Xiu Zhang
Xin Zhang
Wei Wang

Intelligent Information Processing with Matlab

Springer

Intelligent Information Processing with Matlab

Xiu Zhang · Xin Zhang · Wei Wang

Intelligent Information Processing with Matlab

Springer

Xiu Zhang
Tianjin Normal University
Tianjin, China

Xin Zhang
Tianjin Normal University
Tianjin, China

Wei Wang
Tianjin Normal University
Tianjin, China

ISBN 978-981-99-6448-2 ISBN 978-981-99-6449-9 (eBook)
https://doi.org/10.1007/978-981-99-6449-9

Preface

The speed of modern science and technology development is becoming faster and faster. The amount of new science and technology knowledge and information is rapidly increasing. A British scholar pointed out that the doubling cycle of human knowledge was 50 years in the nineteenth century, around 10 years in the first half of the twentieth century, and almost doubled every 3 years by the end of the 1980s. Recently, there are 13,000–14,000 papers being published daily worldwide. The continuous emergence of new theories, materials, processes, and methods has accelerated the pace of knowledge aging. In the past 30 years, the information produced by humans has exceeded the total information production of the past 5000 years. This background shows that we are in an era of information explosion. Thus, intelligent information processing becomes very important.

This book is aimed for sophomore and junior students of university. We focus on intelligent information processing algorithms and their mathematical principles. The presentation of the algorithms is simplified without too many advanced mathematics and object-oriented programming skills. We believe that the concepts and algorithms of intelligent information processing can be learning. We also expect that students could go beyond trial-and-error play. Students can be able to use and apply intelligent information processing algorithms to solve problems in the real world. This book can serve as a bridge to the study of intelligent information processing at the senior undergraduate or postgraduate levels.

The required background is a knowledge of advanced mathematics and programming. From advanced mathematics, the knowledge should include linear algebra, calculus, and probability. From programming, the knowledge should include sequential structure, selection structure, and loop structure. MATLAB programming language is preferred to run the examples of this book. Python programming language is also popular in artificial intelligence field; however, the packages for running Python programs are sometimes hard to manage. Thus, we implement the examples by using MATLAB programming language. In the past course learning process, students were able to implement most examples themselves using Python programming language. Please let us know if you implement the examples by using other programming language, and we will post a link on the book's website.

Chapter 1 describes the artificial neural network model and presents several commonly used neural networks. Chapter 2 presents convolutional neural network and specifies the metrics to evaluate the performance of neural networks. The research progress of neural networks is also given in Chap. 2. Chapter 3 presents an overview of fuzzy computing and specifies the typical problems that fuzzy computing can solve. Chapter 4 describes the fuzzy neural network model. Time series prediction, fuzzy clustering, and research progress of fuzzy computing are also introduced in Chap. 4. Chapter 5 presents an overview of evolutionary computing and specifies four evolutionary algorithms proposed before 2000. Chapter 6 gives a traveling salesman problem test set and a continuous optimization problem test set as well as the metrics to evaluate the performance of evolutionary algorithms. Chapter 6 also introduces two swarm intelligence algorithms proposed after 2000 and the research progress of evolutionary computing.

Tianjin, China

Xiu Zhang
Xin Zhang
Wei Wang

Acknowledgments

This book arose from the intelligent information processing course at Tianjin Normal University. The course is for junior students majoring in "artificial intelligence" and "Intelligent Science and Technology". We would like to thank all the students, teachers, and engineers of Tianjin Key Laboratory of Wireless Mobile Communications and Power Transmission. We would also like to thank all the staffs of College of Artificial Intelligence, College of Electronic and Communication Engineering, Science and Technology Department, and Dean's Office. Without the efforts of them, this book could not have been written.

We would like to thank Bingyue Xu, Liuwei Zhang, Jian Dang, and Ke Chen for their efforts on proofreading the text, table, figure, and program of the book.

We would like to thank the staffs at Springer for their help and support in preparing this book.

Tianjin, China

Xiu Zhang
Xin Zhang
Wei Wang

Contents

Chapter 1
Artificial Neural Network

Abstract Artificial neural network is the core of deep learning algorithms and the forefront of artificial intelligence. Its inspiration comes from neurons within the human brain. Artificial neural network mimics the way biological neurons transmit signals to each other. It can thus achieve the goal of learning experiences. This chapter introduces artificial neuron, perceptron and basic model of artificial neural network. Moreover, the chapter also introduces backpropagation neural network, Hopfield neural network, competitive neural network. Finally, deep neural network is introduced in the chapter. Five examples are given to show the working principle of artificial neural network. The programs for implementing the examples are also provided for better understanding the model of artificial neural network.

1.1 Artificial Neuron

Artificial neural network (ANN) is a computational structure proposed by scientists based on neurobiological research to simulate biological processes and reflect certain properties of the human brain [1]. Artificial neural network is also known as neural network (NN). It is the abstraction, simplification and simulation of human brain nervous system.

The nervous system of human brain is composed of neuron as the basic unit. In order to simulate the neural system of human brain, ANN needs to start from simulating the biological neuron of human brain, which is called artificial neuron. In ANN, artificial neurons are called processing units; from the network point of view, artificial neurons are also called nodes.

The comparison of biological and artificial neurons is shown in Table 1.1. The artificial neuron simulates the biological neuron. Although the simulation could not be exactly the same, the main functional steps are the same [2]. First, the input layer of artificial neurons simulates the dendrites receiving signals from the external input. Second, the function of the cell body is simulated by a weighted summation, which means that each component of the received signal is multiplied by a certain weight and the sum is calculated. Thirdly, the activation function simulates axons controlling the weighted sum. When it reaches a certain threshold, it represents the

X. Zhang et al., *Intelligent Information Processing with Matlab*,
https://doi.org/10.1007/978-981-99-6449-9_1

excitation state; otherwise, it is the inhibition state. Finally, the output layer simulates the synapse to transmit the processed results.

Artificial neuron processes data as follows:

(1) Artificial neuron requires a set of inputs, denote inputs as $(x_1, \ldots, x_i, \ldots, x_n)$. The inputs are sent to artificial neuron, as shown in Fig. 1.1. In Fig. 1.1, both the input nodes and the artificial neuron are represented by circles, and the circle of the artificial neuron is filled with gray. It can be seen that artificial neuron is a multi-input structure.

(2) Artificial neuron generally sums all the inputs together using a weighted summation method, as shown in Fig. 1.2. In Fig. 1.2, each input x_i is assigned a weight w_i.

(3) Artificial neuron typically has an input bias, as shown in Fig. 1.3. In Fig. 1.3, the bias is passing an x_0 to the artificial neuron and assigning it the weight w_0, so that the bias is w_0x_0. In general, x_0 is equal to -1. Note that the bias is also called a deviation, and sometimes the bias is denoted by the symbol b, i.e., $b = w_0x_0$. For ease of expression into a matrix, w_0x_0 is used here to denote the bias.

After the above steps, the input obtained by the artificial neuron can be computed by:

Table 1.1 Comparison of biological neuron and artificial neuron

Biological neuron	Artificial neuron	Function
Dendrite	Input layer	Receive external input signal
Cell body	Weighted summation	Filter and process signal
Axon	Activation function	Set threshold to control signal output
Synapse	Output layer	Output results after processing

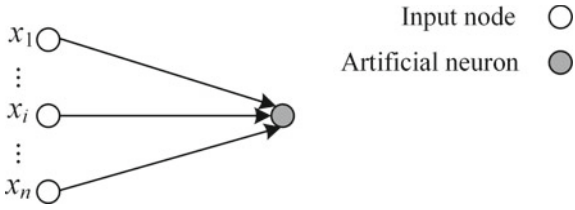

Fig. 1.1 The input of artificial neuron

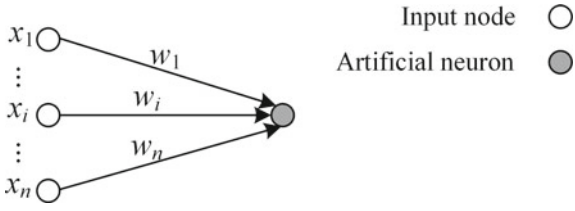

Fig. 1.2 The weighted summation of artificial neuron

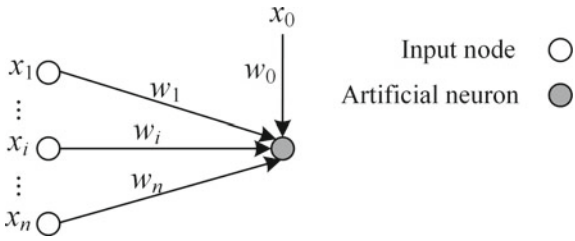

Fig. 1.3 The input bias of artificial neuron

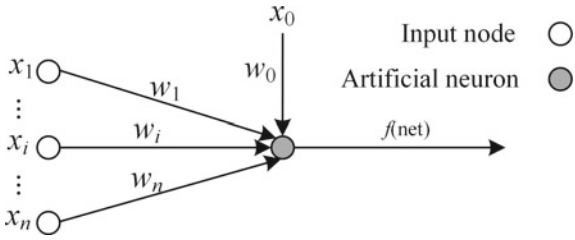

Fig. 1.4 The output of artificial neuron

$$net = \sum_{i=1}^{n} w_i x_i + w_0 x_0 \tag{1.1}$$

Using the notation of vectors, let denote $W = (w_0, w_1, ..., w_i, ..., w_n)^T$, and denote $X = (x_0, x_1, ..., x_i, ..., x_n)^T$, then Eq. (1.1) can be rewritten as:

$$net = W^T X \tag{1.2}$$

(4) The output of the artificial neuron is calculated from the activation function, as shown in Fig. 1.4. In Fig. 1.4, the output function is denoted as f, which is a function of the input *net*, sometimes it is also called activation function, or transfer function.

Let denote the output as y, then:

$$y = f(net) = f\left(W^T X\right) \tag{1.3}$$

It can be seen that the artificial neuron is a multiple-input single-output structure.

The activation functions of artificial neuron generally take the range domain of [0, 1] or [− 1, 1]. The commonly used transfer functions are threshold activation function, nonlinear activation function, piecewise linear activation function, probabilistic activation function, and rectified linear unit (ReLU) activation function. Next, the mathematical models of the transfer functions are introduced, and the independent variable of the transfer functions is always denoted by x for the convenience of presentation, which is different from the input x of artificial neuron.

Threshold-type activation function is expressed as:

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{1.4}$$

It can be seen that the output of the threshold-type activation function is either 1 or 0. In fact, it is a step function. It can model the excitatory and inhibitory states of biological neurons.

Nonlinear activation function is expressed as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.5}$$

The output of the activation function in Eq. (1.5) is a real number between 0 and 1. Sometimes it is also called Sigmoid function or S-type function. It has the advantage of being a monotonically differentiable function in the domain of definition. To obtain a number between -1 and 1, the following activation function can be used:

$$f(x) = \frac{2}{1 + e^{-x}} - 1 = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{1.6}$$

Segmented linear activation functions:

$$f(x) = \begin{cases} 0, & x \leq 0 \\ cx, & 0 < x \leq x_c \\ 1, & x_c < x \end{cases} \tag{1.7}$$

In Eq. (1.7), c is a constant, and x_c is another constant related to c.

Probabilistic activation function:

$$f(x) = \frac{1}{1 + e^{-x/T}} \tag{1.8}$$

In Eq. (1.8), T is a temperature parameter. When the relationship between input and output is uncertain, we need to use a probabilistic activation function to describe the probability that the output state is 1.

Linear rectification activation function:

$$f(x) = \max(0, x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{1.9}$$

The ReLU activation function is generally referred as the ramp function in algebra. Experiments show that the linear rectification activation function maximizes the screening ability of artificial neurons. In addition, the linear rectification activation function converges faster compared to the Sigmoid activation function.

1.2 Overview of Artificial Neural Network

Artificial neural network is composed of several artificial neurons and their connections. Artificial neural network is also called neural network. As shown in Fig. 1.5, the figure shows an artificial neural network composed of two artificial neurons.

As can be seen from Fig. 1.5, the output of the first artificial neuron:

$$y_1 = f\left(\sum_{i=1}^{3} w_{i1}x_i + w_{01}x_{01}\right) \tag{1.10}$$

The output of the second artificial neuron:

$$y_2 = f\left(\sum_{i=1}^{3} w_{i2}x_i + w_{02}x_{02}\right) \tag{1.11}$$

It can be seen from (1.10) and (1.11) that the output formula of each neuron is similar. For convenience of expression, subscripts are often omitted and expressed in vector form. The output of the j-th neuron is as follows:

$$y_j = f\left(W_j^T X\right) \tag{1.12}$$

where X represents the input transmitted to all neurons, W_j represents the weight vector from the input to the j-th artificial neuron, and y_j represents the output of the j-th artificial neuron.

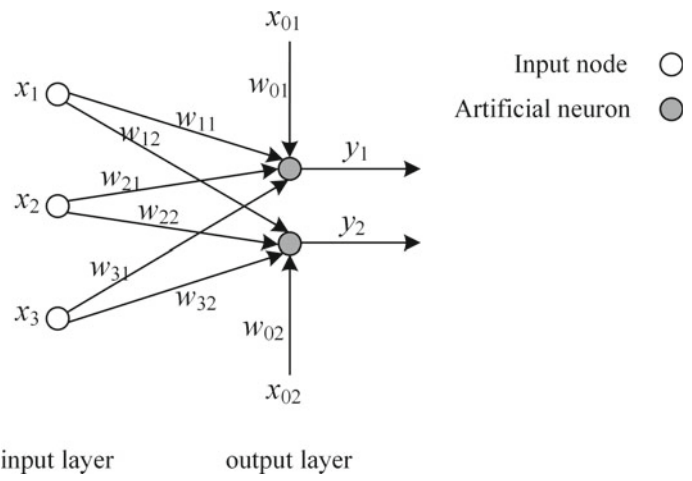

Fig. 1.5 Artificial neural network formed by two artificial neurons

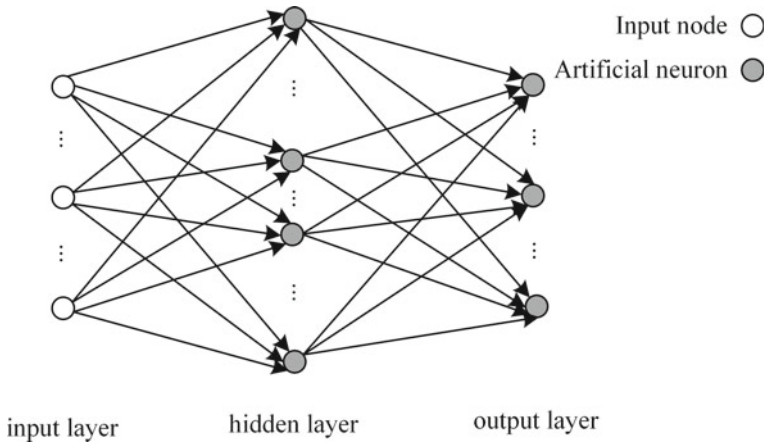

Fig. 1.6 Artificial neural network with hidden layer

Generally, artificial neural network also contains a hidden layer, which is also composed of artificial neurons, as shown in Fig. 1.6.

Figure 1.6 shows an artificial neural network with a hidden layer. From the input layer to the hidden layer, the input data is passed to each hidden layer of neurons. Similarly, from the hidden layer to the output layer, each artificial neuron of hidden layer will also pass information to each artificial neuron in the output layer.

According to the shape of network topology structure, network structure can be classified into hierarchical structure and interconnection structure. The artificial neural network in Fig. 1.6 is a simple hierarchical structure. In addition, the hierarchical structure could have the output layer connected to the input layer structure, and the connections within the hidden layer or the output layer. Interconnection structure includes full interconnection structure, local interconnection structure and sparse interconnection structure.

From the classification of information flow on the network, the network structure can be divided into feedforward neural network and feedback neural network.

For hierarchical neural network, the number of levels can be divided into single-layer neural network, shallow neural network and deep neural network. For example, the neural network in Fig. 1.5 is a single-layer neural network, while that in Fig. 1.6 is a shallow neural network, also known as a common neural network. It is easy to see that the functional of the input layer node is only to transmit the signal to the next layer node without designing other operations, so the number of layers of the neural network does not include the input layer. The deep neural network generally refers to the neural network with more than or equal to 3 layers. For a deep neural network, it contains at least two hidden layers, excluding the input layer and the output layer.

To make an artificial neural network work, the weights of each connection in the network need to be determined, for example, W_j in (1.12). Once the weight is determined, the corresponding output y_j can be calculated. In artificial neural network, the determination of weights is called learning, that is, the learning of artificial

neural network is to determine the weights in the network. Next, we introduce some commonly used learning methods.

Learning method is able to reflect the intelligence characteristics of artificial neural network. Without learning algorithm, artificial neural network will lose the ability of self-organization, self-adaptation and self-learning. The learning process of artificial neural network is its training process. The so-called training is to adjust the connection weights between neurons in a certain way during the input of the sample set composed of sample vectors into the artificial neural network, so that the network can store the connotation of the sample set in the form of connection weight matrix. At last, the network can give appropriate output when data is input.

At present, there are many kinds of neural network learning methods. According to whether there is supervised signal or not, learning methods can be divided into supervised learning, unsupervised learning and reinforcement learning. Note that supervised signals are also known as teacher signals, so supervised learning is known as teacher learning and unsupervised learning is known as teacher-free learning. Reinforcement learning is also known as evaluative learning. In addition, there is no strict unified standard for the classification of learning methods. Some scholars distinguish learning methods from training methods, while some scholars distinguish learning methods from learning modes. We can distinguish the learning methods from the format of the dataset, as shown in Table 1.2. As can be seen from the table, the format of the dataset required for unsupervised learning is the simplest, which is simply to transmit the input signal to the neural network. Supervised learning requires knowing the input of a signal and its corresponding real output. Reinforcement learning requires knowing the input of the signal and the actual output corresponding to some of the input, and in addition, rating the output.

In supervised learning, the output of the network is compared with the desired output, and the weights of the network are adjusted according to the difference between the two, ultimately making the difference smaller. Supervised learning means that there is teacher learning. It is assumed that both teachers and the neural network have to make judgments on training vectors (i.e., examples) extracted from the surrounding environment at the same time, and teachers can provide expected responses for the neural network according to some knowledge they have mastered.

There will be some difference between the output of neural network and the real output, which is generally called error. The error can be expressed by a function, so as to measure the error generated by the whole neural network. This function is also called loss function, sometimes called cost function, objective function, etc. Common loss functions include square loss function, logarithmic loss function, cross entropy

Table 1.2 Learning methods and the format of datasets required

Learning method	Dataset format
Supervised learning	Input and output of neural network
Unsupervised learning	Neural network input
Reinforcement learning	Neural network input, partial output and the associated level

loss function, etc. The value of these loss functions is generally non-negative. The larger the value of the loss function is, the larger the error of the neural network will be, and the worse the ability of the neural network to deal with the problem.

Figure 1.7 is a neural network diagram of supervised learning. The weight parameters of neural network can be adjusted under the comprehensive influence of training vector and error signal. The error signal can be defined as the difference between the predicted output and the real output of the neural network. This adjustment can be carried out gradually and repeatedly, and the ultimate goal is to make the neural network simulate the teacher signal. In this way, the teacher's knowledge of the environment can be transferred to the neural network through training. When meeting certain conditions, the teacher signal can be excluded and the network can cope with the environment completely autonomously.

Figure 1.8 is a neural network diagram of unsupervised learning. In the teacher-free learning mode, after the input mode enters the network, the network automatically adjusts the weight according to pre-set rules (e.g. competition rules), so that the network finally has the function of pattern classification. In the unsupervised learning, there is no teacher monitoring the learning process, that is, the neural network has no examples to learn from.

Unsupervised learning can be divided into two categories: self-organized learning and unsupervised competitive learning.

In the learning process of artificial neural network, the change of weight can be expressed as follows:

$$\Delta W_j = \eta r\big(W_j(t), X(t), d_j(t)\big)X(t) \tag{1.13}$$

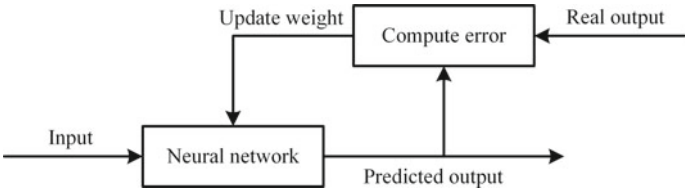

Fig. 1.7 Neural network with supervised learning

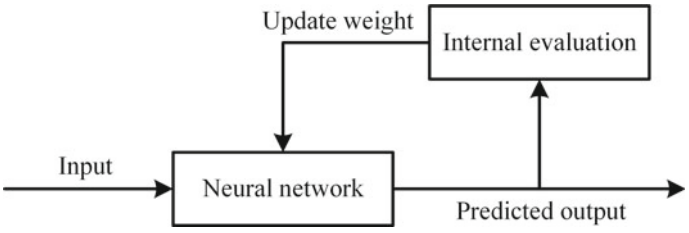

Fig. 1.8 Neural network with unsupervised learning

where, t is the time, also can be understood as the t-th iteration, η is a positive number, known as the learning constant, η determines the learning rate.

For the discrete time adjustment, Eq. (1.13) is:

$$W_j(t+1) = W_j + \eta r\big(W_j(t), X(t), d_j(t)\big)X(t) \tag{1.14}$$

Different learning rules of $r\big(W_j(t), X(t), d_j(t)\big)$ have different definitions, thus forming a variety of neural network. Neural network learning rules are as follows:

(1) Hebb learning rule

When neuron i and neuron j are excited at the same time, the connection strength of the two should increase. When two neurons are connected, it is understood that both the input and the output of the neuron are positive.

Hebb learning rule is a kind of pure feedforward, unsupervised learning, which still plays an important role in various neural network models. The learning signal simply equals to the output of the neuron:

$$r = f\big(W_j^T X\big) \tag{1.15}$$

The weight vector is adjusted by:

$$\Delta W_j = \eta f\big(W_j^T X\big)X \tag{1.16}$$

Hebb learning rule requires pre-set weight saturation values to prevent unconstrained growth of weights when the input and output are always positive and negative. Weight is initialized by assigning a small random number near zero to $W_j(0)$.

(2) Winner-take-all learning rule

Winner-Take-All is a competitive learning rule for unsupervised learning in which a layer of the network is identified as a competitive layer, and for a particular input X, all p neurons in the competitive layer have output responses, where the neuron with the largest response value j^* is the neuron that wins in the competition. The rule is expressed as:

$$W_{j^*}^T X = \max_{i=1,2,\ldots,p}(W_i^T X) \tag{1.17}$$

Only the winning neuron has the right to adjust its weight vector W_j^* by:

$$\Delta W_{j^*} = \eta\big(X - W_{j^*}\big), \eta \in (0, 1] \tag{1.18}$$

Since a larger dot product of two vectors indicates a closer approximation, the adjustment results in making W_j^* further close to the current input X, so that the next time an input pattern similar to X appears, the neuron that won last time is more likely

to win, and thus the weight vector corresponding to each neuron in the competitive layer is gradually adjusted to the clustering center of the input sample space.

Sometimes a winning neighborhood is defined with the winning nerve as the center. In addition to the winning neuron, other neurons in the neighborhood also adjust their weights to varying degrees. Weights are generally initialized to arbitrary values and normalized.

Next, we introduce perceptron, which is a kind of feedforward neural network. Perceptron is a hierarchical neural network which simulates the environment information received by human vision and transmits the information by nerve impulse. Some common feedforward neural networks, such as adaptive linear neural networks, backpropagation neural networks and radial basis function neural networks, belong to perceptron in structure. The structure and function of single-layer perceptron are simple, and the network itself has its inherent limitations, which are overcome by the proposed improved multi-layer perceptron network and the corresponding learning rules.

A single layer perceptron is a forward network with one layer of neurons and a threshold activation function. This forward network has no feedback connections or intra-layer connections and outputs only one node. The single-layer perceptron network model is shown in Fig. 1.4.

In the single-layer perceptron, the net input can be obtained as:

$$net = \sum_{i=0}^{n} w_i x_i \tag{1.19}$$

In a single-layer perceptron, the predicted output is:

$$o_j = sgn\left(\text{net}_j - T_j\right) = sgn\left(\sum_{i=0}^{n} w_i x_i\right) = W^T X \tag{1.20}$$

A single-layer perceptron can do classification. The classification principle is to store the classification knowledge in the weight vector (including threshold) of the perceptron. The classification decision determined by the weight vector divides the input modes into two categories, so as to realize the purpose of classifying the input vector.

Example 1.1 Suppose the input data has 4 sample points, namely (0, 0), (0, 1), (1, 0) and (1, 1). The corresponding real output of the four sample points is 0, 1, 1, 1. Since the output values are only 0 and 1, you can see that this is a binary classification problem. This problem is simulated in Matlab.

The specific programs are as follows:

```
x = [0 0 1 1; 0 1 0 1];
t = [0 1 1 1];
graduate School
net = perceptron('hardlim', 'learnp');
```

```
net = configure(net,x,t);
net.iw{1,1} = [-1.5 -0.5];
net.b{1} = 1;
figure(1);
plotpv(x, t);
hold on; box on; grid on;
plotpc(net.iw{1,1},net.b{1}).
xlabel('×1'); ylabel('×2'); title('').
hold off;
net = train(net,x,t);
view(net)
y = net(x);
figure(2);
plotpv(x, t);
hold on; box on; grid on;
plotpc(net.iw{1,1}, net.b{1})
xlabel('×1'); ylabel('×2'); title('').
hold off;
```

In this example, not only a single-layer perceptron is trained, but also the classification renderings of the untrained perceptron and the trained perceptron are drawn, as shown in Figs. 1.9 and 1.10. In these two figures, the circle and the plus sign respectively represent two types of sample points, and the straight line in the graph is the dividing line determined by the single-layer perceptron.

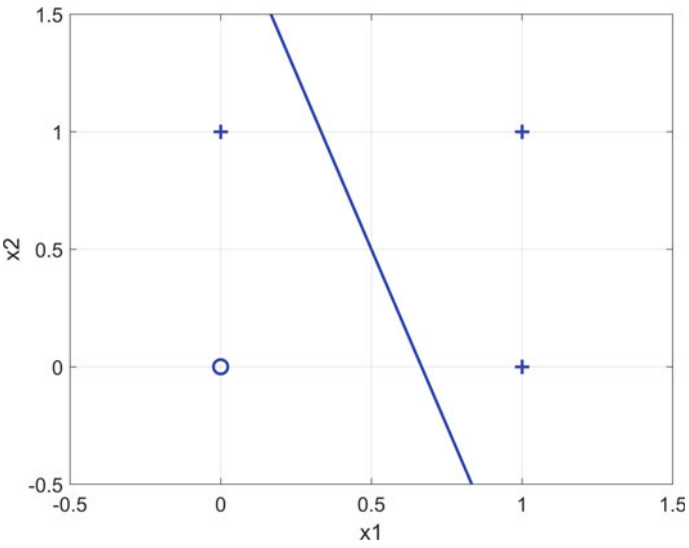

Fig. 1.9 Classification results of untrained single-layer perceptron

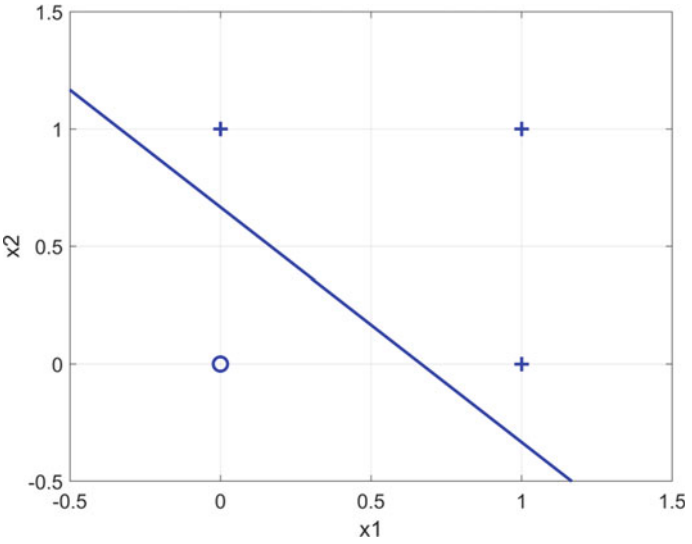

Fig. 1.10 Classification results of trained single-layer perceptron

As can be seen from Fig. 1.9, if the initial weight is arbitrarily set, the single-layer perceptron cannot correctly classify the input sample. As can be seen from Fig. 1.10, the single-layer perceptron can correctly classify input samples after training.

The model of the multi-layer perceptron network (MLP) is shown in Fig. 1.11. Besides the output layer, the multi-layer perceptron also has a mid-layer, called the hidden layer.

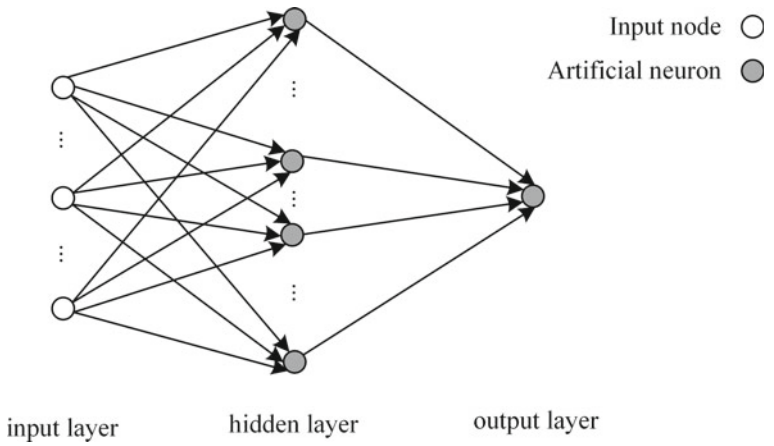

Fig. 1.11 Multi-layer perceptron network model

In the MLP network, the relationship between input and hidden layer processing unit is the same as that in single-layer perceptron network. Each hidden layer processing unit determines the position of a decision line through weight adjustment, and then obtains convex domain through weighted sum to completely separate the two types of data.

Neural network is to learn the input data, constantly adjust the weight value, so that the loss between the output of neural network and the real output is less and less, this process is called neural network training, or training a neural network. For the trained neural network, the new input data is transmitted to the neural network, so as to calculate the output of the neural network, this process is called prediction, or the prediction of the neural network. When a neural network is trained with little error on the trained input data and great error on the new data, this situation is called overfitting of the neural network. If a neural network has a large error in the trained input data after training, it will generally have a large error in the prediction, which is called neural network underfitting.

Both underfitting and overfitting are unsatisfactory, so training a neural network model needs to consider both cases and make a good trade-off. When a neural network predicts new data, that is, it judges the output brought by the new input, which is called generalization of neural network. The new data is also called the test data. The generalization ability of a neural network is what we are most concerned about. We always hope that the obtained neural network model has good generalization ability. Researchers often use regularization method to enhance the generalization ability of neural networks.

In training, the purpose of neural network is to minimize the error, and in generalization, the purpose is to minimize the generalization error. This problem with the smallest error is the problem of finding the optimal weight, which is called the optimization problem in mathematics. From this point of view, the learning of neural network is to solve the optimization problem, so the optimization problem research is also very helpful. The commonly used optimization methods include gradient descent, random gradient descent, batch gradient descent, etc. Sometimes these methods are called traditional optimization methods. At the other end of the spectrum is evolutionary computing, or intelligent optimization. The evolutionary computing approach is described in a later chapter.

1.3 Backpropagation Neural Network

Backpropagation (BP) neural network, also known as BP neural network, belongs to feedforward neural network like perceptron. Different from perceptron, BP neural network adopts weight learning algorithm of backpropagation. The error backpropagation learning algorithm of BP neural network was proposed by Rumelhart and McClelland in 1985 [1–3]. When input data in the input layer, BP neural network transmits information to the output layer through the hidden layer. The results of the output layer were compared with the true results, then the error could be computed.

The transmission direction become from the output layer to the input layer. Each layer could modify the weight, and finally learn a group of good weight.

Theoretically, when BP neural network adopts nonlinear differentiable activation function, such as Sigmoid function, it can approximate any nonlinear function, so BP neural network can effectively solve a variety of nonlinear problems.

