, and little efforts were made to allow for its automatic adaptation. Fourth, little attention was paid to the statistical validation of FNT model, e.g., mostly the single best model was presented as the experimental outcome. However, the evolutionary process and the meta-heuristics being stochastic in nature, statistical validation is inevitably crucial for performance comparisons. Finally, to create a generalized model, an ensemble of FNTs were used, but, FNTs were created separately for making the ensemble. Due to stochastic nature of the evolutionary process, FNT can be structurally distinct when created at different instances. Therefore, no explicit attention was paid to create diverse FNTs within population itself for making ensemble. In this chapter, a heterogeneous FNT, called HFNT was proposed to improve FNT model and its performance by addressing above mentioned shortcomings.

3.3 Multiobjective for flexible neural tree

In this section, first, the Pareto-based multiobjective is discussed. Second, a detailed discussion on FNT and its structure and parameter optimization using NSGA-II-based MOGP and DE, respectively is described. Followed by a discussion on making an evolutionary weighted ensemble of the candidates from the final population.

3.3.1 Pareto-based multiobjective

Usually, learning algorithms owns a single objective, i.e., the approximation error minimization, which is often achieved by minimizing mean squared error (MSE) on the learning data. Let denote MSE as E, which is computed on learning data as per (1.1). Additionally, a statistical goodness measure, called, correlation coefficient r that tells the relationship between two variables (i.e., between the desired output $\mathbf{y}$ and the model's output $\hat{\mathbf{y}}$) may be also used as an objective. Correlation coefficient r is computed as:

$$r = \frac{\sum_{i=1}^{N} (y_i - \bar{\mathbf{y}}) \left(\hat{y}_i - \bar{\hat{\mathbf{y}}}\right)}{\sqrt{\sum_{i=1}^{N} (y_i - \bar{\mathbf{y}})^2 \sum_{i=1}^{N} \left(\hat{y}_i - \bar{\hat{\mathbf{y}}}\right)^2}}, \tag{3.1}$$

where $\bar{\mathbf{y}}$ and $\bar{\hat{\mathbf{y}}}$ are means of the desired output $\mathbf{y}$ and model's output $\hat{\mathbf{y}}$, respectively.

However, single objective comes at the expense of model's complexity or generalization ability on unseen data, where generalization ability broadly depends on the model's complexity [220]. A common model complexity indicator is the number of free parameters in the model. The approximation error (1.1) and the number of free parameters minimization are two conflicting objectives. One approach is to combine these two objectives as:

$$c_{gen} = \lambda E + (1 - \lambda)k(\mathbf{w}), \tag{3.2}$$

where $0 \leq \lambda \leq 1$ is a hyperparameter, E is the MSE (1.1) and $k(\mathbf{w})$ is the total free parameter in a model. The scalarized objective c_{gen} in (3.2), however, has two disadvantages (see Section 2.4). Therefore, both objectives need to be optimized simultaneously. In this chapter, FNT optimization was formulated using the multiobjective optimization framework described in Section 1.2.4.

Algorithm 3.3 is a basic framework of NSGA-II based MOGP, which was used for computing Pareto-optimal solutions from an initial HFNT population. The individuals in MOGP were sorted according to their dominance in population. Moreover, individuals were sorted according to the rank/Pareto-front. MOGP is an elitist algorithm that allowed the best individuals to propagate into next generation. Diversity in the population was maintained by measuring crowding distance among the individuals [229].

Algorithm 3.3 MOGP: Multiobjective genetic programming.

```
 1: procedure MULTIOBJECTIVE GP($W_g, W_A, \mathbf{c}, \epsilon$)
 2:     Initialize $W_g^0$
 3:     Evaluation nondominated sorting of $W_g^0$
 4:     repeat
 5:         Selection: binary tournament selection
 6:         Generation: $W_g^c := \mathrm{GP}_{\mathrm{Operator}}(W_g^t)$
 7:         Recombination: $W_g^g = W_g^t + W_g^c$
 8:         Evaluation: nondominated sorting of $W_g^g$
 9:         Elitism: $W_g^{t+1} = |W_g^t|$ best individuals from $W_g^g$
10:     until Stopping criteria $\epsilon$ satisfied
11:     return a set of Pareto-optimal solutions
12: end procedure
```

3.3.2 Heterogeneous flexible neural tree

HFNT is analogous to a multilayer feedforward neural network that has over-layer connections and activation function at the nodes. HFNT construction has two phases [20]: 1) the *tree construction* phase, in which evolutionary algorithms are applied to construct tree-like structure; and 2) the *parameter-tuning* phase, in which genotype of HFNT (underlying parameters of tree-structure) is optimized by using parameter optimization algorithms.

To create a near-optimum model, phase one starts with random tree-like structures (population of initial solutions), where parameters of each tree are fixed by a random guess. Once a near-optimum tree structure is obtained, *parameter-tuning* phase optimizes its parameter. The phases are repeated until a satisfactory solution is obtained. Fig. 3.1 is a lucid illustration of these two phases. Moreover, evolutionary algorithm allowed HFNT to select activation functions and input feature at the nodes from sets of activation functions and input features, respectively. Thus, HFNT possesses automatic feature selection ability.

Encoding: basic terminology

An HFNT G is a collection of function set F_G and instruction set T_G:

$$G = F_G \cup T_G = \left\{+_2^k, +_3^k, \cdots, +_{tn}^k\right\} \cup \{x_1, x_2, \ldots, x_p\} \tag{3.3}$$

where $+_j^k\,(j = 2, 3, \ldots, tn)$ denotes a non-leaf instruction (a computational node). It receives $2 \leq j \leq tn$ arguments and takes an activation function k from a set of activation functions. Maximum arguments tn to a computational node are predefined. A set of seven activation functions is shown in Table 3.1. Leaf node's instruction $x_1, x_2, \ldots, x_p$ denotes input variables. Fig. 3.2 is an illustration of a typical HFNT. Similarly, Fig. 3.3 is an illustration of a typical node in an HFNT.

The i-th computational node (Fig. 3.3) of a tree (say i-th node in Fig. 3.2) receives n^i inputs (denoted as z_j^i) through n^i connection-weights (denoted as w_j^i) and takes two adjustable parameters a^i and b^i that represents the arguments of the activation function $\varphi_i^k(.)$ at that node. The purpose of using an activation function at a computational node is to limit the output of the computational node within a certain range. For example, if the i-th node contains a Gaussian function $k = 1$ (Table 3.1). Then, its output y_i is computed as:

$$y_i = \varphi_i^k(a_i, b_i, o_i) = \exp\left(-\left(\frac{o_i - a_i}{b_i}\right)\right) \tag{3.4}$$

where o_i is the weighted summation of the inputs z_j^i and weights w_j^i ($j = 1$ to n^i) at the i-th computational node (Fig. 3.3), also known as excitation of the node. The net excitation o^i of the i-th node is computed as:

$$o_i = \sum_{j=1}^{n^i} w_j^i z_j^i \tag{3.5}$$

where $z_j^i \in \{x_1, x_2, \ldots, x_p\}$ or, $z_j^i \in \{y_1, y_2, \ldots, y_m\}$, i.e., z_j^i can be either an input feature (leaf node value) or the output of another node (a computational node output) in the tree.

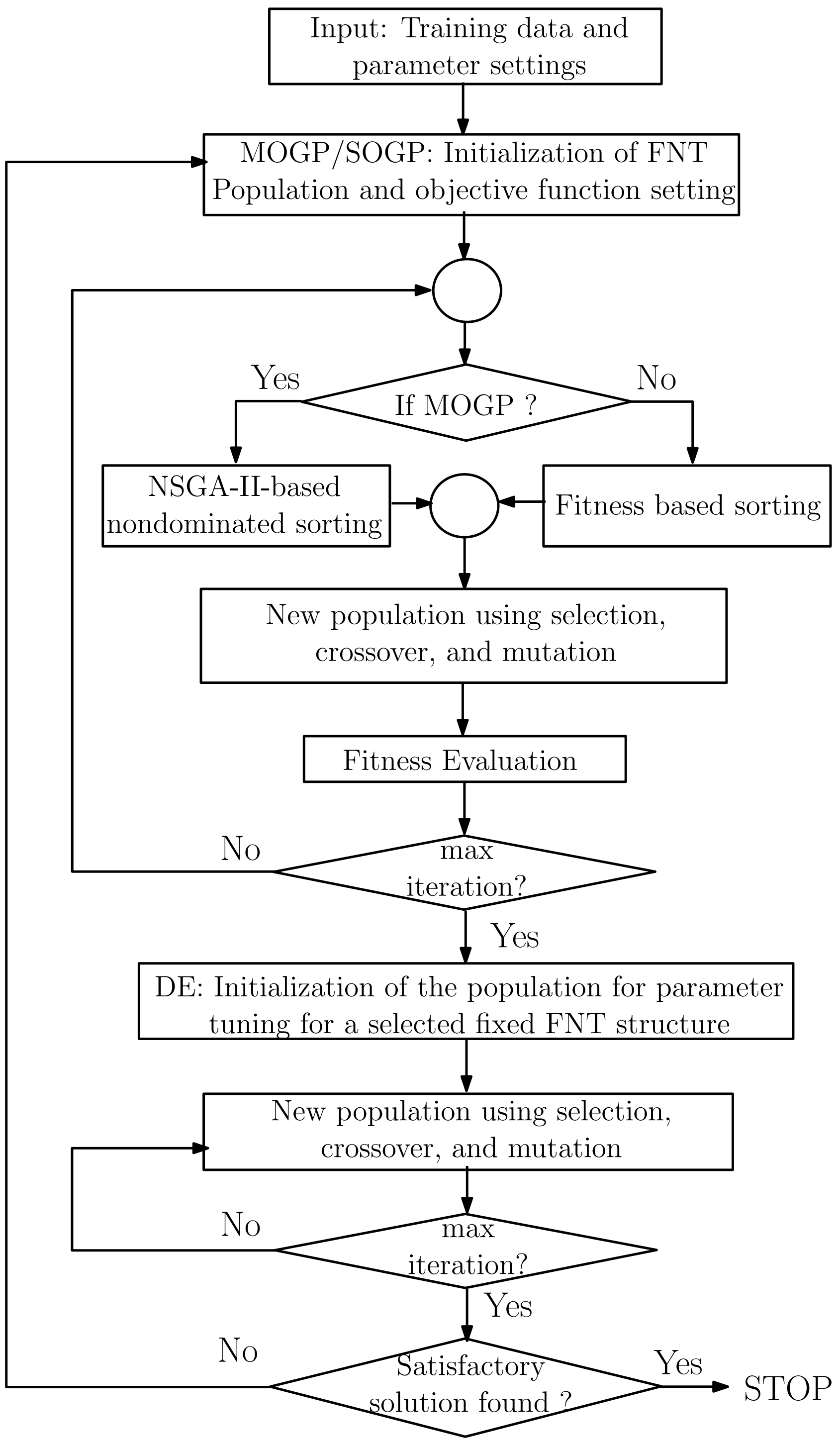

Input: Training data and parameter settings
MOGP/SOGP: Initialization of FNT Population and objective function setting
Yes
If MOGP ?
No
NSGA-II-based nondominated sorting
Fitness based sorting
New population using selection, crossover, and mutation
Fitness Evaluation
No
max iteration?
Yes
DE: Initialization of the population for parameter tuning for a selected fixed FNT structure
New population using selection, crossover, and mutation
No
max iteration?
Yes
No
Satisfactory solution found ?
Yes
STOP

Fig. 3.1 Coevolutionary construction of the heterogeneous flexible neural tree.

Table 3.1 Set of activation function used in neural tree construction

Activation-function	k	Expression for $\varphi_i^k(a,b,x)$
Gaussian Function	1	$\varphi(x,a,b) = \exp\left(-((x-a)^2)/(b^2)\right)$
Tangent-Hyperbolic	2	$\varphi(x) = (e^x - e^{-x})/(e^x + e^{-x})$
Fermi Function	3	$\varphi(x) = 1/(1+e^{-x})$
Linear Fermi	4	$\varphi(x,a,b) = a \times 1/((1+e^{-x})) + b$
Linear Tangent-hyperbolic	5	$\varphi(x,a,b) = a \times (e^x - e^{-x})/(e^x + e^{-x}) + b$
Bipolar Sigmoid	6	$\varphi(x,a) = (1-e^{-2xa})/(a(1+e^{-2xa}))$
Unipolar Sigmoid	7	$\varphi(x,a) = (2\|a\|)/(1+e^{-2\|a\|x})$

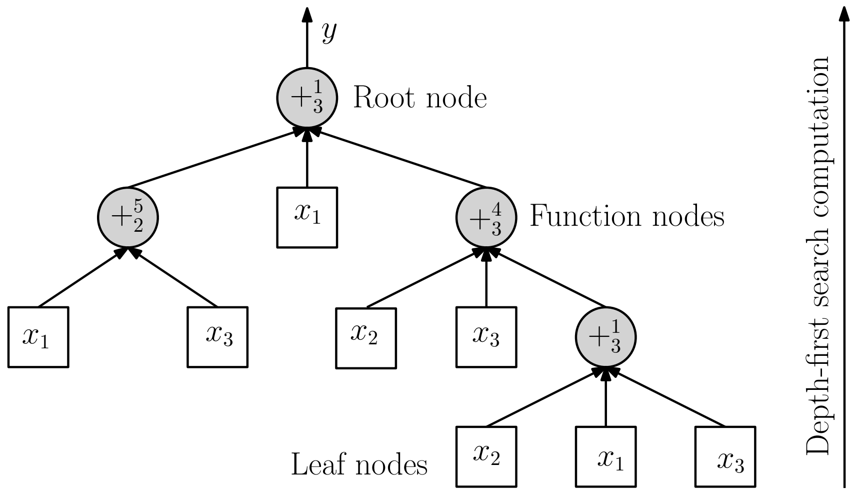


Fig. 3.2 Typical representation of a neural tree $G = F_G \cup T_G$ whose function instruction set $F_G = \left\{+_3^1, +_2^4, +_3^5\right\}$ and terminal instruction set $T_G = \{x_1, x_2, x_3, x_4\}$.

Weight w_j^i is the connection weight of real value in the range $[w_l, w_u]$. Similarly, the output of a tree y is computed from the root node of the tree, which is recursively computed by computing each node's output using (3.4) from right to left in a depth-first method.

The fitness of a tree depends on the problem. Usually, learning algorithm uses *approximation error*, i.e., MSE (1.1). Other fitness measures associated with the tree are *size* and *diversity index*. Tree size is the number of nodes (excluding root node) in a tree, e.g., the number of computational nodes and leaf nodes in the tree in Fig. 3.2 is 11 (three computational nodes and eight leaf-nodes). Hence, in this work, instead the number of free parameter count $k(\mathbf{w})$, the tree size was used as complexity indicator.

The number of distinct activation functions (including root node function) randomly selected from a set of activation functions gives the diversity index of a tree. Total activation functions (denoted as k in $+_j^k$) selected by the tree in Fig. 3.2 is three ($+_3^1, +_3^4,$ and $+_3^5$). Hence, its diversity index is three.

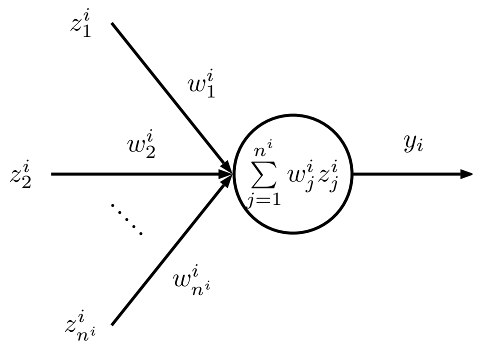


Fig. 3.3 Illustration of a computational node. The variable n^i indicates the number of inputs z_j^i and weights w_j^i received at the i-th node and the variable y^i is the output of the i-th node.

3.3.3 Near optimal tree: structure and parameter learning

A tree that offers the lowest approximation error and the simplest structure is a near optimal tree, which can be obtained by using an evolutionary algorithm such as GP [276], PIPE [271], GEP [307], MEP [284], and so on. To optimize tree parameters, algorithms such as genetic algorithm [109], evolution strategy [109], artificial bee colony [308], PSO [5], DE [6], and so on can be used.

Structure Optimization

Initial Population Two fitness measure was considered: *approximation error E* minimization and number of free parameter count $k(\mathbf{w})$ minimization or tree size minimization (in this chapter tree size minimization, instead $k(\mathbf{w})$, was considered). These two objectives cannot be achieved simultaneously. Hence, during the *structure-tuning* phase using Algorithm 3.3, an initial population W_g^0 of random tree was constructed and sorted according to non-dominance described in [229]. Various components of Algorithm 3.3 is as follows:

Selection In selection operation, a *mating pool* W_g^p of size $size(W_g^0)/2$ was obtained using *binary tournament selection* that selects two candidates randomly at a time from a population W_g^t and the best solution (according to its rank and crowding distance) is copied into the mating pool W_g^p. This process is continued until the mating pool becomes full.

Generation An offspring population W_g^c is generated by using the individuals of the mating pool W_g^p. Two distinct individuals (parents) are randomly selected from the mating pool to create new individuals using genetic operators crossover and mutation for offspring population W_g^c.

Crossover In crossover operation, randomly selected sub-trees of two parent trees are swapped. The swapping includes the exchange of nodes. A detailed description of the crossover operation in genetic programming is available in [287, 109]. The crossover operation is selected with a crossover probability *pc*.

Mutation The mutation operators used in tree are as follows [287, 109]:

a) Replacing a randomly selected terminal $x_i \in T$ with a newly generated terminal $x_j \in T$ for $j \neq i$.

b) Replacing all terminal nodes of a tree with a new set of terminal nodes derived from T.

c) Replacing a randomly selected node $N_i \in F$ with a newly generated node $N_j \in F$ for $j \neq i$.

d) Replacing a randomly selected terminal node $x_i \in T$ with a newly generated node $N_i \in F$.

e) Deleting a randomly selected terminal node $x_i \in T$ or deleting a randomly selected node $N_i \in F$.

The mutation operation is selected with a probability *pm* and the type of mutation operator (a, or b, or c, or d, or e) is selected randomly during the mutation operation.

Recombination The offspring population W_g^c and the main population W_g^t are combined together making a combined population W_g^g.

Elitism In this step, $size(W_g^c)$ worst individuals are weeded out from the combined population W_g^g. In other words, $size(W_g^t)$ best individuals are propagated to new generation $t+1$ as the main population W_g^{t+1}.

Parameter-tuning

In *parameter-tuning* phase, a single objective, i.e., approximation error was used in optimization of HFNT parameter by DE. The tree parameters such as weights of tree edges and arguments of activation functions were encoded into a vector $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$ for the optimization. In addition, a cross-validation (CV) phase was used for statistical validation of HFNTs.

For parameter tuning, DE version "DE/rand-to-best/1/bin" was used [6]. The basic principle of the DE is as follows. First, an initial population matrix $W_h^t = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$

at the iteration $t = 0$ is randomly initialized. The population W_h^t contains m solution vectors. A solution vector $\mathbf{w}$ in the population is an n-dimensional vector representing the free parameters of a model. Secondly, the population W_h^{t+1} is created using binomial trials. Hence, to create a new solution vector for the population W_h^{t+1}, three distinct solution vectors $\mathbf{w}^a$, $\mathbf{w}^b$, and $\mathbf{w}^c$ and the best solution vector $\mathbf{w}^g$ are selected from the population W_h^t. Then, for a random index $k \in [1, n]$ and for the selected trail vector $\mathbf{w}^a = \langle w_1^a, w_2^a, \ldots, w_n^a \rangle$, the j-th variable of the modified trial vector $\mathbf{w}^{a'}$ is created as:

$$w_j^{a'} = \begin{cases} w_j^a + \delta(w_j^g - w_j^a) + \delta(w_j^b - w_j^c) & \text{if } r_j < cr \parallel j = k \\ w_j^a & \text{if } r_j \geq cr \end{cases} \tag{3.6}$$

where $r_j \in [0, 1]$ is a uniform random sample, $cr \in [0, 1]$ is the crossover rate, and $\delta \in [0, 2]$ is the differential weight factor. Similarly, all the variables $j = 1$ to n of the trail vector $\mathbf{w}^{a'}$ is created using (3.6). After creation of the modified trail vector $\mathbf{w}^{a'}$, it is *recombined* as:

$$\mathbf{w}^a = \begin{cases} \mathbf{w}^{a'} & \text{if } c_f(\mathbf{w}^{a'}) < c_f(\mathbf{w}^a) \\ \mathbf{w}^a & \text{if } c_f(\mathbf{w}^{a'}) \geq c_f(\mathbf{w}^a) \end{cases} \tag{3.7}$$

where c_f is the function that returns the fitness of a solution vector using (1.1). The basic framework of the iterative learning of the DE follows Algorithm 1.1. As described in Algorithm 1.1, the application of the DE operators *selection*, *crossover*, and *recombination* are repeated until an optimal solution vector $\mathbf{w}^*$ is found.

3.3.4 Ensemble: making use of MOGP final population

In *tree construction* phase, MOGP provides a population from which tree-models for making ensemble can be selected. Three conflicting objectives such as approximation error, size, and diversity index allows the creation of Pareto-optimal solutions, where solutions are distributed on various Pareto-optimal fronts according to their rank in population. Ensemble candidates can be selected from the first line of solutions (Front 1), or they can be chosen by examining the three objectives depending on the user's need and preference. Accuracy and diversity among the ensemble candidate are important [11]. Hence, in this chapter, approximation error, and diversity among the candidates were given preference over size. Not to confuse "diversity index" with "diversity." The diversity index is an objective in MOGP, and the diversity is the number of distinct candidates in an ensemble.

A collection M of the diverse candidate is called a bag of candidates [10]. In this chapter, any two trees were considered diverse (distinct) if the followings hold: 1) two trees were of different size; 2) if number of function nodes/or leaf nodes in two trees were dissimilar; 3) if two models used different set of input features; and 4) if two models used different set of activation functions. Hence, diversity *div* of ensemble M (a bag of solutions) was computed as:

$$div = \frac{distinct(M)}{|M|}, \tag{3.8}$$

where $distinct(M)$ is a function that returns total distinct models in an ensemble M and $|M|$ is a total number of models in the bag.

Now, the decision of the candidates for a classification problem is combined using (1.3) and for a regression problem is combined using (1.4). A detailed discussion on ensemble learning is provided in Section 1.2.5. The weights (the significance of the candidates in the ensemble) associated with the candidates may be computed according to fitness of the models, or by using a metaheuristic algorithm. In this chapter, DE was applied to compute the ensemble weights, where population size was set to 100 and number of function evaluation was set to 300,000.

3.3.5 Multiobjective: a general optimization strategy

A summary of general HFNT learning algorithm is as follows:

Step 1. Initializing HFNT training parameters.

Step 2. Apply *tree construction* phase to guide initial HFNT population towards Pareto-optimal solutions.

Step 3. Select tree-model(s) from MOGP final population according to their approximation error, size, and diversity index from the Pareto-front.

Step 4. Apply *parameter-tuning* phase to optimize the selected tree-model(s).

Step 5. If no satisfactory solution found, then go to Step 2, else go to Step 6.

Step 6. Using a cross-validation (CV) method to validate the chosen model(s).

Step 7. Use the chosen tree-model(s) for making ensemble (Recommended).

Step 8. Compute ensemble results of the ensemble model (Recommended).

3.4 Input feature analysis

Feature analysis was conducted to understand the significance of the input features. For this purpose, at first, M many HFNT models were created. Second, two performance dimensions were used: feature selection rate R and predictability score P. Feature selection rate R describes the total number of times a particular input feature set was appeared in the list that was prepared out all M models. The feature selection rate is defined as:

$$R_j = \frac{1}{M}\sum_{i=1}^{M}\mathbb{I}(Z_j) \tag{3.9}$$

where R_j is the selection rate of j-th input feature set $Z_j \in \mathbf{Z}$, which is a power set $\mathbf{Z} = \mathcal{P}(Z) - \emptyset$ and $Z = \{z_1, z_2, \ldots, z_p\}$ is an input feature set; M is the total number of models in the list; and function $\mathbb{I}(Z_j)$ is an identity function that returned "1" if j-th input-feature set Z_j is selected by the i-th model, otherwise, returned "0." Feature selection rate R equal to one is the highest (100% selection rate) and R equal to zero is the lowest (0% selection rate). In other words, the value of selection rate R equal to one means an input feature set was selected by all the models in the prepared list, and the value of R equal to zero means an input feature set was selected by none of the models in the prepared list.

Since the models in the list may not be equal in their performances, the predictability score P based on the MSEs/RMSEs of the models was computed. The predictability score P describes the predictability of an input feature set. To compute predictability score P of j-th input-feature set Z_j, at first, the fitness F_j of the corresponding input-feature set Z_j was computed as:

$$F_j = \begin{cases} \sum_{i=1}^{M} E_i \cdot \mathbb{I}(Z_j), & \text{if } |Z_j| = 1 \\ \sum_{i=1}^{M} E_i \cdot \mathbb{I}(Z_j)/\sum_{i=1}^{M}\mathbb{I}(Z_j), & \text{if } |Z_j| > 1 \end{cases} \tag{3.10}$$

where E_i is the MSEs/RMSEs of i-th model. For $|Z_j|$ equal to one, fitness F_j is the sum of MSEs/RMSEs, and for $|Z_j|$ greater than one, fitness F_j is the average of MSEs/RMSEs of all models that selects subset Z_j. Then, the predictability score P_j corresponding to an input-feature set Z_j was computed by normalizing the fitness as:

$$P_j = \frac{F_j}{\max\limits_{j=1 \text{ to } |\mathbf{Z}^s|}(F_j)} \tag{3.11}$$

where function $\max(\cdot)$ determines the maximum fitness value from all F_j and $\mathbf{Z}^s$ is the set of selected feature sets. Similar to the selection rate R, the predictability score P equal to one indicates that the feature set has the highest impact on the predictability of the model and predictability score P equal to zero has the lowest.

Table 3.2 Multiobjective HFNT parameter set-up for the experiments

Parameter	Definition	Default rang	Value
Scaling	Input-features scaling range.	$[dl, du]$, $dl \in \mathbb{R}$, $du \in \mathbb{R}$	[0,1]
Tree height	Maximum depth (layers) of a tree model.	$\{td \in \mathbb{Z}^+ \| td > 1\}$	4
Tree arity	Maximum arguments of a node $+^k_{tn}$.	$\{tn \in \mathbb{Z}^+ \| n \geq 2\}$	5
Node range	Search space of functions arguments.	$[nl, nu]$, $nl \in \mathbb{R}$, $nu \in \mathbb{R}$	[0,1]
Edge range	Search space for edges (weights) of tree.	$[w_l, w_u]$, $w_l \in \mathbb{R}$, $w_u \in \mathbb{R}$	[-1,1]
W_g	MOGP population.	$size(W_g) > 20$	30
Mutation	Mutation probability	pm	0.3
Crossover	Crossover probability	$pc = 1 - pm$	0.7
Mating pool	Size of the pool of selected candidates.	$size(W_g)r$, $0 \leq r \leq 1$	0.5
Tournament	Tournament selection size.	$2 \leq bt \leq size(W_g)$	2
W_h	DE population.	$size(W_h) \geq 50$	50
General i_g	Maximum number of trails.	$\{i_g \in \mathbb{Z}^+ \| i_g > 1\}$	3
Structure i_s	MOGP iterations	$\{i_s \in \mathbb{Z}^+ \| i_s \geq 50\}$	30
Parameter i_p	DE iterations	$\{i_p \in \mathbb{Z}^+ \| i_p \geq 100\}$	1000

3.5 Experimental set-up

Several experiments were designed for evaluating the proposed HFNT$^{\mathrm{M}}$. A careful parameter-setting was used for testing its efficiency. A detailed description of the parameter-setting is given in Table 3.2, which includes: definitions, default range, and selected value. The phases of the algorithm were repeated until the stopping criteria met, i.e., either the lowest predefined approximation error was achieved, or the maximum function evaluations were reached. The repetition holds the key in obtaining a good solution. A carefully designed repetition of these two phases may offer a good solution in fewer of function evaluations.

In this experiment, three general repetitions i_g were used with 30 *tree construction* iterations i_s, and 1000 *parameter-tuning* iterations i_p (Fig. 3.1). Hence, the maximum function evaluation[1] $[size(W_g) + i_g\{i_s(size(W_g) + size(W_g)r) + i_p size(W_h)\}]$ was $154,080$. The DE version $DE/rand-to-best/1/bin$ [6] with cr equal to 0.9 and δ equal to 0.7 was used in the *parameter-tuning* phase.

The experiments were conducted over benchmark (problems of type classification, regression, and time-series) and real-world (a pharmaceutical problem) datasets. A detailed description of the chosen dataset from the UCI machine-learning [309] and KEEL [310] repository is available in Table 3.3. The parameter-setting mentioned in Table 3.2 was used for the experiments over each dataset. Since the stochastic algorithms depend on random initialization, a pseudorandom number generator called, Mersenne Twister algorithm that

[1]Initial GP population + three repetition ((GP population + mating pool size) $\times$ MOGP iterations + MH population $\times$ MH iterations) $= 30 + 3 \times ((30 + 15) \times 30 + 50 \times 1000) = 154,080$.

Table 3.3 Collected datasets for testing HFNT$^{\mathrm{M}}$

Index	Name	Features	Samples	Output	Type
AUS	Australia	14	691	2	Classification
HRT	Heart	13	270	2	
ION	Ionshpere	33	351	2	
PIM	Pima	8	768	2	
WDB	Wdbc	30	569	2	
ABL	Abalone	8	4177	1	Regression
BAS	Baseball	16	337	1	
DEE	DEE	6	365	1	
EVL	Elevators	18	16599	1	
FRD	Fridman	5	1200	1	
MGS	Mackey-Glass	4	1000	1	Time-series
WWR	Waste Water	4	475	1	

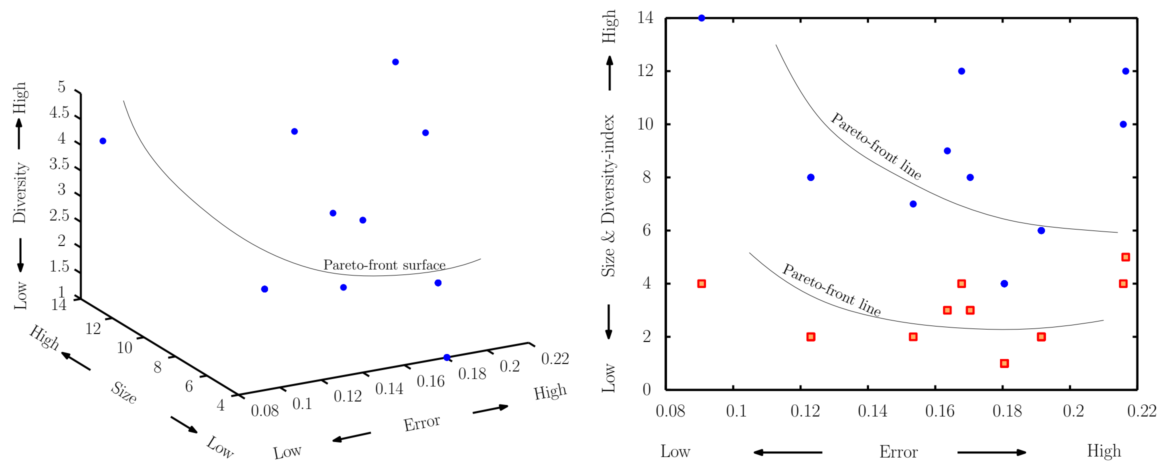
draws random values using probability distribution in a pseudo-random manner was used for initialization of HFNTs [311]. Hence, each run of the experiment was conducted with a random seed drawn from the system. HFNT$^{\mathrm{M}}$ performance with various other approximation models from collected literature were compared. A list of such models is provided in Table 3.4. A developed software tool based on the proposed HFNT$^{\mathrm{M}}$ algorithm for predictive modeling is available at [http://dap.vsb.cz/aat/].

To construct good ensemble systems, highly diverse and accurate candidates were selected in the ensemble bag M. To increase diversity (3.8) among the candidates, the Pareto-optimal solutions were examined by giving preference to the candidates with low approximation error, small size and distinct from others selected candidates. Hence, $|M|$ candidates were selected from a population W_g. An illustration of such selection method is shown in Fig. 3.4, which represents an MOGP final population of 50 candidate solutions computed over dataset MGS.

