Chapter 2
State of the Art

In this chapter, we describe the state of the art of the computational intelligence techniques, which we use as a basis for this work.

2.1 Time Series

Time series is a set of measurements of some phenomenon or experiment sequentially recorded over time. These observations will be denoted by $\{x(t_1), x(t_2), \ldots, x(t_n)\} = \{x(t) : t \in T \subseteq R\}$ con $x(t_i)$ the value of the variable x in the time t_i. If $T = Z$ is said that the time series is discrete and if $T = R$ is said that the time series is continuous [1, 2].

A classic model for a time series, assumes that a $x_{(1)}, \ldots, x_{(n)}$ series can be expressed as the sum or product of its components: trend, cyclical, seasonal and irregular [3]. There are three time series models, which are generally accepted as good approximations to the true relationships between the components of the observed data. These are:

Additive.

$$X(t) = T(t) + C(t) + S(t) + I(t) \tag{2.1}$$

Multiplicative.

$$X(t) = T(t) \cdot C(t) \cdot S(t) \cdot I(t) \tag{2.2}$$

Mixed.

$$X(t) = T(t) \cdot S(t) \cdot I(t) \tag{2.3}$$

J. Soto et al., *Ensembles of Type 2 Fuzzy Neural Models and Their Optimization with Bio-Inspired Algorithms for Time Series Prediction*, SpringerBriefs in Computational Intelligence, https://doi.org/10.1007/978-3-319-71264-2_2

where:

$X(t)$ observed series in time t
$T(t)$ trend component
$C(t)$ cyclic component
$S(t)$ seasonal component
$I(t)$ irregular or random component

A common assumption is that $I(t)$ is a random or white noise component with zero mean and constant variance. An additive model, is suitable, for example, when $S(t)$ does not depend on other components such as $T(t)$, if instead the seasonality varies with the trend, the most suitable model is a multiplicative model. It is clear that the multiplicative model can be transformed into additive, by taking logarithms. The problem that arises is to adequately model the components of the series.

Temporal phenomena are both complex and important in many real-world problems. Their importance stems from the fact that almost every kind of data contains time-dependent components, either explicitly coming in the form of time values or implicitly in the way that the data is collected from a process that varies with time [4]. A time series is an important class of complex data objects [5] and comes as a sequence of real numbers, each number representing a value reported at a certain time instant [6]. The popular statistical model of Box-Jenkins [7] is considered to be one of the most common choices for the prediction of time series. However, since the Box-Jenkins models are linear and most real world applications involve nonlinear problems, it is difficult for the Box-Jenkins models to capture the phenomenon of nonlinear time series and this brings a limitation to the accuracy of the generated predictions [8].

2.2 Interval Type-2 Fuzzy Neural Network

One way to build on IT2FNN is by fuzzifying a conventional neural network (NN). Each part of a NN (the activation function, the weights, and the inputs and outputs) can be fuzzified. A fuzzy neuron is basically similar to an artificial neuron, except that it has the ability to process fuzzy information.

The IT2FNN system is one kind of IT2-TSK-FIS inside a NN structure. An IT2FNN is proposed by Castro et al. in [9], with TSK reasoning and processing elements called IT2FN for defining antecedents, and the IT1FN for defining the consequents of rules R^k.

An IT2FN is composed by two adaptive nodes represented by squares, and two non-adaptive nodes represented by circles. Adaptive nodes have outputs that depend on their inputs, modifiable parameters and transference function while non-adaptive, on the contrary, depend solely on their inputs, and their outputs represent lower $\underline{\mu}_A(x)$ and upper $\overline{\mu}_A(x)$ membership functions. Parameters from adaptive nodes with uncertain standard deviation are denoted by $\left(w \in \left[w_{1,1}, w_{2,1}\right]\right)$