The flow chart of backpropagation neural network is shown in Fig. 1.12. The first step is the preparation of the dataset. For problems with supervised learning, the dataset needs to include input X and real output y. The output is sometimes called the label. The second step is the initialization of the neural network. The weights in the neural network are randomly initialized. The parameters of neural network training, such as learning rate in learning rules, also need to be initialized. In neural network, other parameters besides weights are sometimes called hyperparameters, which have certain influence on the learning ability and generalization ability of neural network. Hyperparameters are usually set by experience, and some scholars have studied the setting of hyperparameters. The third step is to calculate the output and error of the neural network. According to the weight in the neural network, when the data is transferred from the input layer to the output layer, the output can be calculated. The output of the neural network is compared with the real output, and the error can be calculated. The fourth step is error backpropagation. After the error is calculated, the weight value is updated from the output layer forward layer by layer until the input layer ends. The fifth step is to judge whether the learning of the neural network terminates or not. The termination condition can be a pre-set number of iterations, or the loss function can reach a small value. If there is no termination, go back to step 3 and continue the iteration; If terminated, the learned neural network model is output.

As mentioned above, the learning of neural network is a process of adjusting weights to reduce the error between the predicted output and the real output of the network. The error between the network predicted output and the real output is generally measured by loss function. The common loss functions in BP neural network are square loss function and cross entropy loss function. The expression of the square loss function is:

$$L(y) = (o - y)^2 \tag{1.21}$$

where $L(y)$ represents the loss function, o represents the real output, and y represents the network predicted output.

It can be seen from the formula that the square loss function is the square of the error between the network predicted output and the real output. If they are the same, the error is 0; If they are different, the error is greater than 0, and the loss function is proportional to the error. It should be pointed out that the squared loss function is an error measure often used in early neural network research. The learning rules of backpropagation are also derived from the square loss function.

The expression of the cross-entropy loss function is:

$$L(y) = -oln(y) - (1 - o)ln(1 - y) \tag{1.22}$$

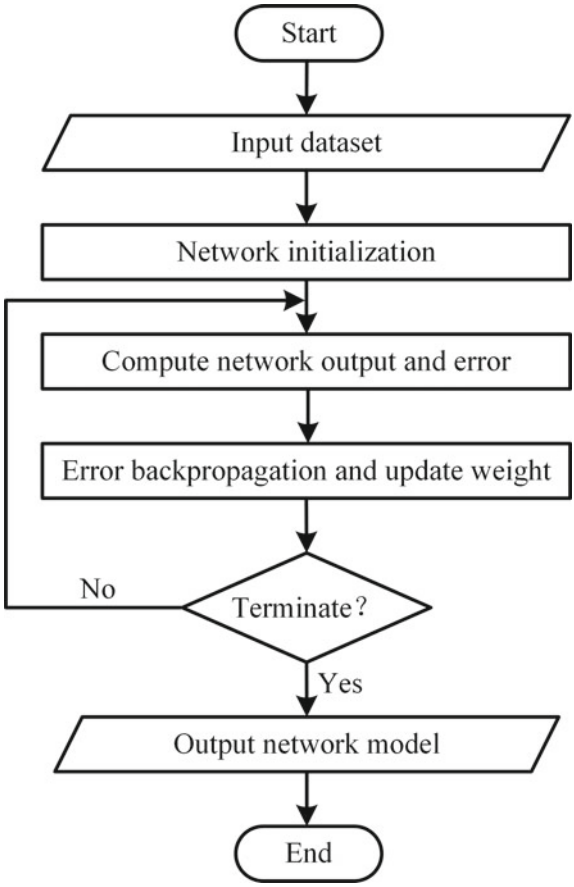

Fig. 1.12 Training process of backpropagation neural network

where $ln(y)$ denotes the natural logarithm function. For the binary classification problem, the value of the real output is either 0 or 1. When $o = 1$, the loss function in the above equation is equivalent to $-ln(y)$, at which point the error increases and $-ln(y)$ tends to positive infinity as y tends to 0, while the error decreases and $-ln(y)$ tends to 0 as y tends to 1. When $o = 0$, the loss function in the above equation is equivalent to $-ln(1 - y)$, when y tends to 0, the error decreases and $-ln(1 - y)$ tends to 0; while when y tends to 1, the error increases and $-ln(y)$ tends to positive infinity. This also shows that the cross-entropy loss function is also proportional to the error, and the cross-entropy loss function is more sensitive to the error than the square loss function, i.e., the cross-entropy loss function increases faster than the squared loss function when the error increases.

Through the simulation analysis of the researchers, it is found that the cross-entropy loss function is better for the classification of supervised learning. In other words, when training the back-propagation neural network, if you choose between the square loss function and the cross-entropy loss function, the cross-entropy loss

function is recommended. For the regression problem in supervised learning, the square loss function learning rule is recommended.

In addition, the purpose of learning rules of neural networks is to reduce the error between network output and real output, that is, to minimize the error, which involves the optimization theory in operations research. For example, for the square loss function, we can get the optimization problem:

$$\begin{aligned} &\min\ L(y) = (o - y)^2 \\ &s.t.\quad y = f\left(W^T X\right) \end{aligned} \tag{1.23}$$

where y is calculated from the weight W of the neural network and the samples X of the training set, while X is known and the weights W are unknown and need to be learned to be determined. This means that the independent variable (1.23) is actually the weight W, which is an optimization model. As mentioned before, machine learning has the problem of overfitting, and the regularization method can solve this problem to some extent. The regularization method is to add the weights to the loss function, whose expression is:

$$\begin{aligned} &\min\ L(y) = (o - y)^2 + \tfrac{1}{2}\lambda \| W \|^2 \\ &s.t.\ \ y = f\left(W^T X\right) \end{aligned} \tag{1.24}$$

One of them is λ a parameter to regulate the proportion of errors and weights in the loss function, and the 1/2 in the expression is introduced for the simplicity of the expression after the derivative is found.

By comparing model (1.23) and model (1.24), it can be seen that the loss function is small enough only when the error and weight are reduced in model (1.24) with regular term added. Conversely, if the error and weight do not decrease, the loss function cannot approach 0. We will not expand on the optimization model theory and its solution, interested readers can refer to the relevant books.

In the backpropagation method, we know that the error of the output layer is reversely transmitted to the hidden layer. The error of the hidden layer is calculated by reverse transmission of the error of the output layer. In the calculation, the same connection weight is used to deal with it. In turn, the error can be transmitted to the hidden layer after the input layer, which realizes the weight adjustment of the whole neural network. The backpropagation method is very important, and it is also commonly used in deep learning.

With a trained neural network model, new input data can be predicted. Usually, a certain amount of data is required for training, that is, the data set contains many samples, so the training of neural network generally takes some time. In the prediction, it only needs to be passed from the input layer to the output layer, and all kinds of weights are determined. Compared with the training, the prediction takes a very short time. Therefore, the neural network model has a good application prospect.

Example 1.2 The dataset for this example comes from the UCI machine learning database and is about monitoring Coronavirus disease (COVID-19). This problem

belongs to supervised learning. The data set includes 14 samples, each of which has 7 attributes, and the data set has 3 kinds of labels. Matlab is used to establish the BP neural network program. After training, all samples are predicted and the prediction accuracy is output.

This problem is simulated in Matlab, and the programs are as follows:

```
P = [ 1 1 1 1 1 -1 -1
      1 1 -1 1 1 -1 -1
      1 1 1 1 -1 1 -1
      1 1 -1 1 -1 1 -1
      1 -1 -1 -1 -1 -1 1
      1 1 1 -1 -1 -1 1
      1 1 -1 -1 -1 -1 1
      1 1 1 1 -1 -1 -1
      1 -1 -1 1 1 -1 -1
      -1 1 -1 1 1 -1 -1
      1 -1 -1 1 -1 1 -1
      -1 1 -1 1 -1 1 -1
      -1 1 -1 -1 -1 -1 1
      -1 -1 -1 -1 -1 -1 1]';
T = [1 1 1 1 1 1 1 1 2 2 2 2 2 3];
hiddenLayerSize = [10, 10];
net = feedforwardnet(hiddenLayerSize);
net.numLayers.
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'logsig';
net.trainFcn = 'traingd';
net.trainParam.goal = 0.01;
net.trainParam.lr = 0.1;
net.trainParam.showWindow = false;
[net, tr] = train(net, P, T);
o = sim(net, P);
o = round(o);
[T; o]
figure1 = figure(1);
axes1 = axes('Parent',figure1);
hold(axes1,'on');
box(axes1,'on');
grid(axes1,'on');
plot(T, 'd', 'MarkerSize',10,'LineWidth',2,'LineStyle','none');
plot(o, '*', 'MarkerSize',10,'LineWidth',2,'LineStyle','none');
hold(axes1,'off');
set(axes1,'FontSize',14);
print('Fig', '-dpng', '-r600')
```

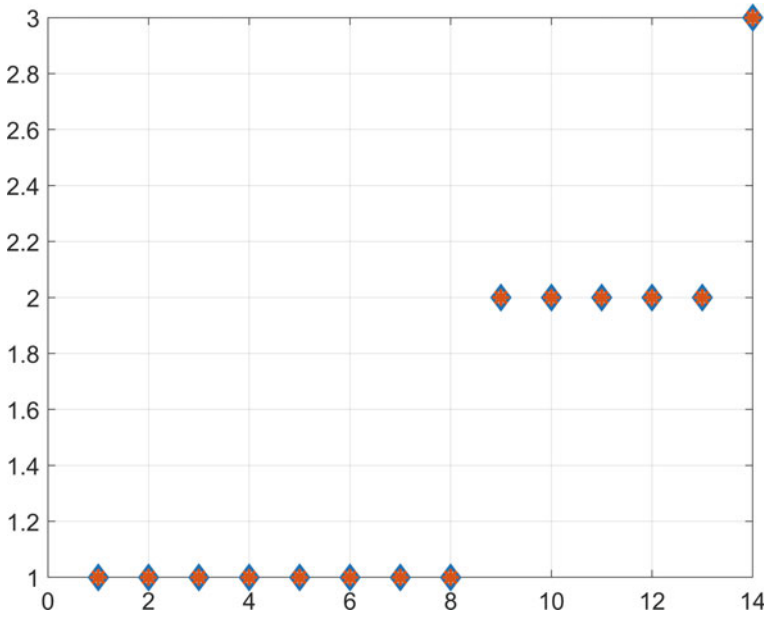

Fig. 1.13 Problem of COVID-19 prediction solved by BP neural network

The running result of this example is shown in Fig. 1.13, where the diamond is the real output and the asterisk is the output predicted by the BP neural network. As can be seen from the figure, for 14 samples in the data set, the trained BP neural network can correctly predict the output results. Due to the use of random initialization weights, so each independent run of the program may not be able to obtain the same results, sometimes the prediction becomes inaccurate.

As can be seen from the above example problems, the number of input layer and output layer nodes of BP neural network depends on the training samples. That is to say, after the training sample of a certain problem is obtained, the number of input layer nodes and output layer nodes of BP neural network can be determined.

BP neural network is also a feedforward neural network. Generally speaking, the structural design of multilayer feedforward neural network needs to solve two problems. One is how many hidden layers should be designed, and the other is how many nodes should be designed for each hidden layer. For these two problems, there is no universal theoretical guidance, but researchers have accumulated a lot of experience through a lot of practice.

(1) Design of the number of hidden layers.
A feedforward neural network with a single hidden layer can approximate any continuous function. Two hidden layers are needed only when learning discontinuous functions. The experience of network design is to give priority to designing a hidden layer. When a hidden layer has a large number of hidden nodes, but cannot improve the network performance, you can consider adding

another hidden layer. Another rule of thumb is that when two hiding layers are used, designing more hidden nodes in the first hiding layer and fewer hidden nodes in the second hiding layer is beneficial to improve network performance.

(2) Design of the number of hidden nodes.
Trial-and-error method is a common method to determine the optimum number of hidden nodes. In the design of the network, a small number of hidden nodes can be set first, and then gradually increase the number of hidden nodes. The same sample set is used for training, from which the number of hidden nodes corresponding to the minimum error can be determined. In the trial-and-error method, some empirical formulas can be used to determine the number of hidden nodes. The number of hidden nodes calculated by these formulas is only a rough estimate, which can be used as the initial value of trial-and-error method. Another method of trial-and-error method is to set more hidden nodes at first. In the training process, the weight with little influence will gradually decline to zero, so the corresponding nodes can be removed, and the remaining is the optimal number of hidden nodes.

1.4 Hopfield Neural Network

Hopfield neural network is a feedback neural network proposed by Hopfield and Tank in 1985. The design of the network follows the principles of physics, and each artificial neuron is composed of an operational amplifier and a capacitor resistor element [1–4]. The input signal is added to each artificial neuron in the form of voltage, and each neuron is connected with each other. After receiving the voltage signal, after a period of time, the current and voltage of each part of the network reach a certain stable state. At this time, the output voltage of the network is the answer to the problem. From the point of view of system, Hopfield neural network is a kind of static nonlinear mapping, which combines simple nonlinear mapping to achieve complex nonlinear processing capability.

Hopfield neural network can be divided into discrete and continuous types according to the input sample processing or activation function. The discrete Hopfield neural network is suitable for the case where the input sample is binary logic; while the continuous Hopfield neural network is suitable for the case where the input sample is analog. Discrete Hopfield neural network activation function is a δ type function (e.g., symbol function), commonly used in associative memory problems; The activation function of continuous Hopfield neural network is S type function (e.g., Sigmoid function), which is generally used for optimization problems.

Discrete Hopfield neural network (DHNN) is a kind of feedback type neural network. As shown in Fig. 1.14, it is a single-layer neural network. Its feature is that the output of each artificial neuron is fed back to all neurons except its own through connection weight, so as to realize that the output of each neuron can be controlled by the output of all neurons except its own, and the output of each neuron can restrict each other.

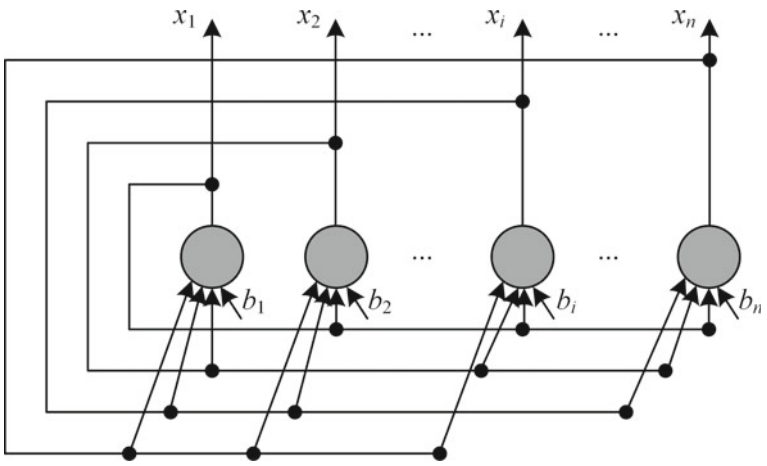

Fig. 1.14 Topology of discrete Hopfield neural network

The input of DHNN is the initial state value of the network, denoted as $X(0) = [x_1(0), x_2(0), \ldots, x_n(0)]^{\mathrm{T}}$. The output of DHNN is the output value of all neuron states, denoted by $X = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$. The connection weight of artificial neuron x_i to x_j is denoted as w_{ij}, that is, the output of x_i is fed back to neuron x_j as input. Each neuron has a threshold b_j.

Under the excitation of the outside world, DHNN enters the dynamic evolution process from the initial state, and the state of each neuron is constantly changing. DHNN usually uses the sign function as the activation function, and the net input of artificial neuron x_j is:

$$net_j = \sum_{i=0}^{n} \left(w_{ij} x_i - b_j\right),\ j = 1, 2, \ldots, n \tag{1.25}$$

The output of artificial neuron x_j is:

$$f\left(net_j\right) = sgn\left(net_j\right) = \begin{cases} 1, & net_j \geq 0, \\ -1, & net_j < 0, \end{cases} \quad j = 1, 2, \ldots, n \tag{1.26}$$

In general, DHNN has $w_{ii} = 0$ and $w_{ij} = w_{ii}$. When DHNN reaches stability, the state of each neuron no longer changes, and the steady state at this time is the output of DHNN. If the network output of DHNN at moment t is denoted as $X(t)$, the output of DHNN in the steady state is $\lim_{t \to \infty} X(t)$.

DHNN works in two ways: asynchronous mode and synchronous mode. The asynchronous mode is a serial mode in which only one neuron at a time adjusts its state according to (1.26) while the DHNN is running, and the state of other neurons remains the same. When adjusting the state of neurons, they can be adjusted in some

prescribed order, or they can be randomly selected for adjustment. Synchronous mode is a parallel mode in which all neurons adjust their state simultaneously while the DHNN is running.

DHNN can store a number of predetermined stable states, that is, the value of the input. When it runs, an $X(0)$ is applied to the network, and the network will feed back the output as the input next time. After several iterations, under certain preconditions, DHNN will finally stabilize at the pre-set stable point. $X(0)$ is known as the initial activation vector of DHNN, which only plays a driving role in the initial scope network. In the following loop iteration, the whole network is in a self-excited state, and $X(0)$ is replaced by the feedback vector as the next input.

DHNN can be regarded as a discrete nonlinear dynamic system, which may have stable state, finite ring state and chaotic state. First, DHNN can be regarded as a discrete nonlinear dynamic system. As mentioned above, it starts with the initial state $X(0)$, and if it can recurse a finite number of times, and its state does not change, so that $X(t+1) = X(t)$, then the network is said to be stable, or the network has a stable state. If DHNN is stable, it can converge from any initial state to a stable state. Secondly, if DHNN is unstable, since the state of each node in the network is binary, that is, there are only 1 and -1 cases, it is impossible for the network to have infinite divergence, but can only be a self-sustained oscillation between 1 and -1, then the network becomes a finite ring network, or the network has a finite ring state. Finally, if the state of a network changes within some definite range, but its state neither repeats nor stops, that is, its state changes infinitely many, and its motion trajectory does not diverge to infinity, then this phenomenon is called chaos. For DHNN, the state of each node is binary, so all possibilities of its network state are limited, so there will be no chaotic phenomenon. In other words, DHNN does not have a chaotic state.

If DHNN has a stable state, then it can realize associative memory function. When the topology structure and weight matrix of the network are given, the Hopfield neural network can store several pre-set stable states. Which stable state the network reaches after running is related to the initial state. If the stable state of the network is used to represent the memory pattern, the process of the initial state converging to the stable state can be regarded as the process of the network searching for the memory pattern. The initial state has part of the information of the memory pattern, and the subsequent evolution of the network is to recall all the information process from part of the information, thus realizing the associative memory function.

The concept of attractor and energy function is introduced next. If X is the state when a network reaches stability, X is called the attractor of the network, also known as the equilibrium point. If the attractor is regarded as the solution of an optimization problem, then the evolution process from the initial state to the attractor is the computational process of finding the optimal solution.

Definition 1.1 If the state X of a network satisfies $X = f(W^T X - b)$, then X is said to be an attractor of the network.

Assuming that X is an attractor of DHNN, the set of all initial states that make DHNN reach X is called the attractor domain of X.

Definition 1.2 If X^a is an attractor of DHNN and DHNN is asynchronous, X is weakly attracted to X^a if there is an adjustment order so that DHNN can evolve from state X to X^a. If DHNN can evolve from state X to X^a for any adjustment order, X is said to be strongly attracted to X^a.

Definition 1.3 If there are some X, which are weakly attracted to X^a, the set of X is said to be the weakly attracted domain of X^a; If there are some X that are strongly attracted to X^a, then the set of X is called the strongly attracted domain of X^a.

A network can always evolve into an attractor starting from the state in the attractor domain. Therefore, when designing the network, it is necessary to make the network have as large an attractor domain as possible so as to enhance the associative memory function.

Theorem 1.1 *If DHNN adjusts the state of the network in an asynchronous manner, and its weight matrix W is a symmetric matrix, then for any initial state, DHNN eventually converges to an attractor.*

Theorem 1.2 *If DHNN adjusts the state of the network in a synchronous manner, and its weight matrix W is a non-negative definite symmetric matrix, then for any initial state, DHNN eventually converges to an attractor.*

Theorems 1.1 and 1.2 point out that no matter which way to adjust the state of the network, as long as certain conditions are satisfied, DHNN can converge to an attractor, that is, DHNN is stable.

If the DHNN network is stable, and steady state is a generalized concept, how do you quantify steady state? The energy function is the solution to this problem. For a system, the more stable it is, the less energy it has, the smaller the value of its energy function. The minimum value of the energy function corresponds to the stable state of the system, so the energy function transforms the problem of finding the attractor into the problem of finding the minimum value of the function. Generally speaking, the energy function of a network is defined as follows:

$$E(t) = -\frac{1}{2}X(t)^T W X(t) + X(t)^T b \tag{1.27}$$

where W is the weight matrix, b is the threshold vector, and E is the energy function.

In addition, the stability of the network is closely related to the energy function, which can be used to optimize the solution function. When the state of the network changes, the energy function of the network automatically tends to the minimum point of energy. If an objective function is expressed in the form of the network energy function, when the energy function tends to the minimum, the corresponding network state is the optimal solution of the problem. The initial state of the network can express the initial solution of the problem. The convergence process of the network from the initial state to the stable state is the process of optimization calculation. This kind of optimization search is automatically completed in the evolution of the network.

Continuous Hopfield Neural Network (CHNN) is proposed on the basis of DHNN, both the principle of which is similar. The input of CHNN is analog, that is, continuous, and each neuron runs in parallel. Therefore, CHNN is closer to biological neural network than DHNN. CHNN is generally used to solve optimization problems, and will not be introduced here.

Example 1.3 The dataset of this example is a manually generated problem, which contains two sample points, namely [1, − 1] and [− 1, 1]. The problem assumes that DHNN contains two neurons and has two attractors. Use Matlab to write DHNN program. After simulation, output DHNN associative memory results.

This problem is simulated in Matlab, and the programs are as follows:

```
T = [1, -1; -1, 1];
figure(1); hold on;
plot(T(1,:), T(2,:), 'ro', 'MarkerSize',10,'LineWidth',2);
axis([-1.1 1.1 -1.1 1.1]);
xlabel('×1');
ylabel('×2');
net = newhop(T);
[Y,Pf,Af] = sim(net,2,[],T);
color = 'rgbmy';
for i = 1:10
    a = {rands(2,1)};
    [y,Pf,Af] = sim(net,{1 20},{},a);
    record = [cell2mat(a) cell2mat(y)];
    start = cell2mat(a);
    plot(start(1,1),start(2,1),'k*',record(1,:),record(2,:),color(rem(i,5) + 1), ...
                    'MarkerSize',10,'LineWidth',2)
end
grid on; box on; hold off;
```

The simulation results of the above problems are shown in Fig. 1.15, where the attractor is represented by a circle symbol, and DHNN is run repeatedly for 10 times. The initial state of each time is represented by an asterisk, and the curve from asterisk to circle represents the trajectory of DHNN iteration.

As can be seen from Fig. 1.15, if the initial state of DHNN is near the upper left, it converges to the upper left attractor. If DHNN starts near the bottom right, it converges to the bottom right attractor. This is the associative memory function of DHNN.

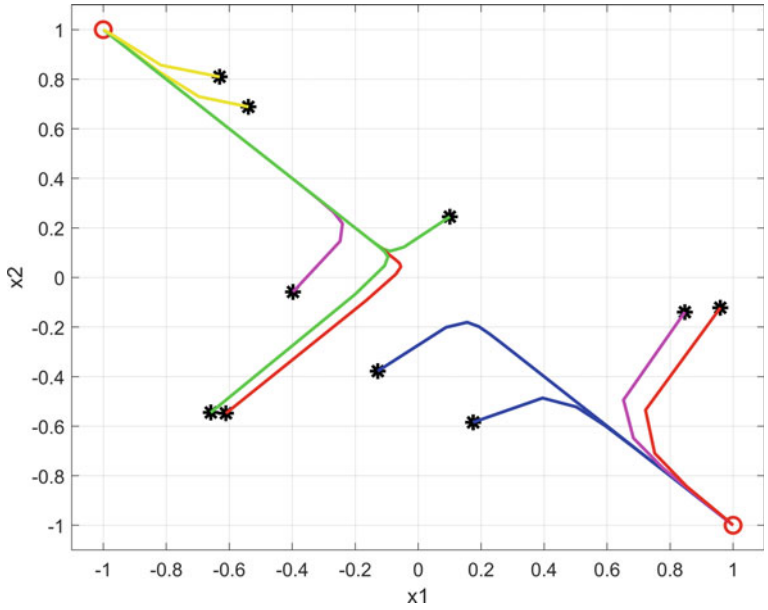

Fig. 1.15 Results of the DHNN

1.5 Competitive Neural Network

The human brain can "learn by itself", that is, through repeated observation, analysis and comparison of objective things, reveal the internal laws of things and correctly classify things with common characteristics. Researchers have used neural networks to achieve this trait, such as the competitive learning neural network described in this section. In addition, this kind of "untaught" way is a kind of learning way without a teacher. The learning way without a teacher is also called self-organized learning, so the "untaught" neural network is also called self-organized neural network. Self-organizing neural network mimics the learning patterns of biological neural network in the human brain. It is characterized by self-organizing and adaptively changing network parameters and structures by automatically searching for intrinsic laws and intrinsic attributes in samples.

Self-organizing neural network is a hierarchical network structure. Different from perceptron and BP neural network, self-organizing neural network has competition layer. As shown in Fig. 1.16, the simplest self-organizing neural network consists of an input layer and a competition layer. The dashed lines in the figure indicate that neurons compete with each other. The input layer accepts the external signal and passes the input pattern to the competition layer. The competition layer analyzes and compares the transmitted patterns to find out the rules in order to correctly classify them. The self-organizing function of self-organizing neural network is realized by competitive learning.

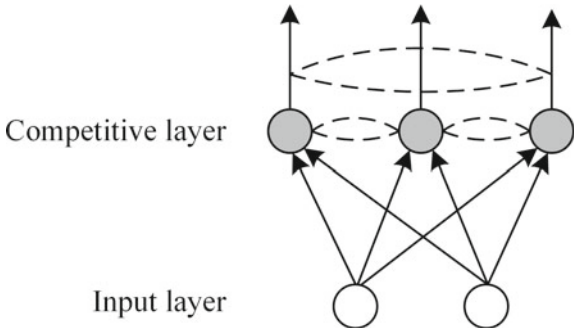

Fig. 1.16 The simplest self-organizing neural network

Competitive learning exists in the human body. There is a lateral inhibition in the retina, spinal cord and hippocampus of the human eye. This is a phenomenon in which the excitation of one nerve cell has an inhibitory effect on the surrounding nerve cells. This lateral inhibition allows the nerve cells to compete with each other. In the initial state, more than one cell may be excited at the same time, but the most excited nerve cell also has the most inhibitory effect on the peripheral nerve cells. As a result, the peripheral nerve cells are less excited, so that this nerve cell is the "winner" of the competition, while other nerve cells lose the competition.

The strongest inhibitory effect is the "only me" effect of the competition winner, which does not allow other nerve cells to excite, this is known as winner-take-all. In competitive learning strategies, winner-take-all is a typical learning rule. The following details the winner-take-all competitive learning rules.

Step (1) Vector normalization. The current input pattern vector X and the internal star vector W ($j = 1, 2,,\ldots, m$) are normalized. After normalization, we obtain $\widehat{X}$ and $\widehat{W}$ ($j = 1, 2, \ldots, m$).

Step (2) Finding the winning neuron. When the network obtains an input pattern vector $\widehat{X}$, the inner star weight vector $\widehat{W}_j$ were compared with $\widehat{X}$. The inner star weight vector most similar to $\widehat{X}$ is judged as the winning neuron, and its weight vector is denoted as $\widehat{W}_{j^*}$. The similarity can be measured by the Euclidean distance between $\widehat{W}_j$ and $\widehat{X}$, or the cosine of the angle between these two vectors. Euclidean distance of these two vectors is:

$$\|\widehat{X} - \widehat{W}_{j^*}\| = \min_{j\in\{1,2,\ldots,m\}}\left\{\|\widehat{X} - \widehat{W}_j\|\right\} \tag{1.28}$$

By expanding the distance of the above Eq. (1.24) and using the property of unit vector, it can be simplified as:

$$\|\widehat{X} - \widehat{W}_{j^*}\| = \sqrt{\left(\widehat{X} - \widehat{W}_{j^*}\right)^T\left(\widehat{X} - \widehat{W}_{j^*}\right)} \tag{1.29}$$

$$= \sqrt{\widehat{X}^T\widehat{X} - 2\widehat{W}_{j^*}^T\widehat{X} + \widehat{W}_{j^*}^T\widehat{W}_{j^*}} = \sqrt{2\left(1 - \widehat{W}_{j^*}^T\widehat{X}\right)} \tag{1.30}$$

As can be seen from (1.29) and (1.30), if the Euclidean distance of two vectors is minimized, it is only necessary to maximize the dot product of the two vectors:

$$\widehat{W}_{j^*}^T\widehat{X} = \max_{j\in\{1,2,\ldots,m\}}\left(\widehat{W}_j^T\widehat{X}\right) \tag{1.31}$$

Note that the dot product of the weight vector and the input vector is exactly the net input of the competing layer neurons. In other words, the winning neuron is the one with the highest net input.

Step (3) Output and weight adjustment. In this learning rule, the output of the winning neuron is 1, and the output of the remaining neurons is 0, as follows:

$$o_j(t+1) = \begin{cases} 1, & j = j^* \\ 0, & j \neq j^* \end{cases} \tag{1.32}$$

It can be seen that only the winning neuron can adjust its weight vector, and the adjusted weight vector is:

$$W_{j^*}(t+1) = \widehat{W}_{j^*}(t) + \Delta W_{j^*} = \widehat{W}_{j^*}(t) + \mu(t)\left(\widehat{X} - \widehat{W}_{j^*}\right) \tag{1.33}$$

For the unwinning neurons, their weight values are not adjusted, which is equivalent to the "victor" neuron j^* applying lateral inhibition to them, not allowing them to excite.

The new vector obtained after the adjustment is not necessarily a unit vector, so it is necessary to re-normalize the adjusted vector. In other words, after Step (3) output and weight adjustment is completed, it is necessary to return to Step (1) vector normalization to continue training until learning rate $\mu(t)$ attenuates to 0.

Next, we introduce the principle of competitive learning. As shown in Fig. 1.17, assuming that the input pattern of a problem is a two-dimensional vector, the normalized input pattern can be regarded as points distributed on the unit circle, represented by "O". It is assumed that the competitive learning neural network has three neurons, and the corresponding three inner star vectors are also distributed on the unit circle after normalization, which is represented by the gray square. From the observation of Fig. 1.17, we can see that the input pattern points can be clustered into three clusters, that is, they can be divided into three categories. In the initial state, the inner star vectors of the neurons in the competition layer are randomly distributed, so how does the competitive learning neural network realize the classification of input patterns?

Before the competitive learning neural network starts training, the inner star vectors of the neurons in the competition layer should be randomly initialized, as shown in Fig. 1.18. We represent the current input pattern to the neural network (the current sample) as a solid circle. According to the above calculation steps, the

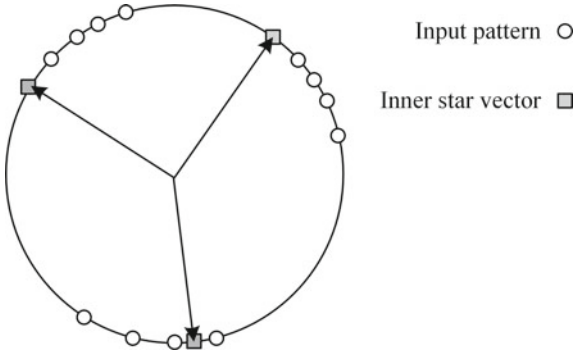

Fig. 1.17 An example of competitive learning

distance between the inner star vector and the current sample needs to be calculated, and the inner star vector closest to the current sample is the winning neuron.

Then, as shown in Fig. 1.19, the winning neuron can adjust its weight, and after adjusting its weight, the winning neuron is further closer to the current input pattern. After weight adjustment, the position of winning neuron moves further to the current sample and its cluster. The next time an input pattern similar to the current sample appears in the same cluster, the neuron that won the last time is more likely to win. After sufficient training in this way, the three inner star vectors on the unit circle will gradually move into the cluster center of each input pattern, so that the weight vector of each neuron in the competition layer becomes a clustering center of the input pattern. After the input pattern training, when a pattern is input into the neural network, the output of the winning neuron in the competition layer is 1, and the winning neuron represents the category of the input pattern.