MOGP simultaneously optimized three objectives. Hence, the solutions were arranged on the three-dimensional map (Fig. 3.4(a)), in which along x-axis, error was plotted; along y-axis, size was plotted; and along z-axis, diversity index (diversity) was plotted. However, for the simplicity, solutions were also arranged in 2-D plots (Fig. 3.4(b)), in which along x-axis computed error was plotted and along y-axis size (indicated by blue dots) and diversity index (indicated by red squares) were plotted. From Fig. 3.4(b), it is evident that a clear choice is difficult since decreasing approximation error increases models size (blue dots in Fig. 3.4(b)). Similarly, decreasing approximation error increases models size and diversity (red squares in Fig. 3.4(b)). Hence, solutions along the Pareto-front (rank-1), i.e., Pareto surface indicated in the 3-D map of the solutions in Fig. 3.4(a) were chosen for the

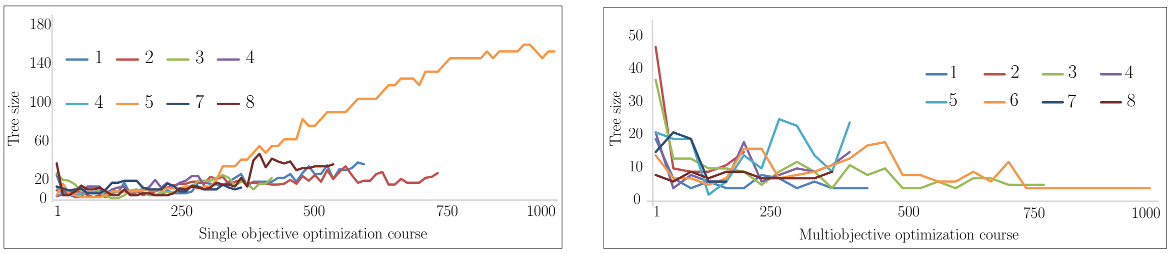
Table 3.4 Algorithms from literature for the comparative study with HFNT^{M}

Ref.	Algorithms	Definition
[33]	MLP	Multi-layer Perceptron
[312]	HDT	Hybrid Decision Tree
[289]	FNT	Flexible Neural Tree
[313]	ANFIS-SUB	Adaptive Neuro-FIS using Subtractive Clustering
[314]	TSK-IRL	Genetic Learning of TSK-Iterative Rule Learning
[315]	LINEAR-LMS	Least Mean Squares Linear Regression
[316]	LEL-TSK	Local Evolutionary Learning of TSK-rules
[317]	RBF	Classical Radial Basis Function
[124]	CPSO	Cooperative Particle Swarm Optimization (PSO)
[318]	PSO-BBFN	PSO-based Beta Basis Function Neural Network
[319]	G-BBFNN	GA-based BBFNN
[320]	HCMSPSO	Hierarchical Cluster-Based Multispecies PSO
[321]	FWNN-M	Fuzzy Wavelet Neural Network Models
[322]	HMDDE-BBFNN	Hierarchical Multidimensional DE-Based BBFNN
[323]	LNF	Local Least-Squares SVM-Based Neuro-Fuzzy Mode
[324]	BPNN	Back-propagation Neural Network
[325]	EFuNNs	Evolving Fuzzy Neural Networks
[326]	FBBFNT-EGP&PSO	Flexible BBFNN-trained by Immune Programming and PSO
[327]	METSK-HD^{e}	Multiobjective Evolutionary Learning of TSK-rules for High-Dimensional Problems



(a) Error versus size versus diversity index (b) Error versus size and diversity index

Fig. 3.4 Pareto-front of a final population of 50 individuals generated on the training dataset of time-series problem MGS. (a) 3-D plot of solutions and a Pareto-front is a surface. (b) 2-D plot of Error versus complexity (in blue dots) and Error versus diversity (in red squares)



(a) Single objective optimization (b) Multiobjective objective optimization

Fig. 3.5 Comparison of single and multiobjective optimization course.

ensemble. For all datasets, ensemble candidates were selected by examining Pareto-fronts in a similar fashion as described for the dataset MGS in Fig. 3.4.

The purpose of the experiment was to obtain sufficiently good prediction models by enhancing predictability and lowering complexity. MOGP for optimization of HFNTs was used. Hence, fitness was compromised by lowering models complexity. In single objective optimization, models fitness is the only objective to be optimized. Therefore, one does not have control over model's complexity. Fig. 3.5 illustrates eight runs of both single and multiobjective optimization course of HFNT, where models size (complexity) is indicated along y-axis and x-axis indicates fitness value of the HFNT models. The results shown in Fig. 3.5 was conducted over MGS dataset. For each single objective GP and multiobjective GP, optimization course was noted, i.e., successive fitness reduction and tree size were noted for 1000 iterations.

It is evident from Fig. 3.5 that the HFNT$^{\mathrm{M}}$ approach leads HFNT optimization by lowering model's complexity. Whereas, in the single objective, model's complexity was unbounded, and was abruptly increased. The average tree size of eight runs of single and eight runs of multiobjective were 39.265 and 10.25, respectively; whereas, the average fitness were 0.1423 and 0.1393, respectively. However, in single objective optimization, given the fact that the tree size is unbounded, the fitness of a model may improve at the expense of model's complexity. Hence, the experiments were set-up for multiobjective optimization that provides a balance between both objectives as described in Fig. 3.4.

3.6 Performance HFNT on benchmark datasets

Experimental results for benchmark datastes were classified into three categories: classification, regression, and time-series. Each category has two parts: 1) First part describes the best and average results obtained from the experiments; and 2) Second part describes ensemble results using tabular and graphical form.

3.6.1 Classification datasets

Five classification datasets were chosen for evaluating HFNT$^{\mathrm{M}}$, and subsequently, the classification accuracy was computed as:

$$c_{acc} = \frac{tp + tn}{tp + fn + fp + tn}, \tag{3.12}$$

where tp is the total positive samples correctly classified as positive samples, tn is the total negative samples correctly classified as negative samples, fp is the total negative samples incorrectly classified as positive samples, and fn is the total positive samples incorrectly classified as negative samples. Here, for a binary class classification problem, the positive sample indicates the class labeled with '1' and negative sample indicates class labeled with '0'. Similarly, for a classification problem of three classes: $\omega_1, \omega_2,$ and ω_3, the samples which are labeled as a class ω_1 are set to 1, 0, 0, i.e., set to positive for class ω_1 and negative for $\omega_2,$ and ω_3, the samples which are labeled as a class ω_2 are set to 0, 1, 0, and the samples which are labeled as a class ω_3 are set to 0, 0, 1.

10-FCV

The experiments on classification dataset were conducted in three batches that produced 30 models, and each model was cross validated using 10-FCV, in which a dataset is equally divided into 10 sets and the training of a model was repeated 10 times. Each time a distinct set was picked for the testing the models and the rest nine set was picked for the training of the model. Accordingly, the obtained results are summarized in Table 3.5. Each batch of the experiment produced an ensemble system of 10 models whose results are shown in Table 3.7.

The obtained results presented in Table 3.5 describes the best and mean results of 30 models. In Table 3.6, a comparative study of the best 10-FCV models of HFNT$^{\mathrm{M}}$ and the results of the literature models are presented. In Table 3.6, the results of HDT and FNT [278] were of 10-FCV results on the test dataset. Whereas, the results of FNT [289] was the best test accuracy and not the CV results. The results summarized in Table 3.6 suggests a comparatively better performance of the proposed HFNT$^{\mathrm{M}}$ over the previous approaches. For the illustration of a model created by HFNT$^{\mathrm{M}}$ approach, the best model of dataset WDB that has a test accuracy of 97.02% (shown in Table 3.5) was chosen. A pictorial representation of the WDB model is shown in Fig. 3.6, where the models size is 7, total input features are 5, (x_3, x_4, x_{12}, x_{17}, and x_{22}) and the selected activation function is tangent hyperbolic ($k = 2$) at both the non-leaf nodes.

Table 3.5 Best and mean results of 30 10-FCV models (300 runs) of HFNT$^{\mathrm{M}}$.

	Best of 30 models				Mean of 30 models			
Data	train c_{acc}	test c_{acc}	*size*	Features	train c_{acc}	test c_{acc}	avg. *size*	*diversity*
AUS	87.41%	87.39%	4	3	86.59%	85.73%	5.07	0.73
HRT	87.41%	87.04%	8	5	82.40%	80.28%	7.50	0.70
ION	90.92%	90.29%	5	3	87.54%	86.14%	6.70	0.83
PIM	78.67%	78.03%	10	5	71.12%	70.30%	6.33	8.67
WDB	97.02%	96.96%	6	5	94.51%	93.67%	7.97	0.73

Table 3.6 Comparative results: 10-FCV test accuracy c_{acc} and variance σ of algorithms.

	AUS		HRT		ION		PIM		WDB	
Algorithms	test c_{acc}	σ	test c_{acc}	σ	test c_{acc}	σ	test c_{acc}	σ	test c_{acc}	σ
HDT [278]	86.96%	2.058	76.86%	2.086	89.65%	1.624	73.95%	2.374		
FNT [278]	83.88%	4.083	83.82%	3.934	88.03%	0.953	77.05%	2.747		
FNT [289]									93.66%	n/a
HFNT$^{\mathbf{M}}$	87.39%	0.029	87.04%	0.053	90.29%	0.044	78.03%	0.013	96.96%	0.005

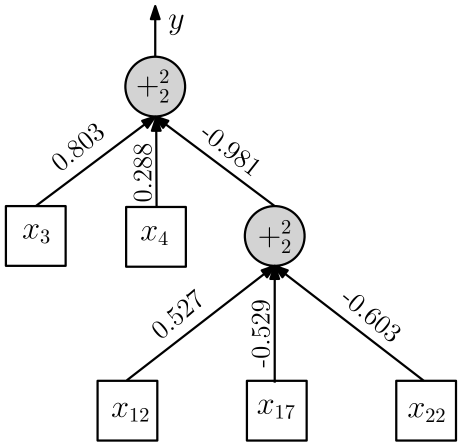


Fig. 3.6 HFNT model of classification dataset WDB (test c_{acc} = 97.02%).

Table 3.7 Ensemble results (10-FCV) of each classification dataset.

Data	Batch	test c_{acc}	avg. *size*	*div* (3.8)	TSF	MSF	MIF
AUS	1	**86.96%**	5	0.7	4	x_6, x_8, x_{10}, x_{12}	x_1, x_2, x_3, x_{11}, x_{14}
	2	85.51%	6	0.7	5		
	3	86.81%	4.2	0.8	5		
HRT	1	77.41%	6.8	0.5	6	x_3, x_4, x_{12}, x_{13}	x_6
	2	70.37%	7.6	0.6	9		
	3	**87.04%**	8.1	1	10		
ION	1	82.86%	7.2	0.9	15	x_2, x_4, x_5, x_{27}	x_{15}, x_{16}, x_{18}, x_{19}, x_{21}, x_{23}, x_{25}, x_{30}, x_{32}
	2	**90.29%**	7.3	1	16		
	3	86.57%	5.6	0.6	6		
PIM	1	**76.32%**	6.9	1	8	x_1, x_3, x_4, x_5, x_6, x_7	x_2
	2	64.74%	5.6	0.7	7		
	3	64.21%	7.4	0.9	8		
WDB	1	94.29%	8.2	0.7	15	x_{21}, x_{22}, x_{24}, x_{25}	x_1, x_5, x_6, x_8, x_{14}, x_{20}, x_{30}
	2	93.75%	5	1	15		
	3	**94.29%**	10.7	0.5	15		

Ensembles

The best accuracy and the average accuracy of 30 models presented in Table 3.5 are the evidence of HFNT$^{\mathrm{M}}$ efficiency. However, as mentioned earlier, a generalized solution may be obtained by using an ensemble. All 30 models were created in three batches. Hence, three ensemble systems were obtained. The results of those ensemble systems are presented in Table 3.7, where ensemble results are the accuracies c_{acc} obtained by weighted majority voting (1.3). In Table 3.7, the classification accuracies c_{acc} were computed over CV test dataset. From Table 3.7, it may be observed that high diversity among the ensemble candidates offered comparatively higher accuracy. Hence, an ensemble model may be adopted by examining the performance of an ensemble system, i.e., average size (complexity) of the candidates within the ensemble and the selected input features.

An ensemble system created from a genetic evolution and adaptation is crucial for feature selection and analysis. Summarized ensemble results in Table 3.7 gives the following useful information about the HFNT$^{\mathrm{M}}$ feature selection ability: 1) TSF—total selected features; 2) MSF—most significant (frequently selected) features; and 3) MIF—most infrequently selected features. Table 3.7 illustrates feature selection results.

3.6.2 Regression datasets

5-FCV

For regression dataset, the performance of HFNT$^{\mathrm{M}}$ was examined by using 5-FCV method, in which the dataset was divided into 5 sets, each was 20% in size, and the process was repeated five times. Each time, four set was used to training and one set for testing. Hence, a total 5 runs were used for each model. As described in [327], MSE $E = 0.5 \times E$ was used for evaluating HFNT$^{\mathrm{M}}$, where E was computed as per (1.1). The training MSE is represented as E_n and test MSE is represented as E_t. Such setting of MSE computation and cross-validation was taken for comparing the results collected from [327]. Table 3.8 presents results of 5-FCV of each dataset for 30 models. Hence, each presented result is averaged over a total 150 runs of experiments. Similarly, in Table 3.9, a comparison between HFNT$^{\mathrm{M}}$ and other collected algorithms from literature is shown. It is evident from comparative results that HFNT$^{\mathrm{M}}$ perform very competitive to other algorithms. The literature results were averaged over 30 runs of experiments; whereas, HFNT$^{\mathrm{M}}$ results were averaged of 150 runs of experiments. Hence, a competitive result of HFNT$^{\mathrm{M}}$ is evidence of its efficiency.

Moreover, HFNT$^{\mathrm{M}}$ is distinct from the other algorithm mentioned in Table 3.9 because it performs feature selection and models complexity minimization, simultaneously. On the other hand, the other algorithms used entire available features. Therefore, the result's comparisons were limited to assessing average MSE, where HFNT$^{\mathrm{M}}$, which gives simple models in comparison to others, stands firmly competitive with the others. An illustration of the best model of regression dataset DEE is provided in Fig. 3.7, where the model offered a test MSE E_t of 0.077, size equal to 10, and four selected input features (x_1, x_3, x_4, and x_5). The selected activation functions were unipolar sigmoid ($+^7_2$), bipolar sigmoid ($+^6_2$), tangent hyperbolic ($+^2_2$), and Gaussian ($+^1_2$). Note that while creating HFNT models, the datasets were normalized as described in Table 3.2 and the output of models were denormalized accordingly. Therefore, normalized inputs should be presented to the tree (Fig. 3.7), and the output y of the tree (Fig. 3.7) should be denormalized.

Ensembles

For each dataset, five ensemble systems were constructed by using 10 models in each batch. In each batch, 10 models were created and cross-validated using five cross two fold cross-validation (5x2-FCV), in which dataset is randomly divided into two equal sets: A and B. Such partition of the dataset was repeated five times and each time when the set A was presented for training, the set B was presented for testing, and vice versa. Hence, total 10 runs of experiments for each model was performed. The collected ensemble results

Table 3.8 Best and mean results of 30 5-FCV models (150 runs) of HFNT$^{\mathrm{M}}$.

	Best of 30 models					Mean of 30 models			
Data	train E_n	test E_t	Size	Features		train E_n	test E_t	*size*	*diversity*
ABL	2.228	2.256	14	5		2.578	2.511	11.23	0.7
BAS	198250	209582	11	5		261811	288688.6	7.69	0.6
DEE	0.076	0.077	10	4		0.0807	0.086	11.7	0.7
ELV[1]	8.33		8.36	11	7	1.35	1.35	7.63	0.5
FRD	2.342	2.425	6	5		3.218	3.293	6.98	0.34

Note: [1]Results of ELV should be multiplied with 10^{-5}, E_n and E_n are MSEs

Table 3.9 Comparative results: 5-FCV training MSE E_n and test MSE E_t of algorithms.

	ABL		BAS		DEE		ELV[1]		FRD	
Algorithms	E_n	E_t	E_n	E_t	E_n	E_t	E_n	E_t	E_n	E_t
MLP	-	2.694	-	540302	-	0.101	-	2.04		3.194
ANFIS-SUB	2.008	2.733	119561	1089824	3087	2083	61.417	61.35	0.085	3.158
TSK-IRL	2.581	2.642			0.545	882.016			0.433	1.419
LINEAR-LMS	2.413	2.472	224684	**269123**	0.081	0.085	4.254	4.288	3.612	3.653
LEL-TSK	2.04	2.412	9607	461402	0.662	0.682			0.322	**1.07**
METSK-HD^e	2.205	**2.392**	47900	368820	0.03	0.103	6.75	7.02	1.075	1.887
HFNT$^{\mathrm{M}}$ [2]	2.578	2.511	261811	2.88688.6	0.0807	**0.086**	1.35	**1.35**	3.218	3.293

Note: [1]ELV results should be multiplied with 10^{-5}, [2]HFNT$^{\mathrm{M}}$ results were averaged over 150 runs compared to MLP, ANFIS-SUB, TSK-IRL, LINEAR-LMS, LEL-TSK, and METSK-HD^e, which were averaged over 30 runs.

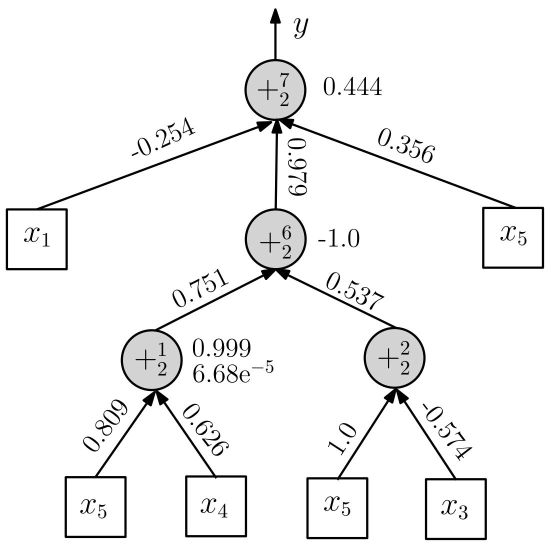


Fig. 3.7 HFNT model of regression dataset DEE (test MSE $E_t = 0.077$).

are presented in Table 3.10, where ensemble outputs were obtained by using weighted arithmetic mean as mentioned in (1.4).

The weights of models were computed by using DE algorithm, where the parameter setting was similar to the one mentioned in classification dataset. Ensemble results shown in Table 3.10 are MSE, and correlation coefficient computed on CV test dataset. From ensemble results, it can be said that the ensemble with higher diversity offered better results than the ensemble with lower diversity. The models of the ensemble were examined to evaluate MSF and MIF presented in Table 3.10. A graphical illustration of ensemble results is shown in Fig. 3.8 using scattered (regression) plots, where a scatter plots show how much one variable is affected by another (here, it is the desired output and the model's output). Moreover, it tells the relationship between two variables, i.e., their correlation. Plots shown in Fig. 3.8 represents the best ensemble batch (numbers indicated bold in Table 3.10) four, five, three, four and five where MSEs are 2.2938, 270706, 0.1085, $1.10E{-}05$ and 2.3956, respectively. The values of R^2 in the plots tell about the regression curve fitting over CV test datasets. In other words, it can be said that the ensemble models were obtained with generalization ability.

3.6.3 Time-series datasets

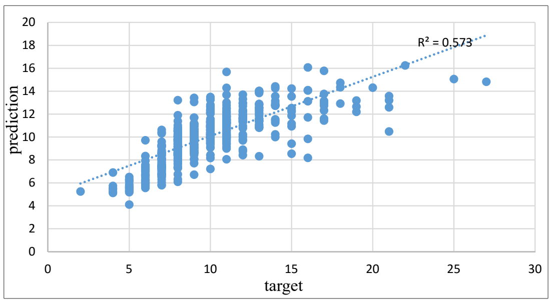
2-FCV

In literature survey, it was found that efficiency of most of the FNT-based models was evaluated over time-series dataset. Mostly, Macky-Glass (MGS) dataset was used for this purpose. However, only the best-obtained results were reported. For time-series prediction problems, the performances were computed using the root of mean squared error

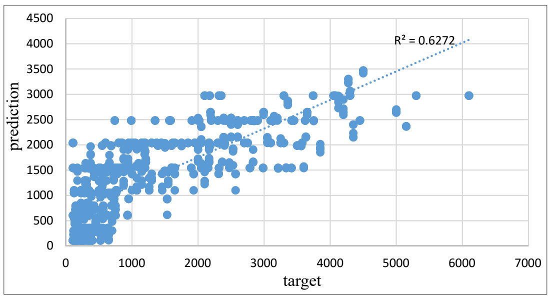
Table 3.10 Ensemble test MSE E_t computed for 5x2-FCV of 10 model in each batch.

Data	batch	MSE E_t	r_t	avg. *size*	*div* (3.8)	TSF	MSF	MIF
ABL	1	3.004	0.65	5	0.1	3	x_2, x_3, x_5, x_6	x_1
	2	2.537	0.72	8.3	1	7		
	3	3.042	0.65	8.5	0.5	5		
	4	2.294	**0.75**	10.7	1	7		
	5	2.412	0.73	11.2	0.7	7		
BAS[1]	1	2.932	0.79	5.6	0.3	5	x_3, x_7, x_8, x_9, x_{11}, x_{13}	x_1, x_2, x_5, x_6, x_{10}
	2	3.275	0.76	8.2	0.3	6		
	3	3.178	0.77	5	0.2	7		
	4	3.051	0.78	5.7	0.3	5		
	5	2.707	**0.81**	7.3	0.7	9		
DEE	1	0.112	0.88	4.3	0.2	4	x_1, x_3, x_4, x_5, x_6	x_2
	2	0.115	0.88	8.9	0.6	6		
	3	0.108	**0.88**	5.4	0.5	3		
	4	0.123	0.87	10.8	0.9	5		
	5	0.111	0.88	5.2	0.6	4		
EVL[2]	1	1.126	0.71	9.3	0.1	12	x_1, x_3, x_4, x_6, x_{17}	x_7, x_8, x_{12}
	2	1.265	0.67	9.6	0.1	12		
	3	1.124	0.71	10.4	0.1	15		
	4	1.097	**0.72**	9.2	0.2	10		
	5	2.047	0.31	3.8	0.4	3		
FRD	1	3.987	0.86	6.2	0.2	4	x_1, x_2, x_4, x_5	x_3
	2	4.154	0.83	8	0.2	4		
	3	4.306	0.83	5.2	0.4	5		
	4	3.809	0.86	7.8	0.5	4		
	5	2.395	**0.91**	7.7	0.4	5		

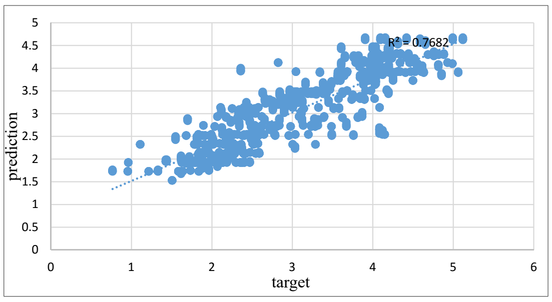
Note: [1]BAS results should be multiplied with 10^5, [2]ELV results should be multiplied with 10^{-5}.



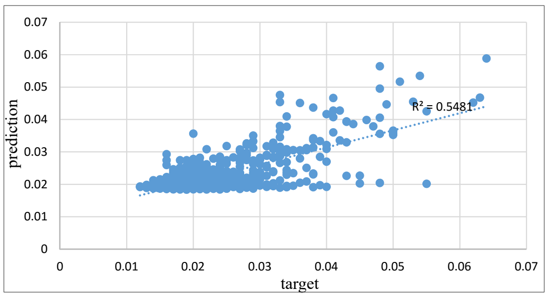
(a) Dataset ABL. $r_t = 0.75$



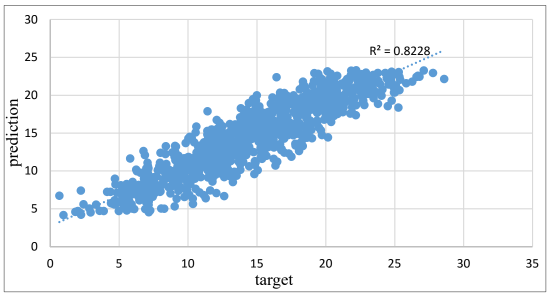
(b) Dataset BAS. $r_t = 0.81$



(c) Dataset DEE. $r_t = 0.88$



(d) Dataset EVL. $r_t = 0.72$



(e) Dataset FRD. $r_t = 0.91$

Fig. 3.8 Regression plots of the best ensemble batches on datasets R1, R2, R3, R4, and R5.

Table 3.11 Best and mean results 2-FCV training RMSE E_n and test RMSE E_t.

	Best of 70 models				Mean of 70 models		
Data	E_n	E_t	*size*	Features	E_n	E_t	*size*
MGS	0.00859	0.00798	21	4	0.10385	0.10568	8.15
WWR	0.06437	0.06349	17	4	0.10246	0.09778	8.05

(RMSE) indecated by E, and E was computed by taking square root of (1.1). Additionally, correlation coefficient (3.1) was also used for evaluating algorithms performance. For the experiments, first 50% of the dataset was taken for training and the rest of 50% was used for testing. Table 3.11 describes the results obtained by HFNT$^{\mathrm{M}}$, where E_n is RMSE for training set and E_t is RMSE for test-set. The best test RMSE obtained by HFNT$^{\mathrm{M}}$ was $E_t = 0.00859$ and $E_t = 0.06349$ on datasets MGS and WWR, respectively.

HFNT$^{\mathrm{M}}$ results are competitive with most of the algorithms listed in Table 3.12. Only a few algorithms such as LNF and FWNN-M reported better results than the one obtained by HFNT$^{\mathrm{M}}$. FNT based algorithms such as FNT [20] and FBBFNT-EGP&PSO reported RMSEs close to the results obtained by HFNT$^{\mathrm{M}}$. The average RMSEs and its variance over test-set of 70 models were 0.10568 and 0.00283, and 0.097783 and 0.00015 on dataset MGS and WWR, respectively. The low variance indicates that most models were able to produce results around the average RMSE value. The results reported by other function approximation algorithms (Table 3.11) were merely the best RMSEs. Hence, the robustness of other reported algorithm cannot be compared with the HFNT$^{\mathrm{M}}$. However, the advantage of using HFNT$^{\mathrm{M}}$ over other algorithms is evident from the fact that the average complexity of the predictive models were 8.15 and 8.05 for datasets MGS and WWR, respectively. The best model obtained for dataset WWR is shown in Fig. 3.9, where the size of the tree is equal to 17 and following are the selected activation functions: tangent hyperbolic, Gaussian, unipolar sigmoid, bipolar sigmoid and linear tangent hyperbolic. The selected input features in the tree (Fig. 3.9) are x_1, x_2, x_3 and x_4.

Ensembles

The ensemble results of time-series datasets are presented in Table 3.13, where the best ensemble system of dataset MGS (marked bold in Table 3.13) offered a test RMSE $E_t = 0.018151$ with a test correlation coefficient $r_t = 0.99$. Similarly, the best ensemble system of dataset WWR (marked bold in Table 3.13) offered a test RMSE $E_t = 0.063286$ with a test correlation coefficient $r_t = 0.953$. However, apart from the best results, most of the ensemble produced low RMSEs, i.e., high correlation coefficients. The best ensemble batches (marked bold in Table 3.13) of dataset MGS and WWR were used for graphical plots in Fig. 3.10. A one-to-one fitting of target and prediction values is the evidence

Table 3.12 Comparative results: training RMSE E_n and test RMSE E_t for 2-FCV.

	MGS		WWR	
Algorithms	E_n	E_t	E_n	E_t
CPSO	0.0199	0.0322		
PSO-BBFN	-	0.027		
HCMSPSO	0.0095	0.0208		
HMDDE-BBFNN	0.0094	0.017		
G-BBFNN	-	0.013		
Classical RBF	0.0096	0.0114		
FNT [20]	0.0071	0.0069		
FBBFNT-EGP&PSO	0.0053	0.0054		
FWNN-M	0.0013	0.00114		
LNF	0.0007	0.00079		
BPNN	-	-	-	0.200
EFuNNs	-	-	0.1063	0.0824
HFNT$^{\mathbf{M}}$	0.00859	0.00798	0.064377	**0.063489**

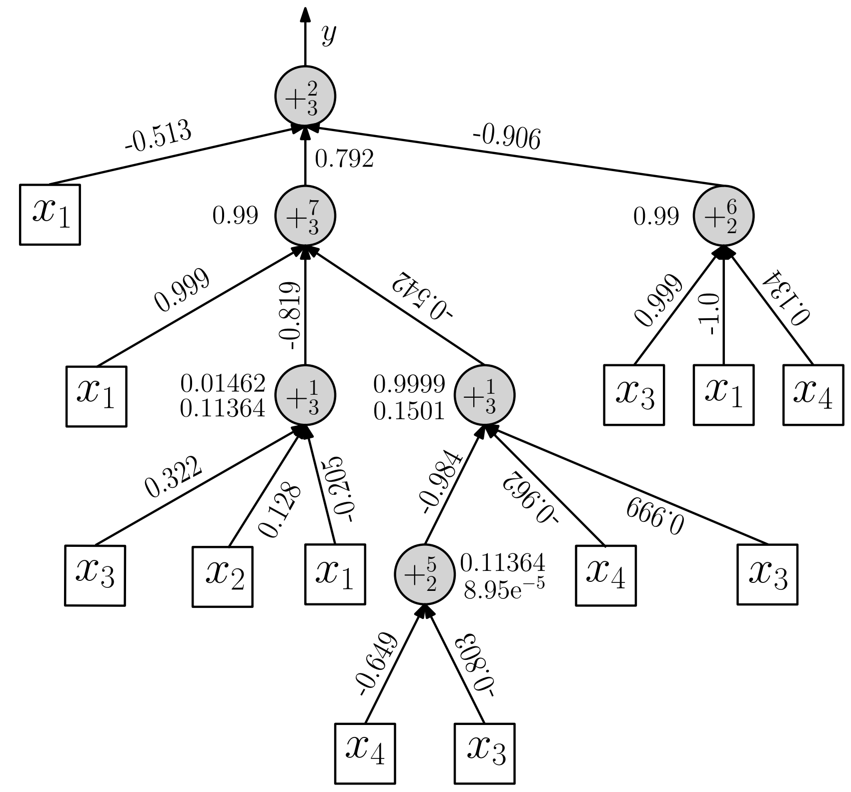
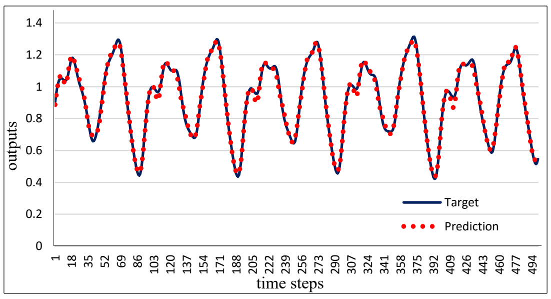


Fig. 3.9 HFNT model of time-series dataset WWR (RMSE = 0.063489).

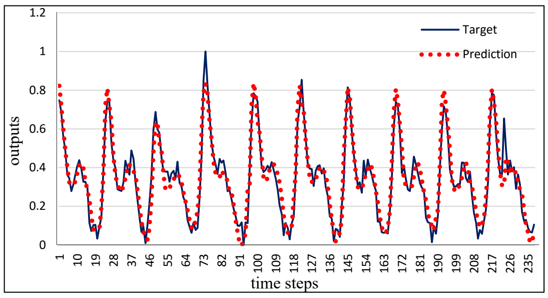
Table 3.13 Ensemble results computed for 50% test samples of time-series datasets.

Data	batch	E_t	r_t	avg. *size*	*div* (3.8)	TSF	MSF	MIF
MGS	**1**	0.018	**0.99**	9.4	0.6	4	x_1, x_3, x_4	-
	2	0.045	0.98	5.8	0.2	3		
	3	0.026	0.99	15.2	0.5	3		
	4	0.109	0.92	5.1	0.4	3		
	5	0.156	0.89	7	0.2	3		
	6	0.059	0.97	8.2	0.5	3		
	7	0.054	0.98	6.4	0.4	4		
WWR	1	0.073	0.94	5	0.1	3	x_1, x_2	-
	2	0.112	0.85	6	0.2	2		
	3	0.097	0.91	10.6	0.3	4		
	4	0.113	0.84	5	0.1	2		
	5	0.063	**0.96**	14.4	0.9	4		
	6	0.099	0.89	8.5	0.7	3		
	7	0.101	0.88	6.9	0.4	3		

Note: E_t, r_t, and *div* indicate test RMSE, test correlation coefficient, and diversity, respectively



(a) Dataset MGS $E_t = 01815$



(b) Dataset WWR $E_t = 0.06328$

Fig. 3.10 Target versus prediction plot obtained for time-series datasets MGS and WWR.

of a high correlation between model's output and desired output, which is a significant indicator of model's efficient performance.

3.7 Performance of HFNT on real-world application

In pharmaceutical industries, it is well recognized that the variation in composition and the quality of tablets (drugs) are determined by material properties and process conditions. One of the greatest challenges in pharmaceutical development is to identify the causal relationship between material properties, intermediate properties, final product properties, and process variables, which is crucial to obtain high-quality products. However, it is a challenging multifactorial problem [328].

Tablets are manufactured by compressing dry powders or granules in a die, i.e., the so-called die compaction. The die compaction process consists of three primary stages: die filling, compaction, and ejection [329]. Simple gravity effect can play a role during die filling of powders into the die. However, flow behavior during die filling process controls the tablet composition, the tablet properties as well as its segregation tendency [330]. Therefore, the study of die filling process parameters has a significant role in controlling tablet properties in manufacturing industry.

3.7.1 Predictive modeling of pharmaceutical granules

In this chapter, HFNT was also applied to predict die filling performance of pharmaceutical granules and to identify significant die filling process variables. The performance of the HFNT was compared with other CI techniques: multilayered perceptron (MLP), Gaussian process regression (GPR), and reduced error pruning tree (REP-Tree). The accuracy of the CI model was evaluated experimentally using die filling as a case study. The die filling experiments were performed using a model shoe system and three different grades of microcrystalline cellulose (MCC) powders (MCC PH 101, MCC PH 102, and MCC DG). The feed powders were roll-compacted and milled into granules. The granules were then sieved into samples of various size classes. The mass of granules deposited into the die at different shoe speeds was measured. From these experiments, a dataset consisting true density, d50, granule size, and shoe speed as the inputs and the deposited mass as the output was generated. Cross-validation (CV) methods such as 10-FCV and 5x2-FCV were applied to develop and to validate the predictive models.