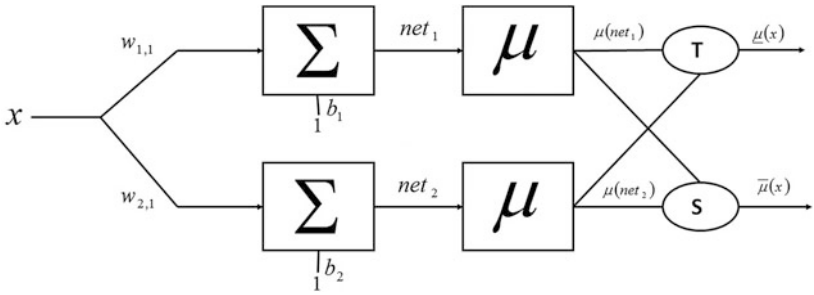

Fig. 2.1 Interval type-2 fuzzy neuron (IT2FN)

and with uncertain mean by $(b \in [b_1, b_2])$. The IT2FN (Fig. 2.1) with crisp input signals (x), crisp synaptic weights (w, b) and type-1 fuzzy outputs $\mu(net_1), \mu(net_2), \underline{\mu}(x), \overline{\mu}(x)$. This kind of neuron is build from two conventional neurons with transference functions $\mu(net_1)$, Gaussian, generalized bell and logistic for fuzzifier the inputs. Each neuron equation is defined as follows: the function μ is often referred to as an activation (or transfer) function. Its domain is the set of activation values, net, of the neuron model; we thus often use this function as $\mu(net_2)$. The variable net is defined as a scalar product of the weight and the vectors:

$$\begin{aligned} net_1 &= w_{1,1} + b_1; \mu_1 = \mu(net_1); \\ net_2 &= w_{2,1} + b_1; \mu_2 = \mu(net_2) \end{aligned} \tag{2.4}$$

The non-adaptive *t*-norm node (*T*) evaluates the lower membership function $\overline{\mu}(x)$ under *t*-norm algebraic product, while s-norm non-adaptive node (*S*), evaluate the upper membership function $\underline{\mu}(x)$ under the *s*-norm algebraic sum, as shown in Eq. (2.5):

$$\begin{aligned} \underline{\mu}(x) &= \mu(net_1) \cdot \mu(net_2), \\ \overline{\mu}(x) &= \mu(net_1) + \mu(net_2) - \underline{\mu}(x) \end{aligned} \tag{2.5}$$

Each IT2FN adapts an interval type-2 fuzzy set [10, 11], $\tilde{A}$, expressed in terms of the output $\underline{\mu}(x)$, of type-1 fuzzy neuron with *T*-norm and $\overline{\mu}(x)$ of type-1 fuzzy neuron with *S*-norm. An internal type-2 fuzzy set is denoted as:

$$\widetilde{A} = \int_{x \in X} \left[\int_{\mu(x) \in \left[\overline{\mu}(x), \underline{\mu}(x)\right]} 1/\mu \right] / x \tag{2.6}$$

An IT1FN (Fig. 2.2) is built from two conventional adaptive linear neurons (ADALINE) [12] for adapting the consequents $y_k^j \in \left[{}_l y_{k \cdot r}^j y_k^j\right]$ from the rules R_k, for the output defined by

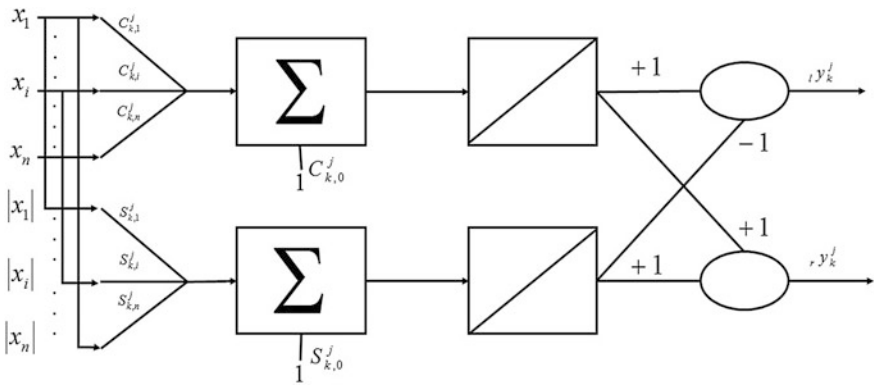

Fig. 2.2 Interval type-1 fuzzy neuron

$$
\begin{aligned}
{}_{l}y_k^j &= \sum_{1=1}^{n} C_{k,i}^j x_i + C_{k,0}^j - \sum_{1=1}^{n} S_{k,i}^j |x_i| - S_{k,0}^j, \\
{}_{r}y_k^j &= \sum_{1=1}^{n} C_{k,i}^j x_i + C_{k,0}^j + \sum_{1=1}^{n} S_{k,i}^j |x_i| + S_{k,0}^j.
\end{aligned}
\tag{2.7}
$$