Example 1.4 The data set of this example comes from UCI machine learning database, which is about the classification problem of iris. This problem belongs to supervised learning. The data set includes 150 samples, each of which has 4

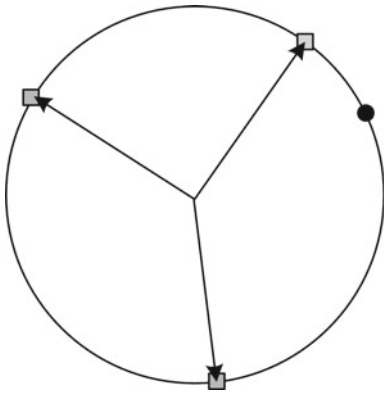

Fig. 1.18 Competitive learning example: initial state

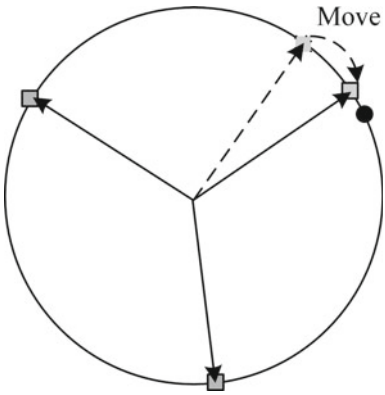

Fig. 1.19 Competitive learning example: weight adjustment

attributes. All samples are divided into 3 categories, with 50 samples in each category. The three types of iris are Sentosa, Versicolour and Virginica. The four attributes of this dataset are sepal length, sepal width, petal length and petal width. Matlab is used to write the program of competitive learning neural network. After training, all samples are predicted and the type of each sample is output.

This problem is simulated in Matlab, and the programs are as follows:

```
[inputs, outputs] = iris_dataset;
outputs = vec2ind(outputs);
net = competlayer(3);
net = configure(net, inputs);
net.trainParam.epochs = 50;
net = train(net,inputs);
y = net(inputs);
y = vec2ind(y);
figure(2); hold on;
plot(outputs, 'd', 'MarkerSize',10,'LineWidth',2,'LineStyle','none');
plot(y, '*', 'MarkerSize',10,'LineWidth',2,'LineStyle','none');
box on; grid on; hold off;
```

The running results of the above program are shown in Fig. 1.20, with the true category of each sample represented by a diamond and the predicted category of each sample represented by an asterisk. As can be seen from the figure, the competitive learning neural network correctly judged the categories of most sample points, while about a dozen sample categories were incorrectly predicted.

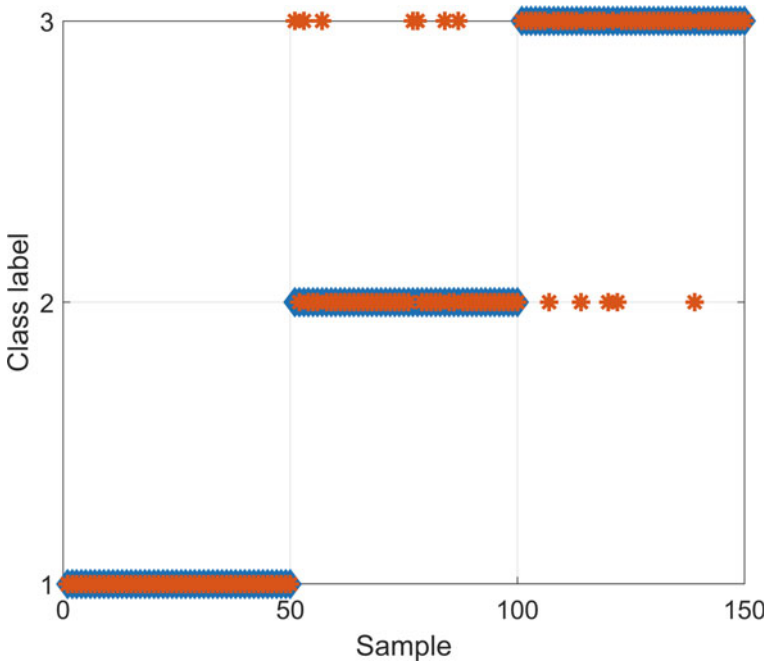

Fig. 1.20 Competitive learning: clustering results of iris data

1.6 Deep Neural Network

Shallow neural networks and deep neural networks are mentioned in Sect. 1.2. Both of these two neural networks have more than or equal to 2 layers, which can be collectively referred to as multi-layer neural networks. With the development of artificial intelligence, deep learning has been deeply rooted in people's hearts, becoming the hottest topic and achieving unprecedented effects [5]. It is just for this reason that deep neural network is separated from multi-layer neural network. The so-called deep learning is actually a machine learning method based on the extension of deep neural network.

Perhaps some readers think that if the hidden layers of the shallow neural network were increased, it would become a deep neural network. While the performance of the obtained deep neural network is not good, the main reasons are:

(1) Gradient disappearance. In the error backpropagation method, as the number of hidden layers increases, the output error cannot be transmitted to the previous hidden layer nodes, which is the problem of gradient disappearance in the backpropagation method.
(2) Overfitting. During the training process, the neural network overlearns the samples, which leads to the neural network has very good performance on

the training set and poor performance on the test set, which is the overfitting problem of the neural network.

(3) The increasing amount of computation. When the number of hidden layers of the neural network is increased, if the fully connected structure is adopted, the number of connection weights between nodes increases very fast, and the dimension of the weight matrix is very large, resulting in a sharp increase in the amount of computation, which is the problem of increasing the amount of computation.

The gradient disappearance problem can be solved by using ReLU activation function and using cross entropy loss function in learning rules. These two small changes can improve the performance of deep neural networks. The ReLU activation function was introduced earlier and, to repeat, its expression is:

$$f(x) = \max(0, x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{1.34}$$

As can be seen from (1.34), when $x > 0$, the value of ReLU activation function is x; When $x \leq 0$, the value of ReLU activation function is 0. It is not difficult to see that the value of the activation function is always non-negative. And the derivative of the ReLU activation function is:

$$\varphi'(x) = \frac{df(x)}{dx} = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{1.35}$$

The expression of the cross-entropy loss function is:

$$L(x) = -\sum_{i=1}^{M} o_i \log_2 o_i \tag{1.36}$$

where M is the number of output nodes, o_i is the predicted output.

Node dropout or regularization methods can solve the overfitting problem. Regularization methods were introduced earlier. Moreover, the validation set can also alleviate the overfitting problem in neural network training. Node dropout method is a simple and effective technique. It means that in the learning process, some nodes are randomly selected for weight update, while the weights of nodes that are not selected are not updated. Since each iteration adopts the method of random selection, the update of node weight is constantly changing. When the dropout technique is used, the dropout percentage of hidden layer nodes is generally set to 50%, while the drop percentage of input layer nodes is generally set to 25%.

The problem of the increasing amount of computation can be solved by using higher performance processors, such as Graphics Processing Unit (GPU), or better performance numerical methods. With the rapid development of computer hardware technology, high-performance GPU can solve the problem of increasing amount of computation to some extent.

Deep learning took off when it was able to solve three problems in which deep neural networks performed poorly. The success of deep learning has been the result of many small techniques and improvements. In this section, deep learning technologies such as convolution, pooling and residual error are not introduced. Instead, a fully connected deep neural network is built from the deep learning toolbox of Matlab, and the results are demonstrated by examples.

Example 1.5 The dataset of this example is the dataset in Matlab, which is about the gear condition classification problem in the transmission system. This problem belongs to supervised learning. The dataset consists of 208 samples, each of which has 18 attributes and each of which has 3 labels. In other words, this problem is a multi-output problem. To facilitate the understanding of the problem, we take the tooth condition label as the output of the classification problem, and convert the sensor state and shaft state into the input of the classification problem. The condition labels for gear teeth include toothless fault and no tooth fault, meaning that the problem is one of two categories. Write out the shallow neural network and deep neural network programs respectively with Matlab. After training, predict the test samples and calculate the accuracy.

The sample size of this example is not large, and in fact, it does not need to use deep neural network. Shallow neural network can also solve this problem. This is just a teaching case to show the difference between shallow neural network and deep neural network. Too many samples will take a long training time, which is not conducive to case presentation.

The problem is simulated in Matlab, and the shallow neural network is used to solve the problem. The programs are as follows:

```
rng(0);
filename = "transmissionCasingData.csv";
tbl = readtable(filename,'TextType','String');
labelName = "GearToothCondition";
tbl = convertvars(tbl,labelName,'categorical');
classNames = categories(tbl{:,labelName});
categoricalInputNames = ["SensorCondition" "ShaftCondition"];
tbl = convertvars(tbl,categoricalInputNames,'categorical');
for i = 1:numel(categoricalInputNames).
    name = categoricalInputNames(i);
    oh = onehotencode(tbl(:,name));
    tbl = addvars(tbl,oh,'After',name);
    tbl(:,name) = [];
end
tbl = splitvars(tbl);
inputs = (table2array(tbl(:, 1:(end-1))))';
outputs = (double(tbl{:,labelName}))';
hiddenLayerSize = [20];
net = feedforwardnet(hiddenLayerSize);
```

```
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
net.trainFcn = 'traingdm';
net.trainParam.epochs = 1000;
[net, tr] = train(net, inputs, outputs);
tstInd = tr.testInd;
YPred = net(inputs(:, tstInd));
YPred(YPred<1.5) = 1;
YPred(YPred>=1.5) = 2;
tstOutputs = outputs(tstInd);
accuracy = sum(YPred = = tstOutputs)/numel(tstOutputs);
figure (1);
confusionchart(tstOutputs,YPred);
```

After running the above program, it can be concluded that the accuracy rate of the shallow neural network on the test set is 90.32%. The program also draws the confusion matrix, which is omitted here. The training process of shallow neural network is shown in Fig. 1.21. As can be seen from the figure, there are 20 neurons in the hidden layer, and the network trains the training data 1000 times. Such training times are called epochs, also known as generations. This means that each training sample has been repeated for 1,000 generations. As shown in Fig. 1.21, the training time of this network is very short, only 1 s.

Next, the problem is simulated in Matlab and the deep neural network is used to solve the problem. The programs are as follows:

```
rng(0);
filename = "transmissionCasingData.csv";
tbl = readtable(filename,'TextType','String');
labelName = "GearToothCondition";
tbl = convertvars(tbl,labelName,'categorical');
classNames = categories(tbl{:,labelName});
categoricalInputNames = ["SensorCondition" "ShaftCondition"];
tbl = convertvars(tbl,categoricalInputNames,'categorical');
for i = 1:numel(categoricalInputNames)
    name = categoricalInputNames(i);
    oh = onehotencode(tbl(:,name));
   tbl = addvars(tbl,oh,'After',name);
end
tbl = splitvars(tbl);
numObservations = size(tbl,1);
numObservationsTrain = floor(0.7*numObservations);
numObservationsValidation = floor(0.15*numObservations);
numObservationsTest = numObservations - numObservationsTrain - numObser-
vationsValidation;
  idx = randperm(numObservations);
```

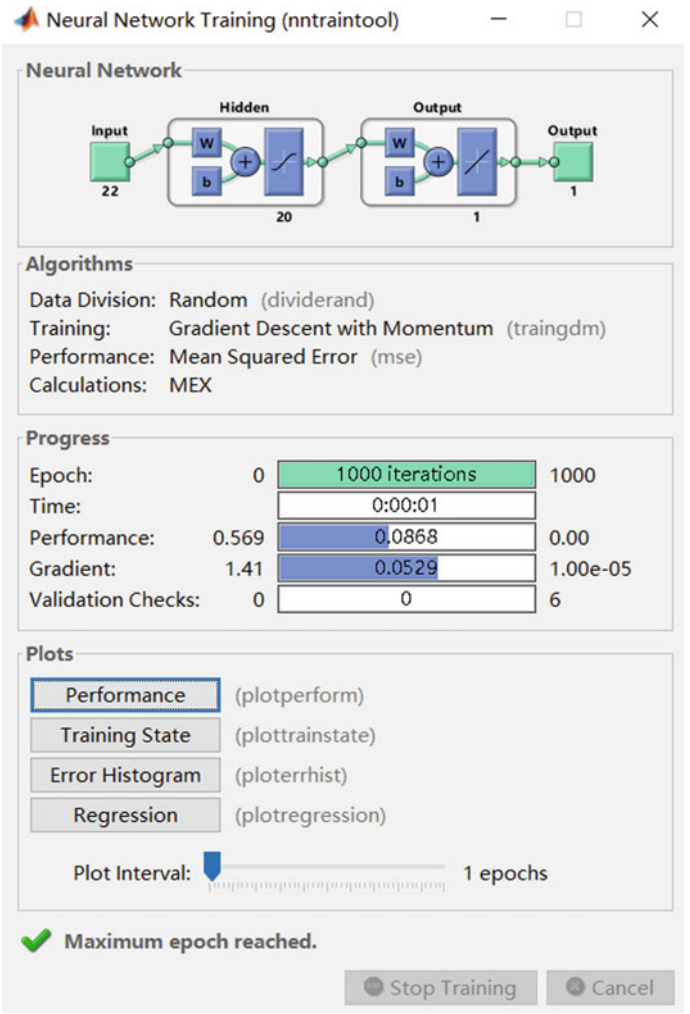

Fig. 1.21 Training process of shallow neural network

```
    idxTrain = idx(1:numObservationsTrain);
    idxValidation = idx(numObservationsTrain + 1: …
      numObservationsTrain + numObservationsValidation);
idxTest = idx(numObservationsTrain + numObservationsValidation + 1:end);
tblTrain = tbl(idxTrain,:);
tblValidation = tbl(idxValidation,:);
tblTest = tbl(idxTest,:);
numFeatures = size(tbl,2) - 1;
numClasses = numel(classNames);
layers = [featureInputLayer(numFeatures,'Normalization', 'zscore').
```

```
    fullyConnectedLayer(20
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(numClasses)
    softmaxLayer.
    classificationLayer];
miniBatchSize = 8;
options = trainingOptions('adam', …
    'MaxEpochs', 30, …
    'MiniBatchSize',miniBatchSize, …
    'Shuffle','every-epoch', …
    'ValidationData',tblValidation, …
    'Plots','training-progress', …
    'Verbose',false);
net = trainNetwork(tblTrain,labelName,layers,options);
YPred = classify(net,tblTest(:,1:end-1),'MiniBatchSize',miniBatchSize);
YTest = tblTest{:,labelName};
accuracy = sum(YPred == YTest)/numel(YTest);
figure(1);
cm = confusionchart(YTest,YPred);
```

After running the above program, it can be concluded that the accuracy of the deep neural network on the test set is 93.75%. The program also draws the confusion matrix, which is omitted here. The training process of this deep neural network is shown in Fig. 1.22. As can be seen from the figure, the deep neural network is run on a single CPU computer, and the accuracy of the verification set is 93.55%, which is similar to that of the test set, indicating that there is no overfitting phenomenon. The network performed 30 rounds of training on the training data, much less than the 1000 rounds of shallow neural network. As shown in Fig. 1.22, the training time of this network is 14 s, indicating that the training time of deep neural network is longer than that of shallow neural network. Moreover, during the training of this deep neural network, the number of iterations per round is 18. This is because deep neural networks generally use batch or minibatch training.

In the learning process of a neural network, samples need to be passed to the input layer, and the weights are adjusted after passing through the network. If the weights are adjusted after each training sample is passed to the input layer, this is the shallow neural network learning method. The so-called batch learning method is to pass all training samples to the input layer before adjusting the weights. In this way, the weights are updated by averaging the changes of all samples. The so-called small-batch learning approach is in between the above two approaches, where the weights are adjusted after passing a portion of the training samples to the input layer, and the average weights are used to update the weights of the neural network. In the above example of deep neural network, the size of small batch is set to 8, i.e., the number of samples passed to the input layer for each iteration is 8, and the number

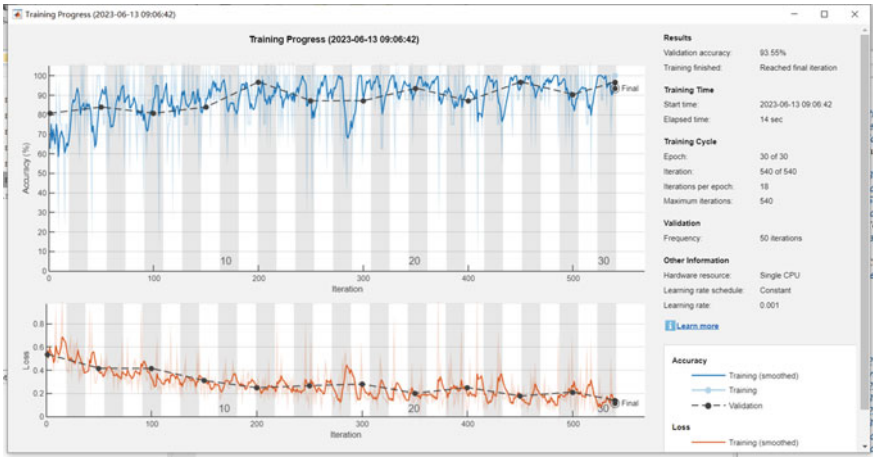

Fig. 1.22 Training process of deep neural network

of training samples is $0.7 \times 208 = 145$, then the number of iterations needed to train all samples is $145/8 \approx 18$.

The previous section describes the way to write programs to implement shallow and deep neural networks, and next we describe the way to implement deep neural networks using Matlab's App. Matlab provides many apps for users to use, which is a graphical user interface. Since the introduction of the App design approach in 2016, Matlab has gradually added many functions and the App has become more and more mature.

Find the Deep Network Design App in the Matlab menu, and by clicking Open, you can build a deep neural network by dragging and dropping. As shown in Fig. 1.23, we build the same structure as the deep neural network program above, the input layer is the "featureInputLayer", which is suitable for numerical data without spatial–temporal characteristics, and for image data you can choose the "imageInputLayer" on the left. the rest of the layers of this deep neural network are not described in detail. It should be noted that Fig. 1.23 shows that the network has 7 layers. The number of layers here is not the number of layers in the network, but the number of components in the whole network. For example, the "featureInputLayer" is the first layer, which is called "input", the second layer is the "fullyConnectedLayer", which is called "fc_1", and so on, and the seventh layer is the "classificationLayer", which is called "classoutput".

After creating the network, the network structure needs to be analyzed to verify the feasibility of the designed network. As shown in Fig. 1.24, after analyzing the network structure, no warnings or errors are given. It should be noted that it is possible to export the designed network, either as a file or to a workspace, or to the corresponding code program. This provides more options for users who prefer to write code for editing, while users who prefer interface interaction can export to a file and continue editing the network structure later.

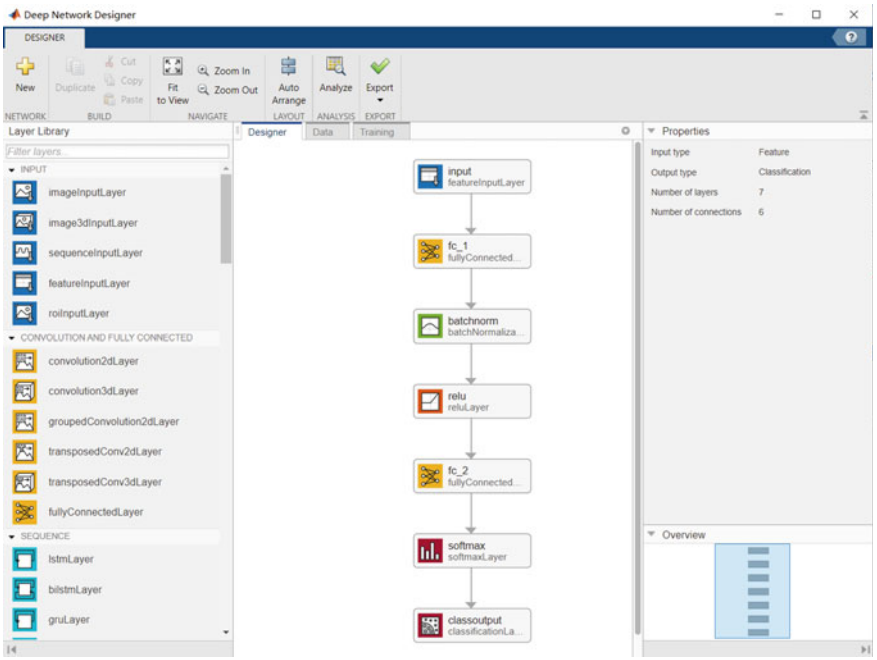

Fig. 1.23 Designing a deep neural network using an App

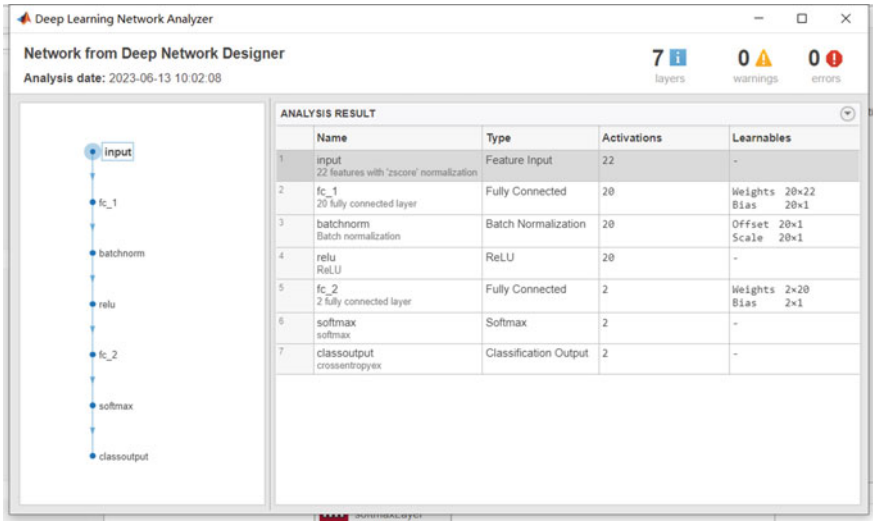

Fig. 1.24 Analyzing a deep neural network using an App

After finishing the design of the network, you can import the existing dataset in the "Data" panel, and also train the deep neural network in the "Training" panel, and these operations will not be described specifically. It should be noted that some features of the Deep Network Designer App are not perfect, and new features are added every year, so users can design the network according to the version of Matlab.

Exercises

(1) Try to write three neuron activation functions and draw the corresponding curves.
(2) Try to draw a schematic diagram of the neuron model, write the mathematical model of the neuron, and explain the meaning of each variable in the model.
(3) Try to write the process of backpropagation neural network algorithm and explain its advantages and disadvantages.

References

1. Zhou ZH (2021) Machine learning. Springer, Singapore. https://doi.org/10.1007/978-981-15-1967-3
2. Ben-Ari M, Mondada F (2015) Elements of robotics. Springer, Cham, pp 203–220. https://doi.org/10.1007/978-3-319-62533-1
3. Chen K, Zhang X, Zhang X (2022) Identifying important attributes for secondary school student performance prediction. In: Liang Q, Wang W, Mu J, Liu X, Na Z (eds) 3rd Artificial intelligence in China, ChangBaiShan, July 2021. Lecture Notes in Electrical Engineering, vol 854. Springer, Singapore, pp 151–158. https://doi.org/10.1007/978-981-16-9423-3_19
4. Keller JM, Liu D, Fogel DB (2016) Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation. Wiley-IEEE Press, New York
5. Aggarwal CC (2018) Neural networks and deep learning—a textbook. Springer, Cham. https://doi.org/10.1007/978-3-319-94463-0

Chapter 2
Convolutional Neural Network

Abstract Convolutional neural network is one of the most important networks in deep learning. Different from common artificial neural network, the main characteristic of convolutional neural network is the convolution operation. It has made remarkable achievements in computer vision and natural language processing. Moreover, convolutional neural network has received extensive attention from industry and academia. This chapter first introduces the convolution operation of the convolutional neural network. Then performance evaluation metrics are introduced. Based on two typical convolutional neural network, transfer learning is demonstrated to use trained convolutional neural network to solve new computer vision problems. Finally, the state-of-the-art research progress of artificial neural network is provided.

2.1 Overview of Convolutional Neural Network

Convolution neural network (CNN) is used to deal with artificial neural network with mesh structure data, its main characteristic is the convolution operation. In the common artificial neural network, such as BP neural network, the parameter calculation of the network is generally realized by matrix multiplication. In convolutional neural network, the parameter calculation of at least one layer of the network is realized by convolutional operation. CNN is one of the most important networks in the field of deep learning. Since CNN has made remarkable achievements in many fields including but not limited to computer vision and natural language processing, it has received extensive attention from industry and academia in the past few years.

In 1998, Yann LeCun first used the term "convolutional" in his paper, which is where the name convolutional neural network came from [1]. LeCun used CNN to solve the problem of handwritten postal code recognition, which involves image processing. It can be said that CNN were originally used for image recognition, which belongs to the field of computer vision. As we all know, vision is the main source of human information about the outside world. Psychologist Treicher has done an experiment on the sources of human access to information, and the experimental results show that 83% of human access to information comes from vision, 11% comes from hearing, and the rest comes from smell, touch and taste. In artificial

X. Zhang et al., *Intelligent Information Processing with Matlab*,
https://doi.org/10.1007/978-981-99-6449-9_2

intelligence, computer vision is also the largest area of research. For example, in the artificial intelligence software market organized by a research institute from China in 2020, artificial intelligence is divided into computer vision, speech recognition and natural language processing, and data science; Computer vision accounted for about 56.6%, speech recognition and natural language processing for 35.6%, and data science for about 7.8%.The research of artificial intelligence is inseparable from computer vision, and the research of computer vision is inseparable from images. Therefore, we need to learn CNN with images as the basic data.

Let's take the MNIST image dataset for an example. The MINST dataset is an open-source handwritten digit recognition dataset maintained by LeCun [1]. In the MNIST dataset, the training set consists of handwritten numbers from 250 different people, 50% of whom are high school students and 50% of whom are Census Bureau staff, and the test set is the same ratio. In the MNIST dataset, each image is composed of 28×28 pixels, and each pixel is represented by a grayscale value, that is, all images are grayscale images of the same size. The dataset includes 10 Arabic numbers ranging from 0 to 9, so it is a classification problem for 10 categories.

In the MNIST dataset, the training set has 60,000 images, while the test set has 10,000 images. We can use the following programs to look at the images in this dataset:

```
oldpath = addpath(fullfile(matlabroot,examples','nnet','main'));
filenameImagesTrain = 'dataset\train-images-idx3-ubyte.gz';
filenameLabelsTrain = 'dataset\train-labels-idx1-ubyte.gz';
filenameImagesTest = 'dataset\t10k-images-idx3-ubyte.gz';
filenameLabelsTest = 'dataset\t10k-labels-idx1-ubyte.gz';
XTrain = processImagesMNIST(filenameImagesTrain);
YTrain = processLabelsMNIST(filenameLabelsTrain);
XTest = processImagesMNIST(filenameImagesTest);
YTest = processLabelsMNIST(filenameLabelsTest);
path(oldpath);
figure(1);
numImages = size(XTrain, 4);
idx = randperm(numImages,9);
for i = 1:length(idx)
    subplot(3,3,i);
    im = extractdata(XTrain(:,:,:,idx(i)));
    imshow(im);
end
```

The result after running the above program is shown in Fig. 2.1. Since 9 images are randomly selected for display, the repeated running of the program may display different images. With the training set and the test set, we have the necessary data for image recognition. Empirically, researchers divide the training set into two parts, one is used to train the neural network model, and the other is used to cross-validate the performance of the model, so researchers now often refer to the dataset as the training set, the validation set, and the test set.

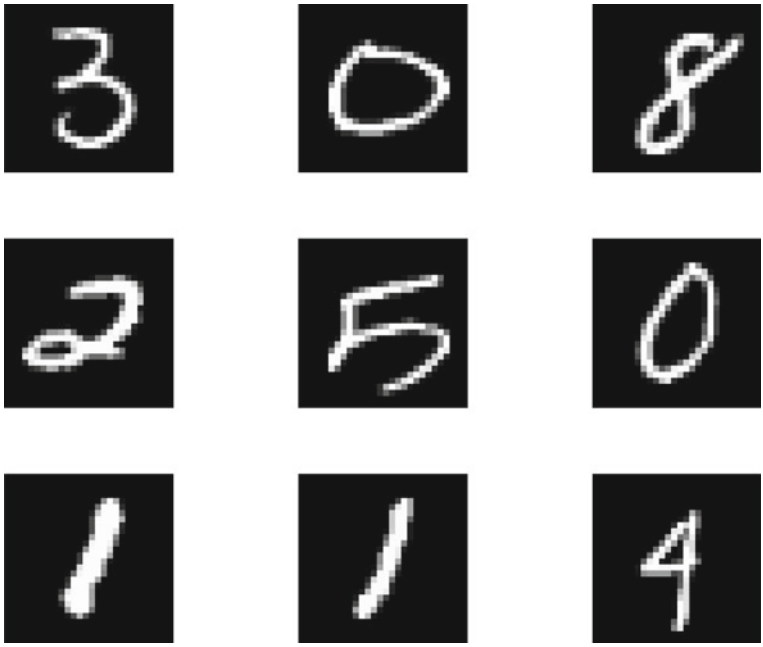

Fig. 2.1 The 9 images in the MNIST dataset

In a BP neural network, suppose there is an input layer, one hidden layer and an output layer. For the binary classification problem, suppose the size of the input sample is N, the number of neurons in the hidden layer is M, the neurons in the output layer is 1. Hence, the number of weight parameters from the input layer to the hidden layer is $N \times (M + 1)$, where 1 denotes the threshold, and the number of weight parameters from the hidden layer to the output layer is $M + 1$, then the number of weight parameters of the BP neural network is $N \times (M + 1) + M + 1$. In a fully connected deep neural network, suppose there is an input layer, two hidden layers and an output layer. For the same binary classification problem, the number of weight parameters of this deep neural network is $N \times (M + 1) + M \times (M + 1) + M + 1$. It can be seen that the growth of the number of parameters is proportional to the size of the input layer, and also proportional to the number of neurons in the hidden layer. For image data, the growth rate of the number of parameters is too fast, which leads to a decrease in the learning efficiency of the neural network, and the convolution operation is an effective method to solve the problem of too large number of parameters.

CNN includes input layer, convolutional layer, pooling layer, fully connected layer, etc. After the analysis of researchers, the role of convolutional layer and pooling layer is feature extraction, i.e., extracting features such as edge, shadow, contour, etc. from the image. While the fully connected layer is the same role as BP neural network. Generally, CNN has multiple fully connected layers, so it is more

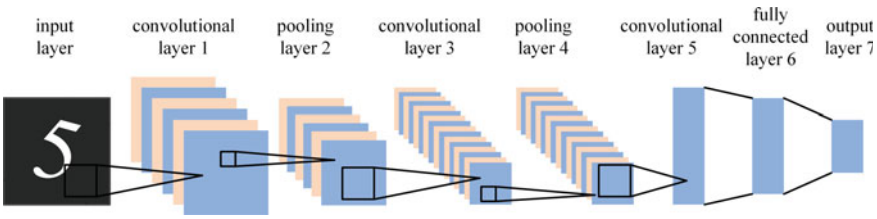

Fig. 2.2 Structure of the convolutional neural network LeNet

like deep neural network, so CNN is a feedforward neural network that contains multiple fully connected layers for convolutional computation. The typical convolutional neural networks (CNNs) are LeNet, AlexNet, VGGNet and GoogLeNet. Unlike AlexNet and VGGNet, GoogLeNet does not rely on deepening the structure of neural networks, but introduces a module called Inception structure.

We use LeNet as an example to describe how to create a CNN. The structure of LeNet is shown in Fig. 2.2, which is a 7-layer network structure. Note that the input layer is not counted in the number of layers.

LeNet is used for handwritten postal code images, so the image of the input layer is assumed to be a 32×32 pixel grayscale map. The first layer of LeNet is the convolutional layer, which is noted as convolutional layer 1. In convolutional layer 1, 6 convolutional kernels are used, each with a size of 5×5 pixels and a step size of 1. Therefore, the size of the image computed by each convolutional kernel is $(32–5 + 1) \times (32–5 + 1) = 28 \times 28$ pixels, 6 convolution kernels generate 6 images of 28×28 pixels, sometimes also called feature maps. The number of neurons from the input layer to the convolutional layer 1 is $28 \times 28 \times 6 = 4704$, and the weight parameters are 6 convolutional kernels and their threshold $(5 \times 5 + 1) \times 6 = 156$. The number of connections from the input layer pixel points to the convolutional layer 1 pixel points is $156 \times 28 \times 28 = 122{,}304$. If the fully connected approach is used, the number of connections is $32 \times 32 \times (4704 + 1) = 4{,}817{,}920$, which shows that the number of connections is greatly reduced by using the convolutional layer.

