Chapter 2
Fundamentals of Intelligent Decision-Making Techniques

Abstract The fundamentals of computational intelligence (CI) techniques are introduced briefly in this chapter. The definition and classification of CI techniques are introduced firstly. Some representative CI techniques, especially those that have been usually used in solving decision-making problems in production and retail operations, are then presented to help readers understand CI techniques used in subsequent chapters. These techniques include evolutionary computation techniques and feedforward neural networks. The fundamentals of these techniques are introduced, including their origins, fundamental characteristics, applications, and the procedures to implement them.

Keywords Evolution algorithms · Neural network · Genetic algorithm · Harmony search

2.1 Computational Intelligence Techniques: A Brief Overview

Computational intelligence (CI) is a multidisciplinary subject which has attracted researchers and practitioners from a variety of fields, such as computing, neuroscience, mathematics, and linguistics. The popularity of CI techniques has been rapidly increasing in recent years, which involves a large variety of subfields in science and engineering, from general-purpose fields, such as decision-making, perception, and logical reasoning, to specific fields, such as robot control and disease diagnosis. CI techniques have received increasing attention from researchers and practitioners in manufacturing and retail industries over the last two decades and have been employed to handling a variety of decision-making problems in production and retail operations, such as plant location selection, production planning and control, and sales forecasting.

Z. Guo, *Intelligent Decision-making Models for Production and Retail Operations*, DOI 10.1007/978-3-662-52681-1_2

2.1.1 What Is CI?

This is a very confusing issue although the name 'computational intelligence' has been used for over 30 years (Bezdek 2013). There is no definite definition of CI, which became a new buzzword that means different things to different people. Researchers from computer science are usually interested in the creation of intelligent systems and programs capable of reproducing human-like behavior, such as understanding languages and learning from experience. On the other hand, engineering researchers place more emphasis on using CI as a problem solver.

The first published definition of CI is attributed to J.C Bezdek who defined CI as 'computational systems depend on numerical data supplied by manufactured sensors and do not rely upon "knowledge"' (Bezdek 1992). Later, in 1994, Bezdek defined that CI is 'low-level computation in the style of the mind.' He stated that 'A system is computationally intelligent when it (1) deals only with numerical (low level) data, (2) has a pattern recognition component, and (3) does not use knowledge in the AI sense' (Bezdek 1994).

Eberhard et al. (1996) defined CI as 'methodology involving computing (whether with a computer, wetware, etc.) that exhibits an ability to learn and/or deal with new situations such that the system is perceived to possess one or more attributes of reason, such as generalization, discovery, association, and abstraction. The output of a computationally intelligent system often includes predictions and/or decisions.'

Engelbrecht (2002) defined CI as the study of 'adaptive mechanisms' which enable or facilitate intelligent behavior in complex and changing environments. In other words, CI is mainly about the design of algorithmic models to solve complex problems. The three main CI paradigms are artificial neural networks, evolutionary computation, and fuzzy logic systems.

Duch (2007) defined broadly CI as 'a branch of computer science studying problems for which there are no effective computational algorithms.'

2.1.2 Why Do We Need CI?

Problem-solving techniques can be roughly classified as traditional and CI ones. Traditional techniques have difficulties in solving complex decision-making problems, such as large-sized NP-hard problems and strongly nonlinear forecasting problem.

CI techniques are able to provide effective solutions to a wide range of complex decision-making problems due to their abilities to emulate intelligent processes, as opposed to traditional techniques. The problems include modeling, classification, optimization, and forecasting, which involve various application domains, including manufacturing and service industries, business and finance, computer science, and telecommunications. Some real-world problems are very complex and

intractable, such as production scheduling and planning, assembly line balancing, and sales forecasting.

CI is also an effective supplement of natural intelligence because it builds intelligence into computer systems, which can effectively perform particular tasks, such as robot control. It is helpful to reduce human labor and mistakes.

2.1.3 Classification of CI Techniques

CI techniques mainly include evolutionary computation, artificial neural networks, and fuzzy logic systems. A brief introduction to these techniques is given below.

2.1.3.1 Evolutionary Computation

Evolutionary computation is an umbrella term for a range of evolutionary computation techniques inspired by optimum-seeking mechanisms from real world, such as natural selection and genetic inheritance, which simulate evolution processes on a computer to iteratively improve the performance of solutions until an optimal (or feasible at least) solution is obtained.

Evolutionary computation techniques make few or no assumptions about the problem to be optimized. They are powerful in dealing with optimization problems with complex problem features such as large solution spaces and NP-hard nature, which traditional techniques fail to handle. These techniques are one of the fastest growing areas of computer science and engineering and are being increasingly widely utilized to handle a large variety of problems, ranging from practical applications in industry to leading-edge scientific research, such as complex production scheduling and stochastic combinatorial optimization.

Broadly speaking, evolutionary computation includes evolutionary algorithms, such as genetic algorithm, evolution strategy and memetic algorithms, and swarm intelligence, such as ant colony algorithms, particle swarm optimization, artificial immune systems, and harmony search. We will introduce several representative evolutionary computation techniques in the field of evolutionary computation in Sect. 2.2.