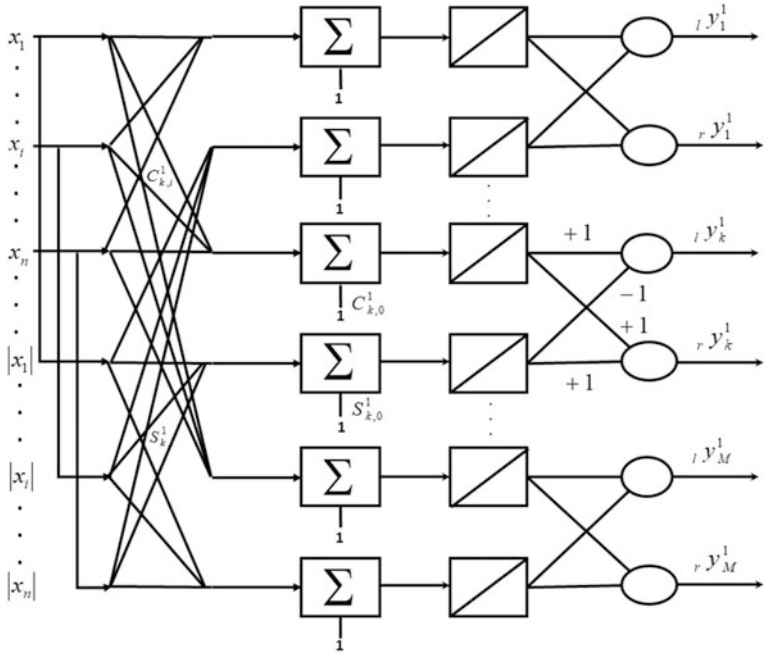

Fig. 2.3 Interval type-2 fuzzy neural network

Thus consequents can be adapted with linear networks. The network weights are the parameters of consequents $C_{k,i}^{j}, S_{k,i}^{j}$ for the kth rule. The outputs represents interval linear MFs of the rule's consequents (Fig. 2.3).

2.3 Ensemble Learning

The Ensemble consists of a learning paradigm where multiple component learners are trained for a same task, and the prediction of the component learners are combined for dealing with future instances [13]. Since an Ensemble is often more accurate than its component learners, such a paradigm has become a hot topic in recent years and has already been successfully applied to optical character recognition, face recognition, scientific image analysis, medical diagnosis and time series [14].

In general, a neural network ensemble is constructed in two steps, i.e. training a number of component neural networks and then combining the component predictions.

There are also many other approaches for training the component neural networks. Some examples are as follows. Hampshire and Waibel [15, 16] utilize different objective functions to train distinct component neural networks. Cherkauer [17] trains component networks with different number of hidden layers. Maclin and Shavlik [18] initialize component networks at different points in the weight space. Krogh and Vedelsby [19, 20] employ cross-validation to create component networks. Opitz and Shavlik [21, 22] exploit a genetic algorithm to train diverse knowledge based component networks. Yao and Liu [23] regard all the individuals in an evolved population of neural networks as component networks [24].

2.4 Interval Type-2 Fuzzy Systems

Type-2 fuzzy sets are used to model uncertainty and imprecision; originally they were proposed by Zadeh [25, 26] and they are essentially "fuzzy–fuzzy" sets in which the membership degrees are type-1 fuzzy sets (Fig. 2.4).

The structure of a type-2 fuzzy system implements a nonlinear mapping of on input to on output space. This mapping is achieved through a set of type-2 if-then fuzzy rules, each of which describes the local behavior of the mapping.

The uncertainty is represented by a region called footprint of uncertainty (FOU). When $\mu_{\tilde{A}}(x,u) = 1, \forall u \in l_x \subseteq [0,1]$ we have an interval type-2 membership function [27–30] (Fig. 2.5).

The uniform shading for the FOU represents the entire interval type-2 fuzzy set and it can be described in terms of an upper membership function $\overline{\mu}_{\tilde{A}}(x)$ and a lower membership function $\underline{\mu}_{\tilde{A}}(x)$.