The second layer of LeNet is the pooling layer, denoted as pooling layer 2. The feature map input to pooling layer 2 is 28×28 pixels, using 2×2 pixel sampling, i.e., 4 pixel values in the 2×2 pixel region are summed, multiplied by a weight parameter, plus a threshold parameter, and using the Sigmoid activation function, pooling layer 2 outputs a 14×14 pixel feature map. Pooling layer 2 uses six 2×2 pixels for sampling, so it yields six 14×14 pixel feature maps, i.e., the number of neurons is $14 \times 14 \times 6$. The number of connections from convolutional layer 1 to pooling layer 2 is $(2 \times 2 + 1) \times 14 \times 14 \times 6 = 5880$. Since the image becomes smaller after sampling, the pooling layer is also called the downsampling layer.

The third layer of LeNet is the convolutional layer, which is called convolutional layer 3. In convolutional layer 3, 16 convolutional kernels are used, and the size of each convolutional kernel is 5×5 pixels with a step size of 1. Therefore, the size of the image computed by each convolutional kernel is $(14–5 + 1) \times (14–5 + 1) = 10 \times 10$ pixels, and 16 convolutional kernels yield 16 feature maps of 10×10

pixels. Since 16 convolutional kernels are used, if all the convolutional kernels are connected to the feature maps obtained in the previous layer, it is easy to cause too many connections. To reduce the computation, the first 6 convolutional kernels of the 16 convolutional kernels are connected to 3 adjacent subsets of the feature maps obtained in the previous layer, the next 6 convolutional kernels are connected to 4 adjacent subsets of the feature maps obtained in the previous layer, and then the next 3 convolutional The next 6 convolutional kernels are connected to the subset of 4 non-adjacent feature maps obtained from the previous layer, the next 3 convolutional kernels are connected to the subset of 4 non-adjacent feature maps obtained from the previous layer, and the last 1 convolutional kernel is connected to all feature maps obtained from the previous layer, as shown in Table 2.1. The number of connections from pooling layer 2 to convolutional layer 3 is $6 \times (3 \times 5 \times 5 + 1) + 6 \times (4 \times 5 \times 5 + 1) + 3 \times (4 \times 5 \times 5 + 1) + 1 \times (6 \times 5 \times 5 + 1) \times 10 \times 10 = 151{,}600$. the number of neurons in convolutional layer 3 is $10 \times 10 \times 16 = 1600$.

In Table 2.1, the serial number in the first row represents the 16 convolutional kernels of convolutional layer 3, while the serial number in the first column represents the 6 convolutional kernels of pooling layer 2. The number 1 after the second column in the second row indicates that one of the convolutional kernels of pooling layer 2 is connected to one of the convolutional kernels of convolutional layer 3, while the number 0 indicates that there is no connection between the two convolutional kernels.

The fourth layer of LeNet is the pooling layer, noted as pooling layer 4. The feature map input to pooling layer 4 is 10×10 pixels, using 2×2 pixels of sampling, which is the same setup as pooling layer 2, and the output is a 5×5 pixel feature map. Pooling layer 4 uses 16 2×2 pixels for sampling, so it generates 16 feature maps of 5×5 pixels, i.e., the number of neurons is $5 \times 5 \times 16 = 400$. The number of connections from convolutional layer 3 to pooling layer 4 is $(2 \times 2 + 1) \times 400 = 2000$.

The fifth layer of LeNet is the convolutional layer, which is called convolutional layer 5. In convolutional layer 5, 120 convolutional kernels are used, and the size of each convolutional kernel is 5×5 pixels with a step size of 1. Therefore, the size of the image computed by each convolutional kernel is $(5–5 + 1) \times (5–5 + 1) = 1 \times 1$ pixel, and 120 feature maps of 1 pixel are obtained from 120 convolutional kernels. The number of neurons in convolutional layer 5 is 120, and each neuron is

Table 2.1 Connection method of pooling layer 2 and convolutional layer 3

Kernel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	0	0	0	1	1	1	0	0	1	1	1	1	0	1	1
2	1	1	0	0	0	1	1	1	0	0	1	1	1	1	0	1
3	1	1	1	0	0	0	1	1	1	0	0	1	0	1	1	1
4	0	1	1	1	0	0	1	1	1	1	0	0	1	0	1	1
5	0	0	1	1	1	0	0	1	1	1	1	0	1	1	0	1
6	0	0	0	1	1	1	0	0	1	1	1	1	0	1	1	1

connected to all the 16 feature maps obtained from pooling layer 4. The number of connections from pooling layer 4 to convolutional layer 5 is $(5 \times 5 \times 16 + 1) \times 120 = 48{,}120$. Since the convolutional kernels of convolutional layer 5 have the same size as the feature maps obtained from pooling layer 4 and are all connected to each other, convolutional layer 5 is equivalent to a fully connected layer.

The sixth layer of LeNet is the fully connected layer, denoted as fully connected layer 6. In fully connected layer 6, the number of neurons is 84, and the number of connections from 120 neurons in convolutional layer 5–84 neurons in this layer is $(120 + 1) \times 84 = 10{,}164$. This layer uses the Sigmoid activation function.

The seventh layer of LeNet is the output layer, denoted as output layer 7, which is also a fully connected layer. In output layer 7, the number of neurons is 10. It uses one-hot encoding method, and 10 neurons can represent the category of numbers from 0 to 9. This layer uses radial basis functions as activation functions.

In Matlab, you can use the Deep Network Designer App to design LeNet, as shown in Fig. 2.3. Export the designed network and select "Generate Code" to generate the corresponding programs, as follows:

```
layers = [
    imageInputLayer([32 32 1],"Name","imageinput")
    convolution2dLayer([5 5],6,"Name","conv1")
```

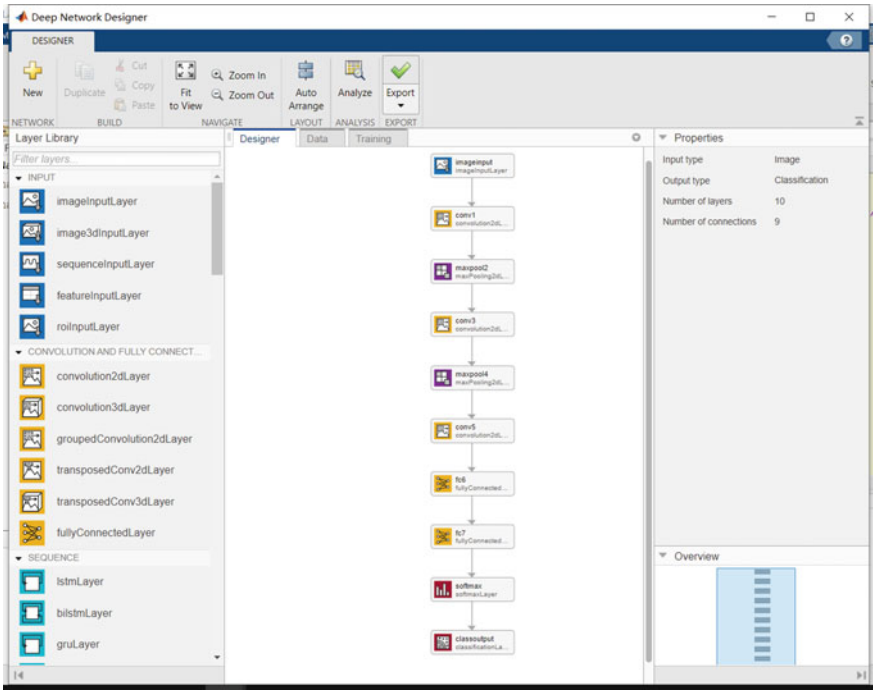

Fig. 2.3 Designing LeNet neural network

```
    maxPooling2dLayer([2 2],“Name”,“maxpool2”,“Padding”,“same”,“Stride”,[2
2])
      convolution2dLayer([5 5],16,“Name”,“conv3”)
    maxPooling2dLayer([2 2],“Name”,“maxpool4”,“Padding”,“same”,“Stride”,[2
2])
      convolution2dLayer([5 5],120,“Name”,“conv5”)
      fullyConnectedLayer(84,“Name”,“fc6”)
      fullyConnectedLayer(10,“Name”,“fc7”)
      softmaxLayer(“Name”,“softmax”)
      classificationLayer(“Name”,“classoutput”)];
plot(layerGraph(layers));
```

For multi-classification problems, “softmaxLayer” and “classificationLayer” are generally required, but they are generally not counted as layers. As mentioned above, the input of LeNet is a 32 × 32 pixel grayscale image, while the image of MNIST is a 28 × 28 pixel grayscale image, there are two ways to solve this problem. One way is to adjust the network structure of LeNet, such as the size of the convolutional kernel and the way to connect between layers; the other way is to adjust the size of the image. Here we use the second method, which is to enlarge the image from 28 × 28 pixels to 32 × 32 pixels. After resizing the image, you can use LeNet to solve the MNIST classification problem. For the image resizing, the following programs can be used:

```
resize3dLayer(“Name”,“resize3d-output-size”,…
    “GeometricTransformMode”,“half-pixel”,“Method”,“nearest”,…
    “NearestRoundingMode”,“round”,“OutputSize”,[32 32 1]).
```

Place the program for adjusting image size between the input layer and convolution layer 1.

The programs to train and test LeNet model are as follows:

```
layers = [
    imageInputLayer([28 28 1],“Name”,“imageinput”)
    resize3dLayer(“Name”,“resize3d-output-size”,…
    “GeometricTransformMode”,“half-pixel”,“Method”,“nearest”,…
    “NearestRoundingMode”,“round”,“OutputSize”,[32 32 1])
    convolution2dLayer([5 5],6,“Name”,“conv1”)
   maxPooling2dLayer([2 2],“Name”,“maxpool2”,“Padding”,“same”,“Stride”,[2
2])
    convolution2dLayer([5 5],16,“Name”,“conv3”)
   maxPooling2dLayer([2 2],“Name”,“maxpool4”,“Padding”,“same”,“Stride”,[2
2])
    convolution2dLayer([5 5],120,“Name”,“conv5”)
    fullyConnectedLayer(84,“Name”,“fc6”)
    fullyConnectedLayer(10,“Name”,“fc7”)
    softmaxLayer(“Name”,“softmax”)
```

```
    classificationLayer("Name","classoutput")];
options = trainingOptions('sgdm', …
    'MaxEpochs',10, …
    'MiniBatchSize',128, …
    'Plots','training-progress');
trainNet = trainNetwork(XTrain,YTrain,layers,options);
save('MNIST_LeNet5.mat','trainNet');
YPred = classify(trainNet, XTest);
accuracy = sum(YPred == YTest)/numel(YPred);
```

The training process is not shown for the saving of space. The accuracy on the test set is 98.42%, which shows that LeNet is able to solve the MNIST classification problem.

It should be noted that when using Matlab to solve classification problems, the classify function is generally used to make predictions, while when using Matlab to solve regression problems, the predict function is generally used to make predictions.

2.2 Neural Network Performance Evaluation

This section describes how to evaluate the performance of a neural network. In the previous section, we used accuracy to describe the performance of a model, but there are many other metrics that can evaluate the performance of a model. Note that the evaluation methods can also be used to assess other machine learning methods such as decision tree and support vector machine.

In the Sect. 2.1, we used the MNIST dataset, which is not stored as images, using the form of an numerical arrays. This is obviously not a figurative way to represent image dataset. In this section we use the Digits dataset as the dataset, which is a similar dataset to MNIST that comes with Matlab. The Digits dataset consists of 10,000 images in ten categories from 0 to 9, with 1000 images in each category. The form of our image data storage divides this dataset into a training set, a validation set and a test set in the ratio of 6:2:2. The programs used are as follows:

```
digitDatasetPath = fullfile(matlabroot,'toolbox',…
    'nnet','nndemos', 'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, …
    'IncludeSubfolders',true,'LabelSource','foldernames');
figure(1);
numImages = length(imds.Files);
idx = randperm(numImages,9);
for i = 1:length(idx)
    subplot(3,3,i);
    imshow(imds.Files{idx(i)});
end
labelCount = countEachLabel(imds);
```

```
img1 = readimage(imds,1);
size(img1)
[imdsTest,imdsTrain] = splitEachLabel(imds,0.2,'randomize');
[imdsValid,imdsTrain] = splitEachLabel(imdsTrain,0.25,'randomize');
labelCountTest = countEachLabel(imdsTest);
labelCountTrain = countEachLabel(imdsTrain);
labelCountValid = countEachLabel(imdsValid);
```

The results of the above program after running are shown in Fig. 2.4. As can be seen from the figure, the Digits dataset is very similar to MNIST, except that the number of samples becomes smaller. There are 600 images for each category in the training set, 200 images for each category in the validation set, and 200 images for each category in the test set.

We still use the LeNet model from the previous section and the network structure can be analyzed using the following program:

```
analyzeNetwork(layers);
```

Analyzing the network structure yields Fig. 2.5, from which we can see the size of the output feature map for each layer and the number of weight parameters to be learned for each layer. For LeNet, the input layer, adjustment size, pooling layer,

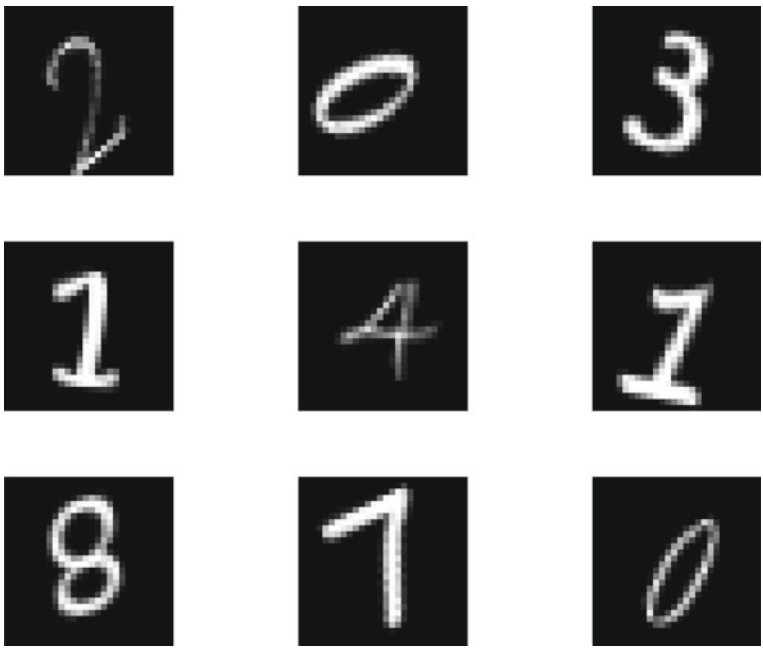

Fig. 2.4 Showing 9 images in the Digits dataset

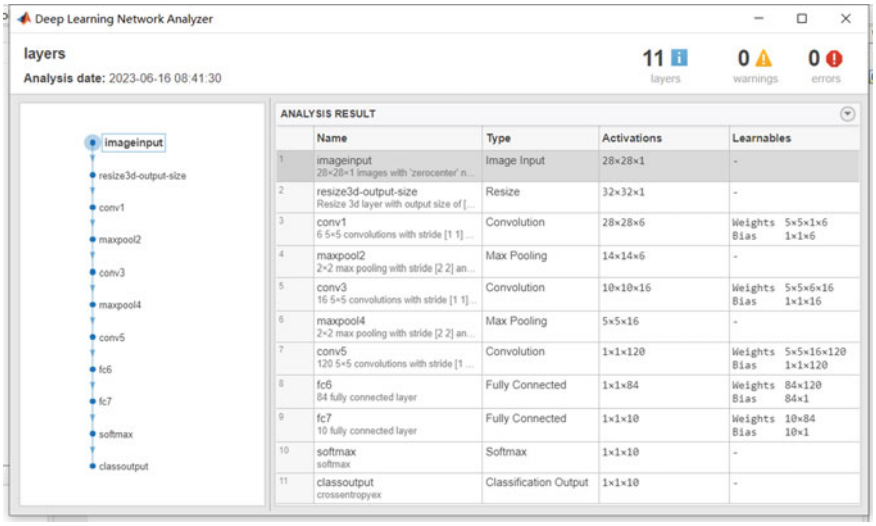

Fig. 2.5 Analyzing the LeNet network

Softmax and output layer do not involve weight parameters, so they are indicated by "-" in the figure.

The training and testing programs for the LeNet network are as follows:

```
layers = [
    imageInputLayer([28 28 1],"Name","imageinput")
    resize3dLayer("Name","resize3d-output-size",…
    "GeometricTransformMode","half-pixel","Method","nearest",…
    "NearestRoundingMode","round","OutputSize",[32 32 1])
    batchNormalizationLayer
    convolution2dLayer([5 5],6,"Name","conv1")
   maxPooling2dLayer([2 2],"Name","maxpool2","Padding","same","Stride",[2
2])
    convolution2dLayer([5 5],16,"Name","conv3")
   maxPooling2dLayer([2 2],"Name","maxpool4","Padding","same","Stride",[2
2])
    convolution2dLayer([5 5],120,"Name","conv5")
    fullyConnectedLayer(84,"Name","fc6")
    fullyConnectedLayer(10,"Name","fc7")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];
options = trainingOptions('sgdm', …
    'MaxEpochs',10, …
    'Shuffle','every-epoch', …
    'MiniBatchSize',128, …
```

```
    'ValidationData',imdsValid, …
    'ValidationFrequency',10, …
    'Verbose',false, …
    'Plots','training-progress');
trainNet = trainNetwork(imdsTrain,layers,options);
save('Digits_LeNet5.mat','trainNet');
YPred = classify(trainNet, imdsTest);
YTest = imdsTest.Labels;
accuracy = sum(YPred == YTest)/numel(YPred);
```

The results of the above program after running are shown in Fig. 2.6. It can be seen that the training process was completed in 78 s on a CPU computer, which is still relatively fast. The accuracy on the validation set is 95.85%, while the accuracy on the test set is 96.65%, which shows that the LeNet network is able to solve the classification problem of the Digits dataset.

Accuracy is the most commonly used and basic metric to evaluate the performance of a classification model, but it is not applicable to all cases. For example, suppose there is an unbalanced dataset which is a binary classification problem containing 10,000 samples, where the number of positive class samples is 9900 and the number of negative samples is 100. If there is a classification model that predicts all the samples as positive class, then its accuracy is 9900/10,000 = 99%. Although the accuracy of this classification model is very high, it cannot determine the negative class samples, and we generally consider the model unconvincing. Especially for the medical diagnosis problem, the number of patients as negative class and the number of normal people as positive class samples, usually the number of patients is much smaller than the number of normal people, such a model cannot effectively determine

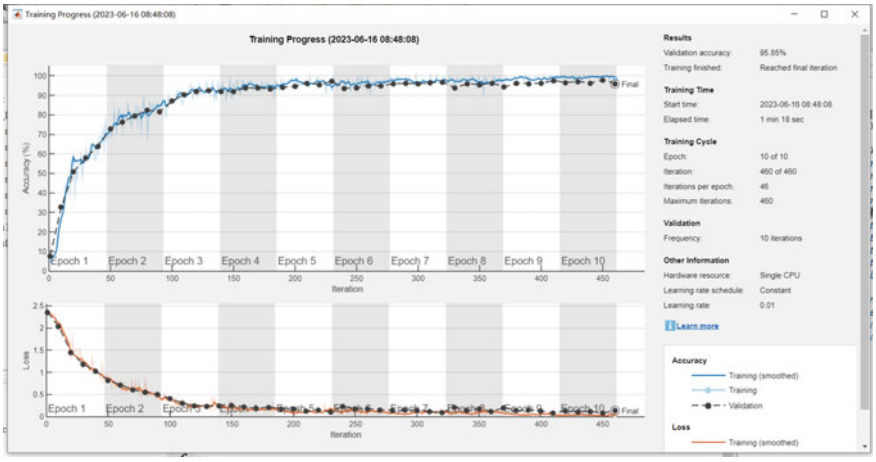

Fig. 2.6 Training process of LeNet network on Digits dataset

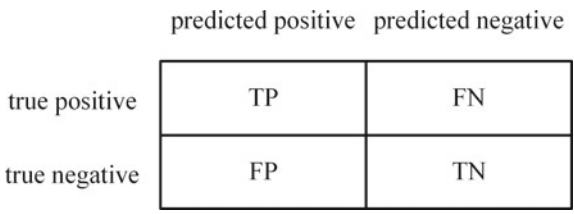

Fig. 2.7 Confusion matrix of binary classification problem

the negative class and is not helpful for identifying patients. Thus, other criteria are needed to evaluate the performance of the classification model.

We first present the evaluation metrics for binary classification problems and then extend to multi-class classification problems. For binary classification problems, in medicine they are generally referred to as positive and negative, while in artificial intelligence we generally refer to samples as positive and negative classes and assume that the positive class is the one we focus on.

As shown in Fig. 2.7, for a binary classification problem, a particular classification model may predict the sample as either positive or negative class, and the true class of the sample has been given, then four scenarios can be combined:

(1) True Positive (TP). True means that a classification model identifies a sample as a positive class, and the true class of the sample is the positive class, which simply means that the positive sample is correctly identified as the positive class.
(2) False Negative (FN). False negative means that a classification model identifies a sample as a negative class, while the true class of the sample is positive, which simply means that the positive sample is incorrectly identified as a negative class.
(3) False Positive (FP). False positive means that a classification model identifies a sample as a positive class, while the true class of the sample is a negative class, which simply means that the negative sample is incorrectly identified as a positive class.
(4) True Negative (TN). True negative means that a classification model identifies a sample as a negative class when the true class of the sample is negative, which simply means that the negative sample is correctly identified as a negative class.

If we use the letter abbreviations in Fig. 2.7 to represent the number of samples for each scenario, the accuracy rate is:

$$ACC = \frac{TP + TN}{TP + FN + FP + TN} \tag{2.1}$$

where ACC denotes accuracy, the numerator is the number of sample categories correctly identified by a model, and the denominator is the number of all samples. Precision is defined as:

$$P = \frac{TP}{TP + FP} \tag{2.2}$$

where P denotes the precision rate, which indicates the proportion of samples that are true positive samples among those identified as positive classes by a particular model. The Recall is:

$$R = \frac{TP}{TP + FN} \tag{2.3}$$

where R denotes the recall rate, which indicates the proportion of samples that are identified as positive classes by a particular model among those that are truly positive classes. In Fig. 2.7, the precision rate can be calculated from the values in the first column, while the recall rate can be calculated from the values in the first row.

Precision and recall express the performance of some aspect of the binary classification problems, and the F-score (F-score), which combines the P and R metrics, is expressed as

$$F_\beta = \left(1 + \beta^2\right) \frac{P \times R}{\left(\beta^2 \times P\right) + R} \tag{2.4}$$

where β is the parameter that balances the precision rate and the recall rate. When $\beta = 2$, the weight of recall is higher than that of precision, and the F-score focuses on recall; when $\beta = 0.5$, the weight of precision is higher than that of recall, and the F-score focuses on precision; and when $\beta = 1$, the weights of precision and recall are equal, and the F-score is also called $F1$-score. $F1$-score is also a common metric to evaluate the performance of the model.

We can also calculate the true positive class rate, whose expression is:

$$TPR = \frac{TP}{TP + FN} \tag{2.5}$$

where *TPR* stands for True Positive Rate. It is clear to see that the *TPR* and the recall rate are the same. Correspondingly, the false positive class rate is:

$$FPR = \frac{FP}{FP + TN} \tag{2.6}$$

where FPR denotes the False Positive Rate. The *TPR* and *FPR* rate can be calculated from the first and second rows in Fig. 2.7. By using the *TPR* as the vertical axis and the *FPR* as the horizontal axis, we can plot a curve called Receiver Operating Characteristic (ROC). Since both *TPR* and *FPR* are between 0 and 1, ROC is a curve located in [0, 1] × [0, 1]. We refer to the area enclosed by the ROC curve and the horizontal axis as the Area Under Curve (AUC). Since AUC is a numerical value, it can quantitatively describe the performance of a classification model.

For binary classification problems, *ACC*, *P*, *R*, $F1$ score and AUC are available evaluation metrics. For multi-class classification problems, it is sufficient to generalize the above formulae. These evaluation metrics are between 0 and 1, and the closer to 1 the better the performance of the classification model. The values of *ACC*, *P*, *R*,

$F1$-score and AUC can be expressed as percentage. Let us take the LeNet network as an example to solve the handwritten postal code recognition problem, which is a ten-class classification problem. The programs to calculate the evaluation metrics on the test set are as follows:

```
M = confusionmat(YTest, YPred);
ACC = sum(diag(M)) / sum(M(:));
P1 = diag(M)./(sum(M,1) + 0.0001)';
R1 = diag(M)./(sum(M,2) + 0.0001);
P = mean(P1);
R = mean(R1);
F1score = 2*P*R/(P + R);
fig = figure;
cm = confusionchart(YTest,YPred,'RowSummary',...
    'row-normalized','ColumnSummary','column-normalized');
CLASSES = unique(YTest);
for i1 = 1:length(CLASSES).
    % compute AUC for Class i.
    [XRF,YRF,TRF,AUCRF(i1)] = perfcurve(YTest,...
        YPredScores(:, i1),CLASSES(i1));
end
AUC = mean(AUCRF);
```

After running the above program, the result in Fig. 2.8 is obtained. The values of all metrics are expressed as percentage. The *ACC* overaged on ten categories is 97.6%, the average precision rate on ten categories is 97.65%, the average recall rate on ten categories is 97.6%, and the average AUC on ten categories is 99.99%. All four metrics are close to 100%, which shows that LeNet shows a very good performance on Digits dataset.

In Fig. 2.8, we can see that the precision rate of each category, the precision rate of number 9 is 93.5%, and the precision rate of the rest of numbers are above 95%; the recall rates of number 3 and number 8 are 92.5% and 94.5% respectively, and the recall rates of the rest of numbers are above 95%. It can be seen that although the precision rate and recall rate of LeNet on the test set are above 95%, its precision or recall of some digits could be less than 95%, which indicates that there is still some room for improvement in the recognition of a specific digit.

2.3 Transfer Learning with Convolutional Neural Network

In this section, we will describe how to use a pre-trained model, i.e., train a convolutional neural network on one dataset, and for another dataset, we just need to modify the previously trained convolutional neural network so that it can match the image size and number of categories of the new dataset. This means that we can reuse the

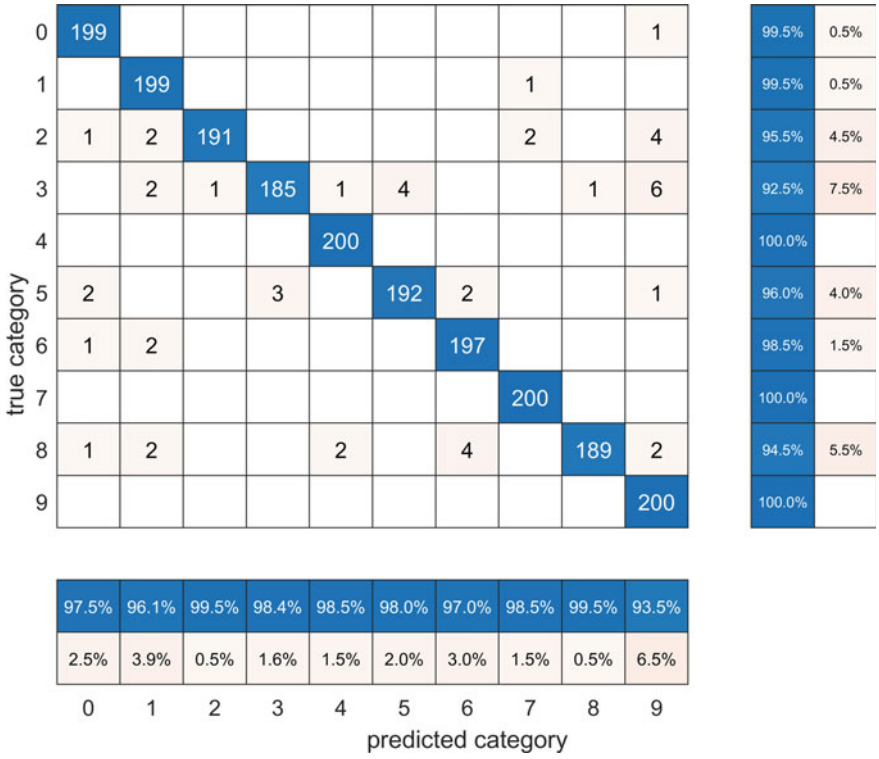

Fig. 2.8 Confusion matrix of the LeNet network on Digits dataset

prior knowledge to solve the unknown problem. In the field of machine learning, this practice is called transfer learning.

Let's first introduce the ImageNet dataset, which is a dataset maintained by Fei-Fei Li at Stanford University. She sponsored the ImageNet competition, which is called ILSVRC (ImageNet Large Scale Visual Recognition Challenge). ImageNet competition has been held since 2010. The dataset is continuously updated, so the size of the dataset has gradually increased. The most commonly used ImageNet dataset is the 2012 dataset with about 1.2 million images in the training set, about 50,000 images in the validation set, and about 100,000 images in the test set [2]. the ImageNet dataset has 1,000 categories, and the size of each image is 227 $\times$ 227 $\times$ 3, i.e., each image is in color. The dataset is widely used for problems such as image classification, image detection and localization, and is also the home of classic convolutional neural networks such as AlexNet (2012), VGG (2014), GoogLeNet (2014) and ResNet (2015), where the number in parentheses indicates the year in which the network model was attended in the ImageNet competition.

In this section, we take two convolutional neural network models, AlexNet and SqueezeNet [3], as examples. Readers can also try other network models. We use the

Table 2.2 Food example image dataset

Label	Amount
caesar_salad	26
caprese_salad	15
french_fries	181
greek_salad	24
hamburger	238
hot_dog	31
pizza	299
sashimi	40
Sushi	124

food example image dataset as the research problem, as shown in Table 2.2, which contains 978 images with 9 categories.

The food example image dataset is a small size dataset and the number of images in each category is different. From Table 2.2, we can see that the category with the least sample is "caprese_salad" with only 15 images and the category with the most sample is "pizza" with 299 images, and the sample ratio of these two categories is about 1:20, so this dataset can be considered as an unbalanced dataset.

AlexNet is a convolutional neural network proposed by Hinton and his student Alex in 2012. It won the first place in the ImageNet dataset competition. AlexNet is a convolutional neural network with 8 layers, where the first 5 layers are convolutional and the last 3 layers are fully connected. Without going into details of the exact structure and computational process of AlexNet, the programs to solve the problem using pre-trained AlexNet are as follows:

```
dataDir = fullfile("ExampleFoodImageDataset");
url = "https://www.mathworks.com/supportfiles/nnet/data/ExampleFoodImageDataset.zip";
if ~ exist(dataDir, "dir")
    mkdir(dataDir);
    downloadExampleFoodImagesData(url,dataDir);
end
imds = imageDatastore('ExampleFoodImageDataset', …
    'IncludeSubfolders',true,'LabelSource','foldernames');
labelCount = countEachLabel(imds);
img1 = readimage(imds,1);
size(img1)
[imdsTest,imdsTrain] = splitEachLabel(imds,0.2,'randomize');
[imdsValid,imdsTrain] = splitEachLabel(imdsTrain,0.25,'randomize');
labelCountTest = countEachLabel(imdsTest);
labelCountTrain = countEachLabel(imdsTrain);
labelCountValid = countEachLabel(imdsValid);
numTrainImages = numel(imdsTrain.Labels);
```

```
idx = randperm(numTrainImages,9);
I = imtile(imds, 'Frames', idx);
figure;
imshow(I);
net = alexnet;
inputSize = net.Layers(1).InputSize;
analyzeNetwork(net);
numClasses = numel(categories(imdsTrain.Labels));
layersTransfer = net.Layers(1:end-3);
layers = [
    layersTransfer
    fullyConnectedLayer(numClasses,'WeightLearnRateFactor',…
    10,'BiasLearnRateFactor',10)
    softmaxLayer
    classificationLayer];
lgraph = layerGraph(layers);
aug = imageDataAugmenter("RandXReflection", true, …
    "RandYReflection", true, …
    "RandXScale", [0.8 1.2], …
    "RandYScale", [0.8 1.2]);
augImdsTrain = augmentedImageDatastore(inputSize(1:2), imdsTrain, …
    'DataAugmentation', aug);
augImdsVal = augmentedImageDatastore(inputSize(1:2), imdsValid);
opts = trainingOptions("adam", …
    "InitialLearnRate", 1e-4, …
    "MaxEpochs", 10, …
    "ValidationData", augImdsVal, …
    "Verbose", false,…
    "Plots", "training-progress", …
    "ExecutionEnvironment","cpu",…
    "MiniBatchSize",128);
netTransfer = trainNetwork(augImdsTrain, lgraph, opts);
save('Food_AlexNet.mat','netTransfer', …
    'imdsTrain','imdsValid','imdsTest');
augImdsTest = augmentedImageDatastore(inputSize(1:2), imdsTest);
[YPred, YPredScores] = classify(netTransfer, augImdsTest);
YTest = imdsTest.Labels;
M = confusionmat(YTest, YPred);
ACC = sum(diag(M)) / sum(M(:));
P1 = diag(M)./(sum(M,1) + 0.0001)';
R1 = diag(M)./(sum(M,2) + 0.0001);
P = mean(P1); % mean precision of all classes
R = mean(R1); % mean recall of all classes
F1score = 2*P*R/(P + R);
fig = figure;
```

```
cm = confusionchart(YTest,YPred,'RowSummary',...
    'row-normalized','ColumnSummary','column-normalized');
CLASSES = unique(YTest);
for i1 = 1:length(CLASSES)
    % compute AUC for Class i
    [XRF,YRF,TRF,AUCRF(i1)] = perfcurve(YTest,...
        YPredScores(:, i1),CLASSES(i1));
end
AUC = mean(AUCRF);
```

The results of the above program after running are shown in Figs. 2.9, 2.10 and 2.11. Figure 2.9 shows the structure analysis of AlexNet. It is known that the AlexNet network model contains more than 60 million parameters to be learned. It should be noted that the figure shows that the AlexNet structure consists of 25 layers, which is because each step from input to output is considered as one layer in Matlab, so the number of layers shown in the figure is larger than the 8 layers introduced earlier. The training process of this model is more time consuming if the model is not pre-trained.