2.1.3.2 Neural Network

An artificial neural network, usually called neural network (NN), is a computational model inspired by research into biologic neural networks, which is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use (Haykin 2001). An NN is composed of a

number of interconnected neurons (or nodes), which are analogous to biologic neurons in the human brain, according to some patterns of connectivity. In most cases, an NN is an adaptive system, which discovers the relationships between training input and associated outputs by adjusting the network parameters on the basis of data patterns of training samples. The history of NNs can be traced back to 1943 when physiologists McCulloch and Pitts created the model of a neuron as a binary linear threshold unit (McCulloch and Pitts 1943). One of the most well-known features of NNs is that it can be used as universal function approximator (Scarselli and Tsoi 1998; Zhang et al. 2012). In view of this feature, NNs have been widely utilized to solve a variety of relevant decision-making problems, including modeling, classification, clustering, and forecasting.

To construct an NN, one needs to determine the following three issues:

(1) Network architecture, including the number of input neurons, the number of hidden layers, the number of hidden neurons in each hidden layer, the number of output neurons, and the interconnections among these neurons;
(2) Activation function, which stipulates the relationship between input and output of a neuron;
(3) Learning algorithm, which determines how the NN adjusts the values of connection weights among network neurons.

According to different settings of the above issues, there exist various types of NNs, such as feedforward NNs (FNNs), recurrent NNs, and random NNs. This chapter will introduce FNNs in Sect. 2.3.

2.2 Evolutionary Computation

The processes of optimum-seeking have been remarkably successful in a variety of real-world phenomena, such as human evolution, food-seeking of ant colonies, and improvisation of musicians. These phenomena work toward a perfect individual to fill a particular environmental niche by using stochastic heuristic individual searches and generation processes. It is naturally expected to develop evolutionary optimization processes by modeling the behaviors of these phenomena. The evolutionary computation techniques were thus proposed to perform this function, which mimics the optimum-seeking processes of these phenomena by a computer program.

This section will introduce several representative evolutionary computation techniques, including genetic algorithms (GAs), evolution strategies (ESs), harmony searches (HSs), and memetic algorithms (MAs).

2.2.1 Optimum-Seeking Mechanism of Evolutionary Computation Techniques

Evolutionary computation techniques have a similar optimum-seeking mechanism although they are inspired by different real-world phenomena. A general flowchart of evolutionary computation techniques is shown in Fig. 2.1. The procedures involved are described as follows:

(1) Generate initial individual population: Each solution individual is generated based on prespecified solution representation and population size.
(2) Evaluate solution individual: Evaluate the performance (fitness) of solution individuals newly generated based on a given performance measure.
(3) Check termination criteria: Check whether termination criteria are met. If so, return the best solution individual as the optimal solution; otherwise, go to the next loop for generating new individuals.
(4) Generate new individuals: Each new solution individual is generated based on one or more individuals in the current population. Different evolutionary computation techniques generate new individuals according to different methods.
(5) Form next individual population: A specified number of individuals are chosen from the newly generated solution individuals and the current population to form the next population (called offspring population).

To design and develop an evolutionary computation technique for handling an optimization problem, one needs to make a variety of design decisions such as:

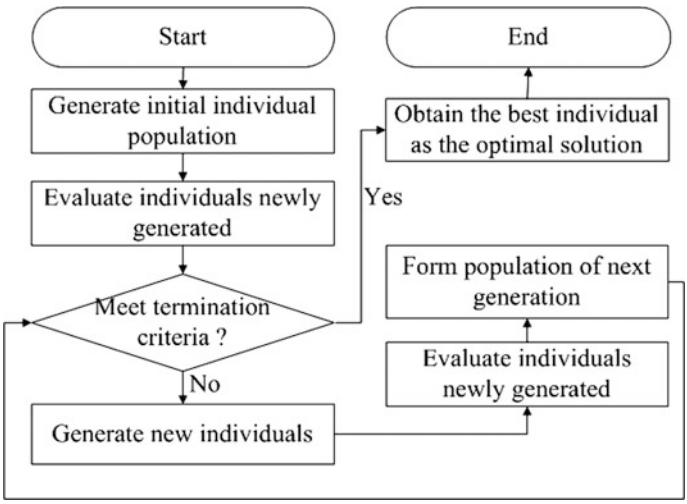

Fig. 2.1 General flowchart of evolutionary computation techniques

- choosing a particular paradigm that is suitable for the problem,
- choosing an appropriate solution representation and population size,
- choosing an appropriate method to generate new solution individuals,
- choosing an appropriate mechanism to form the offspring population,
- choosing an appropriate performance measure to evaluate individuals,
- choosing an appropriate termination criterion.

2.2.2 *Brief Introduction to Genetic Algorithm*

The GA is the most popular technique under the umbrella of evolutionary computation, which is inspired by the principles of genetics and natural selection—Darwin's 'survival of the fittest' theory. The origin of the GA can be traced back to the early 1950s when several biologists used computer programs to simulate biologic systems (Goldberg 1989). However, the popularization of GAs is accredited to the work done in the late 1960s and early 1970s under the direction of John Holland (1975).