This section, first, briefly introduce CI methods. Then, it describes the material and methods, and the data collection process. The concise definitions of these CI techniques are as follows:

- The MLP is a mathematical form of human-like learning, where a network of computational nodes arranged in a layered architecture is trained using a dataset [33].
- In REP-Tree, based on the dataset, a tree-like structure is created, where the tree's internal nodes are binary decision nodes and the leaf nodes are the output nodes [331].
- The GPR is a Gaussian distribution based extension of linear regression technique [332].

These CI techniques are available as an open source library, i.e., as a software tool named WEKA [333]. A detailed description of these models is available in [333].

Materials and methods

Microcrystalline cellulose (MCC) of three different grades was chosen as the raw materials, including Avicel PH-101, Avicel PH 102, and DG. A custom-made gravity-fed roll compactor with two counter rotating smooth rolls of 200 mm in diameter and 46 mm in width was used for ribbon production [334]. The roll gap was set at 1.2 mm and the roll speed was se at 1 rpm. Ribbons were milled using a milling system (SM100, Retsch, Germany) equipped with a mesh size of 4 mm at a constant speed of 1,500 rpm. The granules were sieved into different size classes (0-90, 90-250, 250-500, 500-1000, 1000-1400, 1400-2360 µm) that were used for the die filling experiments. The corresponding upper size limit was used as the granule size in the current study (i.e., granules in the size range 0-90 µm were regarded as granules with a size of 90 µm), as commonly adopted in particle technology.

Data collection

True densities of the three MCC powders were determined using a Helium Pycnometer (AccuPyc II 1340, Micromeritics, UK). Particle size analysis was performed using a size analyzer (Camsizer XT, Retsch, UK). The experiments were run for 2-3 minutes and the data were collected. The mean diameter, d50, defined as the size value below which 50% of the particles lies, was then determined. Die filling experiments were performed using a model die filling system that consists of a shoe driven by a pneumatic driving unit, a positioning controller, and a displacement transducer [335]. For each granule size class, experiments were performed using seven different shoe speeds in the range of 10 $\mathrm{mm\,s^{-1}}$ to 400 $\mathrm{mm\,s^{-1}}$. For each die filling experiment, the powder mass deposited into the die was weighted, and the value was recorded. Each experiment was repeated three times. In total, 389 experiments (3 powders, 6 granule sizes, 7 speeds, 3 repeats) were performed, and 389 data samples were generated.

From the collected experimental data, four parameters were chosen as inputs for the modeling: true density and mean diameter (d50) of raw powders, granule size (µm), and shoe speed ($\mathrm{mm\,s^{-1}}$), and the deposited mass was the only output. Table 3.14 shows a selection of few samples (taken from 389 samples) of the generated dataset.

3.7.2 Predictive modeling results

Predictive modeling with the CI techniques discussed in Section 3.7.1 was performed in the following manner. Two different cases, i.e., 10-FCV and 5x2-FCV methods, were considered. Each of the four CI techniques discussed above (i.e., HFNT, MLP, GPR, and REP-Tree) were used to develop the models, and their performance was evaluated. The collected dataset was pre-processed by normalizing the features of dataset between zero

Table 3.14 Example of few data samples generated for modeling.

#	Samples Name	Input				Output
		True density	d50 ($\mathrm{mm\,s^{-1}}$)	Granules size	Shoe speed	Mass (g)
		Feature #1	Feature #2	Feature #3	Feature #4	
1	MCC PH 101	1581	59.83	90	10	12.81
2	MCC PH 101	1581	59.83	90	10	12.78
:	:	:	:	:	:	:
5	MCC PH 101	1581	59.83	90	20	12.3
6	MCC PH 101	1581	59.83	90	30	9.55
:	:	:	:	:	:	:
135	MCC PH 102	1570.3	94.7	250	50	13.45
136	MCC PH 102	1570.3	94.7	250	60	13.5
:	:	:	:	:	:	:
388	MCC DG	1785.6	52.33	2360	400	9.51
389	MCC DG	1785.6	52.33	2360	400	9.3

Table 3.15 Parameters settings and the values are chosen during MLP, GPR, and REP-Tree training. The settings mentioned are those used in the software tool [333].

#	CI Technique	Parameter Name	Definition/Purpose	Values
1	MLP	Learning rate	Convergence speed.	0.3
2		Momentum rate	Magnitude of past iteration influence.	0.2
3		Hidden Layer	Maximum nodes at hidden layer.	350
4		Iterations	Maximum number of evaluations of parameter optimization	500
5	GPR	Kernel	The function used for implementing covariance function.	RBF Kernel
6	REP-Tree	No. Leaf Instances	Minimum No. of children per node.	2
8		Depth	Maximum limit of tree depth/level.	No limit
9		Pruning	Pruning of tree nodes.	Allowed

and one using a min-max normalization method. Table 3.2 lists the underlying parameters for the HFNT; whereas, Table 3.15 describes the basic parameter set-up for other three CI techniques: MLP, GPR, and REP-Tree. The model accuracy was assessed using RMSE E and correlation coefficient r.

Predictions using the 10-FCV method

Table 3.16 describes the performance of the best models created by using HFNT, MLP, GPR, and REP-Tree. In Table 3.16, the generated models are arranged in descending order (the highest accuracy to the lowest accuracy) of their test correlation values. It may be observed that the performance of the models created using HFNT (when compared the

Table 3.16 Performance of the prediction models and validation over 10-FCV.

Model	Model	Mean of RMSEs		Mean of r		Std over r		Model	Selected
No.	Type	Train	Test	Train	Test	Train	Test	Complexity[1]	Features[2]
1	HFNT	2.0206	2.0571	0.93	0.95	0.0087	0.0383	43	1, 2, 3, 4
2		2.3891	2.3934	0.91	0.91	0.0083	0.0617	34	2, 3, 4
3		2.5491	2.2618	0.88	0.91	0.0078	0.0563	32	3, 4
4	REP-Tree	2.5751	3.1637	0.88	0.82	-	-	99	1, 2, 3, 4
5	GPR	2.9632	3.4023	0.86	0.79	-	-	-	1, 2, 3, 4
6	MLP	3.3687	3.4427	0.81	0.79	-	-	-	1, 2, 3, 4

Note: [1]Complexity is the sum of total nodes in the created tree-model. [2]Features Nos are assigned in Table 3.14

test accuracies, i.e., correlation values) was better than the other CI techniques. Hence, the detail observation of the model Nos. 1, 2, and 3, which were created using HFNT are provided.

The model No. 1 (see: row 1 of Table 3.16) produces a high correlation coefficient (on test set), i.e., 0.95, which indicates high predictability of this model over 10% of unknown samples. However, the model complexity was also high since the total function nodes and leaf nodes in the created model amounted to 43. In addition, there was no feature selection performed by this model. In comparison to model No. 1, the model Nos 2 and 3 had lower test correlation coefficient (performance was slightly poor); but their model's complexity was simpler, and they did offer feature selection, which was advantageous than the model No. 1.

Fig. 3.11 illustrates the performance of the model using regression (scatter) plot and target against predicted value plot. In Fig. 3.11, each model Nos 1, 2, and 3 respectively were tested over 10% of test samples, i.e., 38 randomly chosen test samples, and the scattered plot and the target against predicted values plot were analyzed. It may be observed that the prediction curve follows the target curve. However, the lower values and outliers were slightly out of the reach in the prediction curve.

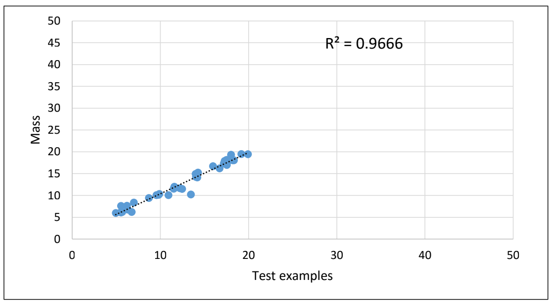
Predictions using the 5x2-FCV method

In Table 3.17, a comparison of the best models created using HFNT, MLP, GPR, and REP-Tree are provided. The models in Table 3.17 are arranged in descending order (the highest accuracy to the lowest accuracy) of their test correlation values. Similar to the modeling in 10-FCV, the modeling in 5x2-FCV and the performance of the models created using HFNT outperformed the models created using MLP, GP, and REP-Tree. Hence, a detail observation on the model Nos. 7, and 8 are provided. Accordingly, Fig. 3.12 illustrates the performance of the model using regression (scatter) plot and target against predicted value plot.

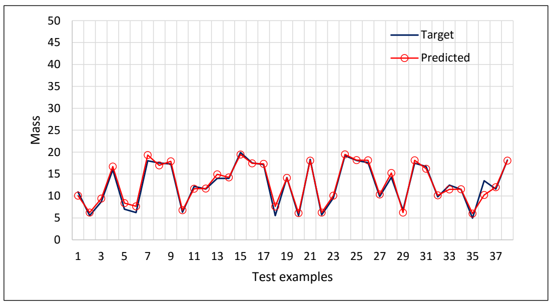
Table 3.17 Performance of the prediction models and validation over 5x2-FCV.

Model No.	Model Type	Mean of RMSEs Train	Mean of RMSEs Test	Mean of r Train	Mean of r Test	Std over r Train	Std over r Test	Model Complexity	Selected Features
7	HFNT	2.5075	2.6481	0.88	0.88	0.0415	0.0403	16	1, 2, 3, 4
8		2.6030	2.6792	0.88	0.87	0.0213	0.0217	17	2, 3, 4
9	REP-Tree	2.4460	3.6691	0.89	0.77	-	-	51	1, 2, 3, 4
10	MLP	2.7698	3.8730	0.86	0.76	-	-	-	1, 2, 3, 4
11	GP Reg.	2.7978	3.7869	0.87	0.75	-	-	-	1, 2, 3, 4

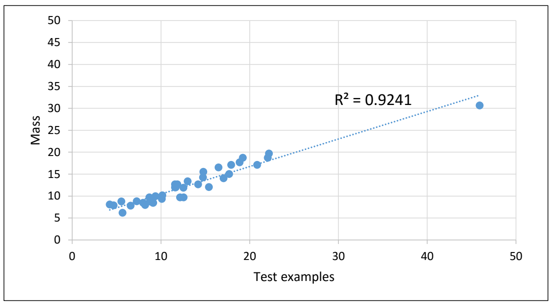
Note: Model Nos. are continued from Table 3.16



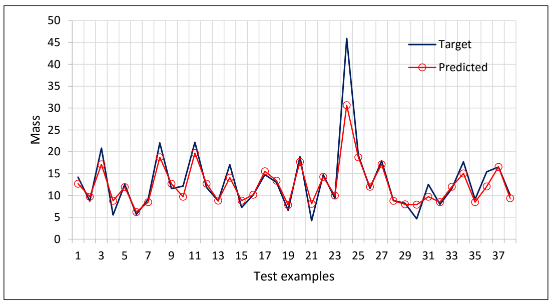
(a) Model No. 1



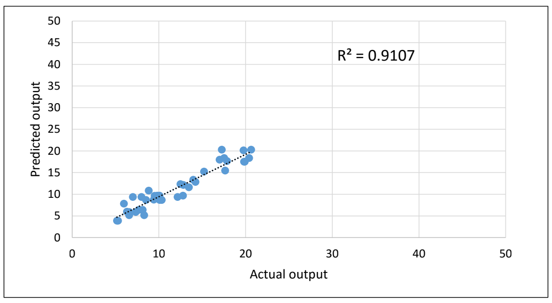
(b) Model No. 1



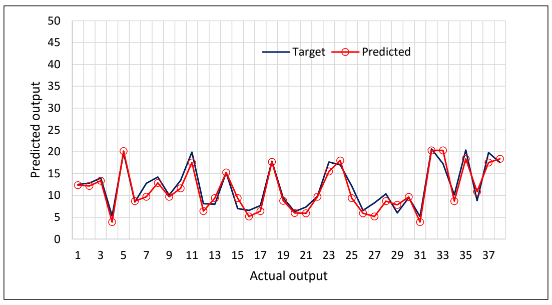
(c) Model No. 2



(d) Model No. 2



(e) Model No. 3



(f) Model No. 3

Fig. 3.11 Models evaluation on unknown test samples. The regression plots (a), (c), and (e) indicates a high correlation between actual and predicted values. The plots (b), (d), and (f) shows the one-to-one mapping of target and prediction of the best models Nos. 1, 2, and 3 (see Table 3.16). The R^2 is the squared value of correlation coefficient r, where R^2 equal to one is the best performance and R^2 equal to zero is the worst performance.

Models evaluation on unknown test samples. The regression plots (a), (c), and (e) in Fig. 3.11 indicates a high correlation between actual and predicted values. The plots (b), (d), and (f) in Fig. 3.11 shows the one-to-one mapping of target and prediction of the best models Nos. 1, 2, and 3 (see Table 3.16). The R^2 is the squared value of correlation coefficient r, where R^2 equal to one is the best performance and R2 equal to zero is the worst performance.

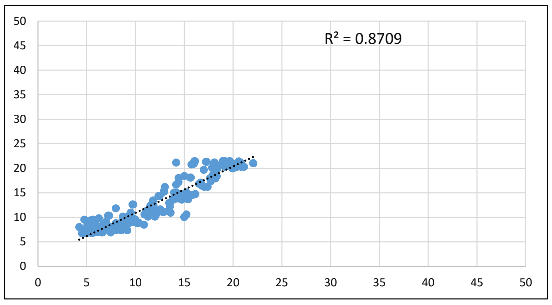
In Fig. 6, each model Nos. 7, and 8, respectively was tested over 50% of test samples, i.e., 194 randomly chosen test samples, and the scattered plot and the target versus prediction plot were analyzed. Similar to the trend as observed in Fig. 3.11, the trend observed in Fig. 3.12 says that the prediction curve follows the target curve. However, the lower values and the outliers were out of the reach of the prediction curve. The outlier is clearly visible in Fig. 6d. The outlier in the dataset was because of the noise or bad value observed during die filling process.

Fig. 3.13 and Fig. 3.14 are the illustrations of the created models, where the root node of the tree indicates the output of the models and the leaf nodes (Square boxes with numbers) indicates the input features. In Fig. 3.13 and Fig. 3.14, the input feature x_1, x_2, x_3, and x_4 indicate the features true density, d50, granule size, and shoe speed, respectively.

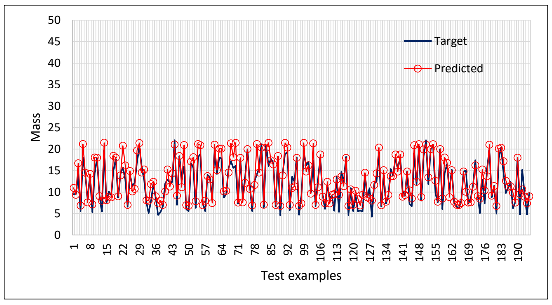
Comparison between 10-FCV and 5x2-FCV methods

Each 10-FCV and 5x2-FCV methods have their advantages. This is the reason both methods were used for creation and validation for the predictive modeling. In 10-FCV, a model uses a large sample for training. Thus, 10-FCV has higher representativeness of real world (data samples) during learning, i.e., the model is trained efficiently. Whereas, in 5x2-FCV a model used an equal proportion of data samples for training and testing (smaller training samples than 10-FCV, but larger test samples than 10-FCV). Thus, 5x2-FCV has higher generalization ability. Hence, at one hand, if a high test correlation coefficient obtained using 10-FCV, it indicates that an effective predictive model can be created from the given dataset. Whereas, if a high test correlation coefficient obtained using 5x2-FCV, it indicates that a general predictive model can be created from the given dataset.

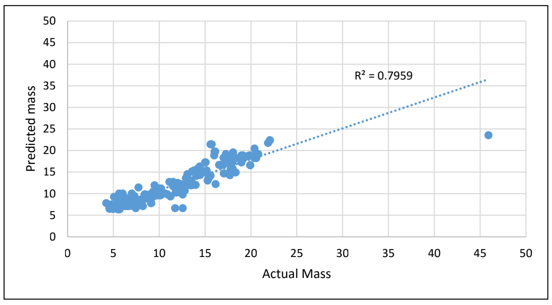
The models 7 and 8 were created using 5x2-FCV, and their test correlation values were found competitive to the model No. 1 using 10-FCV method. From 5x2-FCV results, it was found that the model Nos. 7 and 8 were simple in comparison to model Nos. 1, 2, and 3, but their accuracies (correlation coefficient) were slightly poorer in comparison. However, the model Nos. 7 and 8 were tested over 50% test samples. Hence, it describes that the deposited mass can be efficiently predicted by using die filling process variables



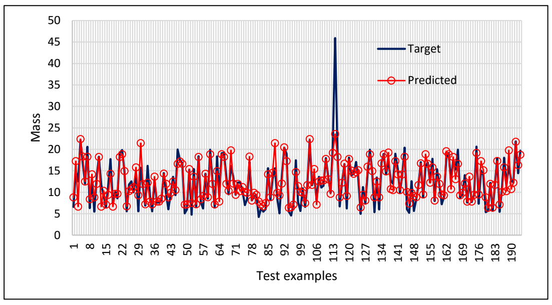
(a) Model No. 7



(b) Model No. 7



(c) Model No. 8



(d) Model No. 8

Fig. 3.12 Models evaluation on unknown test samples. The regression plots (a) and (c) indicates a high correlation between actual and predicted values and plots (b) and (d) shows the one-to-one mapping of target and prediction of the best model Nos. 7 and 8 (Table 3.17). The R^2 is the squared value of correlation coefficient r, where R^2 equal to one is the best performance and R2 equal to zero is the worst performance.

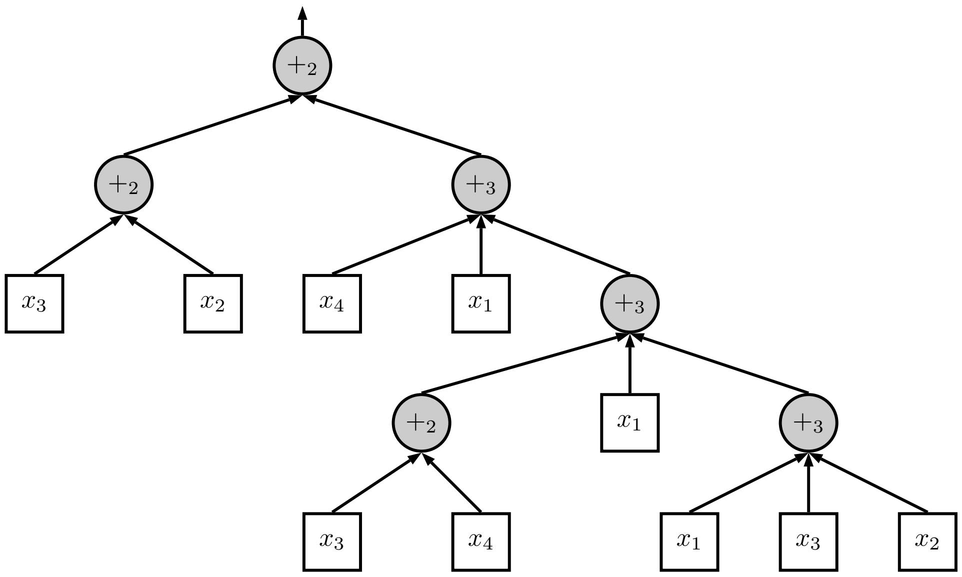


Fig. 3.13 Tree-like structure of predictive model No. 7 created using 5x2-FCV method: Complexity equal to 16 is the sum of the computational nodes (node in circles) and the leaf nodes (node in square).

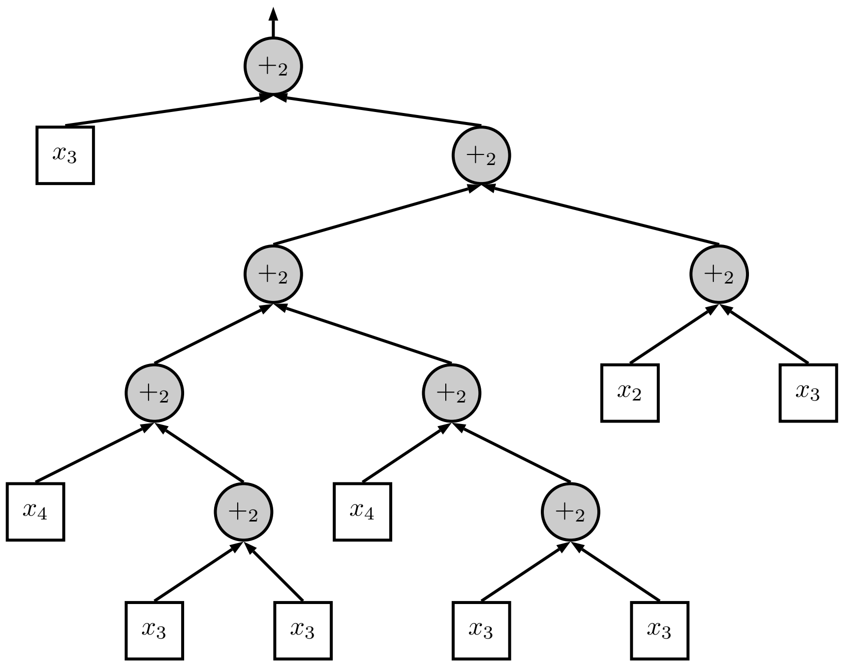


Fig. 3.14 Tree-like structure of predictive model No. 8 created using 5x2-FCV method: Complexity equal to 17 is the sum of the computational nodes(node in circles) and the leaf nodes (node in square).

knowledge. Moreover, choice of model is subjective. Since simple models have higher generalization ability, the competitive accuracies of the model Nos. 7 and 8 to the model Nos. 1, 2, and 3 tells that one had to choose between the accuracies and the generalization ability of the models.

Fig. 3.11 and Fig. 3.12 present the graphical visualization of the 10-FCV and 5x2-FCV models, respectively, where, the test performance of the models over each test samples were examined. The 5x2-FCV models produced similar trend when the models were tested over 50% of samples. However, the 5x2-FCV models (see Fig. 3.12(d)) did not predict the outlier as close as 10-FCV models (see Fig. 3.11(d)) predicted it.

3.7.3 Feature analysis results

A total 30 models were created using HFNT for feature analysis. Since the evolutionary process was used during model creation, the created models selected the input features set that had the highest predictability. Therefore, the RMSEs and selected input feature set by the models were placed into a list. Subsequently, a comprehensive feature analysis was performed. For this purpose, two performance measure dimensions were adopted: feature selection rate R as defined in (3.9) and feature predictability score P as defined in (3.11). The feature analysis was categorized into two phases.

1) The identification of individual input features. Here, for (3.9) and (3.10) the feature set $|Z_j|$ was set to one, which indicates that at a time only one input feature was analyzed.

Table 3.18 Significance of individual input features.

#	Input Features set	Selection Rate (R)	Predictability Score (P)
1	Z_1= True density	0.55173	0.541356
2	Z_2= d50	0.62069	0.586262
3	Z_3= Granule size	1	1
4	Z_4= Shoe speed	0.86207	0.92563

Since there were four input features in the dataset, in this phase, $\mathbf{Z}^s \subset \mathbf{Z}$, i.e., $\mathbf{Z}^s$ was equal to $\{Z_1, Z_2, Z_3, Z_4\}$ (see Table 3.18 for the definition of $Z_1, \ldots, Z_4$), i.e., $|\mathbf{Z}^s|$ in (3.11) was equal to four.

2) The identification of feature subset, i.e., identification of the best combination of input feature. Here, for (3.9) and (3.10) the feature set $|Z_j|$ can be one or two or three or four. After examining the selected feature by the models in the list, there was six different input feature subsets was found. Hence, in this phase, $\mathbf{Z}^s \subset \mathbf{Z}$, i.e., $\mathbf{Z}^s$ was equal to $\{Z_1, Z_2, Z_3, Z_4, Z_5, Z_6\}$ (see Table 3.19 for the definition of $Z_1, \ldots, Z_6$), i.e., $|\mathbf{Z}^s|$ in (3.11) was equal to six.

Identification of the significance of individual input features

Table 3.18 describes the feature analysis results performed for the individual input features. The significance of individual features true density, d50, granule size, and shoe speed was examined. The features true density and d50 represents the powder properties. Whereas, the granule size and shoe speed represents the die filling process variables. It can be observed that the selection rate and predictability score of d50 (0.62069 and 0.58626) were higher than that of the selection rate and predictability score of true density (0.55173 and 0.54136). Therefore, d50 possess comparatively higher importance as a powder property than that of the true density. Similarly, the process variable granule size was more influential than that of shoe speed. However, the difference of significance level was marginal, but when the entire four input variables were compared, the process variables were having significantly higher selection rate and predictability score than that of the properties of the powder. This fact was also evident from feature subset analysis.

Identification of the best set input features.

Table 3.19 shows interesting findings, where it can also be observe that the predictability score of subset Z_4 (process variable granule size and shoe speed) and the subsets Z_1, Z_2 and Z_3 (with both process variables combined with one or two powder properties) were

Table 3.19 Optimal subset of input features.

#	Input Feature set	Selection Rate (R)	Predictability Score (P)
1	Z_1= True density, d50, Granule size, Shoe speed	0.31035	0.969497
2	Z_2= d50, Granule size, Shoe Speed	0.17242	0.941601
3	Z_3= True density, Granule size, Shoe speed	0.13793	1
4	Z_4= Granule size, Shoe speed	0.24138	0.979663
5	Z_5= True density, d50, Granule size	0.10345	0.493741
6	Z_6= d50, Granule size	0.03448	0.470451

higher compared to the subsets Z_5 and Z_6 (subsets where one of the process variables was not used for prediction).

The subset analysis produced a clear picture. It says that although the predictability score of subset Z_3 was highest, the selection rate of subset Z_1 was highest. This indicates that the evolutionary process often preferred to use the combination of entire features, i.e., the set Z_1. However, among the subsets Z_1, Z_2, and Z_3, the selection rate of subset Z_2 was higher, which indicates d50 had the higher ability to represent powder properties than that of true density, but again the difference was marginal (say approximately higher by only four percent).

Accuracy evaluations of the models

Fig. 3.15 and Fig. 3.16 shows the comparison between die filling experimental results and predicted results from the model No 1 of the 10-FCV method using HFNT. This model was chosen because it has the highest value of R^2, which leads to a better fitting between experimental and predicted data. More specifically, Fig. 3.15 and Fig. 3.16 presents the mass collected after each experiment as a function of the shoe speed of the three different MCCs powders for six different granule size ranges. It shows that there is a decrease of mass deposited into the die with increasing shoe speed for all the materials and granules size ranges investigated. Moreover, a general increase of deposited mass at a consistent shoe speed was found with the increasing granule size. MCC DG tends to have higher mass deposited values compare to MCC PH 102 and MCC PH 101 at all the shoe speeds considered and for all the different size ranges analyzed, exception for the granule size range 250-500 µm (Fig. 3.15(c)). MCC PH 101 and MCC PH 102 show an identical trend in all the experiments performed.

Interestingly, results for finest granules (Fig. 3.15(a) and Fig. 3.15(b)) appears to have larger variations (due to the lower experimental reproducibility) compare to those for coarser granules (Fig. 3.16(b) and Fig. 3.16(c)). It is clear that the models give better predictions for coarser granules (Fig. 3.16(b) and Fig. 3.16(c)) than for finer granules

(Fig. 3.15(a) and Fig. 3.15(b)) for all the materials under investigation. In particular, the model gives almost identical values to the measured ones for coarser granules, which proves that the HFNT method can predict die filling behavior for such materials with high accuracy. The accuracy of the model appears to rely on the consistency (or scattering) of the experimental data.

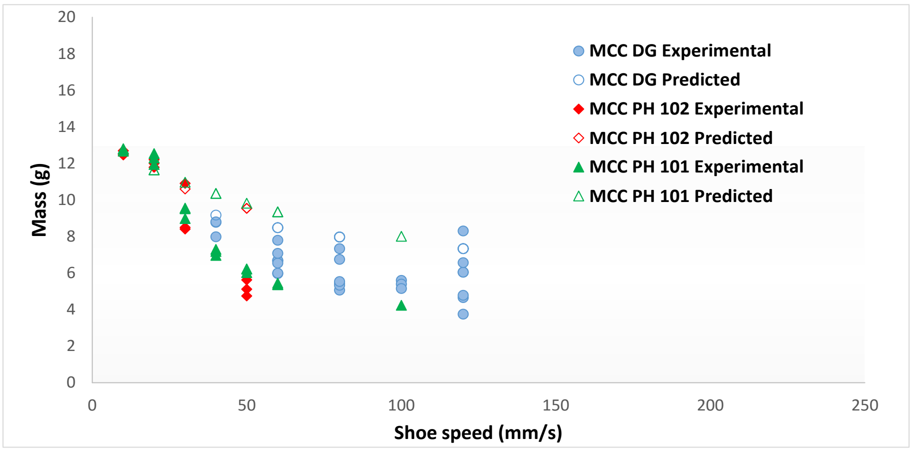
3.7.4 Discussion on die filing results

Heterogeneous flexible neural tree (HFNT) was used to predict die filling behavior of MCCs granules of different size ranges. Two main methods were investigated, 10-FCV and 5x2-FCV. Computational intelligence models were developed using HFNT, MLP, REP-Tree, and GPR. It was observed that the flexible neural tree models performed better than other CI techniques. Additionally, by examining HFNT models of each method, it was found that 10-FCV was an efficient method with a higher correlation coefficient than the 5x2-FCV. The experimental results were used as inputs and outputs of the HFNT models. The constructed model efficiently predicted the deposited mass based on the knowledge gathered from the experimental data. Similarly, the feature analysis discovered that the shoe speed and the granule size are more significant in terms of governing the deposited mass than the raw powder properties (true density and d50). Interestingly, die filling behavior of coarser granules are easier to predict than fine granules for all the materials considered. This is due to the higher reproducibility of the experimental data for larger granules.

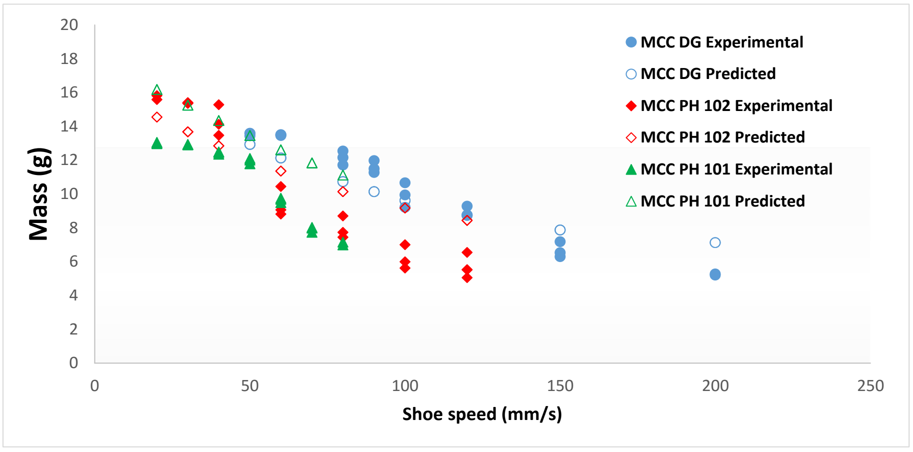
3.8 Summary

HFNT$^{\mathrm{M}}$ was examined over three categories of datasets: classification, regression, and time-series. The results presented in Sections 3.6 and 3.7, clearly suggests a superior performance of HFNT$^{\mathrm{M}}$ approach. In HFNT$^{\mathrm{M}}$ approach, MOGP guided an initial HFNT population towards Pareto-optimal solutions, where HFNT final population was a mixture of heterogeneous FNTs. Alongside, accuracy and simplicity, a Pareto-based multiobjective approach ensured diversity among the candidates in final population. Hence, FNTs in the final population were fairly accurate, simple, and diverse. Moreover, FNTs in the final population were diverse according to structure, parameters, activation function, and input feature. Hence, the model's selection from Pareto-fronts, as indicated in Section 3.5, led to a good ensemble system.