Figure 2.10 shows the training process of the AlexNet network, and it can be seen that the model was re-trained on a CPU computer in less than 6 min based on the pre-trained model. The accuracy on the validation set was 85.71%. The accuracy on the test set was 83.67%. Thus, the AlexNet network is able to solve food example image dataset.

Figure 2.11 shows the confusion matrix of the AlexNet network. Although the model has a high average accuracy and recall on the nine categories, it has shortcomings, for example, in the "sashimi" category, the precision rate of AlexNet is only

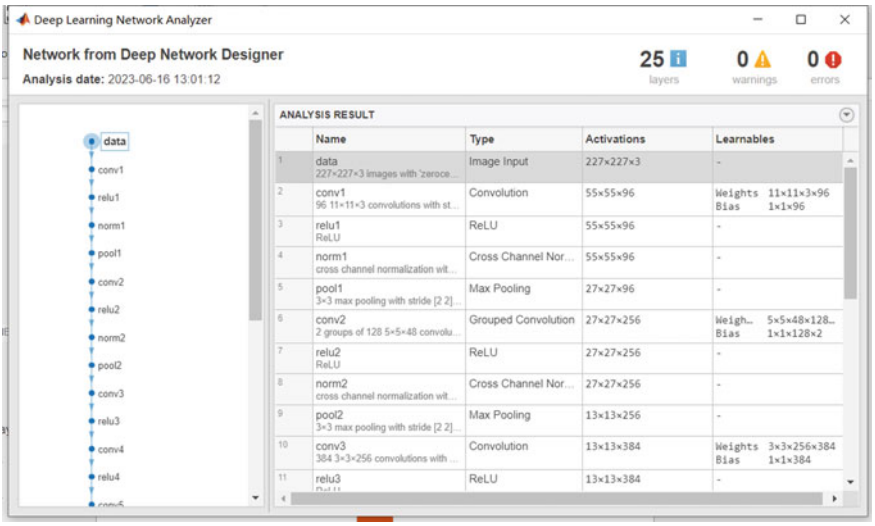

Fig. 2.9 Analysis of the AlexNet network

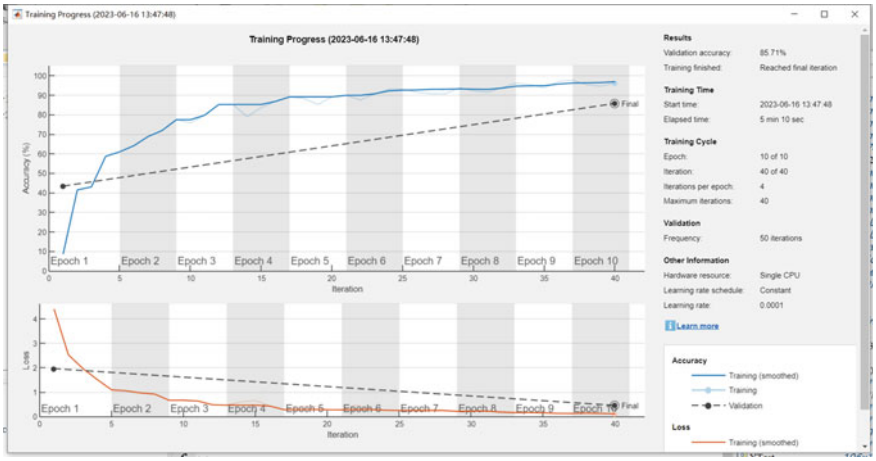

Fig. 2.10 Training process of the AlexNet on food example image dataset

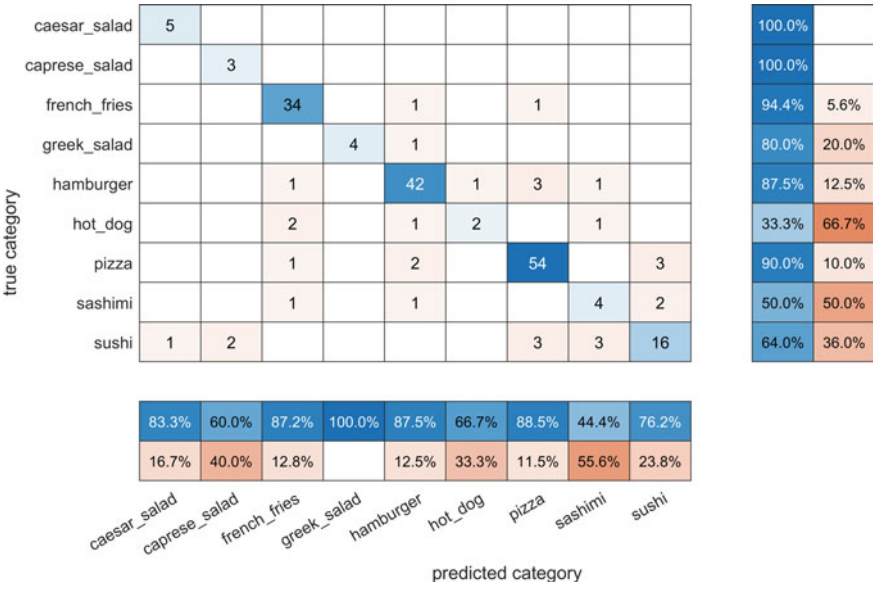

Fig. 2.11 Confusion matrix of the AlexNet on food example image dataset

44.4% and the recall rate of the AlexNet is only 50.0%. The precision rate of the AlexNet in the "hot_dog" category is 66.7%, and the recall in the "hot_dog" category is only 33.3%.

SqueezeNet is a convolutional neural network proposed by Iandola and other scholars [3]. After simulation experiments, SqueezeNet achieves the same accuracy rate as AlexNet, but SqueezeNet has only one-fiftieth of the number of parameters of

AlexNet. SqueezeNet also opens up a new research direction in the field of artificial intelligence, which is to maximize the computational speed without decreasing the accuracy of the model. A pre-trained version from the ImageNet dataset, which is based on more than 1 million images, has been saved in Matlab. The pre-trained network can classify images into 1000 object classes, such as keyboard, mouse, pencil and many animals. Thus, SqueezeNet has learned a rich feature representation of a wide range of images, and its network input image size is 227 × 227.

We use the food example image dataset as the research problem. The programs to solve the problem using pre-trained SqueezeNet are as follows:

```
dataDir = fullfile("ExampleFoodImageDataset");
url = "https://www.mathworks.com/supportfiles/nnet/data/ExampleFoodImageDataset.zip";
if ~ exist(dataDir, "dir").
   mkdir(dataDir);
   downloadExampleFoodImagesData(url,dataDir);
end
imds = imageDatastore('ExampleFoodImageDataset', …
   'IncludeSubfolders',true,'LabelSource','foldernames');
labelCount = countEachLabel(imds);
img1 = readimage(imds,1);
size(img1)
[imdsTest,imdsTrain] = splitEachLabel(imds,0.2,'randomize');
[imdsValid,imdsTrain] = splitEachLabel(imdsTrain,0.25,'randomize');
labelCountTest = countEachLabel(imdsTest);
labelCountTrain = countEachLabel(imdsTrain);
labelCountValid = countEachLabel(imdsValid);
numTrainImages = numel(imdsTrain.Labels);
idx = randperm(numTrainImages,9);
I = imtile(imds, 'Frames', idx);
figure;
imshow(I);
net = squeezenet;
inputSize = net.Layers(1).InputSize;
analyzeNetwork(net);
lgraph = layerGraph(net);
numClasses = numel(categories(imdsTrain.Labels));
newConvLayer = convolution2dLayer ([1,1],numClasses,…
   'WeightLearnRateFactor',10,'BiasLearnRateFactor',…
   10,"Name",'new_conv');
lgraph = replaceLayer(lgraph,'conv10',newConvLayer);
newClassificatonLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,…
   'ClassificationLayer_predictions',newClassificatonLayer);
aug = imageDataAugmenter("RandXReflection", true, …
```

```
    "RandYReflection", true, …
    "RandXScale", [0.8 1.2], …
    "RandYScale", [0.8 1.2]);
augImdsTrain = augmentedImageDatastore(inputSize(1:2), imdsTrain, …
    'DataAugmentation', aug);
augImdsVal = augmentedImageDatastore(inputSize(1:2), imdsValid);
opts = trainingOptions("adam", …
    "InitialLearnRate", 1e-4, …
    "MaxEpochs", 10, …
    "ValidationData", augImdsVal, …
    "Verbose", false,…
    "Plots", "training-progress", …
    "ExecutionEnvironment","cpu",…
    "MiniBatchSize",128);
netTransfer = trainNetwork(augImdsTrain, lgraph, opts);
save('Food_SqueezeNet.mat','netTransfer', …
    'imdsTrain','imdsValid','imdsTest');
augImdsTest = augmentedImageDatastore(inputSize(1:2), imdsTest);
[YPred, YPredScores] = classify(netTransfer, augImdsTest);
YTest = imdsTest.Labels;
M = confusionmat(YTest, YPred);
ACC = sum(diag(M)) / sum(M(:));
P1 = diag(M)./(sum(M,1) + 0.0001)';
R1 = diag(M)./(sum(M,2) + 0.0001);
P = mean(P1); % mean precision of all classes
R = mean(R1); % mean recall of all classes
F1score = 2*P*R/(P + R);
fig = figure;
cm = confusionchart(YTest,YPred,'RowSummary',…
    'row-normalized','ColumnSummary','column-normalized');
CLASSES = unique(YTest);
for i1 = 1:length(CLASSES)
    % compute AUC for Class i.
    [XRF,YRF,TRF,AUCRF(i1)] = perfcurve(YTest,…
        YPredScores(:, i1),CLASSES(i1));
end
AUC = mean(AUCRF);
```

The results of the above program after running are shown in Figs. 2.12, 2.13 and 2.14. Figure 2.12 gives the structural analysis of the SqueezeNet. The SqueezeNet network model contains about one million two hundred thousand parameters to be learned. It should be noted that the figure shows that the SqueezeNet structure consists of 68 layers, which is because in Matlab, each step from input to output is considered as one layer, so the number of layers shown in the figure is larger than the 18 layers introduced earlier.

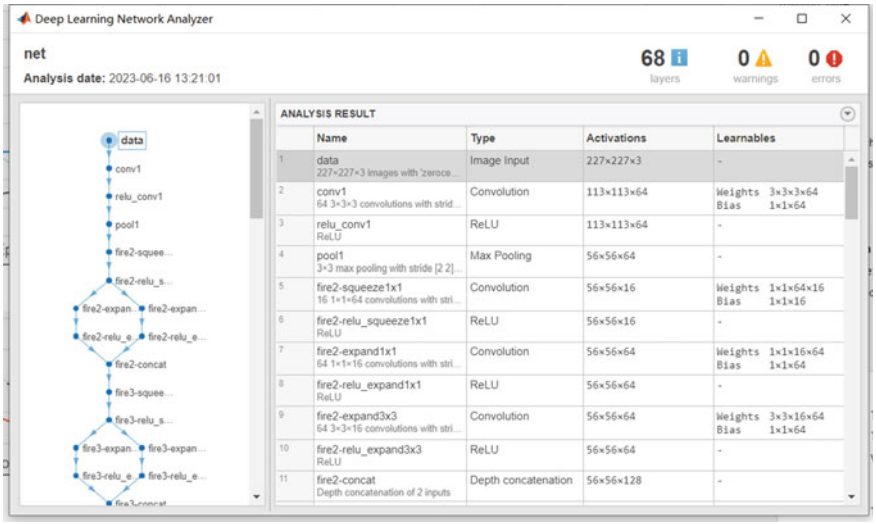

Fig. 2.12 Analysis of the SqueezeNet network

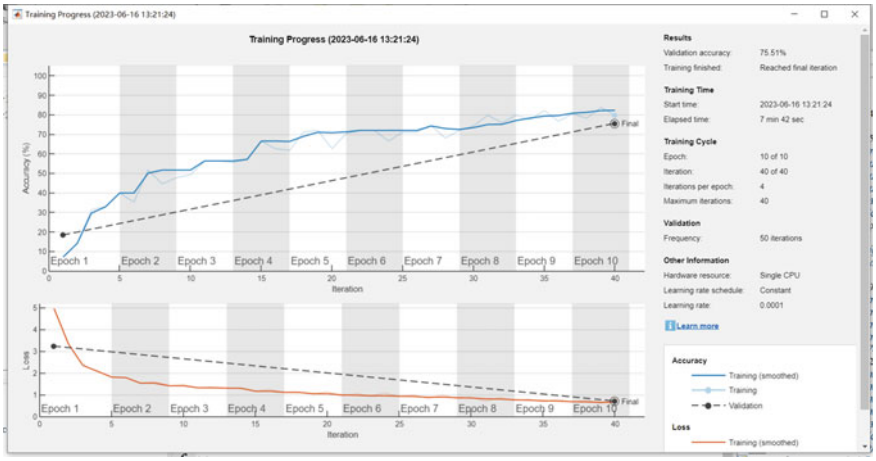

Fig. 2.13 Training process of the SqueezeNet on food example image dataset

The training process of this model is more time consuming if the model is not pre-trained. Figure 2.13 shows the training process of the SqueezeNet. It can be seen that the model was retrained on a CPU computer in less than eight minutes based on the pre-trained model and with an accuracy of 75.51% on the validation set. The accuracy of the SqueezeNet on the test set is 76.02%. Thus, the SqueezeNet is able to solve the food example image dataset.

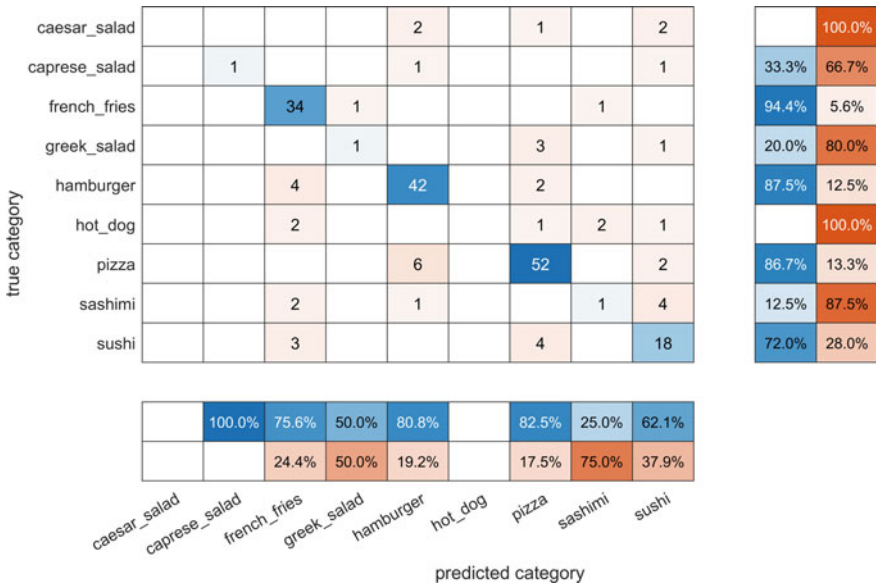

Fig. 2.14 Confusion matrix of the SqueezeNet on food example image dataset

Figure 2.14 shows the confusion matrix of the SqueezeNet. It can be seen that the model's precisioin rate of SqueezeNet does not exceed 50.0% on categories of "greek_salad" and "sashimi". Moreover, the SqueezeNet predicts incorrectly on all samples of the "caesar_salad" category, resulting in an precision rate that cannot be calculated. Except "french_fries", "greek_salad", "sashimi" and "sushi" categories, the recall rates of the SqueezeNet are all less than 33.3%.

Finally, the comparison of the AlexNet and the SqueezeNet performance on the food example image dataset is shown in Table 2.3.

Table 2.3 gives the values of the five metrics *ACC*, *P*, *R*, *F*1-score and AUC introduced in the previous section, where *F*1-score is abbreviated as *F*1. it can be seen that the AlexNet outperforms the SqueezeNet in all metrics. This result is consistent with the experimental results of recent researchers. The total number of parameters of the SqueezeNet is reduced by about 50 times. Although the accuracy of the SqueezeNet is lower than that of the AlexNet, the reduction in the total number of parameters makes the SqueezeNet applicable to real-time image classification problems. Moreover, the SqueezeNet can be laid out in small chips such as field programmable gate array (FPGA).

Table 2.3 Performance of AlexNet and SqueezeNet on food example image dataset

Network	*ACC* (%)	*P* (%)	*R* (%)	*F*1 (%)	AUC (%)
AlexNet	83.67	77.09	77.70	77.39	97.21
SqueezeNet	76.02	52.88	45.16	48.72	94.30

Except MINST and ImageNet dataset, readers can use other datasets. For example, corona virus disease 2019 (COVID-19) is a worldwide outbreak of an infectious disease. There are publicly available datasets on the classification problem of COVID-19 [4]. Readers could easily use AlexNet or SqueezeNet to solve COVID-19 dataset.

2.4 Research Progress of Neural Network

This section reviews the state-of-the-art progress of neural network research. These researches are classified to four categories. They are deep neural network, convolutional neural network, graph neural network and other network model.

(1) Deep neural network

Considering that the application of deep neural network (DNN) in modeling tabular data is still challenging. Borisov et al. outline the most advanced deep learning methods for tabular data [5]. Specifically, they are divided into three groups: data transformation, dedicated architecture, and regularization model. The authors do in-depth study of deep learning methods for tabular data. This work can serve as a valuable starting point to guide researchers and practitioners interested in deep learning of tabular data.

Due to ill-posed problems, gradient disappearance or explosion problems, saddle point problems and other reasons, DNN often have problems of poor performance or even training failure. Liu et al. proposed a new method of applying gradient activation function to gradient to solve these problems [6]. Intuitively, GAF enlarges the small gradient and limits the large gradient. The conditions that GAF needs to meet are given theoretically, and it is proved that GAF alleviates the above problems. In addition, under certain assumptions, it is proved that the convergence speed of SGD with GAF is faster than that of SGD without GAF. The experimental results also show that this method can be applied to various deep neural networks (DNNs) to improve their performance.

Considering that DNNs require high computational time, people always expect to achieve better performance with lower computational complexity. Therefore, the human sense system was studied and a neural network (SpinalNet) was designed to achieve higher accuracy with less computation [7]. The hidden layer in the traditional neural network receives the input of the previous layer, applies the activation function, and then passes the result to the next layer. In SpinalNet, each layer is divided into three parts: input part, middle part and output part. The input part of each layer receives part of the input. The middle part of each layer receives the output of the middle part of the previous layer and the output of the input part of the current layer. Compared with the traditional DNN, the number of input weights is significantly reduced. SpinalNet can also be used as a fully connected layer or classification layer of DNN to support transfer learning.

Aiming at the problem that there are few studies on the interpretation of the physical meaning of CNN structure in the existing literature, Li et al. proposed a new wavelet-driven DNN, called wavelet kernel net (WKN), in which a continuous wavelet convolution (CWConv) layer is designed to replace the first convolution layer of standard CNN [8]. This allows the first CWConv layer to find more meaningful filters. In addition, in the CWConv layer, only the scale parameters and translation parameters are learned directly from the original data. This provides a very effective method to obtain a customized filter bank dedicated to extracting the defect-related impact component embedded in the vibration signal. The experimental results show that the importance of the CWConv layer and the output of the CWConv layer are interpretable. Compared with standard CNN, WKN has fewer parameters, higher fault classification accuracy and faster convergence speed.

In view of the lack of high-quality labels in many practical scenarios, Song et al. first described the learning problem with label noise from the perspective of supervised learning [9]. Next, the authors make a comprehensive review of 62 state-of-the-art robust training methods. All these methods are divided into five groups according to their method differences, and then the six attributes used to evaluate their superiority are systematically compared. The authors put forward several promising research directions, which can be used as guidance for future research.

Elbrächter et al. developed the basic limitations of DNN by describing what may occur without imposing constraints on learning algorithms and training data volume [10]. Specifically, they consider Kolmogorov optimal approximation through DNNs. The guiding theme is the relationship between the complexity of the function to be approximated and the complexity of the approximation network in terms of connectivity and memory requirements for storage network topology and related quantization weights. The theory establishes Kolmogorov optimal approximations of DNNs that are significantly different function classes, such as the unit ball in Besov space and modulation space. The authors also prove that in the approximation of a sufficiently smooth function, a finite-width deep network requires less connectivity than a finite-width wide network.

Apostolidis et al. mainly conducted a comprehensive survey of the existing general video summarization methods based on deep learning [11]. They formulate a video summarization task and discuss the main features of a typical deep learning-based analysis pipeline. The authors report the objective evaluation protocol of video summarization algorithm, and compare the performance of several methods based on deep learning. Based on the results of these comparisons, as well as some literature considerations on the amount of annotated data and the applicability of evaluation schemes, the authors point out potential future research directions.

In view of the fact that there is no comprehensive investigation focusing on the advantages and limitations of using neural evolution methods in DNN, and preventing DNN researchers from using neural evolution methods in their own research, Galván et al. conducted a comprehensive investigation, discussion and evaluation of the latest work on DNN architecture configuration and training using evolutionary algorithms (EAs) [12]. On this basis, this paper focuses on the most relevant problems and

challenges in current neuroevolution research, and points out several promising future research directions.

(2) Convolutional neural network

Among them, Li et al. consider that the existing literature review mainly focuses on the application of CNN in different application scenarios, and does not consider CNN as a whole, nor does it cover some novel ideas recently proposed [13]. They aim to provide new ideas and prospects for this rapidly developing field as much as possible. It involves not only two-dimensional convolution, but also one-dimensional and multi-dimensional convolution. They introduce the classical and advanced CNN models, especially the key points that make them reach the most advanced results. Through experimental analysis, some conclusions are drawn and empirical rules are provided for functional selection. The applications of one-dimensional, two-dimensional and multi-dimensional convolutions are introduced. Finally, some open problems and development directions of CNN are discussed to provide guidance for future work.

With the increasing number of remote sensing data obtained from satellites, the simultaneous processing and analysis of multi-modal remote sensing data pose new challenges to remote sensing researchers. To this end, Wu et al. proposed a new framework for multi-modal remote sensing data classification based on deep learning [14]. CNN is used as the backbone and the advanced cross-channel reconstruction module CCR-Net is used. A large number of experiments on two multi-modal remote sensing datasets, including hyperspectral (HS) and light detection and ranging (LiDAR) data, and synthetic aperture radar (SAR) data are carried out. Compared with several state-of-the-art multi-modal remote sensing data classification methods, the effectiveness and superiority of CCR-Net are proved.

In order to effectively learn features from small training data, an additional sparsity cost is added to the cost function of CNN to modify it. Kumar et al. proposed a new triangular cross entropy function to calculate the sparsity cost [15]. The proposed cost function introduces sparsity by avoiding unnecessary neuron activation in the CNN hidden layer. At the same time, in order to identify bearing defects from small training samples, the transfer learning application based on novel CNN (NCNN) is as follows: First, the original vibration signal and the envelope signal of the source domain machine are obtained. Then, these envelope signals are applied to NCNN. It is used to learn features from large training data released from the source domain. After feature learning, the knowledge obtained from NCNN is transferred to the small training samples in the target domain to fine-tune the NCNN. Then, the test data of the target domain is applied to the fine-tuned NCNN for defect recognition. The experimental results verify that the proposed cross-entropy function introduces sparsity in CNN, thereby creating an effective deep learning that can even work when the training data is insufficient.

(3) Graph neural network

Based on non-Euclidean data type graph neural network (GNN), Bessadok et al. provides a clever way to learn the structure of depth maps, thereby improving the

performance of various neuroscience tasks [16]. The authors review the current GNN-based methods, emphasizing the ways they are used in several brain map-related applications. They also draw a path for the better application of GNN in the diagnosis of neurological diseases.

Grattarola et al. introduced Spectral, an open-source Python library for constructing graph neural networks using TensorFlow and Keras application programming interfaces (API) [17]. Spectral implements a large number of methods for deep learning on graphs, including message passing and pool operators, as well as utilities for processing graphs and loading popular benchmark datasets. The purpose of this library is to provide the building blocks necessary to create a GNN. Therefore, Spectral is very suitable for beginners and deep learning experts.

Aiming at the problem that there is no unified processing method and a standard evaluation benchmark and test platform for the current GNN interpretability methods, Yuan et al. provides a unified and classified view [18]. First, the unified and classified processing of GNN interpretation methods reveal the commonalities and differences of existing methods. At the same time, in order to facilitate the evaluation, the authors provide a test platform for GNN interpretability. A comprehensive experiment is conducted to compare and analyze the performance of many methods.

In recent years, graph neural networks (GNNs) have received extensive attention due to their excellent performance. Although GNNs are very popular and dynamic network models have proved its benefits, few people pay attention to the application of GNNs in dynamic networks. In order to solve the challenges brought by this research across different fields and investigating dynamic neural networks, Skarding et al. establishes a dynamic network foundation with consistent and detailed terms and symbols. A comprehensive survey of dynamic neural network models was conducted using the proposed terms [19].

Aiming at the problem that it is difficult to explain the validity of the GNN model due to the complex nonlinear transformation in the iterative process, Huang et al. proposed a nonlinear feature selection method GraphLIME model [20]. It is a general GNN model interpretation framework, which learns the nonlinear interpretable model locally in the subgraph of the interpreted node. The GraphLIME generates a nonlinear interpretable model from its n-hop neighborhood, and then uses HSIC Lasso to calculate K most representative features as an explanation for its prediction. It is found that GraphLIME has a very high degree of interpretation and is more descriptive than the existing interpretation methods.

Bianchi et al. proposed a new graph convolutional layer inspired by the autoregressive moving average (ARMA) filter [21]. Compared with the polynomial layer, it provides a more flexible frequency response. At the same time, the authors propose a GNN implementation of ARMA filter based on recursive and distributed formulas, and obtain an efficient training convolution layer. Spectrum analysis is conducted to study the filtering effect of the proposed ARMA layer. The results show that the ARMA layer has significant improvement compared with the GNN based on polynomial filter.

Schnake e al. shows that GNN can actually be explained naturally using higher-order expansions. In fact, it turns out that such an explanation can be extracted using

a nested attribution scheme [22]. The output is a set of walks into the input graph related to the prediction. The new interpretation method represented by GNN-LRP is suitable for a wide range of GNNs. The method also extracts practically relevant insights in sentiment analysis of text data and image classification.

Aiming at the problem that the tasks of GNNs on unclassified graphs usually require non-local aggregation, and local aggregation is harmful to some unclassified graphs, Liu et al. proposes a simple and effective non-local aggregation framework [23]. The framework has efficient GNN attention guided sorting. At the same time, the authors develop various non-local GNNs, and conducts thorough experiments to analyze the non-classified graph data set and evaluate non-local GNNs. The experimental results show that nonlocal GNNs outperform the state-of-the-art methods in terms of model performance and efficiency on seven graph benchmark datasets.

GNN is an information processing architecture for processing the signals supported on the graph. They are here a generalization of CNN, where a single layer contains a graph convolutional filter library, rather than a classical convolutional filter library. The filter is composed of point-to-nonlinear and stacked in layers. Ruiz et al. study that the GNN structure is equivalent to the permutation and is stable to the deformation of the graph [24]. These characteristics help to explain the good performance of GNN. It can be observed empirically that if the graph converges to a limit object, then GNN converges to the corresponding limit object. This convergence proves the transferability of GNN between networks with different number of nodes.

Xie et al. provide a unified review of different methods for training GNN using SSL [25]. Specifically, the authors divide the SSL method into a comparison model and a prediction model. The unified processing of SSL methods for GNN reveals the similarities and differences of various methods, which lays a foundation for the development of new methods and algorithms. The authors summarize the different SSL settings and the corresponding datasets used for each setting. A standardized test platform is developed for SSL in GNN, including the implementation of general baseline methods, datasets, and evaluation indicators.

In order to solve the problem of optimal power allocation in single-hop ad hoc wireless networks, Chowdhury et al. propose a neural network architecture inspired by the iterative weighted least mean square error algorithm expansion, which is called UWMMSE [26]. The GNN is used to parameterize the learnable weights in UWMMSE, where the time-varying underlying graph is given by the fading interference coefficient in the wireless network. The experimental results show that the method has robustness to hyperparameter selection and generalization to unknown scenarios.

Aiming at the intrusion detection based on GNN in the Internet of Things (IoT) system with limited budget, Zhou et al. propose a new hierarchical adversarial attack (HAA) generation method, and realize a hierarchical-aware black box adversarial attack strategy [27]. By constructing the shadow GNN model, an intelligent mechanism based on saliency mapping technology is designed to generate adversarial examples by effectively identifying and modifying key feature elements under minimum perturbation. Considering the overall loss changes of nodes in the IoT network, a

hierarchical node selection algorithm based on random walk restart is proposed to select a set of vulnerable and high-priority nodes. The comparison results show that the classification accuracy of the two most advanced GNN models is reduced by more than 30% in the IoT environment.

Liu et al. discuss the use of GNNs and expert knowledge for intelligent contract vulnerability detection methods [28]. Specifically, the authors convert the rich control flow and data flow semantics of the source code into a contract graph. In order to highlight the key nodes in the graph, the node elimination phase to standardize the graph is designed. A new time message propagation network to extract graph features from the normalized graph is designed. The experimental results show that the accuracy of the proposed method is significantly improved compared with the existing methods in three types of vulnerabilities.

Each layer of the neural network uses a graph as a parameterization to reduce the number of parameters and computational complexity to capture node-level details. According to this principle, Isufi et al. propose a unified half framework of the most advanced GNN through the EdgeNet concept [29]. EdgeNet is a GNN architecture that allows different nodes to use different parameters to weigh the information of different neighbors. By extrapolating this strategy to more iterations between adjacent nodes, EdgeNet learns the weights of edge and neighbor dependencies to capture local details. This is a general linear and local operation that a node can perform, and contains all existing graph convolutional neural networks (GCNNs) and graph attention networks (GATs) under a formula.

Chen et al. propose a new few short learning hierarchical GNN (HGNN) [30]. The network consists of bottom-up reasoning, top-down reasoning and final node classification. For bottom-up reasoning, the authors design an intra-class k-nearest neighbor pool and an inter-class layer to learn intra-class and inter-class nodes hierarchically. For top-down reasoning, the authors use the graphical pool layer to restore the down sampled graph to the original size. For the final node classification, the authors propose a skip connection to fuse multi-level features. The parameters of HGNN are learned through the scenario training of node loss signals. The experimental results on benchmark data show that HGNN is significantly superior to other state-of-the-art GNN-based methods for both transduced and non-transduced few short learning tasks.

(4) Other network model

Han et al. classify dynamic neural networks to three categories [31]:

(i) dynamic neural networks using data-related architectures or parameters to process the sample intelligent dynamic model of each sample;
(ii) Spatial intelligent dynamic network for adaptive computing of different spatial locations of image data;
(iii) the temporal intelligent dynamic model that performs adaptive reasoning on sequence data.

Zhang et al. review the interpretability of neural networks [32]. The importance of interpretability is elaborated in detail, and a new classification method is

proposed. The classification method is organized along three dimensions: participation type, interpretation type and focus. This classification provides a meaningful three-dimensional view of the distribution of papers from the relevant literature. The existing interpretability evaluation methods are also summarized.