The optimum-seeking mechanism of a GA is analogous to the biologic evolutionary process. The GA operates on a population of chromosomes (also called solution individuals). Each individual represents a feasible solution to the investigated problem. Different solution representations have been developed to represent individuals, including real-coded representation and order-based representation. According to the evolutionary principle, the individuals adapting to the environment in the parental population have a higher probability to survive and generate offspring by transmitting their biologic heredity to the next population (offspring population). The child individuals are generated by using a set of biologically inspired genetic operators, including selection, crossover, and mutation. The child chromosomes are supposed to inherit good genes from their parents so that the average quality of solutions is superior to ones in previous generations.

Figure 2.2 shows the flowchart of a canonical GA. GA works iteratively. Each single iteration is called a generation. In each generation, the fitness of each individual is evaluated and determined by the fitness function. When the fitness function value of an individual is larger, the individual becomes fitter, indicating that the individual has a bigger opportunity to survive in the next generation. This evolution process is repeated until some termination criteria are met. Selection operators determine which individuals are selected for mating from the current generation. Crossover and mutation operators are employed to create offspring individuals based on individuals selected by selection operators. The entire set of generations is called a run. At the end of a run, one or more individuals with the highest fitness values are selected as optimal solutions.

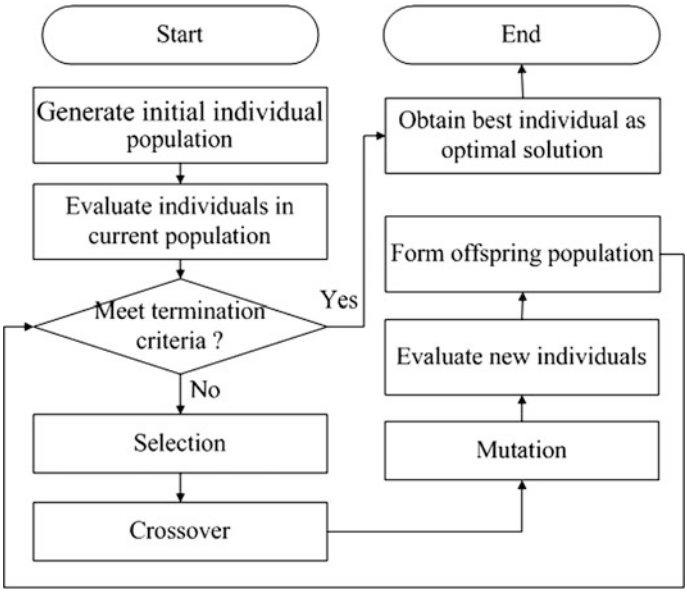

Fig. 2.2 Flowchart of a canonical GA

2.2.3 Brief Introduction to Evolution Strategy

The ES is another evolutionary computation technique of mimicking natural evolution, which was invented by Ingo Rechenberg and Hans-Paul Schwefel in the early 1960s (Rochenberg 1965; Schwefel 1975) to solve parameter optimization problems.

The general flowchart of an ES is shown in Fig. 2.3, which is very similar to that of a GA. The only difference is that an ES does not use the crossover operator and uses only mutation operator. The earliest ES model, termed as (1 + 1)-ES, was based on a population having one solution individual (chromosome) only. Hansen and Kern (2004) have pointed out that ESs with the population of μ ($\mu > 1$) individuals are less prone to getting stuck in the local optima. In these ESs, a new (child) individual is generated by selecting a parental individual randomly to undergo mutation. In each generation, λ child individuals are generated. ESs can be classified into (μ, λ)-ES and ($\mu + \lambda$)-ES. The two types use different strategies to generate child populations:

(μ, λ)-ES: The next population is composed of the μ best individuals from the population of λ newly generated offspring.

($\mu + \lambda$)-ES: The next population is composed of the μ best individuals from μ parents and λ newly generated offspring.

The ES has been modified to tackle combinatorial optimization problems although it was initially developed for continuous optimization. Some researchers

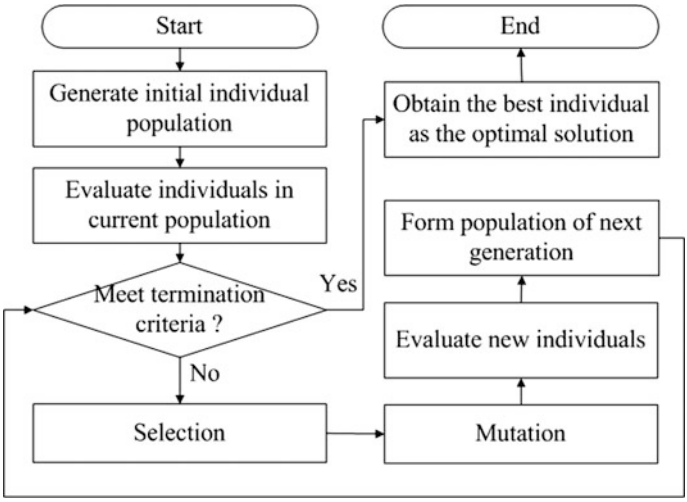

Fig. 2.3 Flowchart of a canonical ES

extended the ES to recombination, which leads to more general notation $(\mu/\rho^+, \lambda)$-ES. ρ denotes the number of parents involved in the generation of one offspring (mixing number). For $\rho = 1$, we have ES cases (μ, λ) and $(\mu + \lambda)$ without recombination operation. For $\rho > 1$, we have ES cases with recombination operation. Facing different optimization problems, like GAs, different solution representations and evolutionary operators in ESs are required to adapt themselves to these problems.