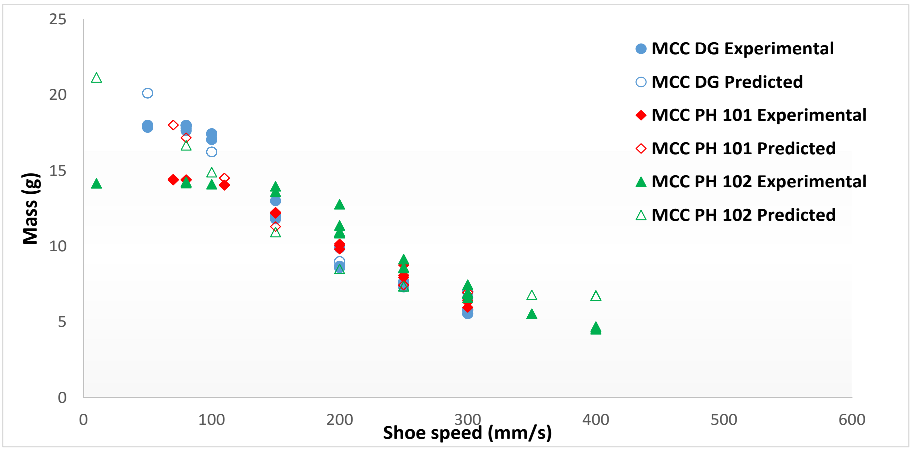
HFNT$^{\mathrm{M}}$ was applied to solve classification, regression, and time-series problems. Since HFNT$^{\mathrm{M}}$ is stochastic in nature, its performance was affected by several factors: random generator algorithm, random seed, the efficiency of the meta-heuristic algorithm used in



(a)

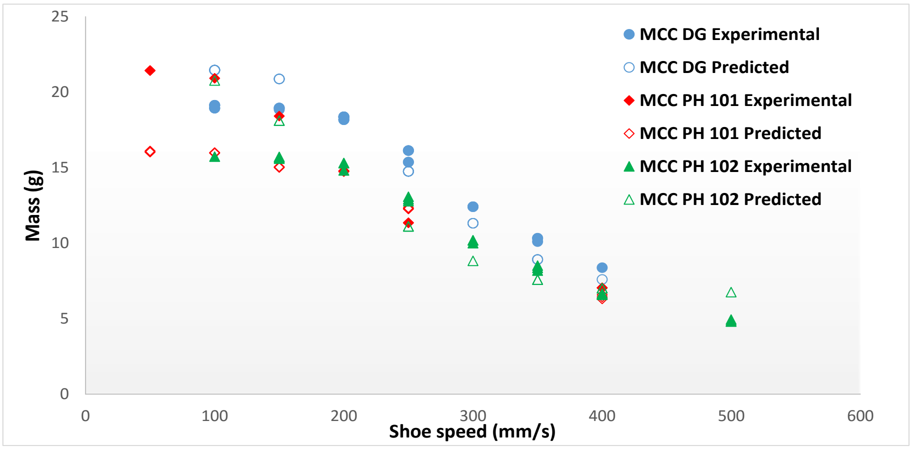


(b)

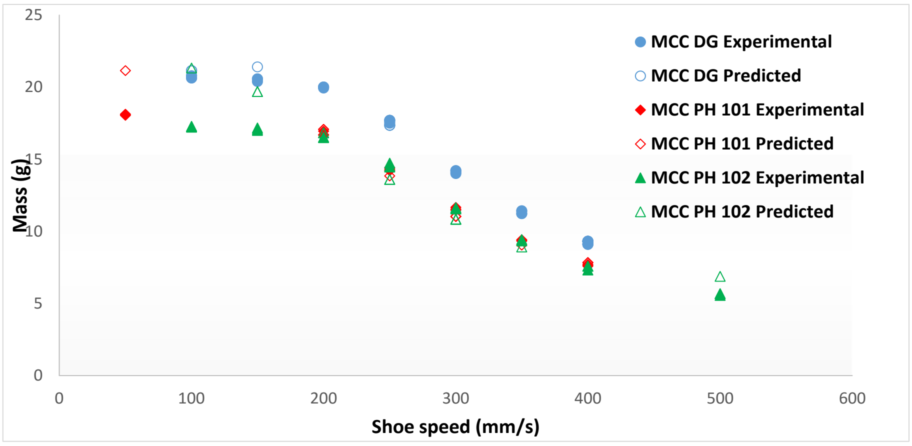


(c)

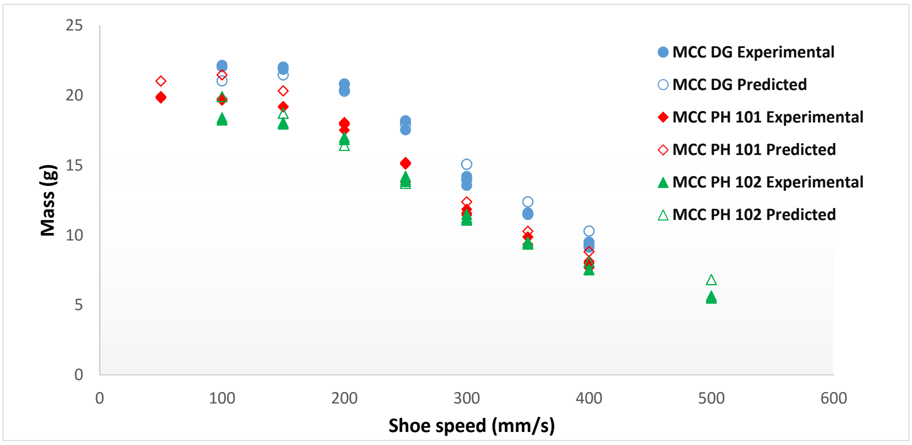
Fig. 3.15 Comparison of experimental results and model predictions for 3 MCC granules of different size ranges: a) 1-90 µm, b) 90-250 µm, c) 250-500 µm.



(a)



(b)



(c)

Fig. 3.16 Comparison of experimental results and model predictions for 3 MCC granules of different size ranges: a) 500-1000 µm, b) 1000-1400 µm, and c) 1400-2360 µm.

Table 3.20 Performance of the activation functions noted for the best-performing ensembles.

	activation function (k)						
Data	1	2	3	4	5	6	7
AUS	10	-	-	2	-	-	-
HRT	10	-	9	4	-	5	3
ION	6	5	-	-	2	4	4
PIM	3	8	2	5	2	1	-
WDB	-	3	-	7	8	10	8
ABL	2	10	-	-	-	10	-
BAS	2	5	-	-	2	10	-
DEE	-	6	6	4	4	10	-
EVL	10	5	-	3	-	-	6
FRD	10	10	-	-	-	-	-
MGS	4	1	-	2	1	10	10
WWR	10	-	4	-	4	7	-
Total	67	53	21	27	23	67	31

Note: 67 is the best and 21 is the worst

parameter-tuning phase, the activation function selected at the nodes, etc. Therefore, to examine the performance of HFNT$^{\mathrm{M}}$, several HFNT-models were created using different random seeds and the best and average approximation error of all created models were examined. In Section 3.6, as far as the best model is concerned, the performance of HFNT$^{\mathrm{M}}$ surpass other approximation model mentioned from literature. Additionally, for each dataset, a very low average value (high accuracy for classification and low approximation errors for regression and time-series) were obtained, which significantly suggests that HFNT$^{\mathrm{M}}$ often led to good solutions. Similarly, for the ensembles, it is clear from the result that combined output of diverse and accurate candidates offered high quality (in terms of generalization ability and accuracy) approximation/prediction model. From the results, it is clear that the final population of HFNT$^{\mathrm{M}}$ offered the best ensemble when the models were carefully examined based on approximation error, average complexity (*size*), and selected features.

Moreover, the performances of the best performing activation functions were examined. For this purpose, the best ensemble system obtained for each dataset were considered. Accordingly, the performance of activation functions was evaluated as follows. The best ensemble system of each dataset had 10 models; therefore, in how many models (among 10) an activation function k appeared, was counted. Hence, for a dataset, if an activation function appeared in all models of an ensemble system, then the total count was 10. Subsequently, counting was performed for all the activation functions for the best ensemble systems of all the datasets. Table 3.20, shows the performance of the activation functions.

It can be observed that the activation function Gaussian ($k = 1$) and Bipolar Sigmoid ($k = 6$) performed the best among all the other activation functions followed by Tangent-hyperbolic ($k = 2$) function. Hence, no one activation function performed exceptionally well. Therefore, the efforts of selecting activation function, adaptively, by MOGP was essential in HFNTs performance.

In the presences of *no free lunch theorem* [91] and the algorithm's dependencies on random number generator, which are platforms, programming language, and implementation sensitive [336], it is clear that performance of the mentioned approach is subjected to careful choice of training condition and parameter-setting when it comes to deal with other real-world problems.

However, the effective use of the final population of the HFNTs evolved using Pareto-based MOGP and the subsequent parameter tuning by DE led to the formation of high-quality ensemble systems. The simultaneous optimization of accuracy, complexity, and diversity solved the problem of structural complexity that was inevitably imposed when a single objective was used. MOGP used in the *tree construction* phase often guided an initial HFNT population towards a population in which the candidates were highly accurate, structurally simple, and diverse. Therefore, the selected candidates helped in the formation of a good ensemble system. The result obtained by HFNT$^{\mathrm{M}}$ approach supports its superior performance over the algorithms collected for the comparison. In addition, HFNT$^{\mathrm{M}}$ provides adaptation in structure, computational nodes, and input feature space. Hence, HFNT$^{\mathrm{M}}$ is an effective algorithm for automatic feature selection, data analysis, and modeling. Particularity, HFNT was found efficient than other CI algorithms in modeling the real-world problem that had data which represented a dynamic environment of die filling process.

Chapter 4

Metaheuristic design of fuzzy inference system

Fuzzy inference system (FIS) has the ability to model uncertain, incomplete, and noisy data more efficiently than the neural network. In the past decades, it has been used for modeling complex real-world problems. Metaheuristic design of fuzzy, particularly the use of evolutionary algorithms, has given a significant contribution in the tuning of FISs: Mamdani-type and Takagi-Sugano-Kang (TSK)-type. Moreover, neuro-fuzzy paradigm, which combines both FNN and FIS allowed us to improve the approximation ability. One basic issue with FIS is in handling input dimensionality, where hierarchical design has played a crucial role in addressing these problems to a great extent. Additionally, the multiobjective optimization of FIS has addressed the interpretation and accuracy trade-offs. This chapter summarizes various FIS paradigms.

4.1 Introduction

The ability of the FIS-based model to explain *how the model offer solution to a problem* has drawn FIS to a broad range of application domain. Moreover, there are two basic types of linguistic fuzzy rule base system (FRBS): Mamdani-type [337], and 2) TSK-type [1]. Both these models have IF-THEN rule structure, i.e., their rules are in the antecedent and consequent form. However, they differ in their consequent part, where the Mamdani has an output action or class, and the TSK has a polynomial function. Thus, they differ in their approximation ability. The former type has a better interpretation ability, and the latter has a better approximation accuracy. However, the hybrid design of FRBSs such as neuro-fuzzy design, multiobjective optimization of FRBSs (multiobjective optimization optimizes both accuracy and interpretation simultaneously) has reduced such difference between these two.

The focus of this chapter is to discuss the basics of FIS (Section 4.2), the FIS design paradigms (Section 4.3), and the challenges and future research directions in FIS design (Section 4.4).

4.2 Fuzzy inference system

4.2.1 Components of FIS

FRBSs are composed of the following components: 1) A *knowledge base* (KB), which contains fuzzy rules of the form IF-THEN, i.e.,

IF a set of conditions is satisfied
THEN a set of consequent can be inferred

2) An *inference engine*, which includes *fuzzification* module that fuzzify crisp input data into fuzzy sets, and the inference engine also does the reasoning to infer knowledge from KB. 3) A *defuzzification* part that translate inferred knowledge from KB by inference engine into a rule action (crisp output). An illustration of such discussion is shown in Fig. 1.2.

Further, the KB is composed of a data base (DB) and a rule base (RB). The DB assigns fuzzy memberships to the linguistic variables, i.e., it transforms linguistic input variables to a fuzzy membership value using membership functions (or a fuzzy sets). On the other hand, in an RB, a set of rules are constructed using DB. Thus, the design of RB governs the type of FRBS that can be designed (Mamdani-type or TSK-type).

The DB contains fuzzy sets that can be either of type-1 or of type-2. The basic form of membership functions (MFs) are coined as type-1 fuzzy set (T1FS); whereas, type-2 fuzzy set (T2FS) allow a membership function to be fuzzy itself by extending membership value into an additional dimension of membership value (the following section explains T2FS, in detail). Hence, type of fuzzy set implies type-1 FIS (T1FIS) and type-2 FIS (T2FIS). The following section describes T1FIS and T2FIS in detail. The detail explanation of FIS is limited to TSK-type FIS only because the other type, Mamadai-type FIS, only differs in its consequent part.

4.2.2 Type-1 and type-2 FISs

Type-1 TSK-fuzzy inference system

A TSK-type FIS is governed by IF–THEN rule of the form [1]:

$$R^i : \text{IF } x_1 \text{ is } A_1^i \text{ and } \ldots \text{ and } x_{p^i} \text{is } A_{p^i}^i \text{ THEN } y^i \text{ is } B^i \tag{4.1}$$

where R^i is the i-th rule in a FIS, A^i is a T1FS, B^i is a function of an input vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_{p^i} \rangle$ that returns a crisp output y^i, and p^i is the total number of inputs presented at the i-th rule. Note that the number of inputs may vary from rule-to-rule. Hence, the dimension of inputs at a rule is denoted as p^i. In TSK, function B^i is usually expressed as:

$$B^i = c_0^i + \sum_{j=1}^{p^i} c_j^i x_j, \tag{4.2}$$

where c_j^i for $j = 0$ to p^i is the free parameters at the consequent part of a rule. The basic building blocks of a FIS is shown in Fig. 1.2 whose defuzzified crisp output is computed as follows. First, the inference engine fires the rules from rule-base. The firing strength f^i of the i-th rule is computed as:

$$f^i = \prod_{j=1}^{p^i} \mu_{A_j^i}(x_j) \tag{4.3}$$

where $\mu_{A_j^i}$ is the value of j-th T1FS MF at the i-th rule. Then, the defuzzified output $\hat{y}$ of FIS is computed as:

$$\hat{y} = \frac{\sum_{i=1}^{M} B^i f^i}{\sum_{i=1}^{M} f^i} \tag{4.4}$$

where M is the number of rules in the rule-base.

An example of T1FS A is illustrated in Fig. 4.1(a), which has the following form:

$$\mu_A(x) = \frac{1}{1 + \left(\frac{x-m}{\sigma}\right)} \tag{4.5}$$

where m and σ are the center and the width respectively of MF $\mu_A(x)$.

Type-2 TSK-fuzzy inference system

A T2FS $\tilde{A}$ is characterized by a 3-dimensional membership function [338]. The three axes of T2FS are defined as follows. The x-axis is called primary variable, the y-axis is called secondary variable (or primary MF, which is denoted by u), and the z-axis is called the MF value (or secondary MF value, which is denoted by μ). Hence, in a universal set X, a T2FS $\tilde{A}$ has the form:

$$\tilde{A} = \left\{ \left((x, u), \mu_{\tilde{A}}(x, u) \right) \mid \forall x \in X, \forall u \in [0, 1] \right\}. \tag{4.6}$$

The MF value μ has a 2-dimensional support called the *footprint of uncertainty* of $\tilde{A}$, which is bounded a lower membership function (LMF) $\underline{\mu}_{\tilde{A}}(x)$ and an upper membership function (UMF) $\bar{\mu}_{\tilde{A}}(x)$. Such form of T2FS that is bounded by lower and upper MFs is called interval type-2 FS (IT2FS). The *footprint of uncertainty* is the area enclosed within

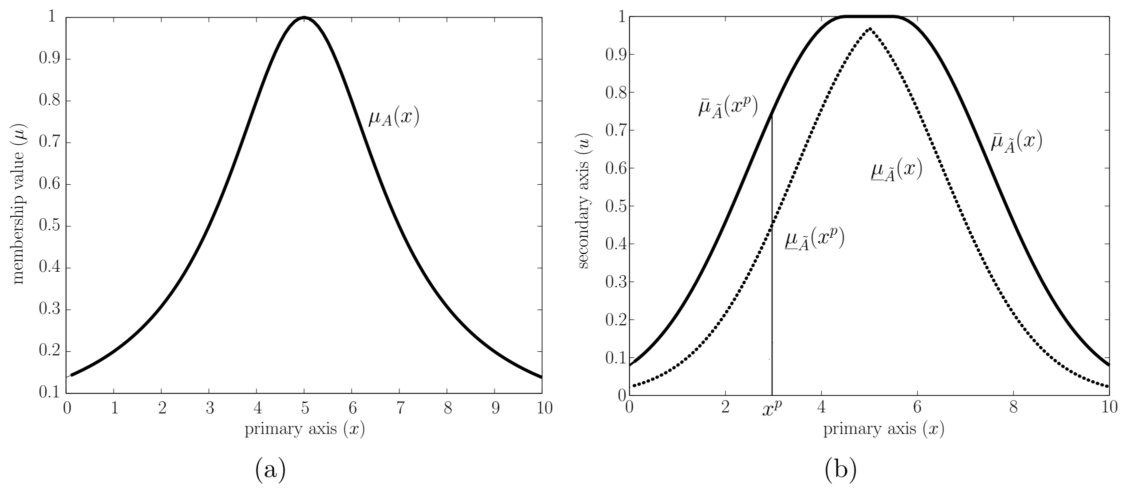


Fig. 4.1 Fuzzy membership functions. (a) Type-1 MF (4.5) with mean $m = 5.0$ and $\sigma = 2.0$. (b)Type-2 Fuzzy MF with fixed $\sigma = 2.0$ and means $m_1 = 4.5$ and $m_2 = 5.5$. UMF $\bar{\mu}_{\tilde{A}}(x)$ as per (4.9) is in solid line and LMF $\underline{\mu}_{\tilde{A}}(x)$ as per (4.8) is in dotted line.

the LMF and the UMF (Fig. 4.1(b)). A Gaussian function with uncertain mean within $[m_1, m_2]$ and standard deviation σ is called an interval type-2 MF (Fig. 4.1(b)), which is expressed as:

$$\mu_{\tilde{A}}(x, m, \sigma) = \exp\left(-\frac{1}{2}\left(\frac{x-m}{\sigma}\right)\right), \quad m \in [m_1, m_2]. \tag{4.7}$$

The LMF of a IT2FS can be defined as [339]:

$$\underline{\mu}_{\tilde{A}}(x) = \begin{cases} \mu_{\tilde{A}}(x, m_2, \sigma), & x \leq (m_1 + m_2)/2 \\ \mu_{\tilde{A}}(x, m_1, \sigma), & x > (m_1 + m_2)/2 \end{cases} \tag{4.8}$$

and the UMF can be defined as [339]:

$$\bar{\mu}_{\tilde{A}}(x) = \begin{cases} \mu_{\tilde{A}}(x, m_1, \sigma), & x < m_1 \\ 1, & m_1 \leq x \leq m_2 \\ \mu_{\tilde{A}}(x, m_2, \sigma), & x > m_2 \end{cases} \tag{4.9}$$

In Fig. 4.1(b), a point x^p along the x-axis of 3-dimensional T2FS MFs cuts the UMF and LMF along the y-axis, and the value of the type-2 MF is considered to be along the z-axis (not shown in Fig. 4.1(b)) are $\bar{\mu}_{\tilde{A}}(x^p)$ and $\underline{\mu}_{\tilde{A}}(x^p)$. Considering T2FS MFs, i-th IF–THEN rule of type-2 TSK-FIS for an input vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_{p^i} \rangle$ takes the following form:

$$R^i : \text{IF } x_1 \text{ is } \tilde{A}_1^i \text{ and } \ldots \text{ and } x_{p^i} \text{ is } \tilde{A}_{p^i}^i \text{ THEN } y^i \text{ is } \tilde{B}^i \tag{4.10}$$

where $\tilde{A}^i$ is a T2FS, $\tilde{B}^i$ is a function of $\mathbf{x}$ that returns a pair $[\underline{b}^i, \bar{b}^i]$ called left and right weights of the consequent part of a rule. In TSK, $\tilde{B}^i$ is usually written as:

$$\tilde{B}^i = [c_0^i - s_0^i, c_0^i + s_0^i] + \sum_{j=1}^{p^i} [c_j^i - s_j^i, c_j^i + s_j^i] x_j, \tag{4.11}$$

where c_j^i for $j = 0$ to p^i is the free parameter at the consequent part of a rule and s_j^i for $j = 0$ to p^i is the deviation factor of the free parameter. The firing strength of IT2FS $F^i = [\underline{f}^i, \bar{f}^i]$ is computed as:

$$\underline{f}^i = \prod_j^{p^i} \underline{\mu}_{\tilde{A}_j^i} \quad \text{and} \quad \bar{f}^i = \prod_j^{p^i} \bar{\mu}_{\tilde{A}_j^i} \tag{4.12}$$

At this stage, inference engine fires the rule and the type-reducer reduces the T2FS to T1FS. Here, center of set type-reducer prescribed by Karnik [339] can be used for type reduction purpose. The center of set y_{cos} is computed as:

$$y_{cos} = \bigcup_{f^i \in F^i,\, b^i \in \tilde{B}^i} \frac{\sum_{i=1}^{M} f^i b^i}{\sum_{i=1}^{M} f^i} = [y_l, y_r], \tag{4.13}$$

where y_l and y_r are left and right end of the interval. For the ascending order of $\underline{b}^i$ and $\bar{b}^i$, y_l and y_r are computed as:

$$y_l = \frac{\sum_{i=1}^{L} \bar{f}^i \underline{b}^i + \sum_{i=L+1}^{M} \underline{f}^i \underline{b}^i}{\sum_{i=1}^{L} \bar{f}^i + \sum_{i=L+1}^{M} \underline{f}^i}, \tag{4.14}$$

$$y_r = \frac{\sum_{i=1}^{R} \underline{f}^i \bar{b}^i + \sum_{i=R+1}^{M} \bar{f}^i \bar{b}^i}{\sum_{i=1}^{R} \underline{f}^i + \sum_{i=R+1}^{M} \bar{f}^i}, \tag{4.15}$$

where L and R are the switch point, determined by

$$\underline{b}^L \leq y_l \leq \underline{b}^{L+1} \text{ and } \bar{b}^R \leq y_r \leq \bar{b}^{R+1},$$

respectively. The defuzzified crisp output $\hat{y}$ is computed as:

$$\hat{y} = \frac{y_l + y_r}{2}. \tag{4.16}$$

4.3 Fuzzy inference system paradigms

Neural network and metaheuristics plays a significant role in the design of FIS paradigms. Fig. 4.2 is a popular and common way to represent the confluence of metaheuristics, NN,

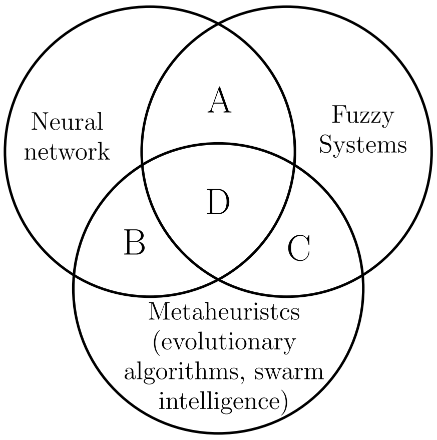


Fig. 4.2 Spectrum of fuzzy inference system paradigms.

and FIS that leads to the indication of several possible designs. Examining Fig. 4.2, the following option can be perceived: 1) neuro-fuzzy system, indicated by letter A; 2) genetic/metaheuristic based fuzzy system, indicated by letter C; 3) metaheuristic based tuning of the neuro-fuzzy system, indicated by letter D; and 4) the metaheuristic-based neural network design, indicated by letter B. Chapter 2 and 3 discussed the option No. 4, in detail. This chapter has a detailed description of the options Nos. 1, 2, and 3.

4.3.1 Metaheuristic based FIS optimization

FRBS can be classified into two major categories: metaheuristic-based tuning and metaheuristic-based learning. Metaheuristic based tuning includes the optimization of the KB components such as MFs parameters and a rule's consequent part. However, it is necessary to a have a KB a priori for the metaheuristic-based tuning. The metaheuristic learning includes the optimization the KB by tuning/evolving the whole KB itself by using either evolutionary algorithm or some other metaheuristic. Learning of KB has a high influence of FRBSs efficiency because it designs the RB by adaptive partitioning/selection of linguistic terms for linguistic variables in DB, by selection of rules, or by the simultaneous optimization of both KB components (DB and RB).

Metaheuristic based optimization (as described in Chapter 2) depends on genotype representation of the problem. Therefore, the approach where a set of rules are encoded into a single chromosome (vectored representation) is known as a *Pittsburgh approach* [15, 340]. In Pittsburgh approach, the first step in the learning of RB is to generate the sets of rules, where each set contains M randomly created rules from DB. From the randomly generated rules, the subset of rules that offered the best fitness computed as per (1.1) is selected. For the selection of the best subset of rules, usually each rule in a FIS are randomly assigned a status "0" (inactive) or "1" (active) that tells whether to select the i-th rule or not. Hence, a population $Q = (\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_k)$ of a total k RBs are generated. The i-th RB $\mathbf{R}_i$ is defined as:

$$\mathbf{R}_i = \{R_{i1}, R_{i2}, \ldots, R_{iM}\} \, \forall \, i = 1, 2, \ldots, k, \tag{4.17}$$

where the status of a rule $R_{ij} \in \mathbf{R}_i$ is randomly set to either “0” or “1.” Such population is a genetic population, where the individuals (say i-th rule-base) are coded into a binary vector that can be optimized by using genetic algorithm [7]. In-steed of the preceding example (the usage of a binary vector for rule base optimization), some other means of manipulation of RB population Q can be adopted to obtain an optimal set of RB. Designing various methods for the manipulation of the population Q is an interesting research problem among the research community.

On the other hand, in *Michigan approach* [15, 341], a single rule or rule parameters are encoded into a genotype. Therefore, for a rule R^i that has a total p^i fuzzy set, the parameter vector $\mathbf{w}^i$ for the i-th rule may be designed as follows:

$$\mathbf{w}^i = \begin{cases} \langle (m,\sigma)^i_1, (m,\sigma)^i_2, \ldots, (m,\sigma)^i_{p^i}, c^i_0, c^i_1, \ldots, c^i_{p^i} \rangle & \text{for T1FS} \\ \langle (m,\lambda,\sigma)^i_1, \ldots, (m,\lambda,\sigma)^i_{p^i}, (c_0, s_0)^i, (c_1, s_1)^i, \ldots, (c_{p^i}, s_{p^i})^i \rangle & \text{for T2FS} \end{cases} \tag{4.18}$$

where $(m,\sigma)^i_j$ is the parameters (center and width) of the j-th T1FS, and $(m,\lambda,\sigma)^i_j$ is the parameters (center, center deviation factor, and width) of the j-th T2FS. Similarly, for type-1 rule, c^i_j, $j = 0$ to p^i are the parameters of the consequent part, and for type-2 rule, the pairs $(c_j, s_j)^i$, $j = 0$ to p^i are the consequent part parameters and their deviation factors. Hence, a metaheuristics algorithm can be employed to optimize the parameter vector $\mathbf{w}^i$. Such kind of learning results in tuning the fuzzy sets MFs shapes.

Similar to Michigan approach, also in *iterative rule learning* scheme and *cooperative-competitive rule leaning*, each rule of an RB are encoded into separate genotypes, and the population of such genotype leads to the formation of RB iteratively. Iterative learning scheme starts with an empty set and adds rules one-by-one to the set by finding an optimum rule from a genetic selection process. For this purpose, the genetic operators such as mutation and crossover operators are applied over one or two rule(s) to make offspring rule(s), and the quality of the generated rule(s) is(are) evaluated using a predefined rule quality measure. Therefore, iteratively selecting rules according to rule quality measure criteria for forms an optimum RB in an iterative rule learning scheme [342]. The cooperative-competitive rule learning is also an RB learning method that determines an optimum RB from competition and cooperation of rules from a genetic/metaheuristic population [343]. A detailed review of metaheuristic-based tuning of FRBS and evolutionary learning of FRBS is available in [344].

4.3.2 Evolving fuzzy systems

Evolving fuzzy system (EFS) concept accommodate the provisions of dynamically (on-line) updating/training of a fuzzy system for streaming (real-time) data [345]. EFS paradigm allowed the dynamic learning of fuzzy rules for every incoming real-time data. Such dynamic learning is offered by adding or removing rules in an RB (vertical direction manipulation), or by adding or removing antecedent part of the rules in an RB (horizontal direction manipulation). Fig. 4.3 is a clear representation of such dynamic training process. In Fig. 4.3, each rule may acquire p^i variables using the evolving clustering methods, i.e., the number of variables are determined automatically [346]; whereas, in traditional clustering methods, the number of clusters has to be predetermined. Moreover, the antecedent part of the rules may expand and contract based on incoming data. Similarly, the number of rules may also be reduced or increased by adding or deleting rules from the RB. Hence, the number of total rules M^t in the RB are time dependent.

Similarly, EFS concept was extended to neuro-fuzzy paradigms, in which the neuro-fuzzy systems are evolved dynamically. Such systems are also called *self-evolving neuro-fuzzy system* or *adaptive neuro-fuzzy system* [347]. In self-evolving neuro-fuzzy system, the network design has two main parts: antecedent section and the consequent section. For every incoming data, the antecedent part learn new information by using unsupervised means of learning through cluster evolving method, and accordingly, the consequent part weights are updated to accommodate the new information contained in the incoming data.

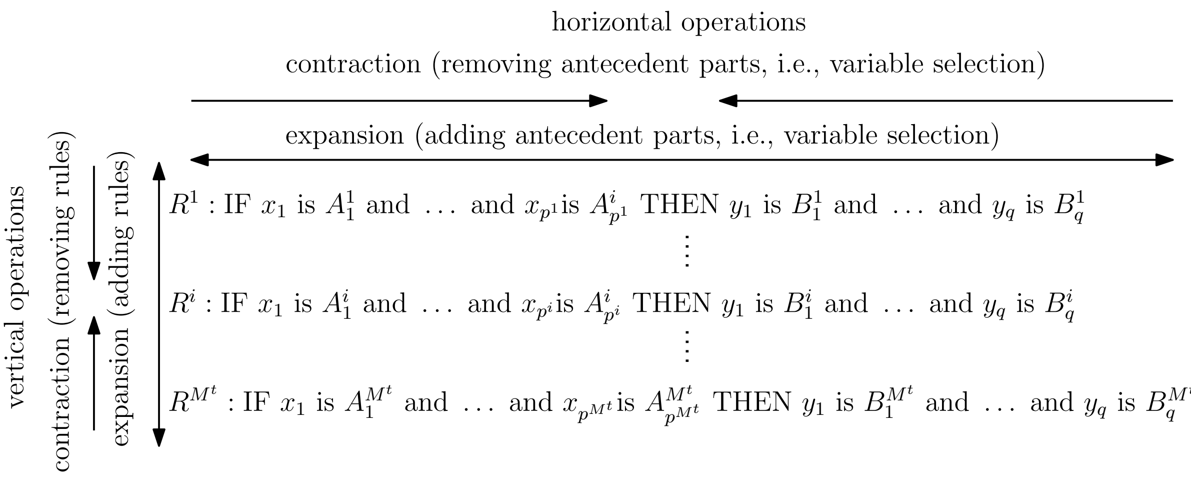

horizontal operations
contraction (removing antecedent parts, i.e., variable selection)
expansion (adding antecedent parts, i.e., variable selection)
vertical operations
contraction (removing rules)
expansion (adding rules)
R^1 : IF x_1 is A_1^1 and ... and x_{p^1} is $A_{p^1}^i$ THEN y_1 is B_1^1 and ... and y_q is B_q^1
R^i : IF x_1 is A_1^i and ... and x_{p^i} is $A_{p^i}^i$ THEN y_1 is B_1^i and ... and y_q is B_q^i
R^{M^t} : IF x_1 is $A_1^{M^t}$ and ... and $x_{p^{M^t}}$ is $A_{p^{M^t}}^{M^t}$ THEN y_1 is $B_1^{M^t}$ and ... and y_q is $B_q^{M^t}$

Fig. 4.3 Evolving fuzzy system: typical dynamic rule-base learning. The symbols in the figure are as follows: p^i and q^i indicates number of inputs x and outputs y variable to a rule i, respectively; M^t is total number of rule in the RB at time t; A^i is the fuzzy set at the antecedent part of a rule; and B^i is the function of the consequent part of a rule.

4.3.3 Neuro-fuzzy systems

The neural network is a data-driven learning model, which does not require a priori knowledge of the problem, but needs sufficient training pattern to learn, and the training model does not explain how to interpret its computational behavior. Hence, it has black-box computational behavior, which does not explain how the output was obtained for a given input data. On the other hand, a fuzzy logic system requires a priori knowledge of the problem and do not have learning ability, but it tells how to interpret its computational behavior, i.e., it explains how the output was obtained for a given input data. Therefore, the shortcomings of both neural network and the fuzzy system can be eliminated by combining them in some manner. Usually, two types of combination are practiced: *cooperative neuro-fuzzy system*, and *hybrid neuro-fuzzy system*.

The cooperative neuro-fuzzy system is the simplest approach, where neural network and the fuzzy system work independently, and the neural network determines parameters of a fuzzy system from training data [348]. Subsequently, fuzzy system performs the required interpretation of the given data.

On the other hand, in hybrid neuro-fuzzy system, both neural network, and fuzzy system are fused together. Moreover, the hybrid neuro-fuzzy system has FNN-like architecture. Some of the hybrid neuro-fuzzy systems are as follows: fuzzy adaptive learning control network (FALCON) [349], adaptive-network-based fuzzy inference system (ANFIS) [313], generalized approximate reasoning based intelligence control (GARIC) [350], neuronal fuzzy controller (NEFCON) [351], fuzzy inference and neural network in fuzzy inference software (FINEST) [352], self-constructing neural fuzzy inference network (SONFIN) [353], dynamic/evolving fuzzy neural network (EFuNN) [354].

A typical illustration of a neuro-fuzzy system is illustrated in Fig. 4.4. In Fig. 4.4, the network architecture is has two parts the antecedent part and the consequent parts. The input layer in the antecedent part is mapped onto the MF layer that transform/assign input variables to fuzzy sets (nodes in this layer are represented as μ). MF layer is then mapped onto a rule layer (product layer) that create rules from the combination of the previous layer connection, i.e., using fuzzy sets. Thus, nodes in the rule layer are represented as $\prod$. Finally, the output of the model is computed from the output layer (consequent layer) that receives fired fuzzy rules from the rule layer.