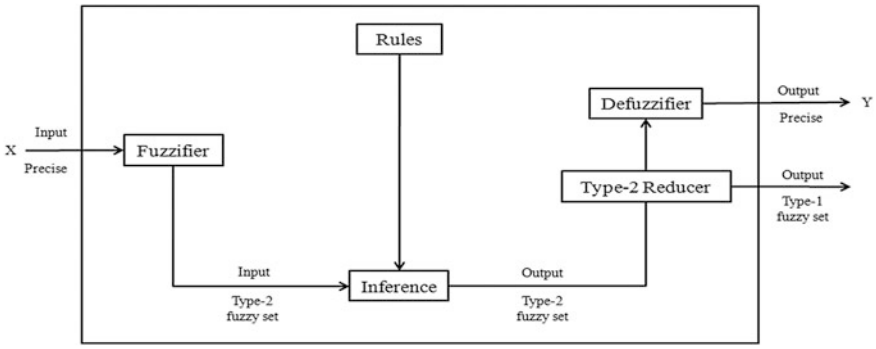

Fig. 2.4 Structure of the interval type-2 fuzzy logic system

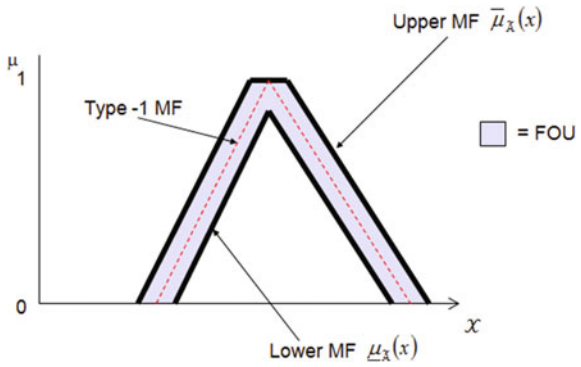

Fig. 2.5 Interval type-2 membership function

A fuzzy logic system (FLS) described using at least one type-2 fuzzy set is called a type-2 FLS. Type-1 FLSs are unable to directly handle rule uncertainties, because they use type-1 fuzzy sets that are certain [9, 31]. On the other hand, type-2 FLSs are very useful in circumstances where it is difficult to determine an exact certainty value, and there are measurement uncertainties.

2.5 Genetic Algorithms

Genetic algorithms (GAs) are adaptive methods that can be used to solve search and optimization problems. They are based on the genetic process of living organisms. Over generations, the populations evolve in nature in accordance with the principles of natural selection and survival of the strongest, postulated by Darwin. By imitating this process, genetic algorithms are able to create solutions to real world problems. The evolution of these solutions towards optimal values of the problem depends largely on proper coding them. The basic principles of genetic

algorithms were established by Holland [32, 33] and are well described in the works of Goldberg [34–36], Davis [37] and Michalewicz [38]. The large field of applications of GA is related to those problems for which there are no specialized techniques. Even if such technical exist and work well, improvements can be made with the same hybrid genetic algorithms.

A GA is a highly parallel mathematical algorithm that transforms a set (population) of individual mathematical objects (typically strings of fixed length which fit the model chains chromosomes), each of which is associated with a fitness in a new population (e.g. the next generation) operations using models according to the principle Darwinian reproduction and survival of the fittest and after having naturally presented a series of genetic operations [39].

To apply the genetic algorithm requires the following five basic components:

1. Representation of the potential solutions to the problem.
2. One way to create an initial population of possible solutions (usually a random process).
3. An evaluation function to play the role of the environment, classifying solutions in terms of their "fitness".
4. Genetic operators that alter the composition of the children that will occur for the next generation.
5. Values for the different parameters using the genetic algorithm (population size, crossover probability, mutation probability, maximum number of generations, etc.).