2.2.4 *Brief Introduction to Harmony Search*

Some evolutionary computation techniques do not originate in natural evolution. The HS is a relatively new evolutionary computation algorithm developed by Geem et al. (2001), which is inspired by musicians' improvisation of their instruments' pitches to search for perfect harmony.

The HS generates a new solution individual (harmony, solution vector) by considering all existing individuals instead of considering only one or two parental individuals like traditional evolutionary algorithms (e.g., ES and GA). This distinct feature of the HS is helpful to improve the algorithm's flexibility so that it can produce better solutions than conventional mathematical methods and GA- and ES-based approaches do (Lee and Geem 2004; Mahdavi et al. 2007).

The flowchart of a canonical HS is shown in Fig. 2.4. The initial harmony memory is generated randomly, in which each individual (harmony, solution vector) v represents a distinct feasible solution of all decision variables. That is, $v = [v_1, v_2, \ldots, v_P]$. The performance (fitness) of each individual is evaluated and

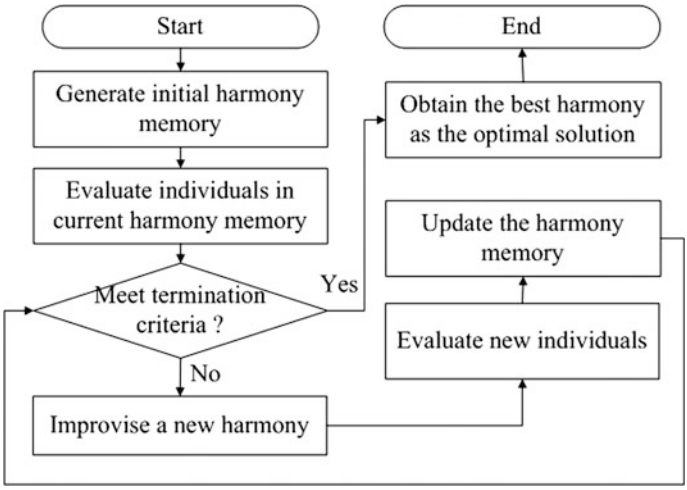

Fig. 2.4 Flowchart of a canonical harmony search

determined by the fitness function. When the fitness function of an individual is larger, the performance of the individual is better. This evolution process is repeated until some termination criteria are satisfied. After the fitness values of all individuals in the population are calculated, two HS procedures, memory consideration and pitch adjustment, are then used to improvise (generate) a new individual. Generating a new individual (harmony) is called improvisation.

2.2.5 Brief Introduction to Memetic Algorithm

The MA represents one of the recent growing research areas in evolutionary computation, which was introduced by P. Moscato of California Institute of Technology in his technical report (Moscato 1989) in 1989. Inspired by both Darwinian principles of natural evolution and Dawkin's notion of a meme, the MA is the algorithmic pairing of a population-based global search method with one local refinement method.

The general flowchart of the MA is shown in Fig. 2.5, which is very similar to that of a GA. The only difference is that the MA hybridizes a local search process to improve locally each newly generated individual. The local search process might be implemented by heuristics techniques such as tabu search and simulated annealing techniques, approximation algorithms, or, sometimes, even (partial) exact methods. The hybridization is helpful either to accelerate the discovery of effective solutions, for which evolution alone would take too long to discover, or to find solutions that would otherwise be unreachable by evolution or a local method alone.