In [313], authors, proposed an adaptive-network-based FIS that implements a network-like structure of the TSK-type FIS model whose parameters were tuned by a gradient-decent method. Similarly, in [355], a TSK-based hierarchical self-organizing learning dynamics was proposed. Moreover, several research works are focused towards FIS and neural network integration and its parameter optimization using various learning methods including

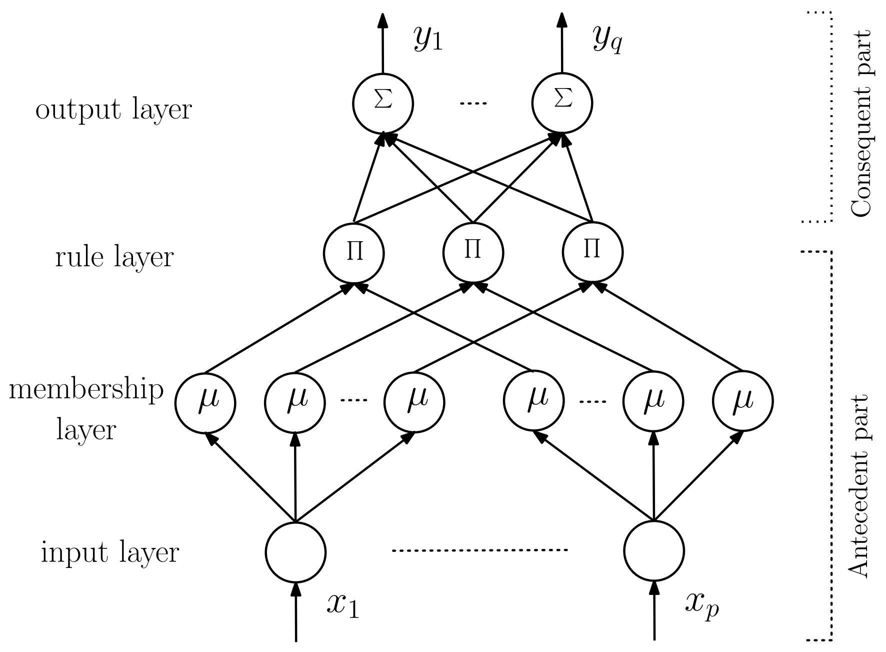


Fig. 4.4 Typical hybrid neuro-fuzzy systems.

gradient-decent and the metaheuristic algorithms [356, 357]. Such neuro-fuzzy system has been found solving several real life application efficiently [16–18].

Self-organizing fuzzy neural network paradigm brought an additional strength to FIS structure optimization. Self-constructing neural fuzzy inference network (SONFIN), proposed by Juang et al. [353], is a six layered network structure and its optimization begins with no rule and the rules are incrementally added during the learning process. SONFIN uses a clustering method to partition the input space that governs the number of rules extracted from the data. Then, the consequent parts of the SONFIN are determined, and the backpropagation algorithm is applied to tune the free parameters. Here, the use of clustering method is crucial since it eliminates the *curse of dimensionality*. Similarly, in [358], a dynamic evolving neural-fuzzy inference system (DENFIS) based on TSK-type FIS model was proposed, which was evolved incrementally by choosing active rules from a set of rules and further the parameters of the rules were optimized by least-square estimator. An evolving clustering method was used in DENFIS for partitioning the input space. In [359], the concept of SONFIN was extended for the construction of T2FIS, where a self-evolving IT2FIS (SEIT2FNN) that implements TSK-type FIS models was proposed. The Kalman-filtering algorithm was applied to tune the parameters of the evolved structure. Similarly, in [360], SEIT2FNN was extended with a simplified type-reduction process.

Self-adaptive fuzzy inference network (SaFIN) was proposed by Tung et al. [361] to overcome the limitations of the self-organizing fuzzy neural network paradigm. SaFIN applied a categorical-learning-induced-partitioning algorithm to eliminate two limitations: 1) the need of predefined numbers of fuzzy clusters during the input space partitioning;

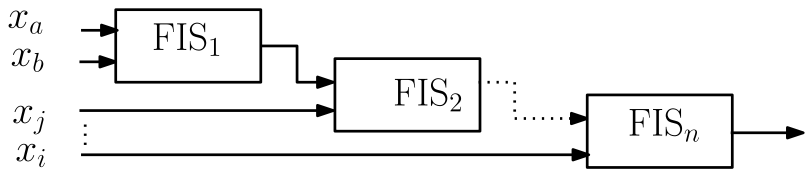


Fig. 4.5 Typical hierarchical fuzzy systems of n low-dimensional fuzzy systems. For a given problem, the inputs x_a, x_b, $x_j, \ldots$ to the subsystems are traditionally selected by applying an expert's knowledge [367].

and 2) the stability–plasticity trade-off problem that addresses the difficulty of finding a balance between past knowledge and current knowledge during the learning process. It also addressed drawbacks of inconsistent rule-base creation by employing a rule consistency check mechanism. SaFIN used Levenberg-Marquardt method for its parameter tuning. In [362], a mutually recurrent interval type-2 neural fuzzy system (MRIT2NFS) was proposed which uses weighted feedback loops at the antecedent part of the formed rules to improve the efficiency of IT2FIS. A gradient-decent learning was applied to update recurrent weights, and the Kalman-filter algorithm was applied to tune rule parameters. In [363], a self-evolving T2FIS model was proposed that employed compensatory operator in the type-2 inference mechanism and a variable-expansive Kalman-filter algorithm for the parameter tuning.

Further, in [364], a simplified interval type-2 fuzzy neural networks (SIT2FIS) that simplified type-reduction was proposed. In [365], a growing on-line self-learning IT2FIS was proposed, which used dynamics of growing Gaussian mixture model for its self-learning dynamics. Recently, a meta-cognitive interval type-2 neuro-fuzzy inference system (McIT2FIS) was proposed in [366], which employs a self-regulatory meta-cognitive system that learns sample-by-sample data patterns and accepts or discards data patterns for extracting the knowledge contained in minimal samples. For the tuning of parameters, McIT2FIS uses Kalman-filtering-based learning algorithm.

4.3.4 Hierarchical fuzzy systems

Self-organizing fuzzy neural network paradigm has to employ clustering method to reduce the dimensionality of the input space for designing the FIS structure, which is network-like. Contrary to that, an approach presented in [367, 368] initiated the design of hierarchical FIS (HFIS) that was composed of low-dimensional fuzzy subsystems. However, the selection of the input variables was left to the expert's knowledge. Thus, the level of hierarchy and number of parameters was fully up to the expert to determine (Fig. 4.5). Universal approximation ability of the HFIS is thoroughly studied in [369, 370]. Torra et al. [371] summarized several contributions related to the design of HFIS. A realization of a feedforward network like HFIS, in which the output of the previous layer subsystem was only fed to the consequent part of next layer was proposed in [372]. In [373], authors developed a two layered HFIS, where for each layer, the knowledge-bases were generated by linguistics rule generation method and the rules at the knowledge-base were selected by GA. In [374], an adaptive fuzzy hierarchical sliding-mode control method was proposed. It was an arrangement of many subsystems and the top layer accommodated all the subsystems outputs. However, structure optimization of the HFIS was explained in [375], which optimizes the hierarchical arrangements of low-dimensional TSK-type FIS using probabilistic incremental program evolution [271]. Similarly, some research work focus on the significance of hierarchical arrangement of the low-dimensional type-2 fuzzy subsystems [22, 376].

4.3.5 Multiobjective farmworker of FIS tuning

The structure optimization of a FIS is inherently a multiobjective problem since accuracy maximization and complexity minimization are two desirable objectives for an optimum FIS [377]. Hence, to make trade-offs between interpretability and accuracy, or in other words, to make trade-offs between approximation error minimization and model complexity minimization, a multiobjective orientation of the FIS optimization can be used [378–380]. Interpretability can be defined in many ways such as reduced number of rules, reduced number of parameters, etc., [381, 380]. Since a single solution may not satisfy both objectives, simultaneously, Pareto-based multiobjective optimization algorithms were used in many FIS optimization that includes rule selection, rule mining, rule learning, etc. [382–385]. Similarly, in [386–390], simultaneous learning of knowledge-base was proposed, which included feature selection, rule complexity minimization together with approximation error minimization, etc. In [391], a co-evolutionary approach that aims towards combining multiobjective approach with single objective approach was presented. In co-evolutionary approach, first, a multiobjective GA determined a Pareto optimal solution by finding a trade-off between accuracy and rule complexity. Then, a single objective GA was applied to reduce training instances. Such process was then repeated until a satisfactory solution was obtained. A summary of many research works focused on multiobjective optimization of FIS is provided in [392].

4.4 Challenges and future scope

Unlike other approximation models, the FIS has interpretation ability. Thus, determining a good interpretability measure is one of the open issues in FRBS design. Additionally, how efficiently an FRBS can handle a high-dimensional data is another big challenge in the research community. In addition, a FRBS also has to address the data related challenges that usually an approximation algorithms has to address.

4.5 Summary

A metaheuristic based design and neuro-fuzzy based design of FIS was discussed in this chapter. At one hand, metaheuristic provides tuning and learning-ability to the FIS components. On the other hand, neuro-fuzzy paradigm eliminates disadvantages of both neural network and fuzzy system by offering a hybrid method to fuse the both into a single entity. Additionally, multiobjective optimization of the FIS address typical interpretability versus accuracy problem. However, design a FRBS that can efficiently address interpretability issues and can effectively manage a high-dimensional data is a big challenge.

Chapter 5

Multiobjective hierarchical fuzzy inference trees

Introduction of the fuzzy-set (type-1) enabled the modeling of uncertain and noisy information and type-2 fuzzy set took this further ahead by allowing fuzzy membership function to be fuzzy itself. Therefore, the design of a fuzzy inference system (FIS) that can offer a viable trade-off between accuracy and complexity is the desirable design. To meet this objective, in this chapter, a multiobjective genetic programming (MOGP) for designing a hierarchical fuzzy inference tree (HFIT), which produces an optimum tree-like structure that accommodates simplicity by combining several low-dimensional fuzzy subsystems was proposed. Such design produces highly accurate FIS models. The constructions of HFIT took place in a coevolutionary manner. First, a nondominated sorting based MOGP was applied to find an optimum tree structure. Then, differential evolution (DE), a metaheuristics algorithm, was applied for tuning the parameters of the tree structure, which are the membership function parameters and the free parameters at the consequent part of the rules. Such process of structure optimization using MOGP and parameter tuning using DE are repeated until an optimal HFIT was obtained. The HFIT offered an automatic feature selection because it uses MOGP for the self-organization of tree-like structures whose nodes are low-dimensional FISs that uses subsets derived from given input set. The HFIT was studied in the context of both type-1 and type-2 FIS, and its performance was evaluated over six application problems. Moreover, the proposed multiobjective HFIT was compared with recently proposed FIS algorithms in literature such as McIT2FIS, TSCIT2FNN, SIT2FNN, RIT2FNS-WB, eT2FIS, MRIT2NFS, IT2FNN-SVR, etc. From the obtained results, it is evident that the proposed HFIT offers an efficient and competitive alternative to the other data mining algorithm for the function approximation and the feature selection.

5.1 Introduction

Initially, only type-1 fuzzy set (T1FS), introduced by Zadeh [21], was used. The development of type-2 fuzzy set (T2FS), however, has drawn much attention towards handling noisy and imprecise data [339]. Type-1 FIS (T1FIS) and type-2 FIS (T2FIS) differs when it comes to the representation of the antecedent part and the consequent part of a rule. The T1FIS uses T1FS MFs; whereas, the T2FIS uses T2FS MFs. Unlike the crisp outputs of the T1FIS MFs, the output of the T2FIS MFs are fuzzy in nature [393]. Such nature of T2FIS MFs is advantageous in processing uncertain information effectively than T1FIS MFs [22]. Hence, T2FIS can overcome the certain limitations of the T1FIS [22]. However, T2FIS is computationally expensive because of a large number of parameters in the optimization and the requirement of type-reduction mechanism at the defuzzification part. Interval T2FIS (IT2FIS) reduces the computational cost to some extent by simplifying the T2FS MFs. The MF of an interval T2FS is bound by a lower MF (LMF) and an upper MF (UMF), and the area between the LMF and the UMF is called the *footprint of uncertainty* [339].

The MFs of a T1FIS or a T2FIS are designed by using optimization. Hence, it is called the optimization of FIS. For both T1FIS and T2FIS, there are two aspects of FIS design:

1. Construction of an appropriate rule-base, which includes the genetic selection of rules at the rule-base, structural/network representation of the rule-base, etc.

2. Optimization of the MF parameters and the free parameters of the designed rule-base that governs the performance of FIS.

Literature contains several works related to structure optimization, which represent the FIS structure in various forms and use of multiobjective orientation to address interpretability and accuracy trade-offs. Therefore, in this chapter, a multiobjective HFIS, called, hierarchical fuzzy inference tree (HFIT) was proposed, which addresses interpretability and accuracy trade-offs indirectly by finding accurate and simple FIS models. Unlike self-organizing paradigm, which has network-like structure and uses clustering algorithm for partitioning of input space, the proposed HFIT constructs a tree-like structure and used dynamics of the evolutionary algorithm for partitioning input space [394]. HFIT is analogous to multilayered network and perform automatic partitioning of input space during the training of structure. The parameter optimization of the HFIT was performed by the DE algorithm, which is a metaheuristics algorithm inspired by the dynamics of the evolutionary process [6]. In the past, researchers used various metaheuristic for FIS optimization [395]. The advantages of using bio-inspired metaheuristic algorithms for the optimization is evident from [396]. Moreover, multiobjective treatment to HFIT was provided by using multiobjective genetic programming (MOGP). In this work, HFIT was developed for both T1FIS and T2FIS, and it implements TSK-type FIS. In the construction of T2FIS, the type-reduction algorithm K-M method was used with an improvement into its termination criteria. The comparison of the proposed algorithms with the algorithms available in the literature suggests that the multiobjective based HFIT design offers a high approximation ability with simple model complexity. Hence, this chapter aims to address the followings:

1. Construction of a hierarchical tree-like arrangement of low-dimensional fuzzy systems using a coevolutionary approach, which involves multiobjective genetic programming for evolving a tree-like structure and metaheuristic for tuning parameters of the evolved tree.

2. Feature selection and the simplification of the rule-base in fuzzy subsystems by exploiting the dynamics of the genetic programming.

3. Implementation of the HFIT for both T1FIS and T2FIS and the comparative evaluation of their single objective and multiobjective orientations with recently proposed FIS algorithms from the literature.

Organization of this chapter is as follows. First, Section 5.2 describes the usage of multiobjective strategy and its formulation for developing the proposed HFIT. A detailed description of the proposed HFIT is provided in Section 5.2.2, which includes detail discussion on structure optimization and parameter optimization of HFIT. In Section 5.3, first, a detailed parameter-setting and performance measures are explained. Then, the four versions HFIT algorithm were proposed: type-1 single objective HFIT (T1HFIT$^{\mathrm{S}}$), type-1 multiobjective objective HFIT (T1HFIT$^{\mathrm{M}}$), type-2 single objective HFIT (T2HFIT$^{\mathrm{S}}$), and type-2 multiobjective objective HFIT (T2HFIT$^{\mathrm{M}}$). These algorithms were compared with the algorithms from literature over six example problems. Finally, the obtained results are discussed in Section 5.6.

5.2 Multiobjective for fuzzy inference tree

5.2.1 Pareto-based multi-objectives

The FIS mentioned here was used for learning from data and usually a learning algorithm own a single objective (approximation error minimization) that is often achieved by minimizing root mean squared error (RMSE) on the learning data, which is computed by taking square root of the cost function (1.1). Let us denote RMSE as E in the context of this chapter. Similarly, let $k(\mathbf{w})$ denote the number of free parameters in the model.

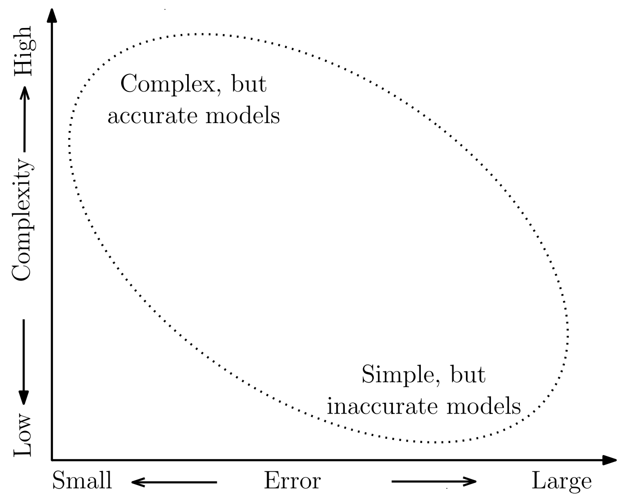


Fig. 5.1 Nondominated fuzzy inference system.

Then, the minimization of the approximation error E and the minimization of the number of free parameters $k(\mathbf{w})$ are necessary for achieving generalization ability.

However, reducing RMSE led to a larger parameter count $k(\mathbf{w})$, and reducing free parameter count $k(\mathbf{w})$ led to a larger RMSE. Fig. 5.1 is a lucid representation of this fact. An alternative is to use a combined objective of RMSE and model's complexity, which can be written as per (3.2). However, as mentioned in Section 2.4 the scalarized objective (3.2) has disadvantages. Therefore, multiobjective optimization of RMSE E and parameter count $k(\mathbf{w})$ simultaneously, as described in Section 1.2.4, is a formidable option to get a generalized solution.

Algorithm 3.3 is a basic framework of NSGA-II [229] based MOGP, which was used for computing Pareto-optimal solutions from an initial population of fuzzy inference trees. The individuals in MOGP were sorted according to their dominance in population. Moreover, individuals were sorted according to the rank (Pareto-front/line). MOGP is an elitist algorithm that allows the best individuals to propagate into next generation. Diversity in population was maintained by measuring the crowding distance among the individuals [229]. Thus, in the final population of MOGP, Pareto-optimal solution are obtained, from which the best solution as per user's preference can be selected.

5.2.2 Hierarchical fuzzy inference tree formation

Hierarchical fuzzy inference tree is a tree-based system. Its hierarchical structure is analogous to multilayer feedforward neural network where nodes (the low-dimensional FIS) are connected using weighted links. The concept of forming hierarchical fuzzy inference tree is inherited from flexible neural tree proposed by Chen et al. [20], which has two phases. First, *tree-construction* phase, where evolutionary algorithms are used

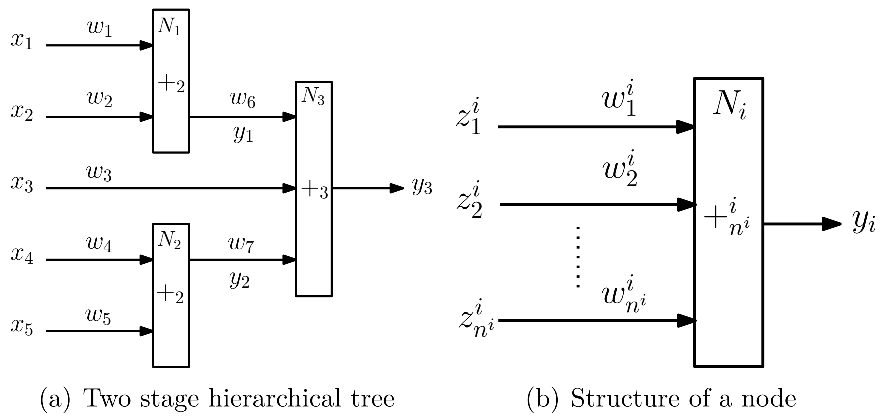


Fig. 5.2 Hierarchical fuzzy inference tree. (a) Complete tree with three nodes N_1, N_2, and N_3 and with inputs x_1, x_2, x_3, x_4, and x_5. (b) Illustration of the i-th node N_i that has n^i inputs $z_j^i \in [x_1, \ldots, x_n]$ for $j = 1$ to n^i and output y_i.

to construct/optimize a tree-like structure. Second, *parameter-tuning* phase, where a genotype representing the underlying parameters of tree-structure are optimized by using parameter optimization algorithms. To create an optimum tree-based model; first, a population of randomly created trees is formed. Once a near-optimum tree structure is obtained using evolutionary algorithm, *parameter-tuning* phase optimizes its parameter. The phases are repeated until a satisfactory solution is obtained. Fig. 3.1 provided in Section 3.3.2 is a lucid representation of the coevolutionary approach for an optimum tree construction. The construction of HFIT follows the similar coevolutionary approach as indicated in Fig. 3.1, but instead of a population of HFNT, a population of HFIT is used, and accordingly the parameters of the HFIT are optimized.

A hierarchical fuzzy inference tree (HFIT), denoted as H, is a collection of FIS node set F_H and terminal node set T_H:

$$H = F_H \cup T_H = \{+_2, +_3, \cdots, +_{tn}\} \cup \{x_1, x_2, \ldots, x_p\} \tag{5.1}$$

where $+_j$ ($j = 2, 3, \ldots, tn$) denotes non-leaf instruction and has $2 \leq j \leq tn$ arguments. Leaf node's instruction $x_1, x_2, \ldots, x_p$ takes no argument and represents input variable/instruction. A typical HFIT is shown in Fig. 5.2(a); whereas, Fig. 5.2(b) is an illustration of i-th node N_i in an HFIT that takes n^i inputs. The inputs $z_j^i \in \{x_1, x_2, \ldots, x_p\}$ for $j = 1$ to n^i to the node N_i is either from the input layer or from another node in HFIT. Each node in HFIT receives a weighted input $x_i w_i$, where w_i is the weight. In this work, the weights in HFIT were set to 1.0. Hence, the inputs were not influenced by the weights of the tree edges. Thus, setting weights fixed to 1.0 led to the reduction in the total parameter count during the *parameter-tuning* phase.

5.2.3 Rule formation

Each node in HFIT is a FIS of either type-1 or type-2. Hence, the rules at a node were created as follows. Considering a reference to the node N_1 from Fig. 5.2(a) that has two arguments/inputs x_1 and x_2 and assuming that each input x_1 and x_2 has two T1FSs A^1_{11}, A^1_{12} and A^1_{21}, A^1_{22}, respectively, the rules for T1FIS are generated as:

$$\begin{aligned}&R^1_{ij} : \text{IF } x_1 \text{ is } A^1_{1i} \text{ and } x_2 \text{ is } A^1_{2j} \text{ THEN}\\ &y^1_{ij} = c^0_{ij} + c^1_{ij}x_1 + c^2_{ij}x_2, \text{ for } i = 1,2 \text{ and } j = 1,2.\end{aligned}$$

The output y^1 of node N_1 is computed as:

$$y^1 = \frac{\sum_{i=1}^{2}\sum_{j=1}^{2}\sigma^1_{ij}y^1_{ij}}{\sum_{i=1}^{2}\sum_{j=1}^{2}\sigma^1_{ij}} \tag{5.2}$$

where

$$\sigma^1_{ij} = \mu_{A_{1i}}(x_1)\mu_{A_{2j}}(x_2) \text{ for } i = 1,2 \text{ and } j = 1,2. \tag{5.3}$$

Similarly, the output of the node N_2 is computed. The output y^3 of the HFIT shown in Fig. 5.2(a) is computed from the node N_3 that revives inputs y^1 and y^2 and x_3, where y^1 and y^2 are the outputs of the nodes N_1 and N_2, respectively.

If the nodes of HFIT in Fig. 5.2(a) are type-2 nodes; then, assuming that the node N_1 has two T2FSs $\tilde{A}^1_{11}, \tilde{A}^1_{12}$ and $\tilde{A}^1_{21}, \tilde{A}^1_{22}$, respectively, the rules for T2FIS are generated as:

$$\begin{aligned}&R^1_{ij} : \text{IF } x_1 \text{ is } \tilde{A}^1_{1i} \text{ and } x_2 \text{ is } \tilde{A}^1_{2j} \text{ THEN}\\ &y^1_{ij} = [c^0_{ij} - s^0_{ij}] + [c^1_{ij} - s^1_{ij}]x_1 + [c^2_{ij} - s^2_{ij}]x_2,\\ &\text{for } i = 1,2 \text{ and } j = 1,2\end{aligned}$$

and the firing strength of the rule at the node N_1 is computed as described in (4.12). Moreover, the type-reduction of the node is performed by using (4.13), where left and right intervals are computed as per (4.14) and (4.15). Finally, the output of the node N_1 is computed as per (4.16). Subsequently, the output of the type-2 HFIT shown in Fig. 5.2(a) is computed from the node N_3.

5.2.4 Near-optimal tree: structure and parameter tuning

Structure tuning

An HFIT that offers the lowest approximation error and simplest structure is the desirable solution. To obtain such set of Pareto-optimal solution nondominated sorting algorithm mentioned in Algorithm 3.3 was applied.

Parameter tuning

In structure tuning phase, an optimum phenotype (HFIT) is derived with the parameters being initially fixed by random guess. Hence, the obtained phenotype is further tuned in the parameter tuning phase by using a parameter optimization algorithm. To tune the parameters of the obtained phenotype, its parameters are mapped onto a genotype, i.e., onto a real vector, called solution vector.

The selection of the best phenotype in a single objective training is solely based on the comparison of the RMSEs, but selecting a solution in a multiobjective training is a difficult choice. In this work, after the multiobjective training of HFIT, the best solution for parameter tuning was picked from the Pareto-front. Strictly, the solution that gave the best RMSE among the solutions in the Pareto-optimal set of rank one was chosen. Fig. 5.3 is an illustration of the solutions that belong to Pareto-front of rank one. The genotype mapping of the T1FIS and the T2FIS differ only with the respect to their number of parameters.

The T1FIS uses the MF mentioned in (4.5), which has two arguments m and σ, and each rule in T1FIS has $p^i + 1$ variables at the consequent part as mentioned in (4.2), where p^i is the number of inputs to the i-th rule. On the other hand, since interval type-2 MF is bounded by a LMF and an UMF (Fig. 4.1(b)), it has two Gaussian means m_1 and m_2 and a variance σ that need to be optimized. The Gaussian means m_1 and m_2 for type-2 Gaussian MF (4.7) were defined as:

$$m_1 = m + \gamma\sigma \tag{5.4}$$

and

$$m_2 = m - \gamma\sigma, \tag{5.5}$$

where $\gamma \in [0, 1]$ was a random variable taken from uniform distribution and m was the center of Gaussian means m_1 and m_2 taken from [0, 1]. Similarly, σ of type-2 Gaussian MF (4.7) was taken from [0, 1]. The consequent part of T2FIS rule was computed as mentioned in (4.11), which led to $2 \times (p^i + 1)$ variables.

Assume that an HFIT (a tree like Fig. 5.2(a)) has k nodes, and each node in the phenotype takes $2 \leq p^i \leq tn$ inputs, where each input is partitioned into two fuzzy sets (membership functions). Then, the number of the fuzzy sets at a node is $2 \times p^i$. Since the number of inputs at a node is p^i and each input is partitioned into two fuzzy sets, the number of rules at a node is 2^{p^i}. Hence, the number of parameters at a T1FIS node is $[2^{p^i} \times (2 \times 2 \times (p^i + p^i + 1))]$ and the number of parameters at a T2FIS node is $[2^{p^i} \times (3 \times 2 \times p^i + 2 \times (p^i + 1))]$. Therefore, the total number of parameters in an HFIT is the summation of the number of parameters at all nodes in the tree. Assuming n is the

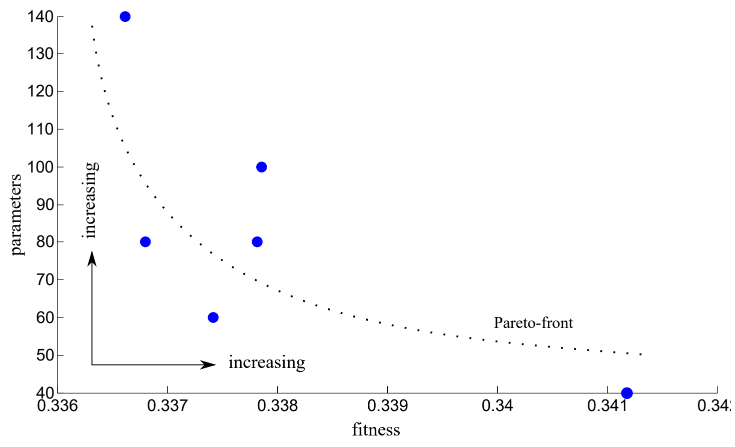


Fig. 5.3 FIS fitness versus FIS parameters mapping across the Pareto-front. This graph was garbed during a multiobjective training of the example–2 mentioned in Section 5.4.2.

total number of parameters in tree, the genotype or the solution vector **w** is expressed as:

$$\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle. \tag{5.6}$$

where elements of the vector are mapped from the phenotype, i.e., from the tree. Therefore, to optimize parameter vector **w**, meta-heuristic algorithms such as genetic algorithms [109], evolution strategy [109], artificial bee colony [308], PSO [5], DE [6], gradient-based algorithms [397], backpropagation [37], Klaman-filter [87], and so on can be used. In this work, differential evolution (DE) [6] was used, which is discussed elaborately in Section 3.3.3.

5.3 Experimental set-up

This section describes the evaluated results of the proposed algorithms T1HFIT^S, T1HFIT^M, T2HFIT^S, and T2HFIT^M on six example problems.

The performance of the algorithms were measured using the RMSE E and correlation coefficient r. For the simplicity, the training and the test RMSEs were denoted by E_n and E_t, respectively. Similarly, the training and the test correlation coefficients were denoted by r_n and r_t, respectively. The parameter-setting mentioned in Table 5.1 was used for the training of the proposed algorithms, which was developed as software tool and is available at [http://dap.vsb.cz//sw/hfit/]. The experiments were conducted on Windows Server R2 that had 20 cores and 700 GB RAM. Each run of experiments was conducted with the random seeds generated from the system. The proposed algorithm was compared with the algorithms from the literature. Table 5.2 provides detail description of the algorithms from the literature.

Table 5.1 Parameter set-up for the experiments.

Algorithm training parameter	Value
Maximum depth (layers) of a tree	4
Maximum inputs to a FIS node	4
Membership function search range	[0,1]
GP population	50
CP mutation probability pm	0.2
GP crossover probability $pc = 1 - pm$	0.8
GP mating pool size	25
GP tournaments selection size	2
GP iterations	500
DE population	50
DE mutation factor δ	0.7
DE crossover factor cr	0.9
DE iterations	5000

Table 5.2 Descriptions of the existing FIS algorithms adopted for the performance comparisons.

FIS	Algorithm	Ref.	Description	Type	Parameter tuning
Type–1	DyEFuNN	[358]	Dynamic evolving neural-fuzzy inference system	TSK	Least-square estimator
	D-FNN	[355]	Dynamic fuzzy neural networks	TSK	Backpropagation
	EFuNN	[354]	Evolving fuzzy neural networks	Mamdani	Widrow–Hoff least square
	FALCON	[349]	ART-based fuzzy adaptive learning control network	−−	Backpropagation
	GNN	[398]	Granular neural networks	−−	Genetic algorithm
	H-TS-FS	[375]	Hierarchical Tukagi–Sugno fuzzy system	TSK	Evolutionary programming
	HyFIS	[399]	Hybrid neural fuzzy inference system	−−	Gradient descent learning
	IFRS and AFRS	[400]	Incremental and aggregated fuzzy relational systems	Mamdani	Backpropagation
	RBF-AFA	[317]	Radial basis function based adaptive fuzzy systems	TSK	Gradient descent learning
	SaFIN	[361]	Self-adaptive fuzzy inference network	Mamdani	Levenberg-Marquardt
	SONFIN	[353]	Self-constructing neural fuzzy inference network	TSK	Backpropagation
	SuPFuNIS	[401]	Subsethood-product fuzzy neural inference system	−−	Gradient descent
	SVR-FM	[402]	Support-vector regression fuzzy model	TSK	Support vector regression
Type–2	eT2FIS	[403]	Evolving type-2 neural fuzzy inference system	Mamdani	Gradient descent learning
	IT2FNN-SVR-N/F	[404]	IT2FNN-support-vector regression-fuzzy and numeric	TSK	Support vector regression
	McIT2FIS-UM/US	[366]	Metacognitive interval type-2 neuro-FIS	TSK	Gradient descent learning
	NNT2FW	[346]	Type-1 and type-2 fuzzy BP neural networks	TSK	Backpropagation
	RIT2FNS-WB	[360]	Reduced IT2NFS-weighted bound-set	TSK	Gradient descent learning
	MRIT2NFS	[360]	Reduced IT2NFS-weighted bound-set	Mamdani	Gradient descent learning
	SEIT2FNN	[359]	Self-evolving IT2FIS	TSK	Kalman filter algorithm
	SIT2FNN	[364]	Simplified Interval Type-2 Fuzzy Neural Networks	TSK	gradient descent learning
	T2FLS	[405]	Interval type-2 fuzzy logic system (TSK and singleton)	TSK	−−
	T2FLS-G	[406]	Gradient-descent based IT2FIS tuning	TSK	Derivation-based learning
	TSCIT2FNN	[363]	Compensatory interval type-2 fuzzy neural network	TSK	Kalman filter algorithm

5.4 Performance of HFIT on benchmark examples

5.4.1 Example 1—system identification

Online identification of the nonlinear system is a widely studied problem. The significance of this problem is evident from its usage in literature for the validation of the approximation algorithms [359, 363, 366, 404]. An interesting paper describing several plant identification is available in [407]. The nonlinear system identification of the plant is described by the following nonlinear difference equation:

$$y_p(k+1) = \frac{y_p(k)}{1 + y_p(k)^2} + u^3(k), \tag{5.7}$$

where $[u(k), y_p(k)]$ is the input–output pair of the single input and the single output plant at the time k and $y_p(k+1)$ is the one step ahead prediction. Hence, the objective is to predict $y_p(k+1)$ of the system based on the sinusoidal input $u(k) = \sin(2\pi k/100)$ and the current output $y_p(k)$. Let us assign the input $x_1 = u(k)$ and the input $x_2 = y(k)$.