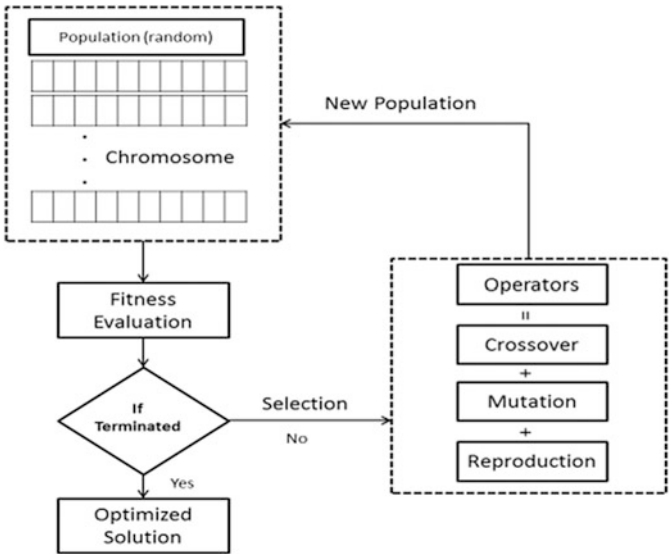

Fig. 2.6 Steps of the genetic algorithm

The basic operations of a genetic algorithm [40] are as follows illustrated in Fig. 2.6:

- Step 1: Represent the problem variable domain as a chromosome of a fixed length; choose the size of a chromosome population N, the crossover probability (pc) and the mutation probability (pm).
- Step 2: Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.
- Step 3: Randomly generate an initial population of chromosomes of size $N : x_1, x_2, \ldots, x_N$
- Step 4: Calculate the fitness of each individual chromosome: $f(x_1), f(x_2), \ldots, f(x_N)$.
- Step 5: Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness.
- Step 6: Create a pair of offspring chromosomes by applying the genetic operators- crossover and mutation.
- Step 7: Place the created offspring chromosomes in the new population.
- Step 8: Repeat Step 5 until the size of the new chromosome population becomes equal to the size of the initial population, N.
- Step 9: Replace the initial (parent) chromosome population with the new (offspring) population.
- Step 10: Go to Step 4, and repeat the process until the termination criterion is satisfied.

2.6 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a bio-inspired optimization method proposed by Eberhart and Kennedy [41–43] in 1995. PSO is as metaheuristic search technique based on a population of particles. The main idea of PSO comes from the social behavior of schools of fish and flocks of birds [44, 45]. In PSO each particle moves in a D-dimensional space based on its own past experience and those of other particles [46, 47]. Each particle has a position and a velocity represented by the vectors $x_i = \left(x_{i,1}, x_{i,2}, \ldots, x_{i,D}\right)$ and $v_i = \left(v_{i,1}, v_{i,2}, \ldots, v_{i,D}\right)$ for the i-th particle. At each iteration, particles are compared with each other to find the best particle [48, 49]. Each particle records its best position as $v_i = \left(v_{i,1}, v_{i,2}, \ldots, v_{i,D}\right)$. The best position of all particles in the swarm is called the global best, and is represented as $G = (G_1, G_2, \ldots, G_D)$. The velocity of each particle is given by Eq. (2.13).

$$v_{id} = wv_{id} + C_1 \cdot rand() \cdot (pbest_{id} - x_{id}) + C_2 \cdot rand() \cdot (gbest - x_{id}) \qquad (2.13)$$

In this equation $i = 1, 2, \ldots, M, d = 1, 2, \ldots D, C_1$ and C_2 are positive constants (known as acceleration constants), $rand_1()$ and $rand_2()$ are random numbers in

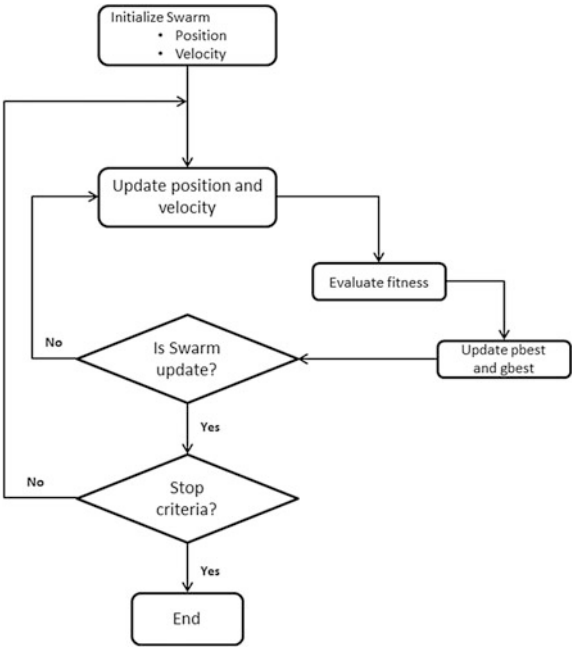

Fig. 2.7 Flowchart of the PSO algorithm

[0, 1], and w, introduced by Shi and Eberhart [50] is the inertia weight. The new position of the particle is determined by Eq. (2.14):

$$x_{id} = x_{id} + v_{id} \tag{2.14}$$

The basic functionally of the PSO is illustrated as follows (Fig. 2.7).