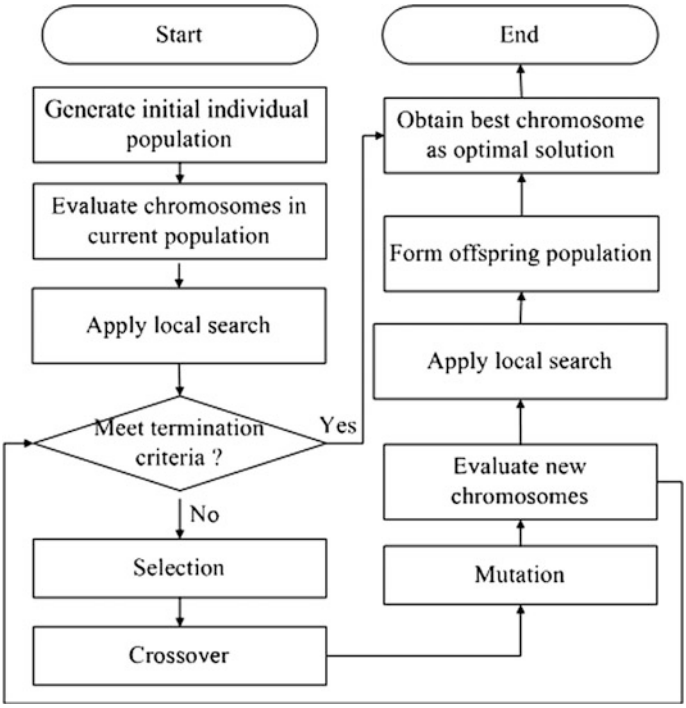

Fig. 2.5 Flowchart of a canonical MA

MAs have been successfully applied to a wide range of domains that cover problems in combinatorial optimization, continuous optimization, multi-objective optimization, and so on. In particular, remarkable success on significant instantiations of MAs across a wide range of real-world applications has been reported, ranging from the field of timetabling (Alkan and Ozcan 2003; Schonberger et al. 2004), production scheduling (Caumond et al. 2008; Guo et al. 2013), vehicle routing (Tavakkoli-Moghadam et al. 2006; Ngueveu et al. 2010), to nonlinear programming problems including aerodynamic design (Ong et al. 2003) and optimal control systems with machine learning (Caponio et al. 2007).

2.3 Feedforward Neural Networks

Feedforward neural networks (FNNs) are the most common type of NNs, which have been used in a wide variety of real-world applications, including modeling, pattern recognition, clustering and classification, and forecasting. Applications of FNNs in production and retail operations involve mainly forecasting, classification, and model identification (Guo et al. 2011).

2.3.1 Brief Introduction to FNNs

FNNs are a type of NNs where connections among units do not travel in a loop but in a single directed path. Typically, an FNN is composed of an input layer, one or more hidden layers, and an output layer of neurons. Each layer consists of one or more neurons. The input layer and output layer form bookends for hidden layers of neurons. Signals are propagated from the input neurons to hidden neurons and then onto output neurons, which output responses of the network to outside users. That is, signals only transfer in a forward direction on a layer-by-layer basis. Figure 2.6 shows a typical FNN with one hidden layer.

In the NN, a neuron is represented as a mathematical function, which is the abstraction of biologic neurons. Figure 2.7 shows a typical neuron. The neuron receives signals from its inputs $x_i (i = 1, \ldots, n)$ (representing one or more dendrites of a biologic neuron) and an externally applied bias b. The weighted summation $X (X = \sum_{i=1}^{n} x_i w_i + b)$ of these input signals is then passed through activation function $f(X)$ to generate output signal Y (representing a biologic neuron's axon). We consider the effects of the bias by (1) adding a new input signal fixed at +1 and (2) adding a new synaptic weight equal to bias b. That is, $x_0 = 1, w_0 = b$. It is clear that $Y = f(X) = f(\sum_{i=0}^{n} x_i w_i)$. The input signal $x_i (i = 1, \ldots, n)$ could be raw data or outputs of other neurons. Output signal Y could be either a final solution to the problem or an input to other neurons. It should be noted that, for simplicity, the NN shown in Fig. 2.6 does not contain bias signals, which is feasible for practical applications.