The training patterns were generated with $k = 1, \ldots, 200$, and $y_p(1) = 0$. Similarly, the test patterns were generated for $k = 201, \ldots, 400$ as mentioned in [360]. Therefore, for the training, the inputs were $u(k)$ and $y_p(k)$, and the desired output was $y_p(k+1)$. The proposed algorithms T1HFIT$^{\mathrm{S}}$, T1HFIT$^{\mathrm{M}}$, T2HFIT$^{\mathrm{S}}$, and T2HFIT$^{\mathrm{M}}$ were trained using the parameter-setting mentioned in Table 5.1. The training was repeated for 10 times, and the collected performance statistics of the proposed algorithms is shown in Table 5.3(a). Since the experiments were repeated 10 times, 10 different models were obtained for each algorithm. The results of the best models (in terms of their RMSE E values) were compared with the results available in the literature (Table 5.3(b)).

The performance statistics shown in Table 5.3(a) is the evidence of the robustness of the proposed algorithms. It shows that the mean correlation coefficients r_n and r_t of training and test sets are 1.00, which indicates that the algorithm consistently performed with a high accuracy. Moreover, such consistency of high accuracy performance is evident from the low standard deviations STDs of training and test RMSEs and correlation coefficients (Table 5.3(a)).

Interestingly, the Pareto-based multiobjective offered less complex models (the mean parameter count $k(\mathbf{w})$ of T1HFIT$^{\mathrm{M}}$ was 34.4 compared to 57.2 of T1HFIT$^{\mathrm{S}}$, and average $k(\mathbf{w})$ of T2HFIT$^{\mathrm{M}}$ was 90.4 compared to 152.0 of T1HFIT$^{\mathrm{S}}$ with high accuracies (Table 5.3(a)). Hence, the Pareto-based multiobjective was advantageous to use, which provided the option of choosing the best solution from a Pareto-front. An example of Pareto-front is shown in Fig. 5.3.

Table 5.3 Performance evaluation on system identification (Example-1).

(a) Performance statistics (10 repetitions)

		T1HFIT$^{\mathrm{S}}$	T1HFIT$^{\mathrm{M}}$	T2HFIT$^{\mathrm{S}}$	T2HFIT$^{\mathrm{M}}$
Training					
E_n	Best	0.0043	0.0041	0.0033	0.0028
	Mean	0.0181	0.0257	0.0123	0.0184
	STD	0.0167	0.0164	0.0074	0.0105
r_n	Best	1.00	1.00	1.00	1.00
	Mean	1.000	0.999	1.000	1.000
	STD	0.0006	0.0007	0.0001	0.0002
Test					
E_t	Best	0.0020	0.0041	0.0034	0.0028
	Mean	0.0169	0.0262	0.0125	0.0187
	STD	0.0173	0.0171	0.0076	0.0109
r_t	Best	1.00	1.00	1.00	1.00
	Mean	1.000	0.999	1.000	1.000
	STD	0.0006	0.0007	0.0001	0.0002
Parameters					
$k(\mathbf{w})$	Best	20	20	72	36
	Mean	57.2	34.4	152	90.4

(b) Performance comparison

	Algorithm	E_n	E_t	$k(\mathbf{w})$
Type–1	FALCON	0.0200		54
	SaFIN		0.0120	
	SONFIN	0.0080	0.0085	36
	T1HFIT$^{\mathrm{S}}$	0.0043	0.0043	60
	T1HFIT$^{\mathrm{M}}$	0.0041	0.0041	40
Type–2	T2FLS (singleton)	0.0306	–	120
	FT2FNN	0.0388	–	36
	T2FLS (TSK)	0.0217	–	120
	TSCIT2FNN	0.0080	–	34
	T2TSKFNS	–	0.0324	24
	T2FNN	–	0.0281	36
	SIT2FNN	–	0.0241	36
	RIT2NFS-WB	0.0073	0.0151	24
	MRI2NFS	0.0042	0.0051	36
	T2FLS-G	0.0214	0.0379	36
	SEIT2FNN	0.0022	0.0022	84
	T2HFIT$^{\mathrm{S}}$	0.0033	0.0034	118
	T2HFIT$^{\mathrm{M}}$	0.0028	0.0028	72

For the performance comparisons, the result of SaFIN was collected from [361], FALCON and SONFIN from [359], T2FLS (singleton) and T2FLS (TSK) from [359], FT2FNN, TSCIT2FNN, T2TSKFNS, and T2FNN from [363], SEIT2FNN, MRI2NFS, RIT2NFS-WB, and T2FLS-G from [360], and SIT2FNN from [364]. Table 5.2 contains the detailed description of these algorithms.

Two parameters may be used for comparing the algorithms: 1) the training and test RMSEs; and 2) the number of parameter count $k(\mathbf{w})$. From the performance comparisons shown in Table 5.3(b), it is found that the proposed algorithms T1HFIT$^{\mathrm{S}}$ and T1HFIT$^{\mathrm{M}}$ were better than the T1FIS algorithms FALCON, SaFIN, and SONFIN. SONFIN offered the test RMSE $E_t = 0.0085$ with the smallest parameter count $k(\mathbf{w}) = 36$; whereas, the proposed algorithm T1HFIT$^{\mathrm{M}}$ offered a better test RMSE $E_t = 0.0041$ with a slightly larger parameter count $k(\mathbf{w}) = 40$. Similarly, the proposed T2FIS algorithms T2HFIT$^{\mathrm{S}}$ and T2HFIT$^{\mathrm{M}}$ offered a better performance compared to the algorithms T2FLS (singleton), T2FLS (TSK), TSCIT2FNN, T2TSKFNS, T2FNN, SIT2FNN, RIT2NFS-WB, MRI2NFS. The algorithm SEIT2FNN reported test RMSE $E_t = 0.0022$, and the parameter count $k(\mathbf{w})$ was 84; whereas, in comparison to SEIT2FNN, the algorithm T2HFIT$^{\mathrm{M}}$ offered a slightly higher test RMSE $E_t = 0.0028$, but had lower parameter count $k(\mathbf{w})$, i.e., 72. Hence, the worth of the proposed model was evident from the statistical performance given in Table 5.3(a) and the performance comparison is given in Table 5.3(b). The target and predicted value plot of 200 samples are shown in Fig. 5.4.

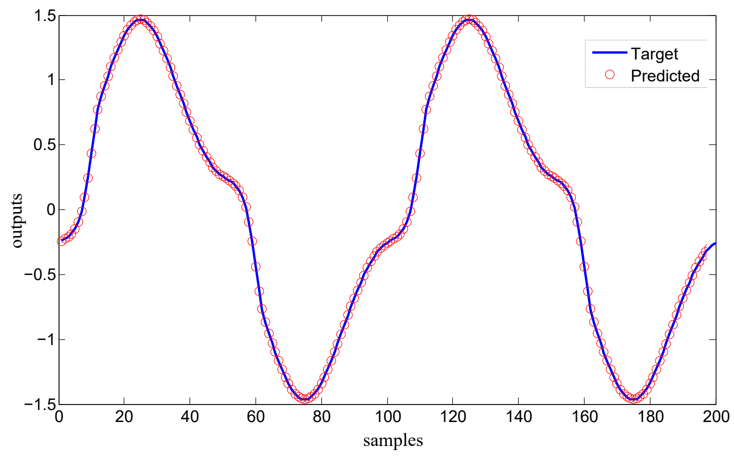
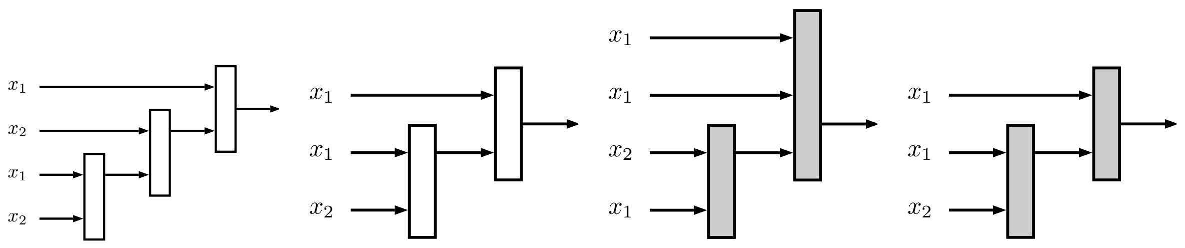


Fig. 5.4 Example–1: target versus predicted test values. The test outputs belong to algorithm T2HFIT$^{\mathrm{M}}$ that has the test RMSE $E_t = 0.0028$.



(a) T1HFIT$^{\mathrm{S}}$: $E_n = 0.0043$ (b) T1HFIT$^{\mathrm{M}}$: $E_n = 0.0041$ (c) T2HFIT$^{\mathrm{S}}$: $E_n = 0.0034$ (d) T2HFIT$^{\mathrm{M}}$: $E_n = 0.0028$

Fig. 5.5 Example–1: designed HFIT. The shaded nodes indicate T2FIS.

The best models obtained using the proposed algorithms are illustrated in Fig.5.5, which shows the hierarchical structure of the obtained models and the selected inputs are indicated by x_i in the models. The rectangular blocks in Fig 5.5 indicate the nodes (a T1FIS or T2FIS) of the tree (hierarchical structure).

5.4.2 Example 2—noisy chaotic time series prediction

Case–clean set

A chaotic time series dataset, the Mackey-Glass chaotic time series, was used in this example, which was generated using the following delay differential equation:

$$\frac{dx(k)}{dk} = \frac{0.2x(k-\tau)}{1+x^{10}(k-\tau)} - 0.1x(k), \tag{5.8}$$

where $\tau > 17$. In this example, the objective was to predict $x(k)$ using the past outputs of the time series as mentioned in [360, 404]. Hence, input–output pattern was of the form:

$$[x(k-24), x(k-18), x(k-12), x(k-6); x(k)] .$$

Let us say the inputs are $x_1 = x(k-24)$, $x_2 = x(k-18)$, $x_3 = x(k-12)$, and $x_4 = x(k-6)$. For the training of the proposed algorithms, a total of 1000 patterns were generated from $k = 124$ to 1123. This set of training patterns were clean (no noise were added). From the generated clean patterns, first 500 patterns (clean training set) were used for training purpose and second 500 patterns (clean test set) were used for test purpose. Ten repetitions of training and test using clean-training and clean-test sets were performed, and the results were collected accordingly (Table 5.4(a)). Table 5.4(b) shows the comparison of results the proposed algorithms with the results reported by algorithms listed in Table 5.2.

For this example (clean set), the performance statistics is shown in Table 5.4(a). The obtained statistics illustrate that the proposed algorithms T1HFIT$^{\mathrm{S}}$, T1HFIT$^{\mathrm{M}}$, T2HFIT$^{\mathrm{S}}$, and T2HFIT$^{\mathrm{M}}$ performed with high accuracies. It shows that the mean correlation coefficient r_n of training set is 1.00, and the mean correlation coefficient r_t of test set of the algorithms T1HFIT$^{\mathrm{S}}$, T1HFIT$^{\mathrm{M}}$, T2HFIT$^{\mathrm{S}}$, and T2HFIT$^{\mathrm{M}}$ are 0.9858, 0.9864, 0.9783, and 0.9912 respectively, i.e., the test correlation coefficients are closer to 1.00 (high positive correlation between target and predicted outputs). Such performance indicates that the algorithms consistently performed with a high accuracy, and the obtained low values of standard deviations (STDs) are the evidence of this fact (Table 5.4(a)).

Moreover, the Pareto-based multiobjective offered less complex models (the mean parameter counts $k(\mathbf{w})$ of T1HFIT$^{\mathrm{M}}$ was 57.6 compared to 71.6 of T1HFIT$^{\mathrm{S}}$ and $k(\mathbf{w})$ of T2HFIT$^{\mathrm{M}}$ was 129.5 compared to 203.4 of T1HFIT$^{\mathrm{S}}$) with high accuracies (Table 5.4(a)). Hence, like example 1, in this example also, the Pareto-based multiobjective was advantageous to use, which provided the option to choose the best solution from a Pareto-front. Fig. 5.3 illustrate a Pareto-front garbed during the multiobjective training of HFIT.

Table 5.4(b) describe the comparison between several algorithm on clean training and test set, where the result of IFRS, AFRS, H-TS-FS1, and H-TS-FS2 was collected from [375], RBF-AFA, HyFIS, D-FNN, and SuPFuNIS from [359], NNT1FW and NNT2FW from [346], and T2FLS (singleton), T2FLS (TSK), and SEIT2FN from [359].

The training and test RMSEs and the number of parameter count $k(\mathbf{w})$ were used for comparing the algorithms, which is shown in Table 5.4(b). In T1FIS comparisons, it was found that the proposed algorithms T1HFIT$^{\mathrm{S}}$ and T1HFIT$^{\mathrm{M}}$ performed better than the algorithms NNT1FW, AFRS, IFRS, H-TS-FS, RBF-AFA, and HyFIS. The algorithms D-FNN and SuPFuNIS had better test RMESs $E_t = 0.008$ and $E_t = 0.005$, but their parameter counts were larger since the number of rules in each case was 10. Since each

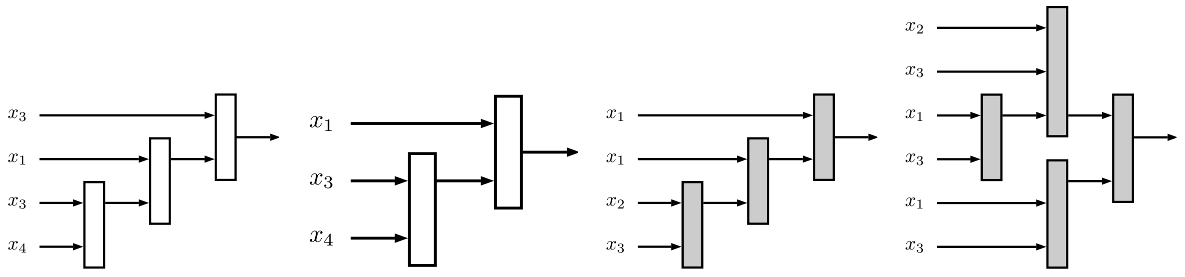
Table 5.4 Performance evaluation on clean set of noisy chaotic time series (Example-2)

(a) Performance statistics (10 repetitions)

		T1HFIT^{S}	T1HFIT^{M}	T2HFIT^{S}	T2HFIT^{M}
Training					
E_n	Best	0.0115	0.0115	0.0108	0.0032
	Mean	0.0345	0.0338	0.0413	0.0224
	STD	0.0163	0.0207	0.0221	0.0203
r_t	Best	1.00	1.00	1.00	1.00
	Mean	0.9858	0.9864	0.9783	0.9912
	STD	0.0117	0.0107	0.0182	0.0154
Test					
E_t	Best	0.0122	0.0119	0.0086	0.0058
	Mean	0.0414	0.0356	0.0427	0.0275
	STD	0.0224	0.0173	0.0234	0.0207
r_t	Best	1.00	1.00	1.00	1.00
	Mean	0.9786	0.9850	0.9769	0.9888
	STD	0.0211	0.0120	0.0195	0.0158
Parameters					
$k(\mathbf{w})$	Best	20	40	72	36
	Mean	71.6	57.6	203.4	129.5

(b) Performance Comparison

	Algorithm	E_n	E_t	$k(\mathbf{w})$
Type–1	NNT1FW	–	0.0550	–
	AFRS	0.0267	0.0256	78
	IFRS	0.0240	0.0253	58
	H-TS-FS[1]	0.0120	0.0129	148
	H-TS-FS[2]	0.0145	0.0151	46
	RBF-AFA	–	0.0128	–
	HyFIS	–	0.0100	–
	D-FNN	–	0.0080	–
	SuPFuNIS	–	0.0057	–
	$\mathbf{T1HFIT}^{\mathbf{S}}$	0.0115	0.0122	60
	$\mathbf{T1HFIT}^{\mathbf{M}}$	0.0115	0.0119	40
Type–2	T2FLS (singleton)	–	0.0426	–
	T2FLS (TSK)	–	0.0431	–
	NNT2FW	–	0.0390	–
	SEIT2FNN[1]	–	0.0034	–
	SEIT2FNN[2]	–	0.0053	–
	$\mathbf{T2HFIT}^{\mathbf{S}}$	0.0108	0.0086	108
	$\mathbf{T2HFIT}^{\mathbf{M}}$	0.0032	0.0058	118



(a) T1HFIT^{S}: $E_n = 0.0115$ (b) T1HFIT^{M}: $E_n = 0.0115$ (c) T2HFIT^{S}: $E_n = 0.0108$ (d) T2HFIT^{M}: $E_n = 0.0032$

Fig. 5.6 Example–2 clean set: designed HFIT. The shaded nodes are T2FIS.

T1FS MF has at least two parameters and each rule has three free parameters at the consequent part, the number of parameter count $k(\mathbf{w})$ for two input variables stands to at least 70 (this is an approximate calculation since the algorithms may have other parameters). Whereas, the algorithms T1HFIT^{S} and T1HFIT^{M} had parameter count $k(\mathbf{w})$ equal to 60 and 40, respectively.

In T2FIS, the proposed algorithms clearly performed better than T2FLS (singleton), T2FLS (TSK), and NNT2FW. Whereas, the performance of the proposed algorithms were competitive with SEIT2FNN[1] (without fuzzy set reduction) and SEIT2FNN[2] (with fuzzy set reduction) whose test RMSEs E_t were 0.0034 and 0.0058, respectively. The algorithm SEIT2FNN[1] had 28 fuzzy sets and SEIT2FNN[2] had 16 fuzzy sets (reduced), and each of these had seven rules. Hence, the parameter count $k(\mathbf{w})$ of these algorithms stands to at least 126 and 90, respectively. On the other hand, the proposed algorithm T2HFIT^{S}

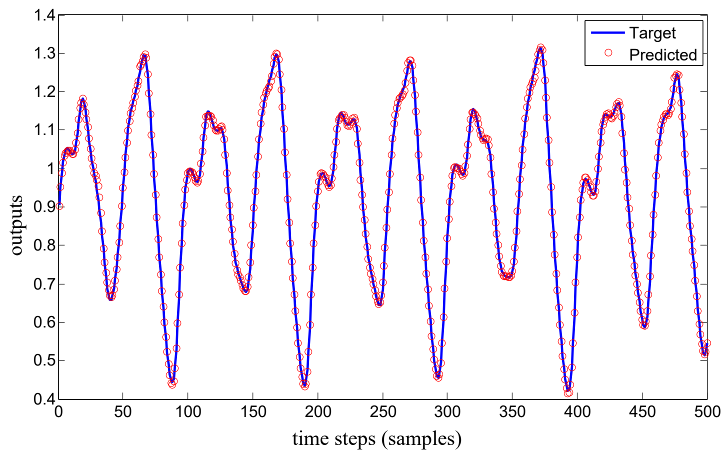


Fig. 5.7 Example–2 Clean set: target versus predicted test values. The test outputs belong to algorithm T2HFIT$^{\mathrm{M}}$ that has the test RMSE $E_t = 0.0058$.

had test RMSE $E_t = 0.0086$ (slightly larger than SEIT2FNN^1 and SEIT2FNN^2), but the parameter count $k(\mathbf{w})$ was 108, which is smaller than SEIT2FNN^1. Similarly, the proposed algorithm T2HFIT$^{\mathrm{M}}$ had test RMSE $E_t = 0.0058$, which is very close to SEIT2FNN^2 and the number of parameter count $k(\mathbf{w})$ was smaller than SEIT2FNN^1 and closer to SEIT2FNN^2. Fig.5.6 illustrates the hierarchical structure of the obtained models using the proposed algorithms. In Fig. 5.7, the target versus prediction plot of test data samples are illustrated.

Case–noisy set

The performances of the proposed algorithms were further evaluated for the noisy patterns. Therefore, three training sets and three test sets were generated by adding Gaussian noise with a mean 0 and standard deviation (STDs) of 0.1, 0.2, and 0.3 to the original data $x(k)$. These noisy training sets (with STDs 0.1, 0.2, and 0.3) were presented for the training of the proposed algorithms. With each training set of STDs 0.1, 0.2, and 0.3, three test sets were presented for testing: clean, STD 0.1, and STD 0.3. The obtained results were compared with the results reported in the literature (Table 5.5).

Table 5.5 describes the comparisons between the results of the algorithms, where the results of SONFIN and SVR-FM were collected from [404], DyEFuNN and EFuNN from [403], SEIT2FNN, T2FLS-G, IT2FNN-SVR(N), and IT2FNN-SVR(F) from [404], and eT2FIS from [364]. It is clear from the comparison of the results that the proposed algorithms performed efficiently for the noisy datasets, and the obtained models were less complex (particularly when T1FISs were compared) than the other models listed in Table 5.5. Moreover, in each case of noisy data (STD 0.1, STD 0.2, and STD 0.3), the proposed algorithms had smaller parameter count $k(\mathbf{w})$ and had smaller training RMSE E_t than the other listed algorithms. In T1FIS comparisons, the SONFIN had slightly better

Table 5.5 Example 2–noisy set: performance comparison

		Train	Test				Train	Test				Train	Test			
FIS	Algorithm	0.1	clean	0.1	0.3	$k(\mathbf{w})$	0.2	clean	0.1	0.3	$k(\mathbf{w})$	0.3	clean	0.1	0.3	$k(\mathbf{w})$
Type–1	SVR-FM	0.128	0.045	0.087	0.200	1127	0.229	0.089	0.109	0.189	1127	0.332	0.138	0.147	0.198	1127
	EFuNN	0.126	–	–	–	–	0.252	–	–	–	–	0.366	–	–	–	–
	DyEFuNN	0.116	–	–	–	–	0.214	–	–	–	–	0.306	–	–	–	–
	SONFIN	0.113	0.054	0.108	0.256	130	0.226	0.116	0.138	0.280	130	0.302	0.195	0.208	0.305	130
	T1HFIT$^{\mathrm{S}}$	0.127	0.050	0.140	0.363	60	0.234	0.111	0.153	0.349	104	0.305	0.100	0.159	0.356	64
	T1HFIT$^{\mathrm{M}}$	0.128	0.042	0.138	0.357	40	0.225	0.085	0.145	0.360	84	0.307	0.119	0.162	0.351	60
Type–2	T2FLS-G	0.133	0.074	0.103	0.220	110	0.238	0.125	0.132	0.200	110	0.357	0.232	0.234	0.264	110
	IT2FNN-SVR(N)	0.128	0.048	0.087	0.193	103	0.234	0.085	0.105	0.186	103	0.349	0.127	0.138	0.188	103
	IT2FNN-SVR(F)	0.127	0.046	0.088	0.215	103	0.233	0.083	0.103	0.180	103	0.347	0.121	0.131	0.184	103
	SEIT2FNN	0.123	0.049	0.097	0.212	110	0.225	0.083	0.113	0.228	110	0.319	0.196	0.197	0.254	110
	eT2FIS	0.120	0.059	0.107	0.214	–	0.225	0.083	0.132	0.247	–	0.327	0.102	0.152	0.278	–
	T2HFIT$^{\mathrm{S}}$	0.128	0.039	0.135	0.355	108	0.227	0.079	0.143	0.348	82	0.314	0.100	0.148	0.354	144
	T2HFIT$^{\mathrm{M}}$	0.123	0.042	0.135	0.365	72	0.233	0.087	0.144	0.348	72	0.311	0.097	0.148	0.356	108

RMSE, but the number of parameters counts $k(\mathbf{w})$ was larger than the proposed algorithms T1HFIT$^{\mathrm{S}}$ and T1HFIT$^{\mathrm{M}}$. Similarly, in T2FIS comparison, the algorithm eT2FIS had slightly better RMSE than the other listed algorithms, but the models obtained using the proposed algorithms were of low complexity, i.e., had small parameter count $k(\mathbf{w})$.

5.4.3 Example 3—miles-per-gallon prediction problem

To evaluate the performance of the proposed algorithms, a real-world MPG problem was used. The objective of this example was to predict or estimate the city-cycle fuel consumption in MPG. The MPG dataset was collected from the UCI machine learning repository [309]. This dataset has 392 sample each of which has six input variables, but in this example, as mentioned in [366], three variables (x_1 = weight, x_2 = acceleration, and x_3 = model year) were selected. In the training process of the algorithms, 50% (196 patterns) samples were randomly selected for training and the rest of the 50% (196 patterns) samples were taken for testing. Such process of training-set and the test-set selection was repeated 10 times. The performance statistics is shown in Table 5.6(a). The performances of the proposed algorithms were compared with the literature (Table 5.6(a)). However, the algorithms chosen from the literature were tested over fewer test samples. Therefore, the comparison shown in Table 5.6(a) was limited to the comparison of the training RMSEs.

To compare, the result of T1FLS was collected from [360] and the results of SEIT2FNN, RIT2NFS-WB, McIT2FIS-UM, and McIT2FIS-US were collected from [366]. The comparisons of the models in this example were based on the mean training and test RMSE E_n and E_t obtained for 10 repetitions. However, the comparison on test RMSEs was limited since only 120 sample were used for testing by the algorithms considered from literature.

Table 5.6 Performance evaluation on miles-per-gallon prediction problem (Example-3).

(a) Performance Statistics (10 repetitions)

		$T1HFIT^S$	$T1HFIT^M$	$T2HFIT^S$	$T2HFIT^M$
Training					
E_n	Best	1.8931	2.2686	2.0881	1.9582
	Mean	2.7115	2.6037	2.4699	2.4052
	STD	0.5144	0.4071	0.4461	0.3774
r_n	Best	0.97	0.96	0.96	0.96
	Mean	0.921	0.941	0.946	0.950
	STD	0.1035	0.0218	0.0244	0.0160
Test					
E_t	Best	2.7550	2.7907	2.8383	2.6623
	Mean	4.2333	3.3349	3.4006	3.3172
	STD	0.5024	0.5720	0.7423	0.6855
r_t	Best	0.97	0.96	0.96	0.96
	Mean	0.921	0.941	0.946	0.950
	STD	0.1035	0.0218	0.0244	0.0160
Parameters					
$k(\mathbf{w})$	Best	20	20	108	118
	Mean	132	78.8	224	207.4

(b) Performance Comparison (10 repetitions)

	Algorithm	Mean E_n	STD	Mean E_t	STD	Samples
Type–1	T1FLS	–	–	3.5960	–	120
	$T1HFIT^S$	2.7115	0.5144	4.2333	0.5024	196
	$T1HFIT^M$	2.6037	0.4071	3.3349	0.5720	196
Type–2	McIT2FIS-US	2.7358	–	2.6770	–	120
	SEIT2FNN	2.7161	–	2.7895	–	120
	McIT2FIS-UM	2.6524	–	2.6486	–	120
	RIT2NFS-WB	2.3685	–	2.7807	–	120
	$T2HFIT^S$	2.4699	0.4461	3.4006	0.7423	196
	$T2HFIT^M$	2.4052	0.3774	3.3172	0.6855	196

Whereas, the algorithms proposed in this work used 196 samples for testing (Table 5.6(b)). It was observed that the proposed algorithms $T2HFIT^S$ and $T2HFIT^M$ outperformed all the other algorithms except RIT2NFS-WB, which had slightly better training RMSE $E_n = 2.3685$ in comparison to the training RMSE $E_n = 2.4699$ and $E_n = 2.4052$ of $T2HFIT^S$ and $T2HFIT^M$ respectively. Since the performance comparisons were based on the average of 10 repetitions, the models hierarchical structures are not presented for this example.

5.4.4 Example 4—abalone age prediction

In this example, a prediction problem was taken, in which the age of a person was predicted based on the physical measurements. The abalone dataset was collected from the UCI machine learning repository [309]. It has 4177 data samples and each of which has seven input variables (x_1 = length, x_2 = diameter, x_3 = height, x_4 = whole weight, x_5 = shucked weight, x_6 = viscera weight, and x_7 = shell weight) and one output variable (rings). To train the proposed algorithms, 80% (3342 patterns) samples were randomly taken for training and the rest of 20% (835 patterns) samples were taken for testing. Such training process was repeated 10 times, and the collected results are summarized in Table 5.7(a).

The obtained results are compared with the results reported in the literature (Table 5.7(b)). For the comparisons, algorithms 'General', HS, CCL, and 'Chen&Cheng' were collected from [360], and the results of SEIT2FNN, RIT2NFS-WB, McIT2FIS-UM, and McIT2FIS-US were collected from [366]. The algorithms 'General' [408], CCL [409],

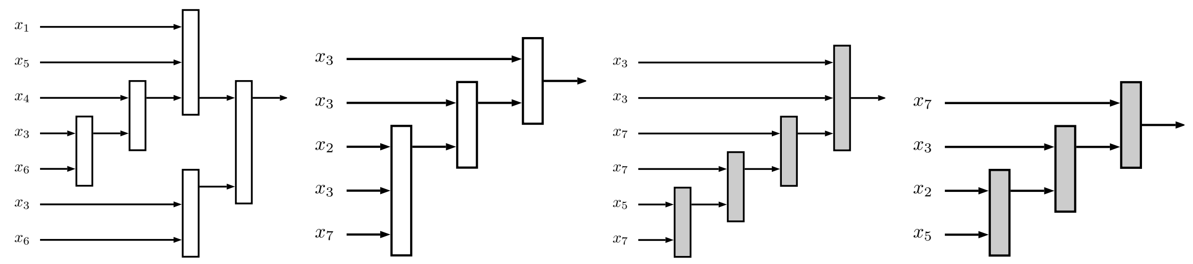
Table 5.7 Performance evaluation on abalone age prediction problem (Example-4).

(a) Performance Statistics (10 repetitions)

		T1HFIT^{S}	T1HFIT^{M}	T2HFIT^{S}	T2HFIT^{M}
Training					
E_n	Best	2.1097	2.2857	2.1154	2.1275
	Mean	2.3267	2.4284	2.2597	2.2404
	STD	0.1534	0.1079	0.1478	0.0627
r_n	Best	0.76	0.71	0.76	0.75
	Mean	0.688	0.655	0.710	0.716
	STD	0.0490	0.0347	0.0481	0.0204
Test					
E_t	Best	2.1260	2.3480	2.1824	2.1428
	Mean	2.3644	2.4843	2.3808	2.3533
	STD	0.1448	0.1029	0.1676	0.1127
r_t	Best	0.76	0.71	0.76	0.75
	Mean	0.688	0.655	0.710	0.716
	STD	0.0490	0.0347	0.0481	0.0204
Parameters					
$k(\mathbf{w})$	Best	20	20	144	72
	Mean	77.6	46.4	188.4	152.9

(b) Performance Comparison

	Algorithm	E_n	E_t	$k(\mathbf{w})$
Type–1	HS	–	3.1600	–
	General	–	3.1500	–
	CCL	–	2.6500	–
	Chen&Cheng	–	2.5900	–
	T1HFIT^{S}	2.1097	2.1260	124
	T1HFIT^{M}	2.2857	2.3480	84
Type–2	RIT2NFS-WB	2.4047	2.1346	131
	McIT2FIS-UM	2.3481	1.8740	115
	SEIT2FNN	2.3388	2.4330	140
	McIT2FIS-US	2.3357	1.8387	115
	T2HFIT^{S}	2.1154	2.1824	226
	T2HFIT^{M}	2.1275	2.1428	108



(a) T1HFIT^{S}: E_n = 2.1097 (b) T1HFIT^{M}: E_n = 2.2857 (c) T2HFIT^{S}: E_n = 2.1154 (d) T2HFIT^{M}: E_n = 2.1275

Fig. 5.8 Example–4: designed HFIT. The shaded nodes are T2FIS.

HS [410], and WFRI-GA [411] were fuzzy interpolate reasoning methods, where WFRI-GA was based on genetic algorithm and the algorithm 'General' implemented Mamdani type FIS. It is evident from the results in Table 5.7(b) that the proposed algorithms (both T1FIS and T2FIS) had outperformed the algorithms considered for comparisons. The best-performing models are illustrated in Fig. 5.8, where the selected input feature is indicated by x_i.