References

1. Cowpertwait, P., Metcalfe, A.: Time Series. Introductory Time Series with R, pp. 2–5. Springer, Dordrecht (2009)
2. Wei, W.W.S.: Time Series Analysis: Univariate and Multivariate Methods. Ed. Addison-Wesley **1**, 40–100 (1994)
3. Durbin, J., Koopman, S.J.: Time Series Analysis by State Space Methods. Oxford University Press (2001)
4. Erland, E., Ola, H.: Multivariate time series modeling, estimation and prediction of mortalities. Insur. Math. Econ. **65**, 156–171 (2015)
5. Weina, W., Witold, P., Xiaodong, L.: Time series long-term forecasting model based on information granules and fuzzy clustering. Eng. Appl. Artif. Intell. **41**, 17–24 (2015)
6. Shu-Xian, L., Xian-Shuang, Y., Hong-Yun, Q., Hai-Feng, H.: A novel model of leaky integrator echo state network for time-series prediction. Neurocomputing **159**, 58–66 (2015)
7. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time Series Analysis: Forecasting and Control, 3rd edn. Prentice Hall, New Jersey (1994)

8. Akhter, M.R., Arun, A., Sastry, V.N.: Recurrent neural network and a hybrid model for prediction of stock returns. Expert Syst. Appl. **42**, 3234–3241 (2015)
9. Castro, J.R., Castillo, O., Melin, P., Rodriguez, A.: A hybrid learning algorithm for interval type-2 fuzzy neural networks: the case of time series prediction, vol. 15a, pp. 363–386. Springer, Berlin (2008)
10. Karnik, N.N., Mendel, J.M.: An Introduction to Type-2 Fuzzy Logic Systems. University of Southern California, Los Angeles, CA (1998)
11. Mendel, J.M.: Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions. Prentice-Hall, NJ (2001)
12. Hagan, M.T., Demuth, H.B., Beale, M.H.: Neural Network Design. PWS Publishing, Boston, MA (1996)
13. Sharkey, A.: Combining artificial neural nets: ensemble and modular multi-net systems. Springer, London (1999)
14. Sollich, P., Krogh, A.: Learning with ensembles: how over-fitting can be useful. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems 8, pp. 190–196. MIT Press, Cambridge (1996)
15. Hampshire, J., Waibel, A.: A novel objective function for improved phoneme recognition using time-delay neural networks. IEEE Trans. Neural Netw. **1**(2), 216–228 (1990)
16. Wei, L.Y., Cheng, C.H.: A hybrid recurrent neural networks model based on synthesis features to forecast the Taiwan Stock Market. Int. J. Innovative Comput. Inf. Control **8**(8), 5559–5571 (2012)
17. Cherkauer, K.J.: Human expert level performance on a scientific image analysis task by a system using combined artificial neural networks. In: Chan, P., Stolfo, S., Wolpert, D. (eds.) Proceedings of AAAI-96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms, Portland, OR, AAAI Press, Menlo Park, CA, pp. 15–21 (1996)
18. Maclin, R., Shavlik, J.W.: Combining the predictions of multiple classifiers: using competitive learning to initialize neural networks. In: Proceedings of IJCAI-95, Montreal, Canada, Morgan Kaufmann, San Mateo, CA, pp. 524–530 (1995)
19. Liu, F., Quek, C., See, G.: Neural network model for time series prediction by reinforcement learning. In: Proceedings of the International Joint Conference on the Neural Networks, Montreal, Canada (2005)
20. Soltani, S.: On the use of the wavelet decomposition for time series prediction. Neurocomputing **48**(1–4), 267–277 (2002)
21. Krogh, A., Vedelsby, J.: Neural network ensembles, cross validation, and active learning. In: Tesauro, G., Touretzky, D., Leen, T. (eds.) Advances in neural information processing systems 7, MIT Press, Denver, CO, Cambridge, MA, pp. 231–238 (1995)
22. Opitz, D.W., Shavlik, J.W.: Generating accurate and diverse members of a neural network ensemble. in: D.S. Touretzky M.C., Mozer M.E., Hasselmo (Eds.), Advances in Neural Information Processing Systems 8, Denver, CO, MIT Press, Cambridge, MA, pp. 535–541, (1996)
23. Yao, X., Liu, F.: Evolving neural network ensembles by minimization of mutual information. Int. J. Hybrid Intell. Syst. **1**, 12–21 (2004)
24. Xue, J., Xu, Z., Watada, J.: Building an integrated hybrid model for short-term and mid-term load forecasting with genetic optimization. Int. J. Innovative Comput. Inf. Control **8**(10), 7381–7391 (2012)
25. Zadeh, L.A.: Fuzzy logic = computing with words. IEEE Trans. Fuzzy Syst. **4**(2), 103–111 (1996)
26. Zadeh, L.A.: Fuzzy logic. Computer **1**(4), 83–93 (1988)
27. Castillo, O., Melin, P.: Optimization of type-2 fuzzy systems based on bio-inspired methods: a concise review. Inf. Sci. **205**, 1–19 (2012)
28. Castro, J.R., Castillo, O., Martínez, L.G.: Interval type-2 fuzzy logic toolbox. Eng. Lett. **15**(1), 89–98 (2007)