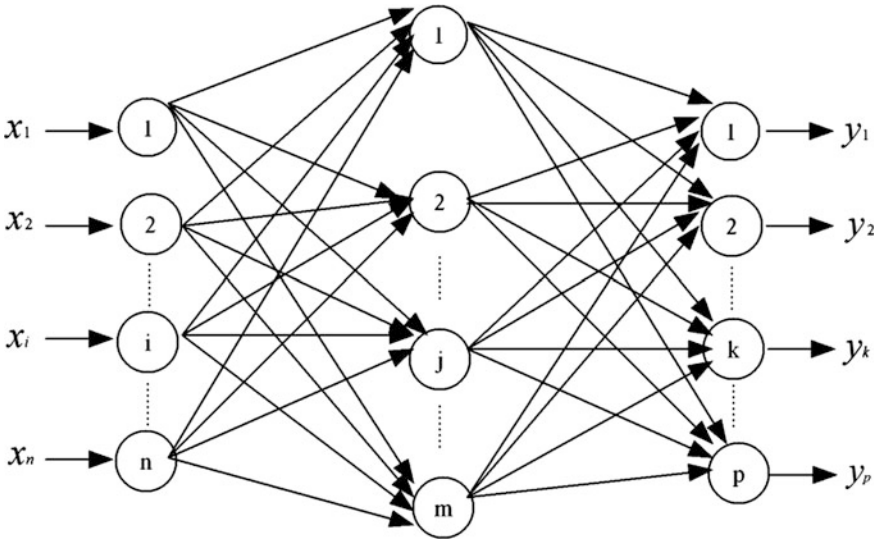

Fig. 2.6 FNN with one hidden layer

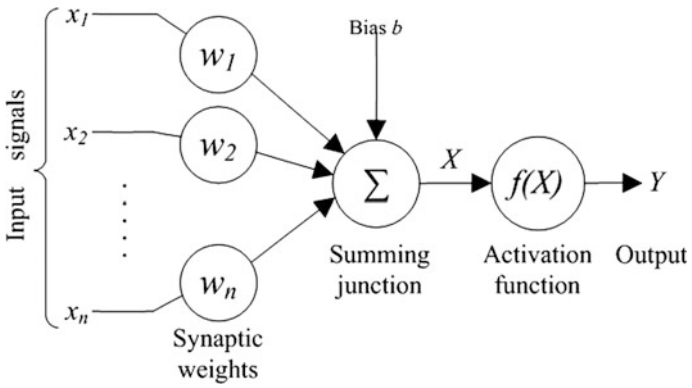

Fig. 2.7 Diagram of a neuron

A variety of FNNs have been presented, including backpropagation (BP) networks, extreme learning machine (ELM) networks, self-organizing map networks, learning vector quantization networks, and radial basis function networks. These FNNs have the capability of approximating generic classes of functions (Scarselli and Tsoi 1998; Zhang et al. 2012) and are constructed in terms of different settings from the following three perspectives.

Network architecture: In traditional FNNs, neurons are by default connected fully between neighboring layers (see Fig. 2.6) in order to simplify the network design although fully connected NNs are biologically unrealistic (Wong et al. 2010). To simplify the network structure of FNNs and improve their generalization capability, some partially connected FNNs have been developed (Elizondo and Fiesler 1997; Wong et al. 2010). However, fully connected FNNs are still dominant because the design of partially connected FNNs is more complicated and usually data-dependent. In FNNs, BP networks can have more than one hidden layer, while ELM networks and radial basis function networks have one hidden layer only.

Activation function: There are generally two different activation functions in a particular NN although each neuron can have its own activation function. Neurons in the input layer use the identity function as the activation function. That is, the output of an input neuron equals its input. The activation functions of hidden and output neurons can be differentiable and nonlinear in theory. Several 'well-behaved' (bounded, monotonically increasing and differentiable) activation functions are commonly used in practice, including (1) the sigmoid function $f(X) = (1 + \exp(-X))^{-1}$; (2) the hyperbolic tangent function $f(X) = (\exp(X) - \exp(-X))/(\exp(X) + \exp(-X))$; (3) the sine or cosine function $f(X) = \sin(X)$ or $f(X) = \cos(X)$; (4) the linear function $f(X) = X$; and (5) the radial basis function. Among these functions, the sigmoid function is the most popular one, while the radial basis function is only used for radial basis function networks.

Learning algorithm: Traditionally, NN learning is an algorithmic procedure whereby NN parameters (such as weights) are estimated. Within this framework, two categories of learning are usually considered: supervised learning and unsupervised learning. Learning can be 'supervised' since an NN needs to fulfill a function known in some or even all data points: A 'teacher' provides sample data of inputs and corresponding outputs of a task that an NN performs. The most popular supervised learning algorithm is the BP algorithm. In contrast to supervised learning, unsupervised learning does not require a 'teacher.' During the learning process, an NN receives a number of input patterns, discovers significant features self-adaptively in these patterns, and learns how to classify input data into categories in an appropriate manner. The most popular unsupervised learning algorithm is the self-organizing map.

2.3.2 *Backpropagation Networks*

The BP network is the most commonly used FNN. Its structure is the same to that shown in Fig. 2.6 except that it can include more than one hidden layer. A BP algorithm is used for the learning process of BP network, which is described in detail as follows.

Given a desired output response vector $d = [d_1, d_2, \ldots, d_p, \ldots, d_P]$, the learning algorithm performs an optimization process to seek the optimal connection weights so that each output error e_p, defined as the error between the desired output d_p and the output of network o_p, is minimized. That is,

$$\min_{w \in R^n} E(w) = \frac{1}{2}\sum_{p=1}^{P} [d_p - o_p]^2 = \frac{1}{2}\sum_{p=1}^{P} e_p^2$$

Consider an FNN with L-1 ($L > 2$) hidden layers. Let neuron(i, l) be the ith neuron in the lth layer and w_{ji}^l be the connection weight between neuron(j, l) and neuron$(i, l-1)$. I_{ji}^l denotes the ith input of neuron(j, l), which is the output o_i^{l-1} of neuron$(i, l-1)$ (i.e., $I_{ji}^l = o_i^{l-1}$). The BP algorithm can be implemented on the basis of the following steps.

Step 1 Set a learning rate η ($0 \le \eta \le 1$).

Step 2 Set all connection weights $w_{ji}^l(0)$ to random numbers uniformly distributed inside a small range. A feasible empirical range is $(-2.4/N_j^l, +2.4/N_j^l)$ (Haykin 1994), where N_j^l is the total number of inputs of neuron(j, l).

Step 3 Select a random input pattern with its corresponding target output from training sample data.

Step 4 Assign the appropriate value of the input vector to each neuron in the input layer. Feed this input to all neurons in the first hidden layer.

Step 5 For neuron(j,l) in hidden and output layers (i.e., $2 \le l \le L$), calculate its total input net_j^l,

$$\text{net}_j^l = \sum_i I_{ji}^l w_{ji}^l$$

where I_{j0}^l equals 1 and w_{j0}^l equals the bias b_j^l applied to neuron(j,l). The output of this neuron is $f(\text{net}_j^l)$. $f(.)$ is the activation function that can be any function with bounded derivatives.