5.4.5 Example 5—box-Jenkins gas furnace problem

In this example, the Box and Jenkins gas furnace dataset that was taken from [412], which has 296 data samples. The objective of this example was to predict the CO$_2$ concentration from the gas-flow rate. The system for the gas furnace is modeled using a series, which is of the form: $y(k) = f(y(k-1), u(k-4)$. For the training of the proposed

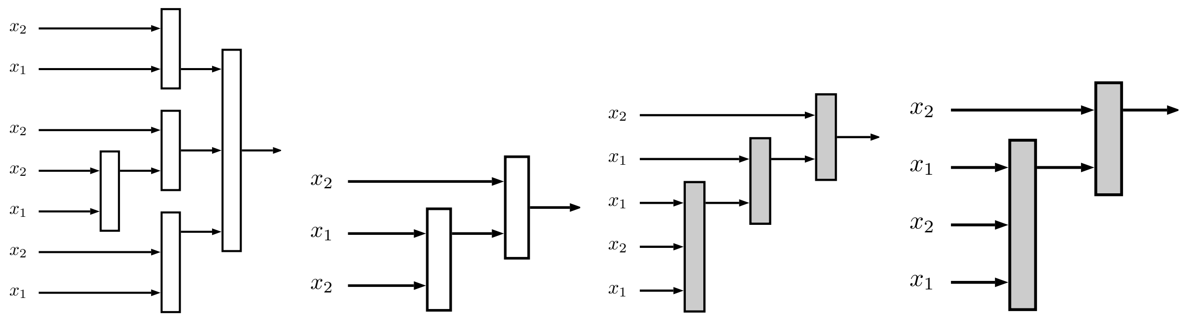
Table 5.8 Performance evaluation on box-Jenkins gas concentration problem (Example-5).

(a) Performance statistics (10 repetitions)

		T1HFIT^S	T1HFIT^M	T2HFIT^S	T2HFIT^M
Training					
E_n	Best	0.246	0.280	0.256	0.275
	Mean	0.303	0.344	0.291	0.301
	STD	0.036	0.043	0.023	0.033
r_n	Best	0.97	0.97	0.97	0.97
	Mean	0.959	0.947	0.963	0.960
	STD	0.010	0.013	0.006	0.010
Parameters					
$k(\mathbf{w})$	Best	40	40	72	72
	Mean	132.8	58.4	286	167.4

(b) Performance comparison

	Algorithm	E_n	$k(\mathbf{w})$
Type–1	T1-NFS	0.4074	–
	GNN^1	0.3114	–
	GNN^2	0.2983	–
	T1HFIT^S	0.2455	124
	T1HFIT^M	0.2838	40
Type–2	SEIT2FNN	0.2690	152
	RIT2NFS-WB	0.3527	90
	McIT2FIS-UM	0.3139	48
	McIT2FIS-US	0.3181	48
	T2HFIT^S	0.2767	154
	T2HFIT^M	0.2840	72



(a) T1HFIT^S: $E_n = 0.2455$ (b) T1HFIT^M: $E_n = 0.2838$ (c) T2HFIT^S: $E_n = 0.2767$ (d) T2HFIT^M: $E_n = 0.2840$

Fig. 5.9 Example–5: designed HFIT. The shaded nodes are T2FIS.

models, 100% (296 patterns) samples were used. The training process was repeated for 10 times, and the collected results are summarized in Table 5.8(a). The performances of the proposed algorithms were compared with the performances of the algorithms reported in the literature (Table 5.8(b)).

To compare the performance of the algorithms, the results of T1-NFS and GNN were collected from [360], and the results of SEIT2FNN, RIT2NFS-WB, McIT2FIS-UM, and McIT2FIS-US were collected from [366]. As reported in Table 5.8(b), the proposed algorithms clearly outperformed the algorithms T1-NFS, GNN^1, and GNN^2 for T1FIS comparisons, and the algorithms SEIT2FNN, RIT2NFS-WB, McIT2FIS-UM, and McIT2FIS-US for T2FIS comparisons. The best-performing models are illustrated in Fig. 5.9.

5.5 Performance of HFIT on real-world application

5.5.1 Poly (lactic-co-glycolic acid) (PLGA) micro- and nanoparticle's dissolution-rate prediction modeling

This example illustrates a pharmaceutical industrial problem related to PLGA dissolution profile prediction, which is a complex problem since a huge number of factors governs its dissolution-rate profile. As per the dataset provided in [413, 13, 12], this problem has a total of 300 potential factors that influences the PLGA protein particle's dissolution-rate [414], The input features are categorized into four groups (protein descriptor, plasticizer, formulation characteristics, and emulsifier), which has 85, 17, 98, 99, and 1 features, respectively. This problem has a very high noise and redundancy because data were obtained from various experimental measurements and instruments.

The significance of the PLGA dissolution profile prediction can be understood from the following description. PLGA micro- and nanoparticles play a significant role in the medical application and toxicity evaluation of PLGA-based multi-particulate dosages [415]. PLGA micro-particles are important diluents used to produce drugs in their correct dosage form. Apart from playing the role as a filler, PLGA as an excipient, and alongside pharmaceutical APIs, plays other crucial roles in various ways. It helps in the dissolution of drugs, thus increasing the absorbability and solubility of drugs [416, 417]. It helps in pharmaceutical manufacturing processes by improving API powders' flow and non-stickiness.

The dataset collected from various academic literature contains 300 input features categorized into four groups, including protein descriptor, plasticizer, formulation characteristics, and emulsifier (Table 5.9). For example, the formulation characteristics group contains features such as PLGA-inherent viscosity, PLGA molecular weight, lactide-to-glycolide ratio, inner and outer phase polyvinyl alcohol (PVA) concentration, PVA molecular weight, inner phase volume, encapsulation rate, mean particle size, PLGA concentration, and experimental conditions (dissolution pH, the number of dissolution additives, dissolution additive concentration, and production method and dissolution time). The protein descriptor, plasticizer, and emulsifier feature groups contain 85, 98, and 101 features, respectively. The regression model sought to predict the dissolution percentage or solubility of PLGA, which is dependent on the features mentioned above. In order to avoid over-fitting, collected data were preprocessed by adding noise to them. The dataset was then normalized, in other words, scaled within the range –1.0 and 1.0.

Table 5.9 PLGA dataset description

Sl No	Group name	Features	Importance
1	Protein descriptors	85	Describes the type of molecules and proteins used
2	Formulation characteristics	17	Describe the molecular properties, e.g., molecular weight, particle size, etc.
3	Plasticizer	98	Describe the properties such as fluidity of the material used
4	Emulsifier	99	Describe the properties of stabilizing the pharmaceutical product life
5	Time in days	1	Time taken to dissolve
6	% of molecules dissolved	1	Output

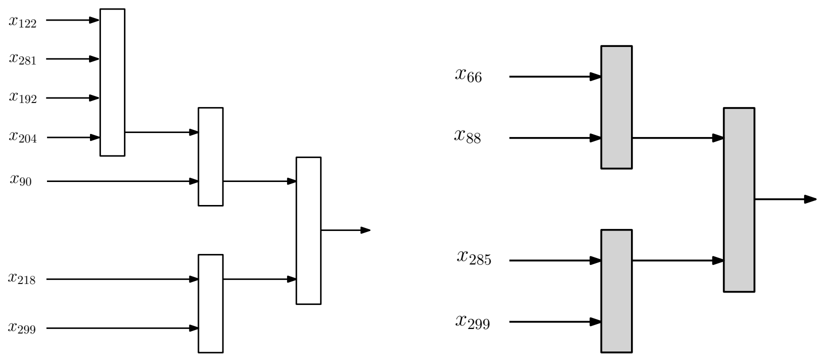
5.5.2 PLGA predictive modeling results

Using the parameter-setting mentioned in Table 5.1 and using 10-fold cross-validation, the proposed algorithm T1HFIT$^{\mathrm{M}}$ was able to select seven input features [PVA Mw (x_{90}), ASA (x_{122}), pH 8 msdon (x_{192}), aromatic bond count (x_{204}), a(xx) (x_{218}), pH 12 msacc (x_{281}), time days (x_{299})] and was able to approximate a test RMSE of $E_t = 18.66$. Similarly, the proposed algorithm T2HFIT$^{\mathrm{M}}$ was able to approximate a test RMSE of $E_t = 15.259$ with only four input features [aromatic atom count (x_{66}), PVA conc inner phase (x_{88}), pH 1 msdon (x_{285}), time days (x_{299})].

The feature reduction is a significant task since it reduces the drug manufacturing cost. Table 5.10 shows a comparison of the results of the proposed T1HFIT$^{\mathrm{M}}$ and T2HFIT$^{\mathrm{M}}$ with the results of the algorithms such as multilayer preceptron (MLP), reduced error pruning tree (REP Tree), heterogeneous flexible neural tree (HFIT), and Gaussian process regression (GPR). It is evident from the results that the proposed algorithm predicted the PLGA dissolution profile with a lower number of features, and its approximation error was very competitive with the performance of other algorithms. Fig. 5.10 illustrates the obtained models for the prediction of PLGA dissolution profile.

Table 5.10 Performance comparison.

Algorithm	Ref.	RMSE E_t	No. of Features
MLP	[413]	14.3	17
HFIT	[291]	13.2	15
REP Tree	[14]	13.3	15
GPR	[14]	14.9	15
MLP	[14]	15.2	15
MLP	[413]	15.4	11
T1HFIT$^{\mathrm{M}}$	This Chapter	18.6	7
T2HFIT$^{\mathrm{M}}$	This Chapter	15.2	**4**



(a) T1HFIT$^{\mathrm{M}}$: $E_t = 18.66$ (b) T2HFIT$^{\mathrm{M}}$: $E_t = 15.25$

Fig. 5.10 Example–6: designed HFIT. The shaded nodes are T2FIS.

5.6 Summary

The proposed HFIT algorithms T1HFIT$^{\mathrm{S}}$, T1HFIT$^{\mathrm{M}}$, T2HFIT$^{\mathrm{S}}$, and T2HFIT$^{\mathrm{M}}$ were evaluated over six examples including a real-world from the pharmaceutical industry. Performance of the proposed algorithms were compared with the algorithms that offer structure optimization (e.g., SEIT2FNN, SONFIN, SaFIN, TSCIT2FNN, etc.), the algorithms that offer hierarchical fuzzy system design (e.g., IFRS, H-TS-FS, etc.), the algorithms that offer dynamic fuzzy system design (e.g., DyEFuNN, D-FNN, etc.), and so forth. The obtained results illustrate the efficiency of the proposed algorithms in comparison to the algorithms collected from the literature. Such performance was obtained by using the parameter-setting mentioned in Table 5.1. Moreover, a comparison using a noisy data [example 2, case 2 (Section 5.4.2)] has proved the approximation efficiency of the proposed algorithms over other algorithms. The HFIT algorithms not only offer the solutions with high accuracy (low approximation error), but they also offer the solutions with low complexity.

The number of clusters needs to be predetermined in a clustering-based partitioning of the input space. Hence, the proposed HFIT, which automatically partition the input space by using the dynamics of the evolutionary process, is particularly significant for the predictive modeling of the problems that have a large number of input features such as example 6 (Section 5.5.1).

In Section 5.3, a comprehensive study of the comparative results (of the proposed algorithms) was presented. It was observed that the proposed HFIT based algorithms gave better performance than the other algorithms available in the literature. For example, in the case of example–1 T1HFIT$^{\mathrm{M}}$ offered better RMSE with lower parameter count. Additionally, T2HFIT$^{\mathrm{M}}$ offered a better RMSE (i.e., 0.0028) with a low complexity (i.e., 72) in comparison to the model SEIT2FNN that gave an RMSE 0.0053. Similarly, for example-2, T2HFIT$^{\mathrm{M}}$ offered a very competitive RMSE (i.e., 0.0058) in comparison to

Table 5.11 Performance summary: single objective versus multiobjective and type-1 versus type-2.

	Single Objective				Multibjective			
	T1HFIT^{S}		T2HFIT^{S}		T1HFIT^{M}		T2HFIT^{M}	
Example	E_n	$k(\mathbf{w})$	E_n	$k(\mathbf{w})$	E_n	$k(\mathbf{w})$	E_n	$k(\mathbf{w})$
1	0.018	57.2	0.012	152.0	0.025	34.4	0.018	90.4
2	0.034	71.7	0.041	203.4	0.033	57.6	0.022	129.5
3	2.711	132.0	2.469	224.0	2.603	78.8	2.405	204.4
4	2.326	77.6	2.259	188.4	2.428	46.4	2.242	152.9
5	0.303	138.8	0.291	286.0	0.344	58.4	0.301	167.4
Average	1.078	95.5	1.014	210.7	1.086	55.1	0.997	148.9

the model SEIT2FNN^2 that gave an RMSE 0.0053. Additionally, the comparison on the noisy dataset, the proposed model offered T2HFIT^{M} offered better training RMSEs with low model complexity in comparison to many of the recently proposed T2FIS algorithms such as SEIT2FNN, IT2FNN-SVR, and T2FLS-G. Moreover, the models developed by the proposed algorithm adapted its structure in each instance of experiments on the noisy dataset; whereas, the other models had a fixed structure in each instance of their experiments (Table 5.5). Therefore, the proposed algorithm was able to accommodate the variance in noise more precisely than the other models. The cases of example–3, example–4, and example–5, the proposed models surpassed the other algorithms such as RIT2NFS-WB, McIT2FIS, and SEIT2FNN. Accordingly, TlHFIT^{M} had performed better than its counterparts. In the case of example–6, the proposed T2HFIT^{M} was highly efficient than the T1HFIT^{M} because T2HFIT^{M} was capable of accommodating the noisy information more efficiently than T1HFIT^{M}, which is evident from the fact that the average RMSE of T2HFIT^{M} was 16.64, and the average RMSE of T1HFIT^{M} was 22.36. Hence, the use of interval type-2 MFs at was worth considering in such kind of high-dimensional and noisy application/problems.

A comparison between single objective and multiobjective, summarized in Table 5.11, suggest that the multiobjective approach has performance superiority over the single objective because multiobjective gives a competitively better approximation error with lower model complexity in both type-1 and type-2 cases. Additionally, it can be observed that the type-2 offers better approximation error against type-1.

Since HFIT algorithms are developed using evolutionary process, the quality of its performance is subjected to the careful setting of parameters (Table 5.1). Hence, results of the algorithms mentioned in this chapter may be further improved with the choice of the different sets of parameters; however, this is a trial-and-error process. For example, the feature selection, i.e., the number of inputs to a node (a fuzzy subsystem) is proportional to the setting of the maximum inputs to a FIS node. Similarly, the hierarchy (number of layers) in an HFIT is proportional to the setting of maximum depth of a tree. Therefore, the complexity of the HFIT can be controlled using these parameters. Additionally, the parameters of MOGP and DE such as their population size, crossover probability, mutation probability, etc. influence the performance of HFIT.

Moreover, FIS for data mining inherently requires a multiobjective solution and the proposed multiobjective design of HFIT stands as a viable option that constructs a tree-like model whose nodes are low-dimensional FIS. The proposed HFIT was developed for both type-1 and type-2 FIS and each node in HFIT implements a TSK model. Both type-1 and type-2 FIS were studied in the scope of single objective and multiobjective optimization using GP. Hence, four versions of HFIT were studied: $\mathrm{T1HFIT^S}$, $\mathrm{T1HFIT^M}$, $\mathrm{T2HFIT^S}$, and $\mathrm{T2HFIT^M}$. The parameters of the MFs and the consequent parts of the rules were tuned using DE algorithm. The optimization procedure of HFIT was a coevolutionary approach, in which structure optimization and parameter optimization was applied one-by-one until a formidable solution was obtained. A comprehensive performance comparison was performed for evaluating the efficiency of the proposed HFIT. The performance of the proposed HFIT algorithm was found to be very efficient and competitive in comparison to the algorithm collected from the literature. Additionally, HFIT can do feature selection. Hence, it is advantageous to use HFIT to solve high-dimensional application problems.

Chapter 6

Conclusions and future research

In this thesis, two novel algorithms for data-driven modeling, i.e., for the feature selection and function approximation were proposed. The objective was to design the adaptive algorithms that can provide an efficient alternative for the data-driven modeling to this date. The conclusions and future research direction derived from the thesis are as follows.

6.1 Conclusions

For the development of the proposed algorithms, feedforward neural network (FNN) and fuzzy inference system (FIS) paradigms were investigated. FNN and FIS faces challenges in predictive modeling for data that are insufficient, imbalanced, high-dimensional, and abundant. Thus, in the literature, several FNN, and FIS based approximation methods has been proposed. Moreover, the components of FNN (weights, structure, activation function, and learning algorithm) and FIS (rules base and membership function) are responsible for design of various algorithms (Chapters 2 and 4). Hence, this thesis proposed to optimize these components of FNN and FIS using an adaptive tree-like model and by using a coevolutionary approach for optimizing the tree. The proposed algorithms were the heterogeneous flexible neural tree (HFNT) and the hierarchical fuzzy inference tree (HFIT). The proposed algorithms were adaptive the following sense: the tree structure were determined automatically (by using the principle of natural selection), nodes of the tree were adapted automatically, and tree parameters were tuned.

Moreover, structure optimization and generalization ability are strongly correlated with each others. Therefore, FNN architecture optimization and for that matter various model designs were of particular interest to the research community. Additionally, FNN node optimization, learning parameter optimization, hybrid metaheuristic training to FNN, and multiobjective treatment to FNN optimization were of particular focus in the past. The proposed HFNT algorithm is an adaptive algorithm that accommodates all these forms of optimization strategies into a single entity (function approximation algorithm).

Similarly, for a connection-based FIS model (e.g., neuro-fuzzy and hierarchical fuzzy systems), the design of the structure holds the key to its approximation ability. Moreover, parameter tuning of FIS rules and the interpretability–accuracy trade-off is utmost important issues in constructing a FIS model. The proposed HFIT algorithm is an adaptive algorithm that addresses multiobjective optimization of FIS structure by evolving a tree -like model, and parameter tuning by using metaheuristic in a coevolutionary manner.

HFNT was a FNN based model, which was a tree-like model whose nodes were neural nodes (Chapter 3). On the other hand, HFIT was a connection-based hierarchical FIS model (tree-like model) whose nodes were low-dimensional FIS of type-1 or type-2 (Chapter 5). The multiobjective evolutionary algorithm minimized the approximation error and the model's complexity simultaneously. Thus, the obtained tree structure of HFNT and HFIT were simple (small in the parameter count). Finally, the parameters were tuned by using differential evolution algorithm that further improved the model's approximation ability.

Both the algorithms were compared with the algorithms from the literature. The obtained results were the evidence of their (the proposed algorithms) superior performance over other algorithms. Moreover, the HFNT was used for modeling a real-world problem of pharmaceutical industry that had data that represented the dynamic environment of the pharmaceutical die filling process. It was found the HFNT offered significantly better result than other computational intelligence models such as multilayer perceptron and reduced error pruning tree. Similarly, the performance of HFIT (its version type-1 HFIT and type-2 HFIT) when tested over benchmark and real-world problems, it surpasses its competitor algorithms in the literature. Additionally, HFIT was used for modeling a real-world pharmaceutical problem that had a high-dimensional data that governed the prediction of drug desolation rate. It was found that the HFIT had a high prediction ability with a small set of features than other algorithms.

FNN and FIS are known as a universal approximator and it is known that for any problem (X, Y), a FIS model can be designed, which can approximate (X, Y) to equal degree that of a FNN model can, and vice versa [418]. Such equivalence of their approximation ability is evident from the results obtained in the experiments conducted in this thesis. The approximation ability of the HFNT and HFIT was fond very similar when they were used for predicting two problems: Mackey-Glass chaotic time series prediction and PLGA drug dissolution prediction. For the former, the approximation quality of the HFIT was found slightly better than the HFNT. On the other hand, the approximation quality of the HFNT was found slightly better than the HFIT for the latter. However, a significant improvement in feature selection was observed during HIT modeling because it was able to predicate to an equal degree of HFNT by using fewer number of features.

The proposed algorithms are capable of feature selection using the dynamics of the natural section and are able to approximate to a very high degree. In addition to that, they offer very simple models. These are the desirable significant attributes to an efficient, effective, and robust algorithm. Therefore, the proposed algorithm stands as a viable alternative to the research community to date.

6.2 Future research directions

The proposed algorithms are efficient tools, which can be used for solving several high-dimensional and noisy real-world problems (Section 2.6 describes data related challenges, which HFNT and HFIT can address to a larger extent).

In this thesis, the tree structure was tuning using multiobjective genetic programming and the parameters were optimized by differential evolution. However, hybrid metaheuristic approach may be advantageous to use (Section 2.3.2 describes advantages of hybrid metaheuristics approach). Similarly, the model's structure, in this thesis, was represented as a tree-like. However, structure representation is an open research problem.

The proposed algorithms may be enhanced, modified to suit a particular class of problem, i.e., the nodes of the HFNT and HFIT may be replaced or designed according to the specification of the possible problems (Section 2.3.1 describes the various directions for the node design and optimization). Moreover, the fuzzy rules at the nodes of HFIT can be further optimized by using iterative learning methods or by using Pittsburgh approach (Section 4.3.1 describes methods of rule learning).

Evolving fuzzy paradigm indicates an opportunity to extend HFNT and HFIT algorithms towards dynamic learning system, in which a dynamic mechanism may be employed to expand and contract a tree such that it can learn the knowledge contained in the incoming (on-line stream) data. Therefore, it can efficiently solve a broad range of dynamic problems (problems that has non-stationary/stream data).

References

[1] T. Takagi and M. Sugeno, “Fuzzy identification of systems and its applications to modeling and control,” *IEEE Trans. Syst. Man Cybern.*, no. 1, pp. 116–132, 1985.

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

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

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

[5] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence.* Morgan Kaufmann, 2001.

[6] S. Das, S. S. Mullick, and P. Suganthan, “Recent advances in differential evolution–an updated survey,” *Swarm Evol. Comput.*, vol. 27, pp. 1–30, 2016.

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

[8] 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, 2005.

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

[10] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009, vol. 2.

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

[12] V. K. Ojha, K. Jackowski, A. Abraham, and V. Snášel, “Feature selection and ensemble of regression models for predicting the protein macromolecule dissolution profile,” in *2014 6th World Congr. on Nature and Biologically Inspired Comput. (NaBIC),.* IEEE, 2014, pp. 121–126.

[13] V. K. Ojha, K. Jackowski, V. Snášel, and A. Abraham, “Dimensionality reduction and prediction of the protein macromolecule dissolution profile,” in *Proc. 5th Int. Conf. Innovations in Bio-Inspired Comput. Appl. IBICA 2014.* Springer, 2014, pp. 301–310.

[14] V. K. Ojha, K. Jackowski, A. Abraham, and V. Snášel, "Dimensionality reduction, and function approximation of poly (lactic-co-glycolic acid) micro-and nanoparticle dissolution rate," *Int. J. Nanomed.*, vol. 10, p. 1119, 2015.

[15] V. K. Ojha, A. Abraham, and V. Snášel, "Metaheuristic tuning of type-II fuzzy inference systems for data mining." in *IEEE World Congr. Comput. Intell., IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE 2016).* IEEE, 2016.

[16] S. Abdelwahab, V. K. Ojha, and A. Abraham, "Neuro-fuzzy risk prediction model for computational grids," in *Proc. 2nd Int. Afro-Eur. Conf. Ind. Adv. AECIA 2015.* Springer, 2016, pp. 127–136.

[17] N. Ahmed, V. K. Ojha, and A. Abraham, "An ensemble of neuro-fuzzy model for assessing risk in cloud computing environment," in *Advances in Nature and Biologically Inspired Computing.* Springer, 2016, pp. 27–36.

[18] L. A. Gabralla, T. M. Wahby, V. K. Ojha, and A. Abraham, "Ensemble of adaptive neuro-fuzzy inference system using particle swarm optimization for prediction of crude oil prices," in *14th Int. Conf. Hybrid Intell. Syst. (HIS), 2014.* IEEE, 2014, pp. 141–146.

[19] 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, 2004.

[20] Y. Chen, B. Yang, J. Dong, and A. Abraham, "Time-series forecasting using flexible neural tree model," *Inform. Sci.*, vol. 174, no. 3, pp. 219–235, 2005.

[21] L. Zadeh, "Fuzzy sets," *Inform. Control*, vol. 8, no. 3, pp. 338 – 353, 1965.

[22] H. A. Hagras, "A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 4, pp. 524–539, 2004.

[23] 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.

[24] 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, 2000.

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

[26] 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, 2002.

[27] Y. Park, T. Murray, and C. Chen, "Predicting sun spots using a layered perceptron neural network," *IEEE Trans. Neural Netw.*, vol. 7, no. 2, pp. 501–505, 1996.

[28] J. Hansen and R. Nelson, "Neural networks and traditional time series methods: a synergistic combination in state economic forecasts," *IEEE Trans. Neural Netw.*, vol. 8, no. 4, pp. 863–873, 1997.

[29] 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, 2006.

[30] 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, 1997.

[31] 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, 1994.

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

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

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

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

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

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

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

[39] 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.

[40] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," in *Int. Jt. Conf. Neural Netw., 1989. IJCNN*, pp. 613–617.

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

[42] 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, 1993.

[43] 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, 2006.

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

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

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

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

[48] 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, 1982.

[49] 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.

[50] 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, 1995.

[51] 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.

[52] 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 Inform. Process. TELEMATIK*, vol. 8, no. 1, p. 39–43, 2002.

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

[54] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *IRE WESCON Conv. Rec.*, vol. 4, Aug 1960, pp. 96–104.

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

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

[57] E. W. Steyerberg, B. V. Calster, and M. J. Pencina, "Performance measures for prediction models and markers: Evaluation of predictions and classifications (in English)," *Revista Española de Cardiología*, vol. 64, no. 9, pp. 788 – 794, 2011.

[58] 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.

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

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

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

[62] 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, 2000.

[63] 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, 1992.

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

[65] 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.

[66] 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, 1992.

[67] 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, 1998.

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

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

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

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

[72] 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, 1997.

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

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

[75] 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.

[76] 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.

[77] 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.

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

[79] 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.

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

[81] 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.

[82] 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.

[83] 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, pp. 817–822.

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

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

[86] 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, 1994.

[87] S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Kalman filtering and Neural Networks.* Wiley Online Library, 2001.

[88] 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.

[89] 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.

[90] I. H. Osman and G. Laporte, "Metaheuristics: a bibliography," *Ann. Oper. Res.*, vol. 63, no. 5, pp. 511–623, 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, 1997.

[92] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Comput.*, vol. 8, no. 7, pp. 1341–1390, 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, 2002.

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

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

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

[97] 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.

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

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

[100] 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, 1999.

[101] 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, 2003.

[102] 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*, pp. 953–961 vol.2.

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

[104] 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.

[105] R. S. Sexton, B. Alidaee, R. E. Dorsey, and J. D. Johnson, “Global optimization for artificial neural networks: a tabu search application,” *Eur. J. Oper. Res.*, vol. 106, no. 2, pp. 570–584, 1998.

[106] 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, 2007.

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

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

[109] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Springer, 2015.

[110] 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.

[111] D. Whitley, T. Starkweather, and C. Bogart, “Genetic algorithms and neural networks: optimizing connections and connectivity,” *Parallel Comput.*, vol. 14, no. 3, pp. 347 – 361, 1990.

[112] 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*, pp. 2331–2336.

[113] 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.

[114] 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.

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

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

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

[118] 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, 2011.

[119] 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 Berlin Heidelberg, 2005, vol. 3496, pp. 660–665.

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

[121] 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, 2003.

[122] 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, pp. 132–137 vol.1.

[123] 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, 2007.

[124] 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.

[125] 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, pp. 2615–2619 vol.5.

[126] 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, 2009.

[127] 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 Berlin Heidelberg, 2013, vol. 182, pp. 193–205.

[128] 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, 1990.

[129] 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, 1996.

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

[131] 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.

[132] V. K. Ojha, A. Abraham, and V. Snášel, "ACO for continuous function optimization: A performance analysis," in *2014 14th Int. Conf. Intell. Syst. Design and Appl.* IEEE, 2014, pp. 145–150.

[133] 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.

[134] 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),*, pp. 105–110.

[135] 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, 2012.

[136] 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, 2010.

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

[138] 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, 2011.

[139] 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, 2010.

[140] 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.

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

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

[143] 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, 2013.

[144] 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.

[145] 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, 1990.

[146] 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.

[147] 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, 1989.

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

[149] 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.*, pp. 392–397.

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

[151] 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.

[152] 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.

[153] 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, 2006.

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

[155] 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, 2006.

[156] 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, 2009.

[157] 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, 2013.

[158] 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, pp. 397–404.

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

[160] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.

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

[162] I. K. Fodor, “A survey of dimension reduction techniques,” 2002.

[163] 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.

[164] 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*, pp. 223–234.

[165] 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, 2013.

[166] J. Kennedy and R. C. Eberhart, “A discrete binary version of the particle swarm algorithm,” in *IEEE Int. Conf. Syst., Man, Cybern., 1997. Comput. Cybern. Simulation., 1997*, vol. 5. IEEE, 1997, pp. 4104–4108.

[167] 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.

[168] 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.

[169] 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.

[170] 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*, pp. 690–695.

[171] 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.

[172] 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.

[173] 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, 2010.

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

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

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

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

[178] 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, 2007.

[179] 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, 2001.

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

[181] 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, 2003.

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

[183] A. Andoni, R. Panigrahy, G. Valiant, and L. Zhang, “Learning polynomials with neural networks,” in *Proc. the 31st Int. Conf. Mach. Learn. (ICML-14)*, 2014, pp. 1908–1916.

[184] L. Zjavka, V. Snášel, W. Pedrycz, and V. K. Ojha, “A substitution of the general partial differential equation with extended polynomial networks,” in *IEEE World Congr. Comput. Intell., IEEE Int. Jt. Conf. Neural Netw. (IJCNN 2016).* IEEE, 2016.

[185] 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.

[186] I. Sporea and A. Grüning, “Supervised learning in multilayer spiking neural networks,” *Neural Comput.*, vol. 25, no. 2, pp. 473–509, 2013.

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

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

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

[190] 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, 1996.

[191] A. Abraham, “Meta learning evolutionary artificial neural networks,” *Neurocomputing*, vol. 56, no. 0, pp. 1 – 38, 2004.

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

[193] X. Yao, “Evolving artificial neural networks,” *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[194] 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.

[195] 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, 1994.

[196] Y. Xin and L. Yong, “A new evolutionary system for evolving artificial neural networks,” *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, 1997.

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

[198] E. Salajegheh and S. Gholizadeh, “Optimum design of structures by an improved genetic algorithm using neural networks,” *Adv. Eng. Softw.*, vol. 36, no. 11–12, pp. 757 – 767, 2005.

[199] 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, 2006.

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

[201] 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, 2003.

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

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

[204] 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)*, pp. 573–580.

[205] L. Peng, B. Yang, L. Zhang, and Y. Chen, “A parallel evolving algorithm for flexible neural tree,” *Parallel Comput.*, vol. 37, no. 10–11, pp. 653–666, 2011.

[206] 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, 2012.

[207] P. Moscato, “On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms,” Tech. Rep., 1989.

[208] 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.

[209] I. C. Trelea, “The particle swarm optimization algorithm: convergence analysis and parameter selection,” *Inform. Process. lett.*, vol. 85, no. 6, pp. 317–325, 2003.

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

[211] 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, 2011.

[212] 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, 2013.

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

[214] 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, 2004.

[215] Y. Da and G. Xiurun, “An improved pso-based ANN with simulated annealing technique,” *Neurocomputing*, vol. 63, no. 0, pp. 527 – 533, 2005.

[216] 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, 2013.

[217] 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, 2013.

[218] 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, 2012.

[219] 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, 2007.

[220] 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.

[221] 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.

[222] 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, 2000.

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

[224] 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, 2003.

[225] 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, 2007.

[226] 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, 2008.

[227] 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, 2010.

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

[229] 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.

[230] 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.

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

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

[233] 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.

[234] 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, 2004.

[235] 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 Berlin Heidelberg, 2006, vol. 16, pp. 629–655.

[236] 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.

[237] 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, 2002.

[238] 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, 2006.

[239] 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.

[240] 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, 1998.

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

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

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

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

[245] 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 Berlin Heidelberg, 2005, vol. 3496, pp. 520–525.

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

[247] 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, 2003.

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

[249] 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, 2002.

[250] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

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

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

[253] 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, 2010.

[254] 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, 2008.

[255] 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, 2008.

[256] 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.

[257] 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, 2010.

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

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

[260] 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.

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

[262] 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.

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

[264] 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, 2012.

[265] 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, 2006.

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

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

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

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

[270] 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.

[271] R. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evol. Comput.*, vol. 5, no. 2, pp. 123–141, 1997.

[272] K. Deb, *Multi-objective optimization using evolutionary algorithms.* John Wiley & Sons, 2001, vol. 16.

[273] Y. Chen, Y. Zhang, J. Dong, and B. Yang, "System identification by evolved flexible neural tree model," in *5th World Congr. Intell. Control and Automat.*, vol. 1. IEEE, 2004, pp. 313–316.

[274] Y. Chen, A. Abraham, and J. Yang, "Feature selection and intrusion detection using hybrid flexible neural tree," in *Advances in Neural Networks–ISNN*, ser. Lecture Notes in Computer Science. Springer, 2005, vol. 3498, pp. 439–444.

[275] L. Sánchez, I. Couso, and J. A. Corrales, "Combining GP operators with SA search to evolve fuzzy rule based classifiers," *Inform. Sci.*, vol. 136, no. 1, pp. 175–191, 2001.

[276] R. Riolo, J. H. Moore, and M. Kotanchek, *Genetic programming theory and practice XI.* Springer, 2014.

[277] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, 2011.

[278] H.-J. Li, Z.-X. Wang, L.-M. Wang, and S.-M. Yuan, "Flexible neural tree for pattern recognition," in *Advances in Neural Networks–ISNN*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, vol. 3971, pp. 903–908.

[279] Y. Chen, Y. Wang, and B. Yang, "Evolving hierarchical RBF neural networks for breast cancer detection," in *Neural Information Processing*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 4234, pp. 137–144.

[280] Y. Chen, F. Chen, and J. Yang, "Evolving MIMO flexible neural trees for nonlinear system identification," in *Int. Conf. Artificial Intell.*, vol. 1, 2007, pp. 373–377.