The purpose of meta-learning is to improve the learning algorithm itself, taking into account the experience of multiple learning. Hospedales et al. describe the landscape of contemporary meta-learning. Firstly, the definition of meta-learning is discussed and it is positioned in related fields, such as transfer learning and hyper-parameter optimization [33]. Then a new classification method is proposed, which provides a more comprehensive subdivision of meta-learning methods. At the same time, the authors also summarize some promising applications and successful cases of meta-learning, including small probability learning, reinforcement learning and architecture search.

Considering that because deep learning methods operate like black boxes, the uncertainty associated with their predictions is often difficult to quantify. Bayesian statistics provide a form to understand and quantify the uncertainty associated with deep neural network prediction. Jospin et al. provide an overview of relevant literature and a complete set of tools for deep learning practitioners to design, implement and evaluate Bayesian neural networks [34].

To further improve the interpretability of DNNs, Fan et al. propose a simple and comprehensive classification of neural network interpretability [35]. The authors systematically review the research on improving the interpretability of neural networks in recent years, describe the application of interpretability in medicine, and discuss the possible future research directions of interpretability.

For the predictive learning of spatio-temporal sequences, Wang et al. proposed a new recurrent network PredRNN [36]. In PredRNN model, a pair of memory units show decoupling and operate in an almost independent transition mode. Specifically, the PredRNN is characterized by a serrated memory flow that propagates in a bottom-up and top-down direction at all layers, allowing the visual dynamics learned by recurrent neural network (RNN) at different levels to communicate. It also uses memory decoupling loss to prevent memory units from learning redundant features. The PredRNN has obtained competitive results on five datasets of predictive learning scenarios.

In order to solve the reconstruction problem in untrained neural networks to accelerate magnetic resonance imaging (MRI), Darestani et al. proposed a highly optimized untrained recovery method based on Deep Decoder variants [37]. The author showed that it was significantly superior to other untrained methods. At the same time, the performance of the training method under ideal settings is compared, where the training and test data are from the same distribution. The authors find that the untrained algorithm achieves similar performance to the baseline trained neural network, but the most advanced training network is superior to the untrained network.

Gurrola-Ramos et al. proposed a residual dense neural network (RDUNet) based on densely connected hierarchical network for image denoising [38]. The encoding and decoding layers of RDUNet are composed of closely connected convolutional layers to reuse feature mapping and local residual learning, so as to avoid the problem

of gradient disappearance and accelerate the learning process. In addition, the model uses a global residual learning method, which no longer directly predicts the denoised image, but predicts the residual noise of the damaged image. An advantage of RDUNet is that the denoising process does not require prior knowledge about the noise level. Experiments show that RDUNet obtains competitive results compared with the most advanced image denoising network.

Aiming at the problem that the label information in DNN is not given and a data point cannot be assigned to a given cluster, Kauffmann et al. proposed a new framework that can explain the clustering assignment of input features in an effective and reliable way for the first time [39]. It is based on the novel idea that clustering models can be rewritten as neural networks. Several cases demonstrate that the method can evaluate the quality of learning clustering and extract new insights from the analyzed data and representations.

For single-phase shunt active power filter, a fractional order sliding mode control scheme based on two hidden layer recurrent neural network (THLRNN) is proposed [40]. A new THLRNN structure is designed to overcome the shortcomings of traditional neural networks like low approximation accuracy. The structure contains two hidden layers, which makes the network have stronger fitting ability and is used to approximate unknown nonlinearity. The fractional order term is added to the sliding mode controller to make the sliding mode controller have larger adjustable space and better optimization space. Simulation results show that compared with the traditional neural network sliding controller, the THLRNN method has satisfactory compensation performance and robustness.

Fei et al. proposed an approximation-based adaptive fractional-order sliding mode control scheme for micro-gyroscopes [41]. Their method used a double-loop recurrent fuzzy neural network (DLRFNN) to approximate the uncertainty and disturbance of the system. The network had two feedback loops to capture the weights and output signals calculated in the previous step and used them as feedback signals for the next step. The proposed DLRFNN combined fuzzy system processing uncertain information with neural network learning from the process. The effectiveness of the DLRFNN method is verified through the simulation analysis.

Exercises

(1) Try to analyze the working of convolutional layer and pooling layer in convolutional neural network and give examples.
(2) Try to analyze the difference between convolutional neural network and fully connected backpropagation neural network.
(3) Novel Corona Virus Disease 2019 (COVID-19) is a worldwide outbreak of an infectious disease. There are publicly available datasets on the classification problem of COVID-19. For example, COVID-CT is a CT image dataset (https://github.com/UCSD-AI4H/COVID-CT) that includes 349 images with COVID-19 and 463 images without COVID-19. Please use Matlab to create a convolutional neural network to solve the classification problem and analyze its performance.

(4) Please download the COVID-19 dataset and solve the problem by transfer learning using pre-trained neural network models in Matlab, e.g., AlexNet and SqueezeNet, and compare the performance of the different models.

References

1. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
2. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. Commun ACM 60(6):84–90
3. Iandola FN, Han S, Moskewicz MW et al (2016) SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5 MB model size. arXiv:1602.07360
4. Zhao J, Zhang Y, He X, Xie P (2020) COVID-CT-Dataset: a CT scan dataset about COVID-19. arXiv:2003.13865
5. Borisov V, Leemann T, SeBler K et al (2022) Deep neural networks and tabular data: a survey. IEEE Trans Neural Netw Learn Syst
6. Liu M, Chen L, Du X et al (2022) Activated gradients for deep neural network. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2022.3229161
7. Kabir HMD, Abdar M, Khosravi A et al (2022) Spinalnet: deep neural network with gradual input. IEEE Trans Artif Intell. https://doi.org/10.1109/TAI.2022.3185179
8. Li T, Zhao Z, Sun C et al (2021) WaveletKernelNet: an interpretable deep neural network for industrial intelligent diagnosis. IEEE Trans Syst, Man, Cybern: Syst 52(4):2302–2312
9. Song H, Kim M, Park D et al (2022) Learning from noisy labels with deep neural networks: A survey. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2022.3152527
10. Elbrächter D, Perekrestenko D, Grohs P et al (2021) Deep neural network approximation theory. IEEE Trans Inf Theory 67(5):2581–2623
11. Apostolidis E, Adamantidou E, Metsai AI et al (2021) Video summarization using deep neural networks: a survey. Proc IEEE 109(11):1838–1863
12. Galván E, Mooney P (2021) Neuroevolution in deep neural networks: current trends and future challenges. IEEE Trans Artif Intell 2(6):476–493
13. Li Z, Liu F, Yang W et al (2021) A survey of convolutional neural networks: analysis, applications, and prospects. IEEE Trans Neural Netw Learn Syst 33(12):6999–7019
14. Wu X, Hong D, Chanussot J (2021) Convolutional neural networks for multimodal remote sensing data classification. IEEE Trans Geosci Remote Sens 60:1–10
15. Kumar A, Vashishtha G, Gandhi CP et al (2021) Novel convolutional neural network (NCNN) for the diagnosis of bearing defects in rotary machinery. IEEE Trans Instrum Meas 70:1–10
16. Bessadok A, Mahjoub MA, Rekik I (2022) Graph neural networks in network neuroscience. IEEE Trans Pattern Anal Mach Intell 45(5):5833–5848
17. Grattarola D, Alippi C (2021) Graph neural networks in tensorflow and keras with spektral [application notes]. IEEE Comput Intell Mag 16(1):99–106
18. Yuan H, Yu H, Gui S et al (2023) Explainability in graph neural networks: a taxonomic survey. IEEE Trans Pattern Anal Mach Intell 45(5):5782–5799
19. Skarding J, Gabrys B, Musial K (2021) Foundations and modeling of dynamic networks using dynamic graph neural networks: a survey. IEEE Access 9:79143–79168
20. Huang Q, Yamada M, Tian Y et al (2023) Graphlime: local interpretable model explanations for graph neural networks. IEEE Trans Knowl Data Eng 35(7):6968–6972
21. Bianchi FM, Grattarola D, Livi L et al (2021) Graph neural networks with convolutional ARMA filters. IEEE Trans Pattern Anal Mach Intell 44(7):3496–3507
22. Schnake T, Eberle O, Lederer J et al (2021) Higher-order explanations of graph neural networks via relevant walks. IEEE Trans Pattern Anal Mach Intell 44(11):7581–7596

23. Liu M, Wang Z, Ji S (2021) Non-local graph neural networks. IEEE Trans Pattern Anal Mach Intell 44(12):10270–10276
24. Ruiz L, Gama F, Ribeiro A (2021) Graph neural networks: architectures, stability, and transferability. Proc IEEE 109(5):660–682
25. Xie Y, Xu Z, Zhang J et al (2022) Self-supervised learning of graph neural networks: a unified review. IEEE Trans Pattern Anal Mach Intell 45(2):2412–2429
26. Chowdhury A, Verma G, Rao C et al (2021) Unfolding WMMSE using graph neural networks for efficient power allocation. IEEE Trans Wireless Commun 20(9):6004–6017
27. Zhou X, Liang W, Li W et al (2021) Hierarchical adversarial attacks against graph-neural-network-based IoT network intrusion detection system. IEEE Internet Things J 9(12):9310–9319
28. Liu Z, Qian P, Wang X et al (2023) Combining graph neural networks with expert knowledge for smart contract vulnerability detection. IEEE Trans Knowl Data Eng 35(2):1296–1310
29. Isufi E, Gama F, Ribeiro A (2021) EdgeNets: edge varying graph neural networks. IEEE Trans Pattern Anal Mach Intell 44(11):7457–7473
30. Chen C, Li K, Wei W et al (2021) Hierarchical graph neural networks for few-shot learning. IEEE Trans Circuits Syst Video Technol 32(1):240–252
31. Han Y, Huang G, Song S et al (2021) Dynamic neural networks: a survey. IEEE Trans Pattern Anal Mach Intell 44(11):7436–7456
32. Zhang Y, Tiňo P, Leonardis A et al (2021) A survey on neural network interpretability. IEEE Trans Emerg Top Comput Intell 5(5):726–742
33. Hospedales T, Antoniou A, Micaelli P et al (2021) Meta-learning in neural networks: A survey. IEEE Trans Pattern Anal Mach Intell 44(9):5149–5169
34. Jospin LV, Laga H, Boussaid F et al (2022) Hands-on Bayesian neural networks—a tutorial for deep learning users. IEEE Comput Intell Mag 17(2):29–48
35. Fan FL, Xiong J, Li M et al (2021) On interpretability of artificial neural networks: a survey. IEEE Trans Radiat Plasma Med Sci 5(6):741–760
36. Wang Y, Wu H, Zhang J et al (2022) PredRNN: a recurrent neural network for spatiotemporal predictive learning. IEEE Trans Pattern Anal Mach Intell 45(2):2208–2225
37. Darestani MZ, Heckel R (2021) Accelerated MRI with un-trained neural networks. IEEE Trans Comput Imaging 7:724–733
38. Gurrola-Ramos J, Dalmau O, Alarcón TE (2021) A residual dense u-net neural network for image denoising. IEEE Access 9:31742–31754
39. Kauffmann J, Esders M, Ruff L et al (2022) From clustering to cluster explanations via neural networks. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2022.3185901
40. Fei J, Wang H, Fang Y (2021) Novel neural network fractional-order sliding-mode control with application to active power filter. IEEE Trans Syst, Man, Cybern: Syst 52(6):3508–3518
41. Fei J, Wang Z, Liang X et al (2021) Fractional sliding-mode control for microgyroscope based on multilayer recurrent fuzzy neural network. IEEE Trans Fuzzy Syst 30(6):1712–1721

Chapter 3
Fuzzy Computing

Abstract Fuzzy computing is a computational intelligence technology based on fuzzy theory. Fuzzy computing can solve various problems such as identification and clustering. Automatic control problems remain the main application area of fuzzy computing. This chapter first introduces the basis of fuzzy computing including fuzzy set and fuzzy membership function. Fuzzy pattern recognition, fuzzy clustering and fuzzy inference are three kinds of problems that fuzzy computing can solve. Then this chapter introduces Mamdani fuzzy control system, which is one of the most important application of fuzzy computing. Finally, this chapter introduces fuzzy logic designer to build fuzzy controller for a control problem.

3.1 Overview of Fuzzy Computing

Fuzzy computing is a computational intelligence technology based on fuzzy set theory, fuzzy linguistic variables, fuzzy inference. Fuzzy computing is also known as Fuzzy logic. In 1965, Zadeh created fuzzy set theory and developed fuzzy logic control theory [1]. In 1974, Mamdani applied fuzzy logic control to boiler and steam engine control and formed fuzzy logic controller. This pioneering work marked the birth of fuzzy logic cybernetics [2].

The phenomenon of "fuzzy" is present in human society and is an important feature of human perception, logical thinking and reasoning decisions [3]. For example, the answer to the question of whether someone is a student or not is definite, i.e., either he is a student or not. For the question of whether a girl is a beauty, the answer is uncertain, i.e., she may be a beauty, she may not be a beauty, or she may be somewhere in between. Through this example, we know that "fuzzy" has more information than "definite" and is more in line with the objective world reality.

In natural human language, there is a large number of concepts that have a rich connotation without an absolute standard of measurement. People in their lives often use words such as "big", "small", "more", "less", "probably" and "probably" [3]. For example, in the column of health status, you can only fill in "good, healthy", but it is difficult to specify exactly what kind of body is "good" and what kind of body is "healthy". In fact, health status is a comprehensive qualitative assessment, and there

X. Zhang et al., *Intelligent Information Processing with Matlab*,
https://doi.org/10.1007/978-981-99-6449-9_3

are often personal subjective judgments. For example, we often hear that someone is "competent", "smart", "beautiful", which are the most common words in natural language, and are very vague concepts.

How do people use fuzziness in their lives to deal with things? For example, when a person goes to the market to buy tomatoes, he knows that the buying guideline is: "The redder the tomato, the riper it is", which is equivalent to fuzzy knowledge or fuzzy rules. When he sees a very red tomato, he immediately assumes that the tomato is very ripe; while when he sees a tomato that is not red, he assumes that the tomato may not be very ripe according to the buying guideline This is fuzzy inference.

Even some definite characteristics that are strictly defined are sometimes described by fuzzy concepts in order to grasp the main characteristics of things from a macro perspective and to facilitate processing. For example, people are divided into "young", "middle-aged" and "old" according to their age. According to height, people are divided into "tall", "medium" and "short". By weight, people are divided into "fat" and "thin", and by speed, people are divided into "fast" and "slow". These examples show that fuzzy phenomena exist in human society in large numbers, and people often use these fuzzy concepts to deal with problems in daily life.

According to the previous discussion, applying conventional mathematical methods to the analysis of problems that are inherently fuzzy is incongruous and it will cause a large gap between theory and practice. This gave rise to the birth of fuzzy mathematics. The birth of fuzzy mathematics was marked by Zadeh's article entitled Fuzzy Sets published in the journal Information and Control in 1965. Zadeh proposed the use of member functions to describe intermediate transitions in the differences of fuzzy phenomena, thus breaking away from the deterministic belong-or-don't-belong relationship of classical set theory.

After that, Mandani proposed fuzzy logic control theory in 1974 and achieved better results than traditional numerical control methods. In 1980, Holmblad and Ostergard in Denmark applied fuzzy logic control theory to the cement kiln problem, which was the first commercially meaningful application of fuzzy logic control theory. In recent years, fuzzy logic cybernetics has been successfully applied to problems such as intelligent drum washing machines, intelligent balancing cars, and automatic driving of cars.

Fuzzy computing can solve problems such as identification and clustering in addition to fuzzy logic cybernetics, but control problems remain the main application area of fuzzy computing.

3.2 Fuzzy Sets

At the end of the nineteenth century, Cantor in Germany founded the theory of sets. The set theory was then gradually integrated into all branches of mathematics and became part of the basic mathematical theory [1–3].

Definition 3.1 Putting together certain things that can be clearly identified with each other is called a set. These things in a set can be either direct objects or abstract objects of thought.

Cantor defined sets as in Definition 3.1. Here, we refer to the set defined by Cantor as classical set, or ordinary set. The theoretical domain is an important concept in ordinary set theory, which refers to the fact that when discussing the extension of a concept or considering the topic of a problem, a scope of discussion is always delineated, and this scope is called the theoretical domain. The domain is often represented by capital letters in italics, e.g., U, E. Each object in the domain is called an element and is often represented by a symbol such as a lowercase letter in italics, e.g., a, b, x, y. In a domain, the whole set of objects with a particular property constitutes a set in the domain. Sets are often represented by italicized uppercase letters, e.g., A, B, C, or X, Y, Z.

We briefly introduce some classical set operations, as follows:

(1) The inclusion relation of a set. For any $x \in A$, there must be $x \in B$. We will say that the set B contains the set A, or the set A is contained in the set B, and it is written as $A \subseteq B$.
(2) Empty set. If for any set A, there is $\emptyset \subseteq A$, then $\emptyset$ is said to be the empty set of any set A. That is an empty set is a set that does not contain any elements.
(3) Power set. Let U be a theoretical domain, and the set consisting of all subsets of U is called the power set of U, denoted as $P(U)$. For example, suppose $U = \{a, b, c\}$, then $P(U) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$.
(4) Union set. The union of a set A and a set B is defined as $A \cup B = \{x | x \in A \, or \, x \in B\}$.
(5) Intersection set. The intersection of a set A and a set B is defined as $A \cap B = \{x | x \in A \, and \, x \in B\}$.
(6) Difference set. The difference set of a set A and a set B is defined as $AB = \{x | x \in A \, and \, x \notin B\}$.
(7) Complementary set. Let U be a theoretical domain and the complement of A to U is defined as $A^c = U - A = \{x | x \in U \, or \, x \notin A\}$.
(8) Equivalence. The set A and the set B are equal noted as $A = B$, i.e., we have $A \subseteq B$ and $B \subseteq A$.

To represent a common set, one can use the enumeration method, the descriptive method and the characteristic function method as follows:

(1) Enumeration method. The enumeration method is to list all the elements in a set, so it generally indicates a finite set. For example, $S =$ {Primary student, Secondary student, University student, Graduate student}, where the set S lists all the elements with the nature of "student".
(2) Descriptive method. The descriptive method is the representation of elements with certain properties by means of language. For example, $\{A = x | x < 0, \, and \, x \, is \, a \, real \, number\}$.
(3) Characteristic function method. The characteristic function method is to represent the elements by means of expressions. For example:

$$F_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \tag{3.1}$$

If there are more than one ordinary set, they can also perform operations on each other and there are some laws as follows:

(1) Exchange law. $A \cup B = B \cup A$, $A \cap B = B \cap A$.
(2) Combination law. $A \cup (B \cup C) = (A \cup B) \cup C$, $A \cap (B \cap C) = (A \cap B) \cap C$.
(3) Absorption rate. $(A \cup B) \cap A = A$, $(A \cap B) \cup A = A$.
(4) Idempotence law. $A \cup A = A$, $A \cap A = A$.
(5) Distributive law. $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$, $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$.
(6) Law of restitution. $(A^c)^c = A$.
(7) Complementary law. $A \cup A^c = U$, $A \cap A^c = \Phi$.
(8) 0–1 law. $A \cup U = U$, $A \cup \Phi = A$, $A \cap U = A$,$A \cap \Phi = \Phi$.
(9) Inversion law, also known as De Morgan's law. $A \cup B^c = A^c \cap B^c$, $A \cap B^c = A^c \cup B^c$.

Based on the concept of classical sets, the concept of fuzzy sets is introduced next.

Definition 3.2 Given a theoretical domain U, any map μ_A from U to the closed interval [0, 1] has the expression:

$$\mu_A : U \to [0, 1] \tag{3.2}$$

The above expression determines a fuzzy set A of U, μ_A is called the member function of the fuzzy set A.

The membership function μ_A reflects the degree to which the elements in a fuzzy set A belong to that set. If the elements in U are represented by x, then $\mu_A(x)$ is called the degree of membership of the element x belonging to the fuzzy set A. From Eq. (3.2), it can be seen that $\mu_A(x)$ takes values in the closed interval [0, 1]. If $\mu_A(x)$ is close to 0, it means that the degree of x belonging to A is low; conversely, if $\mu_A(x)$ is close to 1, it means that the degree of x belonging to A is high.

If a fuzzy set is to be represented, the corresponding method can be used according to the specifics of the theoretical domain. When the theoretical domain $U = \{x_1, x_2, \ldots, x_n\}$ is a discrete finite set, the methods usually used are the Zadeh representation, the ordered pair representation and the vector representation.

(1) Zadeh representation method. The element x_i in the theoretical domain U is represented with its membership function $\mu_A(x_i)$ by the following equation:

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \cdots + \frac{\mu_A(x_n)}{x_n} \tag{3.3}$$

where A is the fuzzy set and the fraction in the expression is just a form and not a division operation.

Example 3.1 Suppose there is a domain of integers 1, 2, …, 10, i.e., the domain $U = \{1, 2, \ldots, 10\}$, the fuzzy set "several" is denoted by A. And let the membership function of each element of μ_A be $\{0, 0.1, 0.3, 0.7, 1, 1, 0.7, 0.3, 0.1, 0\}$ in order.

Solution According to (3.3), the fuzzy set A can be expressed as

$$A = \frac{0}{1} + \frac{0.1}{2} + \frac{0.3}{3} + \frac{0.7}{4} + \frac{1}{5} + \frac{1}{6} + \frac{0.7}{7} + \frac{0.3}{8} + \frac{0.1}{9} + \frac{0}{10} \tag{3.4}$$

$$= \frac{0.1}{2} + \frac{0.3}{3} + \frac{0.7}{4} + \frac{1}{5} + \frac{1}{6} + \frac{0.7}{7} + \frac{0.3}{8} + \frac{0.1}{9} \tag{3.5}$$

The part where the membership is zero can be left out.

(2) ordered pair representation method. Representing A by forming an ordered pair of an element x_i in the theoretical domain with its membership $\mu_A(x_i)$, then:

$$A = \{(x_1, \mu_A(x_1)), (x_2, \mu_A(x_2)), \ldots, (x_N, \mu_A(x_N)) | x \in U\} \tag{3.6}$$

In the ordered pair representation method, terms with zero membership can be omitted. For example, for the above example, the ordered pair representation can be written as:

$$A = \{(1, 0), (2, 0.1), (3, 0.3), (4, 0.7), (5, 1), (6, 1), (7, 0.7), (8, 0.3), (9, 0.1), (10, 0)\} \tag{3.7}$$

$$= \{(2, 0.1), (3, 0.3), (4, 0.7), (5, 1), (6, 1), (7, 0.7), (8, 0.3), (9, 0.1)\} \tag{3.8}$$

(3) vector representation method. Representing A by forming a vector with the membership $\mu_A(x_i)$ of the element x_i in the theoretical domain, i.e., we have

$$A = [\mu_A(x_1)\mu_A(x_2) \ldots \mu_A(x_N)] \tag{3.9}$$

In the vector representation method, terms with zero membership cannot be omitted. For example, for the above example, the vector representation can be written as $A =$ [0 0.1 0.3 0.7 1 1 0.7 0.3 0.1 0].

When the theoretical domain U is a continuous finite set, the Zadeh representation is generally used to represent the fuzzy set, and its expression is:

$$A = \int_U \frac{\mu_A(x)}{x} \tag{3.10}$$

where U is the theoretical domain, x is an element in the domain, and $\mu_A(x)$ is the membership of x. In the Zadeh representation of a finite continuous domain, the part of the membership that is zero can be left out.

For example, if we take age as the domain and set $U = [0, 200]$, let Y denote the fuzzy set "young" and O denote the fuzzy set "old". The membership function of "young" is known as $\mu_Y(x)$, and its expression is:

$$\mu_Y(x) = \begin{cases} 1, & 0 \le x \le 25 \\ 1 + \left(\frac{x-25}{5}\right)^2, & 25 < x \le 200 \end{cases} \tag{3.11}$$

The subordinate function of "old" is $\mu_O(x)$, and its expression is:

$$\mu_O(x) \begin{cases} 0, & 0 \le x \le 25 \\ 1 + \left(\frac{5}{x-50}\right)^2, & 50 < x \le 200 \end{cases} \tag{3.12}$$

Then, using the Zadeh representation method, the "young" fuzzy set Y can be expressed as:

$$Y = \{(x, 1)|0 \le x \le 25\} + \left\{ \left(x, \left[1 + \left(\frac{x-25}{5} \right)^2 \right]^{-1} \right) \middle| 25 < x \le 200 \right\} \tag{3.13}$$

After finishing the above equation, we get:

$$Y = \int_{0 \le x \le 25} \frac{1}{x} + \int_{25 < x \le 200} \frac{\left[1 + \left(\frac{x-25}{5} \right)^2 \right]^{-1}}{X} \tag{3.14}$$

Using the Zadeh representation method, the fuzzy set O of "old" can be expressed as:

$$O = \{(x, 0)|0 \le x \le 50\} + \left\{ \left(x, \left[1 + \left(\frac{5}{x-50} \right)^2 \right]^{-1} \right) \middle| 50 < x \le 200 \right\} \tag{3.15}$$

After finishing the above equation, we get:

$$O = \int_{50 < x \le 200} \frac{\left[1 + \left(\frac{5}{x-50} \right)^2 \right]^{-1}}{x} \tag{3.16}$$

It should be noted that the integral notation in the Zadeh representation method does not mean the integral in calculus, and the notation is simply used here for the representation of fuzzy set.

From the concept of fuzzy set, the membership function plays a very important role. After the membership function is determined, the membership of each element is determined, and then the fuzzy set is determined. Generally speaking, the determination of the membership function is both objectivity and subjectivity. Objectivity means that the membership function is essentially a description of something, so it should be objective; subjectivity means that each person does not have the same understanding of fuzzy concepts, so the subjectivity function should have people's subjective perception. Therefore, the subjectivity function is usually given by experts or authorities in the field, and it expresses the experience accumulated in practice, or the knowledge derived from data statistics.

There are three general methods to determine the subjectivity function, which are: the fuzzy statistics method, the objective scale method and the assignment method.

Fuzzy statistics is a method based on data statistics to obtain a membership function, which requires designing and distributing a questionnaire to form a membership function from the data in the questionnaire.

Example 3.2 Suppose we want to define the membership function of "young" with the domain $U = [0,100]$, i.e. we discuss which age is young between 0 and 100 years. Suppose that 136 experts have empirically given the age range of "young", as shown in Table 3.1. In this table, the age range given by each expert is represented as a closed interval. Suppose we want to count the degree to which a certain age μ_0 is "young", and count the number of times these closed intervals cover μ_0, i.e., the frequency.

For Table 3.1, it is easy to see that the minimum age is 14 years and the maximum age is 36 years. For the integer ages located in the interval [14, 36], we can count their occurrences in Table 3.1, and the results are shown in column 2 of Table 3.2.

In Table 3.2, column 1 is age from 14 to 36, column 2 is the frequency per age, and column 3 is the relative frequency. The value in column 3 is each number in column 2 divided by the largest frequency in column 2. The final relative frequency of each age is obtained as a number that lies between 0 and 1. We can represent the young membership function graphically with the following programs:

```
load('IIP3_1.mat');
agemin = min(data(:));
agemax = max(data(:));
agefreq = zeros(agemax-agemin+1, 1);
idx = 1;
for i1 = agemin:agemax
   for i2 = 1:size(data, 1)
      if data(i2, 1) <= i1 && i1 <= data(i2, 2)
         agefreq(idx,1) = agefreq(idx,1) + 1;
      end
```

Table 3.1 Age range of "young" given by experts

[18, 25]	[17, 30]	[17, 28]	[18, 25]	[16, 35]	[14, 25]	[18, 30]	[18, 35]
[18, 35]	[16, 25]	[15, 30]	[18, 35]	[17, 30]	[18, 25]	[18, 35]	[20, 30]
[18, 30]	[16, 30]	[20, 35]	[18, 30]	[18, 25]	[18, 35]	[15, 25]	[18, 30]
[15, 28]	[16, 28]	[18, 30]	[18, 30]	[16, 30]	[18, 35]	[18, 25]	[18, 30]
[16, 28]	[18, 30]	[16, 30]	[16, 28]	[18, 35]	[18, 35]	[17, 27]	[16, 28]
[15, 28]	[18, 25]	[19, 28]	[15, 30]	[15, 26]	[17, 25]	[15, 36]	[18, 30]
[17, 30]	[18, 35]	[16, 35]	[16, 30]	[15, 25]	[18, 28]	[16, 30]	[15, 28]
[18, 35]	[18, 30]	[17, 28]	[18, 35]	[15, 28]	[15, 25]	[15, 25]	[15, 25]
[18, 30]	[16, 24]	[15, 25]	[16, 32]	[15, 27]	[18, 35]	[16, 25]	[18, 30]
[16, 28]	[18, 30]	[18, 35]	[18, 30]	[18, 30]	[18, 30]	[17, 30]	[18, 30]
[18, 35]	[16, 30]	[18, 28]	[17, 25]	[15, 30]	[18, 25]	[17, 30]	[14, 25]
[18, 26]	[18, 29]	[18, 35]	[18, 28]	[18, 35]	[18, 25]	[16, 35]	[17, 29]
[18, 25]	[17, 30]	[16, 28]	[18, 30]	[16, 28]	[15, 30]	[18, 30]	[16, 30]
[20, 30]	[20, 30]	[16, 25]	[17, 30]	[15, 30]	[18, 30]	[16, 30]	[18, 28]
[15, 35]	[16, 30]	[15, 30]	[18, 35]	[18, 35]	[18, 30]	[17, 30]	[16, 35]
[17, 30]	[15, 25]	[18, 35]	[15, 30]	[15, 25]	[15, 30]	[18, 30]	[17, 25]
[18, 29]	[18, 28]	[18, 35]	[18, 25]	[18, 30]	[15, 30]	[17, 30]	[18, 30]

```
    end
    idx = idx + 1;
end
ageprob = agefreq / max(agefreq);
figure1 = figure(1);
axes1 = axes('Parent',figure1);
hold(axes1,'on');
plot((agemin:agemax), ageprob', 'b', 'LineWidth', 2);
plot((agemin:agemax), ageprob', 'bo','LineWidth', 2,'LineStyle','none');
ylabel('membership function');
xlabel('age');
box(axes1,'on');
grid(axes1,'on');
hold(axes1,'off');
set(axes1,'FontSize',14);
```

The results shown in Fig. 3.1 were obtained after running the above program. From Fig. 3.1, it can be seen that most experts consider 20–25 years old as the "young", while the probability of belonging to the "young" after 31 years old decreases rapidly.

The objective scaling method for describing the membership function is based on empirical knowledge of human. For example, if the theoretical domain is "household" and the fuzzy set is "well-off family", then whether a family element belongs to "well-off family" can be measured using Engel's Coefficient to measure whether a

Table 3.2 Frequency statistics of age

Age	Frequency	Relative frequency
14	2	0.0147
15	27	0.1985
16	52	0.3824
17	69	0.5074
18	131	0.9632
19	132	0.9706
20	136	1
21	136	1
22	136	1
23	136	1
24	136	1
25	135	0.9926
26	109	0.8015
27	107	0.7868
28	105	0.7721
29	86	0.6324
30	83	0.6103
31	28	0.2059
32	28	0.2059
33	27	0.1985
34	27	0.1985
35	27	0.1985
36	1	0.0074

specific family element is a "well-off family". For example, if the theoretical domain is "equipment" and the fuzzy set is "normal", then whether an equipment element is "normal", we can use the equipment intactness rate to measure.

The assignment method is to use a certain distribution matching the nature of the problem to be described as the membership function, and the method is highly subjective. When using the real number field R as the theoretical domain, i.e., $U = R$, the membership function is generally made a fuzzy distribution. To solve the problem, we can choose from common fuzzy distributions, such as rectangular fuzzy distribution, trapezoidal fuzzy distribution, k-th parabolic fuzzy distribution, normal fuzzy distribution, Cauchy fuzzy distribution, S-type fuzzy distribution, etc.