Step 6 Calculate the error signal at output neuron neuron(k,L),

$$\delta_k^L = -\frac{\partial E}{\partial \text{net}_k^L} = -\frac{\partial E}{\partial o_k^L} \cdot \frac{\partial o_k^L}{\partial \text{net}_k^L} = (d_k^L - o_k^L) \cdot f'(\text{net}_k^L)$$

Step 7 Calculate the error signal for each neuron neuron(j,l) in hidden layers $(2 \le l \le L-1)$,

$$\delta_j^l = -\frac{\partial E}{\partial \text{net}_j^l} = -\frac{\partial E}{\partial o_j^l} \cdot \frac{\partial o_j^l}{\partial \text{net}_j^l} = f'(\text{net}_j^l) \sum_k \delta_k^{l+1} w_{kj}^{l+1}$$

Step 8 Update the weights for all layers $w_{ji}^l(n+1) = w_{ji}^l(n) + \eta \delta_j^l I_{ji}^l$

Step 9 Go to Step 3 and continue until the value of the error function has become sufficiently small.

2.3.3 *Extreme Learning Machine Networks*

The major drawback of BP network is its slow convergence speed caused by the local minima. The ELM network has the capability of providing better generalization and much faster learning speed than BP networks. The ELM network is a novel type of FNNs, which is proposed by Huang et al. at Nanyang Technological University, Singapore, in 2004 (Huang et al. 2004). Within contrast to BP networks, ELM networks have only one hidden layer and utilize ELM as learning algorithms.

Figure 2.6 shows the structure of the ELM network. It is assumed that the ELM network with m hidden neurons and activation function $f(x)$ is trained to approximate N distinct sample data (u_i, y_i) with zero error means, where u_i is the input of sample data and $u_i = [u_{i1}, u_{i2}, \ldots, u_{in}]^{\text{T}} \in R^n$, y_i is the output of sample data and $y_i = [y_{i1}, y_{i2}, \ldots, y_{ip}]^{\text{T}} \in R^p$. In ELM networks, input weights and hidden biases are randomly generated. The nonlinear ELM network can thus be converted into the following linear system.

$$M\beta = T,$$

where $M = \{h_{ij}\}$ $(i = 1, \ldots, N$ and $j = 1, \ldots, m)$ denotes the hidden layer output matrix and $h_{ij} = f(w_j \cdot u_i + b_j)$ is the output of the jth hidden neuron neuron$(j, 2)$ with respect to u_i; $w_j = [w_{j1}, w_{j2}, \ldots, w_{jn}]^{\mathrm{T}}$ denotes the weight vector connecting neuron$(j, 2)$ and input neurons, and b_j is the bias of neuron$(j, 2)$; $w_j \cdot u_i$ denotes the inner product of w_j and u_i; $\beta = [\beta_1, \ldots, \beta_j, \ldots, \beta_m]^{\mathrm{T}}$ $(j = 1, \ldots, m)$ is the matrix of output weights and $\beta_j = [\beta_{j1}, \beta_{j2}, \ldots, \beta_{jp}]^{\mathrm{T}}$ represents the weight vector connecting neuron$(j, 2)$ and output neurons; $Y = [y_1, y_2, \ldots, y_N]^{\mathrm{T}}$ denotes the matrix of targets (desired outputs).

The determination of output weights between hidden and output layers is to find the least-square solution to the given linear system. The minimum norm least-square solution to linear system $(M\beta = T)$ is

$$\hat{\beta} = M^{\dagger} Y,$$

where $M^{\dagger}$ is the Moore–Penrose generalized inverse of matrix M. The minimum norm least-square solution is unique and has the smallest norm among the least-square solutions.

In contrast to BP algorithms, ELM has much faster learning and convergence speed because its network weights are obtained by using random generation and a least-mean squares method based on the Moore–Penrose's generalized inverse, instead of using iterative weight adjustment. Moreover, ELM can avoid difficulties experienced by BP algorithms, such as selection of termination criteria, learning rate, and learning epochs due to its distinct learning mechanism.

2.4 Summary

This chapter provides a brief introduction to the family of CI techniques so that readers can gain a basic understanding of the CI family and various CI techniques and understand the subsequent chapters more easily. This chapter introduces the definition of computational intelligence and presents a brief overview of computational intelligent techniques. Some representative CI techniques are briefly introduced, all of which have been used for decision-making in the fashion supply chain. We also discuss the origins of these techniques, fundamental characteristics, possible applications, as well as the procedures to implement them.

A number of research outputs show the effectiveness of CI techniques for decision-making in the fashion industry, as well as their superiority over classical approaches (Guo et al. 2011). The subsequent chapters will introduce several representative applications of CI in supply chain operations.

References

Alkan, A., & Ozcan, E. (2003). Memetic algorithms for timetabling. In: *Proceedings of CEC: 2003 Congress on Evolutionary Computation* (Vols 1–4, pp. 1796–1802).