[281] P. Wu and Y. Chen, "Grammar guided genetic programming for flexible neural trees optimization," in *Advances in Knowledge Discovery and Data Mining.* Springer, 2007, pp. 964–971.

[282] Y. Shan, R. McKay, R. Baxter, H. Abbass, D. Essam, and H. Nguyen, "Grammar model-based program evolution," in *Congr. Evol. Comput.*, vol. 1, 2004, pp. 478–485.

[283] G. Jia, Y. Chen, and Q. Wu, "A MEP and IP based flexible neural tree model for exchange rate forecasting," in *4th Int. Conf. Natural Comput.*, vol. 5. IEEE, 2008, pp. 299–303.

[284] M. Oltean and C. Groşan, "Evolving evolutionary algorithms using multi expression programming," in *Advances in Artificial Life.* Springer, 2003, pp. 651–658.

[285] P. Musilek, A. Lau, M. Reformat, and L. Wyard-Scott, "Immune programming," *Inform. Sci.*, vol. 176, no. 8, pp. 972–1002, 2006.

[286] B. Yang, L. Wang, Z. Chen, Y. Chen, and R. Sun, "A novel classification method using the combination of FDPS and flexible neural tree," *Neurocomputing*, vol. 73, no. 4–6, pp. 690 – 699, 2010.

[287] S. Bouaziz, H. Dhahri, A. M. Alimi, and A. Abraham, "Evolving flexible beta basis function neural tree using extended genetic programming & hybrid artificial bee colony," *Appl. Soft Comput.*, 2016.

[288] Y. Chen, B. Yang, and A. Abraham, "Flexible neural trees ensemble for stock index modeling," *Neurocomputing*, vol. 70, no. 4–6, pp. 697 – 703, 2007.

[289] Y. Chen, A. Abraham, Y. Zhang *et al.*, "Ensemble of flexible neural trees for breast cancer detection," *Int. J Inform. Tech. Intell. Comput.*, vol. 1, no. 1, pp. 187–201, 2006.

[290] B. Yang, M. Jiang, Y. Chen, Q. Meng, and A. Abraham, "Ensemble of flexible neural tree and ordinary differential equations for small-time scale network traffic prediction," *J. Comput.*, vol. 8, no. 12, pp. 3039–3046, 2013.

[291] V. K. Ojha, A. Abraham, and V. Snášel, "Ensemble of heterogeneous flexible neural tree for the approximation and feature-selection of Poly (Lactic-co-glycolic Acid) micro-and nanoparticle," in *Proc. 2nd Int. Afro-European Conf. Ind. Adv. AECIA 2015.* Springer, 2016, pp. 155–165.

[292] M. Ammar, S. Bouaziz, A. M. Alimi, and A. Abraham, "Multi-agent evolutionary design of flexible beta basis function neural tree," in *Int. Jt. Conf. Neural Netw.* IEEE, 2014, pp. 1265–1271.

[293] G. Weiss, *Multiagent systems: A modern approach to distributed artificial intelligence.* MIT Press, 1999.

[294] T. Burianek and S. Basterrech, "Performance analysis of the activation neuron function in the flexible neural tree model," in *Proc. Dateso 2014 Annual Int. Workshop DAtabases, TExts, Specifications and Objects*, April 2014, pp. 35–46.

[295] S. Bouaziz, H. Dhahri, A. M. Alimi, and A. Abraham, "A hybrid learning algorithm for evolving flexible beta basis function neural tree model," *Neurocomputing*, vol. 117, pp. 107–117, 2013.

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

[297] C. Micheloni, A. Rani, S. Kumar, and G. L. Foresti, "A balanced neural tree for pattern classification," *Neural Netw.*, vol. 27, pp. 81–90, 2012.

[298] G. L. Foresti and C. Micheloni, "Generalized neural trees for pattern classification," *IEEE Trans. Neural Netw.*, vol. 13, no. 6, pp. 1540–1547, 2002.

[299] A. Rani, G. L. Foresti, and C. Micheloni, “A neural tree for classification using convex objective function,” *Pattern Recognit. Lett.*, vol. 68, pp. 41–47, 2015.

[300] Q. Shou-Ning, L. Zhao-lian, C. Guang-qiang, Z. Bing, and W. Su-juan, “Modeling of cement decomposing furnace production process based on flexible neural tree,” in *Int. Conf. Inform. Manage., Innovation Manage. Ind. Eng., 2008. ICIII’08.*, vol. 3. IEEE, 2008, pp. 128–133.

[301] Y. Chen, L. Peng, and A. Abraham, “Gene expression profiling using flexible neural trees,” in *Intelligent Data Engineering and Automated Learning*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 4224, pp. 1121–1128.

[302] Y. Chen and A. Abraham, “Hybrid-learning methods for stock index modeling,” in *Artificial Neural Networks in Finance and Manufacturing.* IGI Global, 2006, p. 64.

[303] Z. Chen, B. Yang, Y. Chen, A. Abraham, C. Grosan, and L. Peng, “Online hybrid traffic classifier for peer-to-peer systems based on network processors,” *Appl. Soft Comput.*, vol. 9, no. 2, pp. 685–694, 2009.

[304] Y. Chen, A. Abraham, and B. Yang, “Hybrid flexible neural-tree-based intrusion detection systems,” *Int. J. Intell. Syst.*, vol. 22, no. 4, pp. 337–352, 2007.

[305] Y. Guo, Q. Wang, S. Huang, and A. Abraham, “Flexible neural trees for online hand gesture recognition using surface electromyography,” *J. Comput.*, vol. 7, no. 5, pp. 1099–1103, 2012.

[306] S. Abdelwahab, V. K. Ojha, and A. Abraham, “Ensemble of flexible neural trees for predicting risk in grid computing environment,” in *Innovations in Bio-Inspired Computing and Application.* Springer, 2016, pp. 151–161.

[307] C. Ferreira, *Gene expression programming: mathematical modeling by an artificial intelligence.* Springer, 2006, vol. 21.

[308] D. Karaboga and B. Basturk, “A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm,” *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, 2007.

[309] M. Lichman, “UCI machine learning repository,” 2013, http://archive.ics.uci.edu/ml Accessed on: 01.05.2016.

[310] J. Alcala-Fdez, L. Sanchez, S. Garcia, M. J. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas *et al.*, “Keel: a software tool to assess evolutionary algorithms for data mining problems,” *Soft Comput.*, vol. 13, no. 3, pp. 307–318, 2009.

[311] M. Matsumoto and T. Nishimura, “Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.

[312] Z.-H. Zhou and Z.-Q. Chen, “Hybrid decision tree,” *Knowledge-Based Syst.*, vol. 15, no. 8, pp. 515–528, 2002.

[313] J.-S. R. Jang, “ANFIS: adaptive-network-based fuzzy inference system,” *IEEE Trans. Syst. Man Cybern.*, vol. 23, no. 3, pp. 665–685, 1993.

[314] O. Cordón and F. Herrera, “A two-stage evolutionary process for designing TSK fuzzy rule-based systems,” *IEEE Trans. Syst. Man Cybern., Part B: Cybern.*, vol. 29, no. 6, pp. 703–715, 1999.

[315] J. S. Rustagi, *Optimization techniques in statistics.* Academic Press, 1994.

[316] R. Alcalá, J. Alcalá-Fdez, J. Casillas, O. Cordón, and F. Herrera, “Local identification of prototypes for genetic learning of accurate tsk fuzzy rule-based systems,” *Int. J. Intell. Syst.*, vol. 22, no. 9, pp. 909–941, 2007.

[317] K. B. Cho and B. H. Wang, “Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction,” *Fuzzy Sets Syst.*, vol. 83, no. 3, pp. 325–339, 1996.

[318] A. M. A. H. Dhahri and F. Karray, “Designing beta basis function neural network for optimization using particle swarm optimization,” in *IEEE Jt. Conf. Neural Netw.*, 2008, pp. 2564—-2571.

[319] C. Aouiti, A. M. Alimi, and A. Maalej, “A genetic designed beta basis function neural network for approximating multi-variables functions,” in *Int. Conf. Artificial Neural Nets and Genetic Algorithms.* Springer, 2001, pp. 383–386.

[320] C.-F. Juang, C.-M. Hsiao, and C.-H. Hsu, “Hierarchical cluster-based multispecies particle-swarm optimization for fuzzy-system optimization,” *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 1, pp. 14–26, 2010.

[321] S. Yilmaz and Y. Oysal, “Fuzzy wavelet neural network models for prediction and identification of dynamical systems,” *IEEE Trans. Neural Netw.*, vol. 21, no. 10, pp. 1599–1609, 2010.

[322] H. Dhahri, A. M. Alimi, and A. Abraham, “Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network,” *Neurocomputing*, vol. 97, pp. 131–140, 2012.

[323] A. Miranian and M. Abdollahzade, “Developing a local least-squares support vector machines-based neuro-fuzzy model for nonlinear and chaotic time series prediction,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 2, pp. 207–218, 2013.

[324] N. K. Kasabov, *Foundations of neural networks, fuzzy systems, and knowledge engineering.* Marcel Alencar, 1996.

[325] N. Kasabov, “Evolving fuzzy neural networks for adaptive, on-line intelligent agents and systems,” in *Recent Advances in Mechatronics.* Springer, Berlin, 1999.

[326] S. Bouaziz, A. M. Alimi, and A. Abraham, “Extended immune programming and opposite-based PSO for evolving flexible beta basis function neural tree,” in *IEEE Int. Conf. Cybern.* IEEE, 2013, pp. 13–18.

[327] M. J. Gacto, M. Galende, R. Alcalá, and F. Herrera, “METSK-HD^e: A multiobjective evolutionary algorithm to learn accurate TSK-fuzzy systems in high-dimensional and large-scale regression problems,” *Inform. Sci.*, vol. 276, pp. 63–79, 2014.

[328] V. K. Ojha, S. Schiano, C. Wu, V. Snášel, and A. Abraham, “The development of a machine learning tool for modelling die filling of pharmaceutical granules.” in *Int. Congr. Particle Tech., 2016, PARTEC.* German National Library, 2016.

[329] O. Coube, A. Cocks, and C.-Y. Wu, “Experimental and numerical study of die filling, powder transfer and die compaction,” *Powder Metall.*, vol. 48, no. 1, pp. 68–76, 2005.

[330] L. Schneider, I. Sinka, and A. Cocks, “Characterisation of the flow behaviour of pharmaceutical powders using a model die–shoe filling system,” *Powder Technol.*, vol. 173, no. 1, pp. 59–71, 2007.

[331] R. Kohavi and J. R. Quinlan, “Data mining tasks and methods: Classification: decision-tree discovery,” in *Handbook of Data Min. Knowl. Discovery.* Oxford University Press, Inc., 2002, pp. 267–276.

[332] C. E. Rasmussen and C. Williams, *Gaussian processes for machine learning.* MIT Press, 2006, vol. 2, no. 3.

[333] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[334] J. Zhang, C. Pei, S. Schiano, D. Heaps, and C.-Y. Wu, “The application of terahertz pulsed imaging in characterising density distribution of roll-compacted ribbons,” *Eur. J. Pharm. Biopharm.*, 2016.

[335] C.-Y. Wu, L. Dihoru, and A. C. Cocks, “The flow of powder into simple and stepped dies,” *Powder Technol.*, vol. 134, no. 1, pp. 24–39, 2003.

[336] P. L’Ecuyer and F. Panneton, “Fast random number generators based on linear recurrences modulo 2: Overview and comparison,” in *Proc. 2005 Winter Simulation Conf.* IEEE, 2005, pp. 10–pp.

[337] E. H. Mamdani, “Application of fuzzy algorithms for control of simple dynamic plant,” in *Proc. Inst. of Elect. Eng.*, vol. 121, no. 12. IET, 1974, pp. 1585–1588.

[338] J. M. Mendel, “On km algorithms for solving type-2 fuzzy set problems,” *IEEE Trans. Fuzzy Syst.*, vol. 21, no. 3, pp. 426–446, 2013.

[339] N. N. Karnik, J. M. Mendel, and Q. Liang, “Type-2 fuzzy logic systems,” *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 6, pp. 643–658, 1999.

[340] S. F. Smith, “A learning system based on genetic adaptive algorithms,” Ph.D. dissertation, Pittsburgh, PA, USA, 1980.

[341] L. B. Booker, “Intelligent behavior as an adaptation to the task environment,” Ph.D. dissertation, Ann Arbor, MI, USA, 1982.

[342] G. Venturini, “Sia: a supervised inductive algorithm with genetic search for learning attributes based concepts,” in *Machine Learning: ECML-93.* Springer, 1993, pp. 280–296.

[343] D. P. Greene and S. F. Smith, “Competition-based induction of decision models from examples,” *Mach. Learn.*, vol. 13, no. 2-3, pp. 229–257, 1993.

[344] F. Herrera, “Genetic fuzzy systems: taxonomy, current research trends and prospects,” *Evol. Intel.*, vol. 1, no. 1, pp. 27–46, 2008, 403.

[345] P. Angelov, “Evolving fuzzy systems,” *Encyclopedia Complexity Syst. Sci.*, pp. 3242–3255, 2009.

[346] P. P. Angelov and D. P. Filev, “An approach to online identification of Takagi-Sugeno fuzzy models,” *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 34, no. 1, pp. 484–498, 2004.

[347] P. P. Angelov and D. P. Filev, “Flexible models with evolving structure,” *Int. J. Intell. Syst.*, vol. 19, no. 4, pp. 327–340, 2004.

[348] S. Sahin, M. R. Tolun, and R. Hassanpour, “Hybrid expert systems: A survey of current approaches and applications,” *Expert Syst. Appl.*, vol. 39, no. 4, pp. 4609–4617, 2012.

[349] C.-J. Lin and C.-T. Lin, “An ART-based fuzzy adaptive learning control network,” *IEEE Trans. Fuzzy Syst.*, vol. 5, no. 4, pp. 477–496, 1997.

[350] H. R. Berenji and P. Khedkar, “Learning and tuning fuzzy logic controllers through reinforcements,” *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 724–740, 1992.

[351] D. Nauck and R. Kruse, “Neuro-fuzzy systems for function approximation,” *Fuzzy Sets Syst.*, vol. 101, no. 2, pp. 261–271, 1999.

[352] S. Tano, T. Oyama, and T. Arnould, “Deep combination of fuzzy inference and neural network in fuzzy inference software—finest,” *Fuzzy Sets Syst.*, vol. 82, no. 2, pp. 151–160, 1996.

[353] C.-F. Juang and C.-T. Lin, “An online self-constructing neural fuzzy inference network and its applications,” *IEEE Trans. Fuzzy Syst.*, vol. 6, no. 1, pp. 12–32, 1998.

[354] N. Kasabov, “Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning,” *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 31, no. 6, pp. 902–918, 2001.

[355] S. Wu and M. J. Er, “Dynamic fuzzy neural networks-a novel approach to function approximation,” *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 30, no. 2, pp. 358–364, 2000.

[356] J. J. Buckley and Y. Hayashi, “Fuzzy neural networks: A survey,” *Fuzzy Sets Syst.*, vol. 66, no. 1, pp. 1–13, 1994.

[357] A. Fernández, V. López, M. J. del Jesus, and F. Herrera, “Revisiting evolutionary fuzzy systems: Taxonomy, applications, new trends and challenges,” *Knowledge-Based Syst.*, vol. 80, pp. 109–121, 2015.

[358] N. K. Kasabov and Q. Song, “DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction,” *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 2, pp. 144–154, 2002.

[359] C.-F. Juang and Y.-W. Tsao, “A self-evolving interval type-2 fuzzy neural network with online structure and parameter learning,” *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 6, pp. 1411–1424, 2008.

[360] C.-F. Juang and K.-J. Juang, “Reduced interval type-2 neural fuzzy system using weighted bound-set boundary operation for computation speedup and chip implementation,” *IEEE Trans. Fuzzy Syst.*, vol. 21, no. 3, pp. 477–491, 2013.

[361] S. W. Tung, C. Quek, and C. Guan, “SaFIN: A self-adaptive fuzzy inference network,” *IEEE Trans. Neural Netw*, vol. 22, no. 12, pp. 1928–1940, 2011.

[362] Y.-Y. Lin, J.-Y. Chang, N. R. Pal, and C.-T. Lin, “A mutually recurrent interval type-2 neural fuzzy system (MRIT2NFS) with self-evolving structure and parameters,” *IEEE Trans. Fuzzy Syst.*, vol. 21, no. 3, pp. 492–509, 2013.

[363] Y.-Y. Lin, J.-Y. Chang, and C.-T. Lin, “A TSK-type-based self-evolving compensatory interval type-2 fuzzy neural network (TSCIT2FNN) and its applications,” *IEEE Trans. Ind. Electron.*, vol. 61, no. 1, pp. 447–459, 2014.

[364] Y.-Y. Lin, S.-H. Liao, J.-Y. Chang, and C.-T. Lin, “Simplified interval type-2 fuzzy neural networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 5, pp. 959–969, 2014.

[365] A. Bouchachia and C. Vanaret, “GT2FC: an online growing interval type-2 self-learning fuzzy classifier,” *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 4, pp. 999–1018, 2014.

[366] A. K. Das, K. Subramanian, and S. Sundaram, “An evolving interval type-2 neurofuzzy inference system and its metacognitive sequential learning algorithm,” *IEEE Trans. Fuzzy Syst.*, vol. 23, no. 6, pp. 2080–2093, 2015.

[367] G. Raju, J. Zhou, and R. A. Kisner, “Hierarchical fuzzy control,” *Int. J. Control*, vol. 54, no. 5, pp. 1201–1216, 1991.

[368] M. Brown, K. Bossley, D. Mills, and C. Harris, “High dimensional neurofuzzy systems: overcoming the curse of dimensionality,” in *Proc. 1995 IEEE Int. Fuzzy Syst., 1995. Int. Jt. Conf. of the 4th Int. Conf. Fuzzy Syst. The 2nd Int. Fuzzy Eng. Symp..*, vol. 4, pp. 2139–2146.

[369] L.-X. Wang, “Analysis and design of hierarchical fuzzy systems,” *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 5, pp. 617–624, 1999.

[370] X.-J. Zeng and J. A. Keane, “Approximation capabilities of hierarchical fuzzy systems,” *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 5, pp. 659–672, 2005.

[371] V. Torra, “A review of the construction of hierarchical fuzzy systems,” *Int. J. Intell. Syst.*, vol. 17, no. 5, pp. 531–543, 2002.

[372] M. G. Joo and J. S. Lee, “A class of hierarchical fuzzy systems with constraints on the fuzzy rules,” *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 2, pp. 194–203, 2005.

[373] A. Fernández, M. J. del Jesus, and F. Herrera, “Hierarchical fuzzy rule based classification systems with genetic rule selection for imbalanced data-sets,” *Int. J. Approximate Reasoning*, vol. 50, no. 3, pp. 561–577, 2009.

[374] C.-L. Hwang, C.-C. Chiang, and Y.-W. Yeh, “Adaptive fuzzy hierarchical sliding-mode control for the trajectory tracking of uncertain underactuated nonlinear dynamic systems,” *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 2, pp. 286–299, 2014.

[375] Y. Chen, B. Yang, A. Abraham, and L. Peng, “Automatic design of hierarchical takagi–sugeno type fuzzy systems using evolutionary algorithms,” *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 3, pp. 385–397, 2007.

[376] A. Mohammadzadeh, O. Kaynak, and M. Teshnehlab, “Two-mode indirect adaptive control approach for the synchronization of uncertain chaotic systems by the use of a hierarchical interval type-2 fuzzy neural network,” *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 5, pp. 1301–1312, 2014.

[377] H. Ishibuchi, “Multiobjective genetic fuzzy systems: review and future research directions,” in *IEEE Int. Fuzzy Syst. Conf., 2007. FUZZ-IEEE 2007*, pp. 1–6.

[378] H. Ishibuchi, T. Murata, and I. Türkşen, “Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems,” *Fuzzy Sets Syst.*, vol. 89, no. 2, pp. 135–150, 1997.

[379] R. Alcalá, M. J. Gacto, F. Herrera, and J. Alcalá-Fdez, “A multi-objective genetic algorithm for tuning and rule selection to obtain accurate and compact linguistic fuzzy rule-based systems,” *Int. J. Uncertainty Fuzziness Knowledge Based Syst.*, vol. 15, no. 05, pp. 539–557, 2007.

[380] O. Cordón, “A historical review of evolutionary learning methods for Mamdani-type fuzzy rule-based systems: Designing interpretable genetic fuzzy systems,” *Int. J. Approximate Reasoning*, vol. 52, no. 6, pp. 894–913, 2011.

[381] S. Guillaume, “Designing fuzzy inference systems from data: an interpretability-oriented review,” *IEEE Trans. Fuzzy Syst.*, vol. 9, no. 3, pp. 426–443, 2001.

[382] H. Ishibuchi and Y. Nojima, “Evolutionary multiobjective optimization for the design of fuzzy rule-based ensemble classifiers,” *Int. J. Hybrid Intell. Syst.*, vol. 3, no. 3, pp. 129–145, 2006.

[383] H. Ishibuchi and Y. Nojima, “Analysis of interpretability-accuracy trade-off of fuzzy systems by multiobjective fuzzy genetics-based machine learning,” *Int. J. Approximate Reasoning*, vol. 44, no. 1, pp. 4–31, 2007.

[384] M. J. Gacto, R. Alcalá, and F. Herrera, “Adaptation and application of multi-objective evolutionary algorithms for rule reduction and parameter tuning of fuzzy rule-based systems,” *Soft Comput.*, vol. 13, no. 5, pp. 419–436, 2009.

[385] P. Ducange, B. Lazzerini, and F. Marcelloni, “Multi-objective genetic fuzzy classifiers for imbalanced and cost-sensitive datasets,” *Soft Comput.*, vol. 14, no. 7, pp. 713–728, 2010.

[386] O. Cordón, M. J. Del Jesus, F. Herrera, L. Magdalena, and P. Villar, “A multiobjective genetic learning process for joint feature selection and granularity and contexts learning in fuzzy rule-based classification systems,” in *Interpretability issues in fuzzy modeling.* Springer, 2003, pp. 79–99.

[387] H. Wang, S. Kwong, Y. Jin, W. Wei, and K.-F. Man, “Multi-objective hierarchical genetic algorithm for interpretable fuzzy rule-based knowledge extraction,” *Fuzzy Sets Syst.*, vol. 149, no. 1, pp. 149–186, 2005.

[388] R. Munoz-Salinas, E. Aguirre, O. Cordón, and M. García-Silvente, “Automatic tuning of a fuzzy visual system using evolutionary algorithms: single-objective versus multiobjective approaches,” *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 2, pp. 485–501, 2008.

[389] R. Alcalá, P. Ducange, F. Herrera, B. Lazzerini, and F. Marcelloni, “A multiobjective evolutionary approach to concurrently learn rule and data bases of linguistic fuzzy-rule-based systems,” *IEEE Trans. Fuzzy Syst.*, vol. 17, no. 5, pp. 1106–1122, 2009.

[390] M. Antonelli, P. Ducange, B. Lazzerini, and F. Marcelloni, “Learning knowledge bases of multi-objective evolutionary fuzzy systems by simultaneously optimizing accuracy, complexity and partition integrity,” *Soft Comput.*, vol. 15, no. 12, pp. 2335–2354, 2011.

[391] M. Antonelli, P. Ducange, and F. Marcelloni, “Genetic training instance selection in multiobjective evolutionary fuzzy systems: A coevolutionary approach,” *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 2, pp. 276–290, 2012.

[392] M. Fazzolari, R. Alcala, Y. Nojima, H. Ishibuchi, and F. Herrera, “A review of the application of multiobjective evolutionary fuzzy systems: Current status and further directions,” *IEEE Trans. Fuzzy Syst.*, vol. 21, no. 1, pp. 45–65, 2013.

[393] L. A. Zadeh, “The concept of a linguistic variable and its application to approximate reasoning—I,” *Inform. Sci.*, vol. 8, no. 3, pp. 199–249, 1975.

[394] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming.* Lulu. com, 2008.

[395] O. Cordón, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena, “Ten years of genetic fuzzy systems: current framework and new trends,” *Fuzzy Sets Syst.*, vol. 1, no. 141, pp. 5–31, 2004.

[396] O. Castillo and P. Melin, “Optimization of type-2 fuzzy systems based on bio-inspired methods: a concise review,” *Inform. Sci.*, vol. 205, pp. 1–19, 2012.

[397] J. Snyman, *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms.* Springer Science & Business Media, 2005, vol. 97.

[398] Y.-Q. Zhang, B. Jin, and Y. Tang, “Granular neural networks with evolutionary interval learning,” *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 2, pp. 309–319, 2008.

[399] J. Kim and N. Kasabov, “HyFIS: adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems,” *Neural Netw.*, vol. 12, no. 9, pp. 1301–1319, 1999.

[400] J.-C. Duan and F.-L. Chung, “Multilevel fuzzy relational systems: structure and identification,” *Soft Comput.*, vol. 6, no. 2, pp. 71–86, 2002.

[401] S. Paul and S. Kumar, “Subsethood-product fuzzy neural inference system (SuPFuNIS),” *IEEE Trans. Neural Netw*, vol. 13, no. 3, pp. 578–599, 2002.

[402] J.-H. Chiang and P.-Y. Hao, “Support vector learning mechanism for fuzzy rule-based modeling: a new approach,” *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 1, pp. 1–12, 2004.

[403] S. W. Tung, C. Quek, and C. Guan, “eT2FIS: an evolving type-2 neural fuzzy inference system,” *Inform. Sci.*, vol. 220, pp. 124–148, 2013.

[404] C.-F. Juang, R.-B. Huang, and W.-Y. Cheng, "An interval type-2 fuzzy-neural network with support-vector regression for noisy regression problems," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 4, pp. 686–699, 2010.

[405] J. M. Mendel, *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions.* Upper Saddle River, NJ: Prentice-Hall, 2001.

[406] J. M. Mendel, "Computing derivatives in interval type-2 fuzzy logic systems," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 1, pp. 84–98, 2004.

[407] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw*, vol. 1, no. 1, pp. 4–27, 1990.

[408] P. Baranyi, L. T. Kóczy, and T. T. D. Gedeon, "A generalized concept for fuzzy rule interpolation," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 6, pp. 820–837, 2004.

[409] Y.-C. Chang, S.-M. Chen, and C.-J. Liau, "Fuzzy interpolative reasoning for sparse fuzzy-rule-based systems based on the areas of fuzzy sets," *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 5, pp. 1285–1301, 2008.

[410] Z. Huang and Q. Shen, "Fuzzy interpolation and extrapolation: A practical approach," *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 1, pp. 13–28, 2008.

[411] S.-M. Chen and Y.-C. Chang, "Weighted fuzzy rule interpolation based on GA-based weight-learning techniques," *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 4, pp. 729–744, 2011.

[412] G. E. P. Box and G. M. Jenkins, *Time Series Analysis, Forecasting and Control.* San Francisco, CA, USA: Holden–Day, 1976.

[413] J. Szlek, A. Pacławski, R. Lau, R. Jachowicz, and A. Mendyk, "Heuristic modeling of macromolecule release from PLGA microspheres," *Int. J. Nanomed.*, vol. 8, no. 1, pp. 4601–4611, 2013.

[414] C. E. Astete and C. M. Sabliov, "Synthesis and characterization of PLGA nanoparticles," *J. Biomater. Sci., Polym. Ed.*, vol. 17, no. 3, pp. 247–289, 2006.

[415] R. Langer and D. A. Tirrell, "Designing materials for biology and medicine," *Nature*, vol. 428, no. 6982, pp. 487–492, 2004.

[416] P. Graham, K. Brodbeck, and A. McHugh, "Phase inversion dynamics of PLGA solutions related to drug delivery," *J. Controlled Release*, vol. 58, no. 2, pp. 233–245, 1999.

[417] H. K. Makadia and S. J. Siegel, "Poly lactic-co-glycolic acid (plga) as biodegradable controlled drug delivery carrier," *Polymers*, vol. 3, no. 3, pp. 1377–1397, 2011.

[418] Y. Hayashi and J. J. Buckley, "Approximations between fuzzy expert systems and neural networks," *Int. J. Approximate Reasoning*, vol. 10, no. 1, pp. 63–73, 1994.

Appendix A

Publications list

Articles/Journals

Published

J1. **Ojha, V. K.**, Jackowski, K., Abraham, A., and Snášel, V. (2015). Dimensionality reduction, and function approximation of poly (lactic-co-glycolic acid) micro-and nanoparticle dissolution rate. *International Journal of Nanomedicine*, 10, 1119. (**IF:** 4.38, Quartile: Q1).

Under review

J2. **Ojha, V. K.**, Abraham, A., and Snášel, V. (2015). Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research, *Engineering Applications in Artificial Intelligence.* (**IF:** 2.21)

J3. **Ojha, V. K.**, Abraham, A., and Snášel, V. (2015). Ensemble of Heterogeneous Flexible Neural Trees Using Multiobjective Genetic Programming, *Applied Soft Computing.* (One round revision completed) (**IF:** 2.81)

J4. **Ojha, V. K.**, Schiano, S., Wu, C.Y., Snášel, V., and Abraham, A. (2016) Predictive Modeling of the die filling process of the pharmaceutical granules using Flexible Neural Tree. *Neural Computing Application.* (One round revision completed) (**IF:** 1.57)

J5. **Ojha, V. K.**, Snášel, V., and Abraham, A., (2015). Multiobjective Fuzzy Inference System for feature selection and function approximation. *IEEE Transaction on Fuzzy Systems.* (**IF:** 8.75)

Conference proceedings

C1. **Ojha, V. K.**, Abraham, A., and Snášel, V. (2016) Metaheuristic Tuning of Type-II Fuzzy Inference Systems for Data Mining. IEEE World Congress on Computational Intelligence, IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016). (Accepted) (Indexing: SCOPUS)

C2. Zjavka, L., Snášel, V., Pedrycz, W., and **Ojha, V.K.**, A Substitution of the General Partial Differential Equation with Extended Polynomial Networks, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, 2016, (IJCNN, IEEE). (Accepted) (Indexing: SCOPUS)

C3. **Ojha, V. K.**, Abraham, A., and Snášel, V. (2016). Ensemble of Heterogeneous Flexible Neural Tree for the Approximation and Feature-Selection of Poly (Lactic-co-glycolic Acid) Micro-and Nanoparticle. In Proceedings of the 2nd International Afro-European Conference for Industrial Advancement AECIA 2015 (pp. 155-165). Springer. (Indexing: SCOPUS)

C4. Abdelwahab, S., **Ojha, V. K.**, and Abraham, A. (2016). Ensemble of Flexible Neural Trees for Predicting Risk in Grid Computing Environment. In Proceedings of the 6th International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2015 (pp. 151-161). Springer. (Indexing: SCOPUS)

C5. Abdelwahab, S., **Ojha, V. K.**, and Abraham, A. (2016). Neuro-Fuzzy Risk Prediction Model for Computational Grids. In Proceedings of the 2nd International Afro-European Conference for Industrial Advancement AECIA 2015 (pp. 127-136). Springer. (Indexing: SCOPUS)

C6. Ahmed, N., **Ojha, V. K.**, and Abraham, A. (2016). An Ensemble of Neuro-Fuzzy Model for Assessing Risk in Cloud Computing Environment. In Advances in Nature and Biologically Inspired Computing (pp. 27-36). Springer. (Indexing: SCOPUS)

C7. **Ojha, V. K.**, Schiano, S., Wu, C.Y., Abraham, A., and Snášel, V. (2016) The development of a machine learning tool for modelling die filling of pharmaceutical granules. International Congress on Particle Technology, 2016, PARTEC. (German National Library).

C8. **Ojha, V. K.**, Abraham, A., and Snášel, V. (2014). ACO for continuous function optimization: A performance analysis. In 14th International Conference on Intelligent Systems Design and Applications (ISDA), 2014 (pp. 145-150). IEEE. (Indexing: SCOPUS)

C9. **Ojha, V. K.**, Abraham, A., and Snášel, V. (2015). Simultaneous optimization of neural network weights and active nodes using metaheuristics. In 14th International Conference on Hybrid Intelligent Systems (HIS), 2014 (pp. 248-253). IEEE. (Indexing: SCOPUS)

C10. Gabralla, L. A., Wahby, T. M., **Ojha, V. K.**, and Abraham, A. (2014). Ensemble of adaptive neuro-fuzzy inference system using particle swarm optimization for prediction of crude oil prices. In 14th International Conference on Hybrid Intelligent Systems (HIS), 2014 (pp. 141-146). IEEE. (Indexing: SCOPUS)

C11. **Ojha, V. K.**, Jackowski, K., Snášel, V., and Abraham, A. (2014). Dimensionality Reduction and Prediction of the Protein Macromolecule Dissolution Profile. In Proceedings of the 5th International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2014 (pp. 301-310). Springer. (Indexing: SCOPUS)

C12. **Ojha, V. K.**, Jackowski, K., Abraham, A., and Snášel, V. (2014). Feature selection and ensemble of regression models for predicting the protein macromolecule dissolution profile. In 6th World Congress on Nature and Biologically Inspired Computing (NaBIC), 2014 (pp. 121-126). IEEE. (Indexing: SCOPUS)

Appendix B

Adaptive approximation tools

Multiobjective heterogeneous flexible neural tree

http://dap.vsb.cz/aat/

Multiobjective hierarchical fuzzy inference tree

http://dap.vsb.cz/sw/hfit

Index