The expression for the rectangular fuzzy distribution is:

$$\mu_A(x) = \begin{cases} 0, \ x < a \ or \ x > b \\ 1, \ a \leq x \leq b \end{cases} \tag{3.17}$$

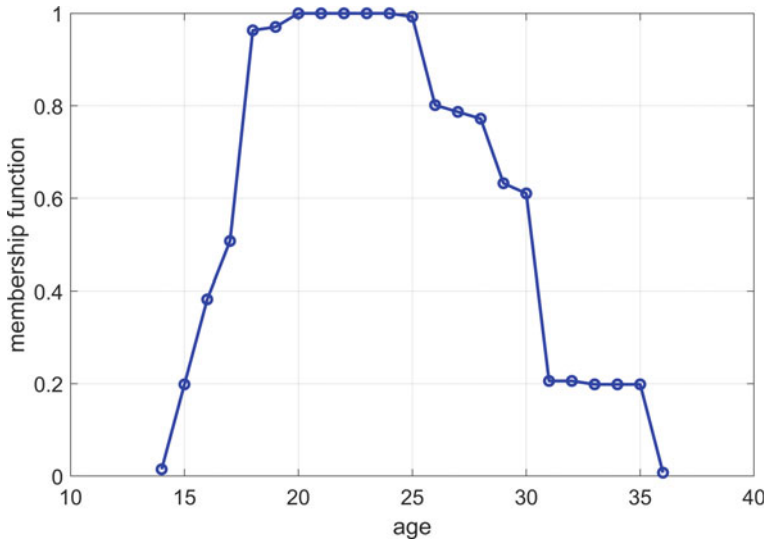

Fig. 3.1 Membership function of "young"

In the above equation a, b are two parameters with $a < b$. When the element x lies between a and b, the affiliation is 1; otherwise, it is 0. In addition, a one-sided distribution can be defined as follows:

$$\mu_A(x) = \begin{cases} 1, & x \leq a \\ 0, & x > a \end{cases} \tag{3.18}$$

$$\mu_A(x) = \begin{cases} 0, & x < a \\ 1, & x \geq a \end{cases} \tag{3.19}$$

In (3.18), the membership is 1 when the element x lies to the left of a; while on the right side is the degree of affiliation 0. In (3.19), the membership is 0 when the element x lies to the left of a; while on the right side is the degree of affiliation 1. It is easy to see that (3.17) gives the case where x takes the value 1 in the middle of the domain, while (3.18) and (3.19) give the case where x takes the value 1 on one side of the domain is 1. Therefore, the case where x takes a value of 1 in the middle of the domain is generally referred to as the intermediate fuzzy distribution, the case where x takes a value of 1 on the left side of the domain is referred to as the small fuzzy distribution, and the case where x takes a value of 1 on the right side of the domain is referred to as the large fuzzy distribution.

Similarly, other fuzzy distributions can be subdivided into small, intermediate and large fuzzy distributions. The expressions for the small, intermediate and large fuzzy distributions of the trapezoidal fuzzy distribution are:

$$\mu_A(x) = \begin{cases} 1, & x < a \\ \frac{b-x}{b-a}, & a \le x \le b \\ 0, & x > b \end{cases} \tag{3.20}$$

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \le x < b \\ 1, & b \le x < c \\ \frac{d-x}{d-c}, & c \le x < d \\ 0, & x \ge d \end{cases} \tag{3.21}$$

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \le x \le b \\ 1, & x > b \end{cases} \tag{3.22}$$

where a, b, c, d are the four parameters and satisfy $a < b < c < d$.

The expressions for the small, intermediate and large fuzzy distributions of the k-th parabolic fuzzy distribution are:

$$\mu_A(x) = \begin{cases} 1, & x < a \\ \left(\frac{b-x}{b-a}\right)^k, & a \le x \le b \\ 0, & x > b \end{cases} \tag{3.23}$$

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \left(\frac{x-a}{b-a}\right)^k, & a \le x < b \\ 1, & b \le x < c \\ \left(\frac{d-x}{d-c}\right)^k, & c \le x < d \\ 0, & x \ge d \end{cases} \tag{3.24}$$

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \left(\frac{x-a}{b-a}\right)^k, & a \le x \le b \\ 1, & x > b \end{cases} \tag{3.25}$$

where k, a, b are three parameters and satisfy $a < b$ and $k > 0$.

The expressions for the small, intermediate and large fuzzy distributions of the normal fuzzy distribution are:

$$\mu_A(x) = \begin{cases} 1, & x \le a \\ e^{-((x-a)/\sigma)^2}, & x > a \end{cases} \tag{3.26}$$

$$\mu_A(x) = e^{-((x-a)/\sigma)^2} \tag{3.27}$$

$$\mu_A(x) = \begin{cases} 0, & x < a \\ 1 - e^{-((x-a)/\sigma)^2}, & x \ge a \end{cases} \tag{3.28}$$

where a and $\sigma > 0$ are the two parameters.

The expressions for the small, intermediate and large fuzzy distributions of the Cauchy fuzzy distribution are:

$$\mu_A(x) = \begin{cases} 1, & x < a \\ \frac{1}{1+\alpha(x-\alpha)^{\beta}}, & x \geq a \end{cases} \tag{3.29}$$

$$\mu_A(x) = \frac{1}{1+\alpha(x-\alpha)^{\beta}} \tag{3.30}$$

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \frac{1}{1+\alpha(x-\alpha)^{-\beta}}, & x \geq a \end{cases} \tag{3.31}$$

where a, α, β are three parameters and satisfy $\alpha > 0$ and $\beta > 0$.

S in the S-type fuzzy distribution refers to the Sigmoid function, which has been studied in Chap. 1, and its expressions for the small, intermediate and large fuzzy distributions are:

$$\mu_A(x) = 1 - \frac{1}{1+e^{-a(x-b)}} \tag{3.32}$$

$$\mu_A(x) = \left| \frac{1}{1+e^{-a(x-b)}} - \frac{1}{1+e^{-c(x-d)}} \right| \tag{3.33}$$

$$\mu_A(x) = \frac{1}{1+e^{-a(x-b)}} \tag{3.34}$$

where a, b, c, d are the four parameters. In Matlab, the S-shaped fuzzy distribution can be plotted using the following programs:

```
x = 0:0.1:10;
y1 = 1-sigmf(x,[5 2]);
y2 = dsigmf(x,[5 2 5 7]);
y3 = sigmf(x,[5 7]);
figure1= figure(1);
axes1 = axes('Parent',figure1);
hold(axes1,'on');
plot(x, y1, '-', 'LineWidth', 2);
plot(x, y2, '--', 'LineWidth', 2);
plot(x, y3, ':', 'LineWidth', 2);
ylabel('membership');
xlabel('x');
box(axes1,'on');
grid(axes1,'on');
hold(axes1,'off');
set(axes1,'FontSize',14);
```

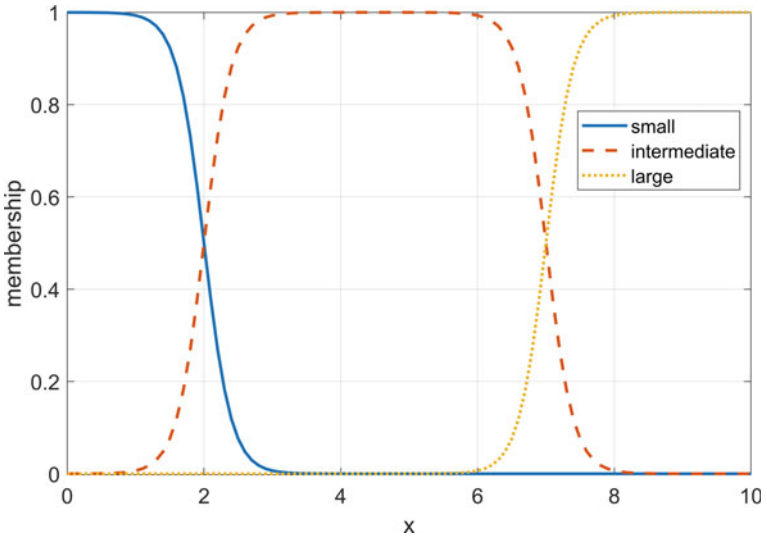

Fig. 3.2 S-type membership function

After running the above program, the small, intermediate and large S-type fuzzy distributions are obtained as shown in Fig. 3.2.

The fuzzy distributions of the above membership functions all contain parameters, and the determination of these parameters is the problem to be solved by the fuzzy computing. In practical applications, we can test the effect of the membership function in practice and make adjustments in order to obtain a more exact membership function. The following tips should be noted in the application:

(1) the fuzzy set of the membership function must be a convex fuzzy set. For example: "moderate speed" of the membership function, it is from the largest membership point to both sides of the extension, the value of its membership must be monotonically decreasing, but not allowed to have undulating waviness, thus, generally the membership function of triangles and trapezoids is more suitable.
(2) membership function is generally symmetric and balanced. For example: "moderate speed" of the membership function, if one side to take "high speed", then generally the other side to take "low speed", to meet the symmetry property.
(3) The membership function should conform to the semantic order of people and avoid inappropriate expressions.

The operation of ordinary sets is described by the characteristic function; while the membership function is a generalization of the characteristic function, thus, the operation of fuzzy sets can be described by the membership function.

(1) Equality of fuzzy sets. If there are two fuzzy sets A and B with $\mu_A(x) = \mu_B(x)$ for all $x \in U$, then the fuzzy set A is said to be equal to the fuzzy set B, denoted as $A = B$.

(2) The inclusion relation of fuzzy sets. If there are two fuzzy sets A and B with $\mu_A(x)\mu_B(x)$ for all $x \in U$, then the fuzzy set A is said to be contained in the fuzzy set B, or A is a subset of B, denoted as $A \subseteq B$.
(3) Fuzzy empty set. A fuzzy set A is said to be empty if $\mu_A(x) = 0$ for all $x \in U$, and is denoted as $A = \emptyset$.
(4) Union of fuzzy sets. The union of fuzzy sets is also known as the maximal operator of fuzzy sets, or the sum operator of fuzzy sets. If there are three fuzzy sets A, B and C, for all $x \in U$, all have:

$$\mu_C(x) = \mu_A(x) \vee \mu_B(x) = \max[\mu_A(x), \mu_B(x)] \tag{3.35}$$

Then the fuzzy set C is said to be the union of A and B, denoted as $C = A \cup B$. In addition, the union of fuzzy sets can also be defined as:

$$\mu_C(x) = \mu_A(x) \vee \mu_B(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x) \tag{3.36}$$

Both definitions of (3.35) and (3.36) are feasible, but the first form is more commonly used.

(5) Intersection of fuzzy sets. The intersection of fuzzy sets is also known as the minimum algorithm of fuzzy sets, or the product operator of fuzzy sets. If there are three fuzzy sets A, B and C, for all $x \in U$, all have:

$$\mu_C(x) = \mu_A(x) \wedge \mu_B(x) = \min[\mu_A(x), \mu_B(x)] \tag{3.37}$$

Then the fuzzy set C is said to be the intersection of A and B, denoted as $C = A \cap B$. Furthermore, the intersection of fuzzy sets can be defined as:

$$\mu_C(x) = \mu_A(x) \wedge \mu_B(x) = \mu_A(x)\mu_B(x) \tag{3.38}$$

Both definitions of (3.37) and (3.38) are feasible, but the first form is more commonly used.

(6) Complement of a fuzzy set. If there are two fuzzy sets A and B with $\mu_B(x) = 1 - \mu_A(x)$ for all $x \in U$, then B is said to be the complement of A, and is denoted as $B = A^c$.

Take Gaussian membership function for example, the programs for union, intersection and complement of fuzzy sets are as follows:

```
x = 0:0.1:10;
A = gaussmf(x,[1 4]);
B = gaussmf(x,[1 6]);
AORB = max([A;B],[], 1);
AANDB = min([A;B],[], 1);
Ac = 1 - A;
figure1 = figure;
subplot1 = subplot(2,2,1,'Parent',figure1);
```

```
hold(subplot1,'on');
plot1 = plot(x,[A;B],'Parent',subplot1,'LineWidth',2);
set(plot1(1),'DisplayName','A');
set(plot1(2),'DisplayName','B','LineStyle','—');
ylabel('membership');
xlabel('x');
box(subplot1,'on');
hold(subplot1,'off');
set(subplot1,'XGrid','on','XTick',[0 2 4 6 8 10],'YGrid','on');
legend1 = legend(subplot1,'show');
set(legend1,'Position',[0.36 0.840 0.094 0.073]);
subplot2 = subplot(2,2,2,'Parent',figure1);
hold(subplot2,'on');
plot(x,AORB,'DisplayName','union of A and B','Parent',subplot2,'LineWidth',2);
ylabel('membership');
xlabel('x');
box(subplot2,'on');
hold(subplot2,'off');
set(subplot2,'XGrid','on','XTick',[0 2 4 6 8 10],'YGrid','on');
legend2 = legend(subplot2,'show');
set(legend2,'Position',[0.774 0.855 0.125 0.040]);
subplot3 = subplot(2,2,3,'Parent',figure1);
hold(subplot3,'on');
plot(x,AANDB,'DisplayName','intersection of A and
B','Parent',subplot3,'LineWidth',2);
ylabel('membership');
xlabel('x');
ylim(subplot3,[0 1]);
box(subplot3,'on');
hold(subplot3,'off');
set(subplot3,'XGrid','on','XTick',[0 2 4 6 8 10],'YGrid','on');
legend3 = legend(subplot3,'show');
set(legend3,'Position',[0.327 0.398 0.125 0.040]);
subplot4 = subplot(2,2,4,'Parent',figure1);
hold(subplot4,'on');
plot(x,Ac,'DisplayName','complement of A','Parent',subplot4,'LineWidth',2);
ylabel('membership');
xlabel('x');
box(subplot4,'on');
hold(subplot4,'off');
set(subplot4,'XGrid','on','XTick',[0 2 4 6 8 10],'YGrid','on');
legend4 = legend(subplot4,'show');
set(legend4,'Position',[0.769 0.334 0.131 0.040]);
```

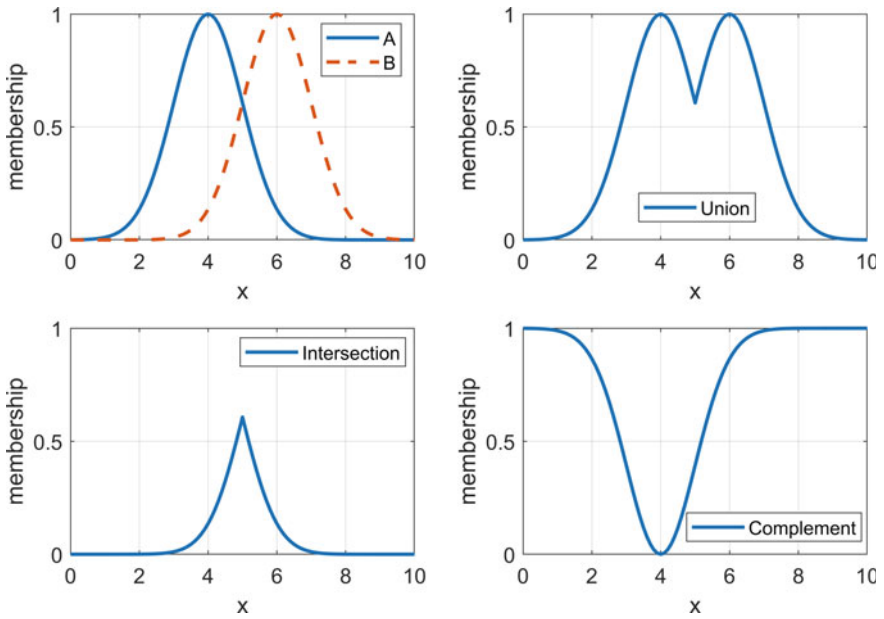

Fig. 3.3 Union, intersection and complement of fuzzy sets

After running the above program, the result is shown in Fig. 3.3. In Fig. 3.3, the upper left graph shows the membership function of fuzzy set A and B. The upper right graph shows the membership function of the union of A and B. The lower left graph shows the membership function of the intersection of A and B. The lower right graph shows the membership function of the complement of A.

Example 3.3 Suppose the theoretical domain $U = \{u_1, u_2, u_3, u_4, u_5\}$, on which there are two fuzzy sets:

$$A = \frac{0.2}{u_1} + \frac{0.7}{u_2} + \frac{1}{u_3} + \frac{0.5}{u_5} \tag{3.39}$$

$$B = \frac{0.5}{u_1} + \frac{0.3}{u_2} + \frac{0.1}{u_4} + \frac{0.7}{u_5} \tag{3.40}$$

Try to find the union and intersection of the two fuzzy sets.

Solution Then, according to the operations on fuzzy sets introduced above, we have:

$$A \cup B = \frac{0.2 \vee 0.5}{u_1} + \frac{0.7 \vee 0.3}{u_2} + \frac{1 \vee 0}{u_3} + \frac{0 \vee 0.1}{u_4} + \frac{0.5 \vee 0.7}{u_5}$$

$$= \frac{0.5}{u_1} + \frac{0.7}{u_2} + \frac{1}{u_3} + \frac{0.1}{u_4} + \frac{0.7}{u_5} \tag{3.41}$$

$$A \cap B = \frac{0.2 \wedge 0.5}{u_1} + \frac{0.7 \wedge 0.3}{u_2} + \frac{1 \wedge 0}{u_3} + \frac{0 \wedge 0.1}{u_4} + \frac{0.5 \wedge 0.7}{u_5}$$
$$= \frac{0.2}{u_1} + \frac{0.3}{u_2} + \frac{0.5}{u_5} \tag{3.42}$$

The operations on fuzzy sets satisfy the following eight properties, as follows:

(1) Power law. $A \bigcup A = A$, $A \bigcap A = A$.
(2) Exchange law. $A \bigcup B = B \bigcup A$, $A \bigcap B = B \bigcap A$.
(3) Combination law. $A \bigcup (B \bigcup C) = (A \bigcup B) \bigcup C$, $A \bigcap (B \bigcap C) = (A \bigcap B) \bigcap C$.
(4) Distributive law. $(A \bigcup B) \bigcap C = (A \bigcap C) \bigcup (B \bigcap C)$.
(5) Absorption rate. $(A \bigcup B) \bigcap A = A$, $(A \bigcap B) \bigcup A = A$.
(6) Law of restitution. $(A^c)^c = A$.
(7) Pairwise law. $A \bigcup B^c = A^c \bigcap B^c$, $A \bigcap B^c = A^c \bigcup B^c$.
(8) 0–1 law. $A \bigcup U = U$, $A \bigcup \Phi = A$, $A \bigcap U = A$, $A \bigcap \varnothing = \varnothing$.

It can be seen that the properties of fuzzy sets are formally similar to those of classical sets, but fuzzy sets do not satisfy the complementary law in classical sets, i.e., $A \bigcup A^c = U$, $A \bigcap A^c = \varnothing$. For example, suppose there is a fuzzy set $A = (0.2, 0.7)$, then its complementary set $A^c = (0.8, 0.3)$. Thus, we have $A \bigcup A^c = (0.2 \vee 0.8, 0.7 \vee 0.3) = (0.8, 0.3) \neq U$ and $A \bigcap A^c = (0.2 \wedge 0.8, 0.7 \wedge 0.3) = (0.8, 0.3) \neq \varnothing$. In particular, when the fuzzy set $A = (0.5, 0.5)$, then its complement set A^c=(1–0.5, 1–0.5) = (0.5, 0.5). This shows that there exist fuzzy sets in which their complements are equal to themselves. This is unbelievable in classical sets, but it reflects the phenomenon of "both here and there" in practice. This feature of fuzzy sets is important in fuzzy information processing, which makes the results of fuzzy information processing more consistent with the actual situation.

Fuzzy intercept set is an important concept of fuzzy set operations, which is generally called λ-cut. For example, if there are fuzzy sets:

$$\text{society} = \frac{1}{\text{Xia}} + \frac{1}{\text{Shang}} + \frac{0.9}{\text{Xizhou}} + \frac{0.7}{\text{Chunqiu}} + \frac{0.5}{\text{Zhanguo}} + \frac{0.4}{\text{Qin}} + \frac{0.3}{\text{Xihan}} + \frac{0.1}{\text{Donghan}} \tag{3.43}$$

If a level of at least 0.5 is required, there are Xia, Shang, Xizhou, Chunqiu and Zhanguo with membership of 0.5. If a level of at least 0.7 is required, there are Xia, Shang, Xizhou and Chunqiu that meet the condition.

Definition 3.3 Suppose U is a theoretical domain with fuzzy set $A \in F(U)$, $\lambda \in [0, 1]$, we call $A_\lambda = \{x | A(x) \geq \lambda\}$ the λ-cut set of A; while $A_\lambda = \{x | A(x) > \lambda\}$ is

.

called the strong λ-cut set of A. Note that there is a small dot directly below λ in the definition of the strong cut set.

Obviously, a strong cut set is a subset of a cut set, i.e., $A_{\dot{\lambda}} \subseteq A_{\lambda}$. In addition, the λ in the cut set is also called the threshold or confidence level.

Example 3.4 Suppose there is fuzzy set:

$$A = \frac{0.5}{u_1} + \frac{0.6}{u_2} + \frac{1}{u_3} + \frac{0.7}{u_4} + \frac{0.3}{u_5} \tag{3.44}$$

Take λ=1, λ=0.7 and λ=0.3, respectively, and try to find their λ-cut sets.

Solution When λ=1, only the membership of u_3 reaches 1, so $A_1 = \{u_3\}$. When λ=0.7, only the membership of u_3 and u_4 reaches 0.7, so $A_{0.7} = \{u_3, u_4\}$. When λ=0.3, there are u_1, u_2, u_3, u_4 and u_5 whose membership reaches 0.3, so $A_{0.3} = \{u_1, u_2, u_3, u_4, u_5\}$. The fuzzy sets of these three cut sets are:

$$A_1 = \frac{1}{u_3} \tag{3.45}$$

$$A_{0.7} = \frac{1}{u_3} + \frac{1}{u_4} \tag{3.46}$$

$$A_{0.3} = \frac{1}{u_1} + \frac{1}{u_2} + \frac{1}{u_3} + \frac{1}{u_4} + \frac{1}{u_5} \tag{3.47}$$

From the Example 3.3, we can see that λ-cut set is an ordinary set, while the elements satisfying the level of λ correspond to a fuzzy set. λ-cut set provides a way to convert between fuzzy and classical sets, which is very useful when dealing with practical problems.

3.3 Fuzzy Pattern Recognition

Pattern recognition is a human perception and awareness of things. It is also a human thinking activity. The pattern recognition of human can be the recognition of concrete things such as pictures, words and language, or the recognition of abstract things such as an argument or an opinion.

Fuzzy pattern recognition is the introduction of fuzzy mathematical theory into pattern recognition, using fuzzy logic methods to classify and recognize things. There are two main types of methods for fuzzy pattern recognition, one is the direct method and the other is the indirect method. Direct methods generally use the maximum membership principle to classify categories and are usually used for individual pattern recognition. Indirect methods generally use the principle of proximity to classify categories and are usually used for pattern recognition of groups.

(1) Maximum membership principle.

Definition 3.4 Suppose there are n patterns which are represented as n fuzzy sets A_i ($i = 1, 2, \ldots, n$) on a theoretical domain U. $u_0 \in U$ is an identified object if there exists $i \in \{1, 2, \ldots, n\}$ such that

$$A_i(\mu_0) = \vee_{j=1}^{n} A_j(\mu_0) = \max\{A_1(\mu_0), A_2(\mu_0), \ldots, A_n(\mu_0)\} \tag{3.48}$$

Then u_0 is said to belong relatively to pattern A_i.

If the object being identified is definite and the pattern is ambiguous, then the maximum membership principle can be used.

Example 3.5 Suppose the theoretical domain $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ denotes the set of 6 goods whose categories are lagging goods A_1, out-of-stock goods A_2 and popular goods A_3, i.e., we have:

$$A_1 = \frac{1}{u_1} + \frac{0.1}{u_2} + \frac{0}{u_3} + \frac{0.6}{u_4} + \frac{0.5}{u_5} + \frac{0.4}{u_6} \tag{3.49}$$

$$A_2 = \frac{0}{u_1} + \frac{0.1}{u_2} + \frac{0.1}{u_3} + \frac{0}{u_4} + \frac{0}{u_5} + \frac{0.05}{u_6} \tag{3.50}$$

$$A_3 = \frac{0}{u_1} + \frac{0.8}{u_2} + \frac{1}{u_3} + \frac{0.4}{u_4} + \frac{0.4}{u_5} + \frac{0.5}{u_6} \tag{3.51}$$

Try to find the category of u_2.

Solution To determine which category u_2 belongs to, the calculation is performed according to the principle of maximum membership:

$$\vee_{j=1}^{n} A_j(\mu_2) = 0.1 \vee 0.1 \vee 0.8 = A_3(\mu_2) \tag{3.52}$$

It is known that u_2 belongs to the popular goods A_3. Similarly, to determine to which category u_5 belongs, the calculation is performed according to the principle of maximum membership:

$$\vee_{j=1}^{n} A_j(\mu_5) = 0.5 \vee 0 \vee 0.4 = A_1(\mu_5) \tag{3.53}$$

It is known that u_5 belongs to lagging goods A_1.

(2) Nearest principle.

We first introduce the concepts of inner product, outer product and nearness degree of fuzzy sets.

Definition 3.5 Suppose there are two fuzzy sets A and B on the theoretical domain U. Then:

$$A \circ B = \vee_{u \in U}(A(u) \wedge B(u)) \tag{3.54}$$

is called the inner product of A and B.

Definition 3.6 Suppose there are two fuzzy sets A and B on the theoretical domain U. Then:

$$A \otimes B = \wedge_{u \in U}(A(u) \vee B(u)) \tag{3.55}$$

is called the outer product of A and B.

When the theoretical domain $U = \{u_1, u_2, \ldots, u_n\}$, according to the above definition, we can obtain:

$$A \circ B = \vee_{i=1}^{n}(A(u_i) \wedge B(u_i)) \tag{3.56}$$

$$A \otimes B = \wedge_{i=1}^{n}(A(u_i) \vee B(u_i)) \tag{3.57}$$

The nearness degree of two fuzzy sets can measure the similarity of fuzzy sets. Commonly used nearness degree includes the lattice nearness degree, the average nearness degree and the max–min nearness degree.

The formula for lattice nearness degree is as follows:

$$N(A, B) = (A \circ B) \wedge (A \otimes B)' \tag{3.58}$$

The formula for the average nearness degree is as follows:

$$N(A, B) = \frac{1}{2}\left[(A \circ B) + (A \otimes B)'\right] \tag{3.59}$$

The formula for the max–min nearness degree is as follows:

$$N(A, B) = \frac{\sum_{i=1}^{n}(A(u_i) \wedge B(u_i))}{\sum_{i=1}^{n}(A(u_i) \vee B(u_i))} = \frac{\int_{-\infty}^{+\infty}(A(u_i) \wedge B(u_i))dx}{\int_{-\infty}^{+\infty}(A(u_i) \vee B(u_i))dx} \tag{3.60}$$

lattice nearness degree is the most common method of nearness degree.

Definition 3.7 Suppose there are n patterns which are represented as n fuzzy sets A_i ($i = 1, 2, \ldots, n$) on a theoretical domain U. Another fuzzy set B is an identified object if there exists $i \in \{1, 2, \ldots, n\}$ such that:

$$N(A_i, B) = \max\{N(A_1, B), N(A_2, B), \ldots, N(A_n, B)\} \tag{3.61}$$

Then B is said to be closest to A_i and it is decided that B belongs to mode A_i.

Example 3.6 Suppose the domain $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ represents the set of 6 commodities, and there are 5 categories of patterns A_1, A_2, A_3, A_4 and A_5 on the domain, where $A_1 = (0.6, 0.3, 0.2, 0, 0.5, 0.1)$, $A_2 = (0.7, 1, 0.3, 0, 0.8, 0.9)$, $A_3 = (0.2, 1, 0.8, 0.4, 0.5, 0.1)$, $A_4 = (0.8, 0, 0.4, 0.5, 0.7, 0)$, $A_5 = (0.5, 0.3, 0.6, 1, 0, 0.4)$, For commodity B, use the lattice nearness degree to determine the category to which B belongs, where $B = (0.7, 0.4, 0.6, 0.3, 0.4, 0.8)$.

Solution Lattice nearness degree requires to compute the inner product and outer produce of two fuzzy sets. The inner product of A_1 and B is $A_1 \circ B = \max(\min(0.6, 0.7), \min(0.3, 1), \min(0.2, 0.3), \min(0, 0), \min(0.5, 0.8), \min(0.1, 0.9)) = \max(0.6, 0.3, 0.2, 0, 0.5, 0.1) = 0.6$. Similarly, we can compute $A_2 \circ B = 0.8$, $A_3 \circ B = 0.6$, $A_4 \circ B = 0.7$, $A_5 \circ B = 0.6$.

The outer products are $A_1 \otimes B = 0.3$, $A_2 \otimes B = 0.3$, $A_3 \otimes B = 0.4$, $A_4 \otimes B = 0.4$, $A_5 \otimes B = 0.4$. Based on the equation of complement set, we have $A_1 \otimes B' = 0.6$, $A_2 \otimes B' = 0.7$, $A_3 \otimes B' = 0.6$, $A_4 \otimes B' = 0.6$, $A_5 \otimes B' = 0.6$.

Based on lattice nearness degree (3.58), we have $N(A_1, B) = (A_1 \circ B) \wedge (A_1 \otimes B)' = 0.6 \wedge 0.7 = 0.6$. Similarly, we have $N(A_2, B) = 0.7$, $N(A_3, B) = 0.6$, $N(A_4, B) = 0.6$, $N(A_5, B) = 0.6$.

We use the nearest principle and get $\max\{N(A_1, B), N(A_2, B), \ldots, N(A_n, B)\} = N(A_2, B) = 0.7$, so we can determine that the fuzzy set B belongs to the second category A_2. We can also perform the category determination with the help of Matlab, whose programs are as follows:

```
A = [0.6,0.3,0.2,0,0.5,0.1;
  0.7,1,0.3,0,0.8,0.9;
  0.2,1,0.8,0.4,0.5,0.1;
  0.8,0,0.4,0.5,0.7,0;
  0.5,0.3,0.6,1,0,0.4];
B = [0.7,0.4,0.6,0.3,0.4,0.8];
for i1 = 1:size(A, 1)
  tmp = [A(i1, :); B];
  N(i1)=min([max(min(tmp)), 1-min(max(tmp))]);
end
[Nmax, BClass] = max(N);
```

After running the above program, the variable BClass is equal to 2. The result indicates that the pattern of B is the second category A_2.

3.4 Fuzzy Clustering

This section will introduce data clustering by using fuzzy computing method. First, we introduce the direct product in classical set theory. Suppose there are two sets A and B. Then we say:

$$A \times B = \{(x, y) | x \in A, y \in B\} \tag{3.62}$$

where $A \times B$ is the direct product of A and B. For example, suppose $A = \{1, 3\}$ and $B = \{2, 4, 6\}$, then their direct product is: $A \times B = \{(1, 2), (1, 4), (1, 6), (3, 2), (3, 4), (3, 6)\}$. It can be seen that the direct product of two sets is the set of ordered pairs of the two sets.

Next, we introduce the relation of sets. The direct product of two sets is the set of all ordered pair. If we restrict the ordered pairs, we will get a subset of the direct product, which is a binary relation of two sets, or the relation of two sets for short. The relation of two sets can be represented by a matrix, generally noted as R, whose elements in the i-th row and j-th column are:

$$r_{ij} = \begin{cases} 1, & (x, y) \in R \\ 0, & (x, y) \notin R \end{cases}, \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, m \tag{3.63}$$

When the theoretical domain is a finite set, for two sets A and B, we can represent the fuzzy relation between A and B by a fuzzy matrix. Specifically: suppose A and B are two non-empty sets, a fuzzy set R in the direct product is called a binary fuzzy relation from A to B, or fuzzy relation for short, and is denoted as $R_{x \times y}$.

Suppose $A = \{x_1, x_2, \ldots, x_n,\}$, $B = \{y_1, y_2, \ldots, y_n,\}$, and $R_{x \times y}$ denotes the fuzzy relation defined on $A \times B$, then it is represented by the matrix as:

$$R_{x \times y} = \begin{bmatrix} \mu_R(x_1, y_1) & \mu_R(x_1, y_2) & \cdots & \mu_R(x_1, y_m) \\ \mu_R(x_2, y_1) & \mu_R(x_2, y_2) & \cdots & \mu_R(x_2, y_m) \\ \vdots & \vdots & \ddots & \vdots \\ \mu_R(x_n, y_1) & \mu_R(x_n, y_2) & \cdots & \mu_R(x_n, y_m) \end{bmatrix} \tag{3.64}$$

where $\mu_R(x_i, y_j)$ denotes the membership of x_i and y_j for the fuzzy relation $R_{x \times y}$, which is generally abbreviated as r_{ij} and $R_{x \times y}$ can be abbreviated as R.

Similar to the binary fuzzy relation, we can define n-element fuzzy relation, which are a subset of the direct product of n sets. Because a fuzzy relation is a fuzzy set defined on the direct product space, the operation rules of fuzzy relation are similar to the operation rules of fuzzy sets.