Bezdek, J. C. (1992). On the Relationship between Neural Networks, Pattern Recognition, and Intelligence. *International Journal of Approximate Reasoning, 62*(2), 85–107.

Bezdek, J. C. (1994). What is computational intelligence? In J. Zurada, B. Marks & C. Robinson (eds.), *Computational intelligence imitating life*. Piscataway, NJ: IEEE Press.

Bezdek, J. C. (2013). The history, philosophy and development of computational intelligence (How a simple tune became a monster hit). In H. Ishibuchi (ed.), *Computational intelligence*. Oxford, UK: Eolss Publishers.

Caponio, A., Cascella, G. L., Neri, F., Salvatore, N., & Sumner, M. (2007). A fast adaptive memetic algorithm for online and offline control design of PMSM drives. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics, 37*(1), 28–41.

Caumond, A., Lacomme, P., & TcherneVa, N. (2008). A memetic algorithm for the job-shop with time-lags. *Computers & Operations Research, 35*(7), 2331–2356.

Duch, W. (2007). What is computational intelligence and where is it going? In W. Duch & J. Mandziuk (eds.), *Challenges for computational intelligence* (Vol. 63, pp. 1–13). Berlin, Heidelberg: Springer.

Eberhart, R., Dobbins, R. W., & Simpson, P. K. (1996). *Computational intelligence PC tools*. New York: Academic Press Professional.

Elizondo, D., & Fiesler, E. (1997). A survey of partially connected neural networks. *International Journal Neural Systems, 8*(5–6), 535–558.

Engelbrecht, A. P. (2002). *Computational intelligence: An introduction*. Chichester: John Wiley.

Geem, Z., Kim, J., & Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation, 76*(2), 60–68.

Goldberg, D. E. (1989). *Genetic algorithms in search optimization and machine learning*. Reading, MA: Addison-Wesley.

Guo, Z., Wong, W., & Leung, S. (2013). A hybrid intelligent model for order allocation planning in make-to-order manufacturing. *Applied Soft Computing, 13*(3), 1376–1390.

Guo, Z., Wong, W., Leung, S., & Li, M. (2011). Applications of artificial intelligence in the apparel industry: a review. *Textile Research Journal, 81*(18), 1871–1892.

Hansen, N., & Kern, S. (Eds.). (2004). *Evaluating the CMA evolution strategy on multimodal test functions. Parallel problem solving from nature—PPSN VIII*. Berlin, Heidelberg: Springer.

Haykin, S. (1994). *Neural networks: A comprehensive foundation*. New York: Macmillan College Publishing Company.

Haykin, S. (2001). Feedforward neural networks: An introduction. In *Nonlinear dynamical systems: Feed forward neural network perspectives*. New York: John Wiley.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN2004).* Budapest, Hungary.

Lee, K., & Geem, Z. (2004). A new structural optimization method based on the harmony search algorithm. *Computers & Structures, 82*(9–10), 781–798.

Mahdavi, M., Fesanghary, M., & Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation, 188*(2), 1567–1579.

McCulloch, W. S., & Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115–133.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program (report 826).*

Ngueveu, S., Prins, C., & Calvo, R. (2010). An effective memetic algorithm for the cumulative capacitated vehicle routing problem. *Computers & Operations Research, 37*(11), 1877–1885.

Ong, Y. S., Nair, P. B., & Keane, A. J. (2003). Evolutionary optimization of computationally expensive problems via surrogate modeling. *American Institute of Aeronautics and Astronautics Journal, 41*(4), 687–696.

Rochenberg, I. (1965). *Cybernetic Solution Path of an Experimental Problem.* Ministry of Aviation, Royal Aircraft Establishment, Library Translation No. 1122, August.

Scarselli, F., & Tsoi, A. (1998). Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks, 11*(1), 15–37.

Schonberger, J., Mattfeld, D., & Kopfer, H. (2004). Memetic Algorithm timetabling for non-commercial sport leagues. *European Journal of Operational Research, 153*(1), 102–116.

Schwefel, H. P. (1975). Evolutionsstrategie und numerische Optimierung. *Dissertation*, TU, Berlin, Germany.

Tavakkoli-Moghadam, R., Saremi, A., & Ziaee, M. (2006). A memetic algorithm for a vehicle routing problem with backhauls. *Applied Mathematics and Computation, 181*(2), 1049–1060.

Wong, W., Guo, Z., & Leung, S. (2010). Partially connected feedforward neural networks on Apollonian networks. *Physica A-Statistical Mechanics and Its Applications, 389*(22), 5298–5307.

Zhang, R., Lan, Y., Huang, G.-B., & Xu, Z.-B. (2012). Universal Approximation of Extreme Learning Machine With Adaptive Growth of Hidden Nodes. *IEEE Transactions on Neural Networks and Learning Systems, 23*(2), 365–371.

http://www.springer.com/978-3-662-52679-8

Intelligent Decision-making Models for Production and Retail Operations

Guo, Z.

2016, XI, 324 p. 84 illus., Hardcover
ISBN: 978-3-662-52679-8