29. Karnik, N.N., Mendel, J.M., Qilian, L.: Type-2 fuzzy logic systems. Fuzzy Syst. IEEE Trans. **7**(6), 643–658 (1999)
30. Mendel, J.M.: Why we need type-2 fuzzy logic systems. Article is provided courtesy of Prentice Hall, By Jerry Mendel (2001)
31. Wu, D., Mendel, J.M.: A vector similarity measure for interval type-2 fuzzy sets and type-1 fuzzy sets. Inf. Sci. **178**, 381–402 (2008)
32. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Michigan (1975)
33. Holland, J.H.: Outline for a logical theory of adaptive systems. J. Assoc. Comput. Mach. **3**, 297–314 (1962)
34. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company (1989)
35. Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: motivation, analysis, and first results. Complex Syst. **3**, 493–530 (1989)
36. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Gregory, J.E.R. (ed) Foundations of Genetic Algorithms, pp. 69–93. Morgan Kaufmann Publishers, San Mateo, California (1991)
37. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold (1991)
38. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. AI Series. Springer, New York (1994)
39. Koza, J.R.: Genetic Programming. On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge (1992)
40. Buckles, B.P., Petry, F.E.: Genetic Algorithms. IEEE Computer Society Press (1992)
41. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS), pp. 39–43 (1995)
42. Eberhart, R., Shi, Y., Kennedy, J.: Swarm Intelligence. Morgan Kaufmann, San Mateo, California (2001)
43. Engelbrech, P.: Fundamentals of Computational of Swarm Intelligence: Basic Particle Swarm Optimization, pp. 93–129. Wiley, New York (2005)
44. Escalante, H.J., Montes, M., Sucar, L.E.: Particle swarm model selection. J. Mach. Learn. Res. **10**, 405–440 (2009)
45. Kennedy, J., Eberhart, R.: Particle swam optimization. Proc. IEEE Int. Conf. Neural Network (ICNN) **4**, 1942–1948 (1995)
46. Clerc, M., Kennedy, J.: The particle swarm-explosion, stability, and convergence in a multimodal complex space. IEEE Trans. Evol. Comput. **6**, 58–73 (2002)
47. Clerc, M.: The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. Proc. IEEE Congr. Evol. Comput. **3**, 1951–1957 (1999)
48. Eberhart, R., Shi, Y.: Comparing inertia weights and constriction factors in particle swarm optimization. Proc. IEEE Congr. Evol. Comput. **1**, 84–88 (2000)
49. Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimization Intelligence: Advances and Applications. Information Science Reference, pp. 18–40. USA (2010)
50. Shi, Y., Eberhart, R.C.: A modified particle swarm optimizer. In: Proceedings of the IEEE Congress of Evolutionary Computation, pp. 69–73 (1998)