For binary fuzzy relation, we introduce the synthetic operation between multiple fuzzy relations. Suppose there are theoretical domains A, B, C are finite, $R_{x \times y} = (r_{ij})_{n \times m}$ is a fuzzy relation from A to B, and $S_{y \times z} = (s_{jk})_{m \times l}$ is a fuzzy relation from B to C. Then $R_{x \times y}$ and $S_{y \times z}$ can be synthesized, denoted as $T_{x \times z} = (t_{ik})_{n \times l}$:

$$T_{x \times z} = R_{x \times y} \circ S_{y \times z} \tag{3.65}$$

The above equation can be abbreviated as $T = R \circ S$. At this point, such a synthesis can be expressed in the form of a matrix with the values of the i-th row and k-th column of $T_{x \times z}$ as:

$$t_{ik} = \vee_{j=1}^{m}\left(r_{ij} \wedge s_{jk}\right) \tag{3.66}$$

The synthesis of the above equation is also called maximum-minimum (max–min) synthesis. The max–min synthesis is commonly used when introducing synthetic operations for fuzzy relations. It should be noted that, in addition to max–min synthesis, one can also use:

$$t_{ik} = \vee_{j=1}^{m}\left(r_{ij} \cdot s_{jk}\right) \tag{3.67}$$

The synthesis of the above equation is called maximum-product (max-product) synthesis.

Example 3.7 Suppose the theoretical domain is family members, A is the son and daughter in the family, B is the father and mother in the family, and the fuzzy relation R is that the children and parents look alike and have:

$$R = \begin{matrix} \\ x_1 \\ x_2 \end{matrix} \begin{matrix} \begin{matrix} y_1 & y_2 \end{matrix} \\ \begin{bmatrix} 0.8 & 0.3 \\ 0.3 & 0.6 \end{bmatrix} \end{matrix} \tag{3.68}$$

If C is the grandfather and grandmother in the family and the fuzzy relationship S is that the parents and grandparents look alike and have:

$$S = \begin{matrix} \\ y_1 \\ y_2 \end{matrix} \begin{matrix} \begin{matrix} z_1 & z_2 \end{matrix} \\ \begin{bmatrix} 0.7 & 0.5 \\ 0.1 & 0.1 \end{bmatrix} \end{matrix} \tag{3.69}$$

Try to find the fuzzy relationship matrix from children to grandparents.

Solution According to the max–min synthesis method, the fuzzy relationship matrix from children to grandparents is:

$$T = R \circ S = \begin{bmatrix} 0.8 & 0.3 \\ 0.3 & 0.6 \end{bmatrix} \circ \begin{bmatrix} 0.7 & 0.5 \\ 0.1 & 0.1 \end{bmatrix} \tag{3.70}$$

$$= \begin{bmatrix} (0.8 \wedge 0.7) \vee (0.3 \wedge 0.1) & (0.8 \wedge 0.5) \vee (0.3 \wedge 0.1) \\ (0.3 \wedge 0.7) \vee (0.6 \wedge 0.1) & (0.3 \wedge 0.5) \vee (0.6 \wedge 0.1) \end{bmatrix} \tag{3.71}$$

$$= \begin{bmatrix} 0.7 \vee 0.1 & 0.5 \vee 0.1 \\ 0.3 \vee 0.1 & 0.3 \vee 0.1 \end{bmatrix} = \begin{bmatrix} 0.7 & 0.5 \\ 0.3 & 0.3 \end{bmatrix} \tag{3.72}$$

The above three equations give the detailed calculation procedure of the max–min synthesis rule. We can also use Matlab to implement the synthetic operation of fuzzy relations with the following programs:

```
R = [0.8, 0.3; 0.3, 0.6];
S = [0.7, 0.5; 0.1, 0.1];
n = size(R, 1);
l = size(S, 2);
for i1 = 1:n
   for j1 = 1:l
      T(i1, j1) = max(min([R(i1,:); S(:, j1)']));
   end
end
```

After running the above program, the value of T is the result of the synthesis. From the above example, we can see that fuzzy relations can be synthesized to form more complex fuzzy logic operations.

"Clustering" is the process of distinguishing and classifying things according to certain requirements and laws, in which there is no a priori knowledge about classification, but only relies on the similarity between things as a criterion for class classification [5]. Thus, clustering belongs to unsupervised learning.

"Cluster analysis" is a mathematical method to study and deal with the classification of a given thing. Cluster analysis is also known as a mathematical method to classify the things studied according to certain criteria. It is a multivariate statistical "things in a class" of a classification method. Human beings to understand the world must distinguish between different things and recognize the similarity between things.

We call the binary fuzzy relation R from the set A to itself as R is a fuzzy relation on A, denoted as $R \in F(A \times A)$, which has the following properties:

(1) Self-reflexivity. A necessary and sufficient condition for R to be self-transitive is $I \subseteq R$, i.e., $I(x, y) = 1$ when $x = y$, and $I(x, y) = 0$ when $x \neq y$.
(2) Symmetry. The necessary and sufficient condition for R to be symmetric is that $R = R^T$.
(3) Transitivity. The necessary and sufficient condition for R to be transitive is that $R^2 \subset R$, i.e., $R \circ R \subset R$.

Suppose R is a fuzzy relation on A. We say that R is transmitted if for any $\lambda \in [0, 1]$ and any $x, y, z \in A$, $R(x, y) \geq \lambda$, $R(y, z) \geq \lambda$ can be introduced by $R(x, z) \geq \lambda$ holds.

R is said to be a fuzzy similarity relation on A if R is a fuzzy relation on A and it has self-reflexivity and symmetry properties. R is said to be a fuzzy equivalence relation on A if R is a fuzzy relation on A and it has self-reflexivity, symmetry and transitivity properties.

We can apply fuzzy relations for fuzzy clustering. When solving problems, we generally use the results of "some level" for classification, thus achieving fuzzy cluster analysis. The conditions of fuzzy equivalence relations are demanding, and

the actual problem often cannot satisfy all three properties at the same time; on the other hand, the actual problem is often a fuzzy similarity relation. We need to transform the fuzzy similarity relation into fuzzy equivalence relation in cluster analysis. Transitive closure is a common method to transform the fuzzy similarity relation into fuzzy equivalence relation. Transitive closure is also called transition closure.

Assuming that U is a finite theoretical domain, the transitive closure method starts from the fuzzy similarity relation R. It synthesizes it with itself repeatedly to compute R^2, R^4, … in order to obtain $t(R) = R^k$, when $R^k \circ R^k = R^k$ occurs for the first time. Next, we learn the process of applying the transitive closure method for cluster analysis by an example.

Example 3.8 Suppose we want to assess the contamination of the environment in the region, which needs to be measured by the content of 4 elements in the pollutant. We choose the theoretical domain U to be the set of the content of the 4 elements, and get the pollution data from 5 locations in the region by measuring $x_1 = (80, 10, 6, 2)$, $x_2 = (50, 1, 6, 4)$, $x_3 = (90, 6, 4, 6)$, $x_4 = (40, 5, 7, 3)$, $x_5 = (10, 1, 2, 4)$. Try to cluster the data by using the transitive closure method.

Solution We expressed the collected data in matrix form as follows:

$$X^* = \begin{bmatrix} 80 & 10 & 6 & 2 \\ 50 & 1 & 6 & 4 \\ 90 & 6 & 4 & 6 \\ 40 & 5 & 7 & 3 \\ 10 & 1 & 2 & 4 \end{bmatrix} \tag{3.73}$$

Since the value of the data varies greatly, we use the maximum value method to speciate the data, where each element of the matrix is divided by the maximum value of the column it is in, i.e.:

$$x_{ij} = \frac{x_{ij}^*}{\max\limits_{1 \le k \le 5} x_{kj}} \tag{3.74}$$

This results in the specified data matrix:

$$X = \begin{bmatrix} 0.89 & 1 & 0.86 & 0.33 \\ 0.56 & 0.10 & 0.86 & 0.67 \\ 1 & 0.60 & 0.57 & 1 \\ 0.44 & 0.50 & 1 & 0.50 \\ 0.11 & 0.10 & 0.29 & 0.67 \end{bmatrix} \tag{3.75}$$

Then, we use the max–min method to calculate the fuzzy similarity matrix, where each element is calculated by the formula:

$$r_{ij} = \frac{\sum_{k=1}^{4}(x_{ik} \wedge x_{jk})}{\sum_{k=1}^{4}(x_{ik} \vee x_{jk})} \tag{3.76}$$

From this, the fuzzy relation matrix can be obtained as:

$$R = \begin{pmatrix} 1 & 0.54 & 0.62 & 0.63 & 0.24 \\ 0.54 & 1 & 0.55 & 0.70 & 0.53 \\ 0.62 & 0.55 & 1 & 0.56 & 0.37 \\ 0.63 & 0.70 & 0.56 & 1 & 0.38 \\ 0.24 & 0.53 & 0.37 & 0.38 & 1 \end{pmatrix} \tag{3.77}$$

Calculating R^2, R^4, R^8 one at a time according to the transitive closure method yields:

$$R^2 = \begin{pmatrix} 1 & 0.63 & 0.62 & 0.63 & 0.53 \\ 0.63 & 1 & 0.56 & 0.70 & 0.53 \\ 0.62 & 0.56 & 1 & 0.62 & 0.53 \\ 0.63 & 0.70 & 0.62 & 1 & 0.53 \\ 0.53 & 0.53 & 0.53 & 0.53 & 1 \end{pmatrix} \tag{3.78}$$

$$R^4 = \begin{pmatrix} 1 & 0.63 & 0.62 & 0.63 & 0.53 \\ 0.63 & 1 & 0.62 & 0.70 & 0.53 \\ 0.62 & 0.62 & 1 & 0.62 & 0.53 \\ 0.63 & 0.70 & 0.62 & 1 & 0.53 \\ 0.53 & 0.53 & 0.53 & 0.53 & 1 \end{pmatrix} \tag{3.79}$$

$$R^8 = R^4 \circ R^4 \tag{3.80}$$

Thus, we have $t(R) = R^4$.

Then, by choosing an appropriate confidence level $\lambda \in [0, 1]$, we intercept $t(R)$ based on the λ level. We sort the elements in $t(R)$ in descending order: $1 > 0.70 > 0.63 > 0.62 > 0.53$. Hence, we can choose $\lambda = 1$ and the clustering matrix is:

$$t(R)_1 = \begin{pmatrix} 1\,0 & 0\,0\,0 \\ 0\,1 & 0\,0\,0 \\ 0\,0 & 1\,0\,0 \\ 0\,0 & 0\,1\,0 \\ 0\,0 & 0\,0\,1 \end{pmatrix} \tag{3.81}$$

From the above equation, it can be seen that when $\lambda = 1$, the data are clustered into 5 classes.

$$t(R)_{0.70} = \begin{pmatrix} 1\,0 & 0\,0\,0 \\ 0\,1 & 0\,1\,0 \\ 0\,0 & 1\,0\,0 \\ 0\,1 & 0\,1\,0 \\ 0\,0 & 0\,0\,1 \end{pmatrix} \tag{3.82}$$

From the above equation, it can be seen that when $\lambda = 0.70$, the data are clustered into 4 classes, where x_2 and x_4 belong to the same class.

$$t(R)_{0.63} = \begin{pmatrix} 1\,1 & 0\,1\,0 \\ 1\,1 & 0\,1\,0 \\ 0\,0 & 1\,0\,0 \\ 1\,1 & 0\,1\,0 \\ 0\,0 & 0\,0\,1 \end{pmatrix} \tag{3.83}$$

From the above equation, it can be seen that when $\lambda = 0.63$, the data are clustered into 3 classes, where x_1, x_2 and x_4 belong to the same class.

$$t(R)_{0.62} = \begin{pmatrix} 1\,1 & 1\,1\,0 \\ 1\,1 & 1\,1\,0 \\ 1\,1 & 1\,1\,0 \\ 1\,1 & 1\,1\,0 \\ 0\,0 & 0\,0\,1 \end{pmatrix} \tag{3.84}$$

From the above equation, it can be seen that when $\lambda = 0.62$, the data are clustered into 2 classes, where x_1, x_2, x_3 and x_4 belong to the same class.

$$t(R)_{0.53} = \begin{pmatrix} 1\,1 & 1\,1\,1 \\ 1\,1 & 1\,1\,1 \\ 1\,1 & 1\,1\,1 \\ 1\,1 & 1\,1\,1 \\ 1\,1 & 1\,1\,1 \end{pmatrix} \tag{3.85}$$

From the above equation, it can be seen that when $\lambda = 0.53$, the data are clustered into 1 class, i.e., all data samples belong to the same class.

Figure 3.4 gives the results of the fuzzy cluster analysis, and it can be seen that as the value of λ becomes smaller, the number of categories becomes smaller; conversely, as the value of λ becomes larger, the number of categories becomes larger. For the unsupervised learning problem, the number of categories into which the data samples are divided is unknown, thus Fig. 3.4 is able to show the degree of

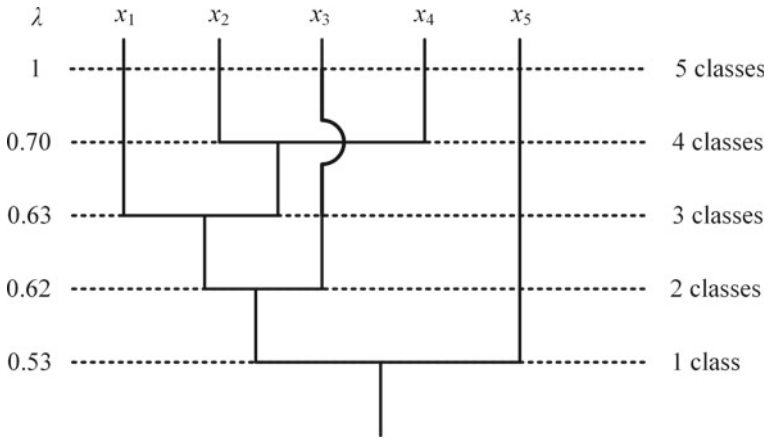

Fig. 3.4 Clustering results of pollutant data

similarity between the samples. The number of categories can also be determined in advance according to specific needs, so that the corresponding categories can be divided.

In the Example 3.8, the degree of similarity between data samples is measured using the max–min method. In addition to this, there are a number of methods, as follows:

(1) Scalar product method. This method uses the scalar product of vectors to define the degree of similarity:

$$r_{ij} = \begin{cases} 1, & i = j \\ (x_i \cdot x_j)/M, & i \neq j \end{cases} \tag{3.86}$$

where $x_i \cdot x_j$ denotes the scalar product of vectors and $M > 0$ is a parameter and satisfies $M \geq max\{x_i \cdot x_j | i \neq j\}$.

(2) Angle cosine method. This method uses the angle of the vectors to define the degree of similarity:

$$r_{ij} = \frac{|x_i \cdot x_j|}{\|x_i\| \|x_j\|} \tag{3.87}$$

where $\|x_i\|$ denotes the 2-norm of the vector x_i.

(3) Correlation coefficient method. This method uses the correlation coefficient to define the degree of similarity:

$$r_{ij} = \frac{\sum_{k=1}^{m} |x_{ik} - \overline{x}_i||x_{jk} - \overline{x}_j|}{\|x_i - \overline{x}_i\| \|x_j - \overline{x}_j\|} \tag{3.88}$$

where $\overline{x}_i$ denotes the mean value of vector x_i.

(4) Nearness degree method. A data sample $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$

can be considered as a fuzzy vector if each component of the sample is between 0 and 1. The degree of similarity between samples can be measured by the nearness degree. The methods to calculate the nearness degree are: max–min method, arithmetic mean-min method, and geometric mean-min method. The max–min method has been used in the Example 3.8, and the expression of the arithmetic mean-min method is

$$r_{ij} = \frac{\sum_{k=1}^{m}\left(x_{ik} \wedge x_{jk}\right)}{\left(\sum_{k=1}^{m}\left(x_{ik} + x_{jk}\right)\right)/2} \tag{3.89}$$

The expression for the geometric mean-min method is:

$$r_{ij} = \frac{\sum_{k=1}^{m}\left(x_{ik} \wedge x_{jk}\right)}{\sum_{k=1}^{m}\sqrt{x_{ik}x_{jk}}} \tag{3.90}$$

(5) Distance method. This method uses the distance between vectors to define the degree of similarity. The closer the distance between two vectors, the greater the degree of similarity between them; conversely, the farther the distance between two vectors, the less similar the two vectors are. Commonly used distances include Euclidean distance, Chebyshev distance, Hamming distance and Minkowski distance. The expressions for the definition of these distances will not be introduced.

(6) Absolute value reciprocal method. This method uses the reciprocal of the absolute value of the difference of two vectors to define the degree of similarity, and its expression is:

$$r_{ij} = \begin{cases} 1, & i = j \\ c/\sum_{k=1}^{m}\left(x_{ik} - x_{jk}\right), & i \neq j \end{cases} \tag{3.91}$$

where c is a properly chosen positive number such that $r_{ij} \in [0, 1]$.

3.5 Fuzzy Inference

This section introduces fuzzy inference. We start from introducing the concept of fuzzy linguistic variables. A linguistic variable can be represented by a quintuple as follows:

$$(x, T(x), U, G, M) \tag{3.92}$$

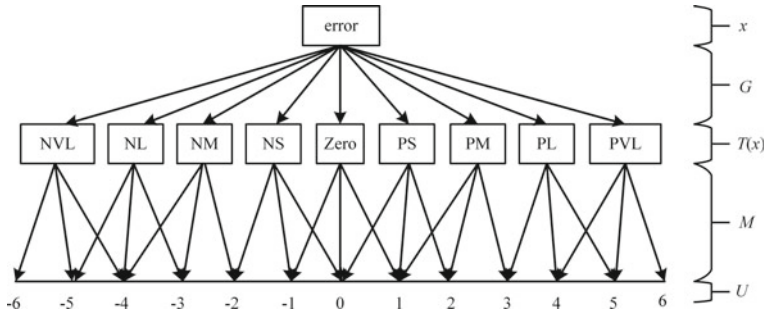

Fig. 3.5 Quintuplet diagram of "error"

where x is the name of the linguistic variable, $T(x)$ is the set of linguistic variable values, U is the domain of the variable x, G is a grammatical rule, and M is a semantic rule. The grammatical rule G is used to generate the names of linguistic variables x, while the semantic rule M is used to generate the membership function of the fuzzy set.

For example, in a control system, we take "error" as a linguistic variable x and choose the domain $U = [-6, 6]$. The atomic words of the linguistic variable "error" are "large", "medium", "small", "zero". By applying the appropriate tone operator to these atomic words, multiple linguistic values can be formed, such as "very large" [3]. Moreover, we can consider that the error has positive and negative cases. Then, the set of linguistic variables $T(x)$ can be expressed as $T(x) = \{$negative very large (NVL), negative large (NL), negative medium (NM), negative small (NS), zero, positive small (PS), positive medium (PM), positive large (PL), positive very large (PVL)$\}$. Figure 3.5 shows the schematic diagram of the quintuple of fuzzy linguistic variables with "error" as the theoretical domain.

As can be seen from Fig. 3.5, the grammatical rule G acts to convert the linguistic variable x into a linguistic variable value, while the semantic rule M acts to map the linguistic variable value into the theoretical domain. A fuzzy linguistic variable is equivalent to a fuzzy set. After the grammatical rule acts on the linguistic variable. That is, it uses the tone operator to add "very", "relatively", "slightly", etc. to the linguistic variables to obtain different values of the linguistic variables. These values need to be mapped to different affiliation values, which is achieved by the semantic rules.

If we denote a fuzzy linguistic variable by A, whose membership function is denoted by μ_A. We add to A the inflections "very", "fairly", "comparatively", "slightly". The membership function of the linguistic variable values can be transformed into $\mu_{\text{very } A} = \mu_A^2$, $\mu_{\text{relatively } A} = \mu_A^{1.5}$, $\mu_{\text{slightly } A} = \mu_A^{0.75}$. The complement of the fuzzy set can be $\mu_{\text{not } A} = 1 - \mu_A$. Note that this is only an example to illustrate the correspondence variation of grammatical rules and membership functions, this correspondence form is not fixed and needs specific analysis in each example.

In fuzzy computing, a fuzzy logic rule is a fuzzy implication relation. An implication relation is essentially a kind of reasoning or inference. One of the most commonly

used fuzzy implication relations is: if x is A, then y is B, denoted as $A \to B$. In ordinary logic, $A \to B$ has a strict definition. In fuzzy logic, $A \to B$ is not a simple generalization of ordinary logic and has many ways of definition. The commonly used operations for fuzzy implication relations are:

(1) Fuzzy implication minimum operation. This operation is given by Mamdani. Its expression is:

$$R_c = A \to B = A \times B = \int_{X \times Y} \frac{\mu_A(x) \wedge \mu_B(y)}{(x, y)} \tag{3.93}$$

From (3.93) the fuzzy implication relation is defined as the multiplication of two fuzzy sets, which in turn transforms into the intersection of two fuzzy sets.

(2) Fuzzy implication arithmetic operation. This operation is given by Zadeh and its expression is:

$$R_c = A \to B = \left(\overline{A} \times Y\right) \oplus (X \times B) = \int_{X \times Y} \frac{1 \wedge (1 - \mu_A(x) + \mu_B(y))}{(x, y)} \tag{3.94}$$

From the above equation, it is clear that the fuzzy implication relation is defined as an arithmetic operation, which in turn transforms into complement and intersection operations of two fuzzy sets.

(3) Fuzzy implication max–min operation. This operation is given by Zadeh and its expression is:

$$R_c = A \to B = (A \times B) \cup \left(\overline{A} \times Y\right) = \int_{X \times Y} \frac{(\mu_A(x) \wedge \mu_B(y)) \vee (1 - \mu_A(x))}{(x, y)} \tag{3.95}$$

From the above equation, it can be seen that the fuzzy implication relation is defined as multiplication and concatenation of sets, which in turn transforms into intersection, complement and concatenation of fuzzy sets.

Among these three fuzzy implication relations, the most commonly used are fuzzy implication minimum operation and fuzzy implication arithmetic operation.

Example 3.9 Suppose we have two fuzzy sets $A = [1, 0.8, 0.7, 0.4, 0.1]$ and $B = [1, 0.7, 0.3, 0]$, try to compute the fuzzy implication relation $A \to B$ using the fuzzy implication minimum operation.

Solution Based on (3.93), the fuzzy implication relation $A \to B$ is:

$$R_c = \int_{X\times Y} \frac{\mu_A(x) \wedge \mu_B(y)}{(x, y)} = \begin{bmatrix} 1 & 0.7 & 0.3 & 0 \\ 0.8 & 0.7 & 0.3 & 0 \\ 0.7 & 0.7 & 0.3 & 0 \\ 0.4 & 0.4 & 0.3 & 0 \\ 0.1 & 0.1 & 0.1 & 0 \end{bmatrix} \tag{3.96}$$

The example can also be calculated using Matlab with the following programs:

```
A = [1, 0.8, 0.7, 0.4, 0.1];
B = [1, 0.7, 0.3, 0];
m = length(A);
n = length(B);
for i1 = 1:m
   for j1 = 1:n
      Rc(i1,j1)=min(A(i1), B(j1));
   end
end
```

After running the above program, the variable Rc is the matrix of the fuzzy implication relation.

Fuzzy inference is the process of determining the mapping from input to output using fuzzy logic. After determining the mapping from input to output, fuzzy identification or fuzzy decision making can be performed. We introduce fuzzy inference in terms of both simple fuzzy conditional statements and multiple fuzzy conditional statements.

(1) Simple fuzzy conditional statements.

Suppose the existing precondition is: if x is A, then y is B. For the input: if x is A', then the output is y is B'. This is the simple fuzzy conditional statement, where the conclusion B' is based on the synthesis of the fuzzy implication relation $A \to B$ and the fuzzy set A', i.e.:

$$B' = A'^{\circ}(A \to B) = A'^{\circ}R \tag{3.97}$$

where R is a fuzzy implication relation and ^{0} is a synthetic operation.

Example 3.10 Suppose there is a fuzzy implication relation $A \to B$, where $A =$ [0,0.1,0.4,0.5,0.9] and $B =$ [0,0,0.2,0.5,1], for the input $A' =$ [1, 0.9, 0.6, 0.5, 0.1], try to find its output B'.

Solution According to the above Eq. (3.97), it is obtained that:

$$B' = A'^{\circ}R = A'^{\circ}\begin{pmatrix} 0\ 0 & 0.0\ 0.0\ 0.0 \\ 0\ 0 & 0.1\ 0.1\ 0.1 \\ 0\ 0 & 0.2\ 0.4\ 0.4 \\ 0\ 0 & 0.2\ 0.5\ 0.5 \\ 0\ 0 & 0.2\ 0.5\ 0.9 \end{pmatrix} = (0, 0, 0.2, 0.5, 0.5) \tag{3.98}$$

The Matlab programs for the Example 3.10 are as follows:

```
A = [0, 0.1, 0.4, 0.5, 0.9];
B = [0, 0, 0.2, 0.5, 1];
Aapo = [1, 0.9, 0.6, 0.5, 0.1];
m = length(A);
n = length(B);
for i1 = 1:m
   for j1 = 1:n
      Rc(i1,j1)=min(A(i1), B(j1));
   end
end
n = size(Aapo, 1);
l = size(Rc, 2);
for i1 = 1:n
   for j1 = 1:l
      Bapo(i1, j1) = max(min([Aapo(i1,:); Rc(:, j1)']));
   end
end
```

After running the above program, the variable Bapo is the inference result of the simple fuzzy statement.

(2) Multiple fuzzy conditional statements

Multiple fuzzy conditional statements can be divided into fuzzy conditional statements connected by "and" and fuzzy conditional statements connected by "also".

Fuzzy conditional statements using "and" concatenation are a common way of fuzzy inference. Suppose the existing precondition is: if x is A and y is B, then z is C. For the input: if x is A' and y is B', then the output is: z is C'. Compared with the simple fuzzy conditional statement, the preconditions and inputs have been added with "and" concatenation. If there is only one "and", it can be regarded as a double fuzzy conditional statement. Similarly, if there are more than one "and", it is a multiple fuzzy conditional statement.

We introduce the operation of double fuzzy conditional statements as an example. The fuzzy preconditions x is A and y is B can be regarded as a fuzzy set on the direct product space $X \times Y$, denoted as $A \times B$. Its membership function can be defined as:

$$\mu_{A\times B}(x, y) = min\{\mu_A(x), \mu_B(y)\} \tag{3.99}$$

The membership function can also be defined as:

$$\mu_{A\times B}(x, y) = \mu_A(x)\mu_B(y) \tag{3.100}$$

The fuzzy implication relation can be expressed as $A \times B \to C$. Based on the fuzzy implication minimum operation, the fuzzy implication relation can be defined as:

$$R = A \times B \to C = A \times B \times C = \int_{X\times Y\times Z} \frac{\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)}{(x, y, z)} \tag{3.101}$$

The fuzzy input x is A' and y is B' can be defined as $A' \times B'$, so the conclusion z is C' can be defined as:

$$C' = \left(A' \times B'\right)^{\circ}(A \times B \to C) = \overrightarrow{\left(A' \times B'\right)}^{\circ} R \tag{3.102}$$

where $\overrightarrow{\left(A' \times B'\right)}$ denotes stretching of $A' \times B'$ by rows.

Example 3.11 Reasoning voltage by knowing the temperature and its variation. If the fuzzy sets A, B, C denote low temperature, fast temperature change and high voltage respectively. We have $A = (1, 0.4, 0.1)$, $B = (0.1, 0.6, 1)$, $C = (0.3, 0.7, 1)$, and A', B' denote higher temperature and faster temperature change respectively. We have $A' = (0.3, 0.5, 0.7)$, $B' = (0.4, 0.5, 0.9)$. The precondition is that if x is A and y is B, then z is C. The implication is that if the temperature is low and the temperature is changing fast, then there is a high voltage. For the input: if x is A' and y is B', try to find the conclusion C'. That is to infer the voltage if the temperature is higher and the temperature is changing faster.

Solution The Matlab programs for the Example 3.11 are as follows:

```
A = [1, 0.4, 0.2];
B = [0.1, 0.6, 1];
C = [0.3, 0.7, 1];
m = length(A);
n = length(B);
for i1 = 1:m
   for j1 = 1:n
      RAB(i1,j1) = min(A(i1), B(j1));
   end
end
RABLaShen = reshape(RAB', 1, size(RAB, 1) * size(RAB, 2));
m = length(RABLaShen);
n = length(C);
for i1 = 1:m
   for j1 = 1:n
```

```
        RABC(i1,j1) = min(RABLaShen(i1), C(j1));
    end
end
Aapo = [0.3, 0.5, 0.7];
Bapo = [0.4, 0.5, 0.9];
m = length(Aapo);
n = length(Bapo);
for i1 = 1:m
    for j1 = 1:n
        RAapoBapo(i1,j1) = min(Aapo(i1), Bapo(j1));
    end
end
m = size(RAapoBapo, 1);
n = size(RAapoBapo, 2);
RAapoBapoLaShen = reshape(RAapoBapo', 1, m*n);
n = size(RAapoBapoLaShen, 1);
l = size(RABC, 2);
for i1 = 1:n
    for j1 = 1:l
        Capo(i1, j1) = max(min([RAapoBapoLaShen(i1,:); RABC(:, j1)']));
    end
end
```

After running the above program, the variable Capo is the value of conclusion C'. We have $C' = (0.3, 0.4, 0.4)$.

Next, we introduce the multiple fuzzy conditional statements using the "also" conjunction. Suppose the existing preconditions are: if x is A_1 and y is B_1, then z is C_1, also if x is A_2 and y is B_2, then z is C_2, …, also if x is A_n and y is B_n, then z is C_n. For the input: if x is A' and y is B', then the output: z is C'. Compared to the fuzzy conditional statement connected by "and", the precondition has more conditions connected by "also".

If the fuzzy implication relation for the i-th conditional rule, i.e., if x is A_i and y is B_i, then z is C_i, is written as:

$$R_i = A_i \times B_i \to C_i \tag{3.103}$$

Then the total fuzzy implication relation for all n precondition rules is:

$$R = \cup_{i=1}^{n} R_i \tag{3.104}$$

Finally, the output z is C' as:

$$C' = \left(A' \times B'\right) \circ R \tag{3.105}$$

From the above expressions, we can see that the multiple fuzzy conditional statements connected with "also" are defined on the basis of the fuzzy conditional statements connected with "and". The final output can be obtained by following the formula step by step.

3.6 Fuzzy Control System

Fuzzy control systems are arguably the most important application of fuzzy computing. Fuzzy control systems are also known as fuzzy controllers. A fuzzy control system is a comprehensive use of fuzzy logic theory [5]. A fuzzy control system includes modules such as fuzzification, knowledge base, fuzzy inference, and defuzzification, as shown in Fig. 3.6.

The part in the dashed box in Fig. 3.6 is the fuzzy control system. The fuzzy control system regulates the control object and solves real-life problems by controlling changes in the object. Fuzzification refers to the conversion of the input deterministic variable into a fuzzy variable. The deterministic variable is also called the clear variable. Fuzzy inference refers to the use of fuzzy implication relations and inference rules in fuzzy logic to make decisions. Defuzzification refers to the conversion of fuzzy variables obtained by fuzzy inference into definite variables for control purposes. Knowledge base refers to the knowledge of the real-life problem and the object to be controlled, which usually includes a database and a fuzzy control rule base.

(1) Fuzzification. The fuzzification operation is to map the input observations into a fuzzy set over the theoretical domain.

First, the input observations are processed so that they are converted into input variables suitable for the fuzzy controller. For example, if the input observation is denoted as r and the output is denoted as y, in general, we need to calculate the error $e = r\text{-}y$ and the rate of change of the error $e' = de/dt$.

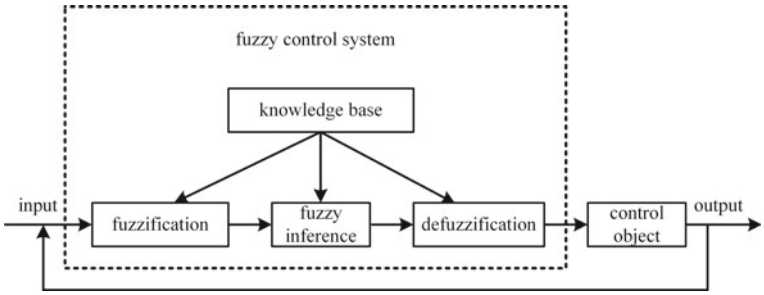

Fig. 3.6 Diagram of a fuzzy control system

Second, the input variables obtained from the processing are to be scaled so that they are mapped to their respective theoretical domain ranges.

Finally, the input variables transformed to the range of the theoretical domain are to be fuzzed to obtain the corresponding fuzzy sets [6].

The method of scale transformation of the input variables can be linear or nonlinear, while the theoretical domain can be discrete or continuous. If the theoretical domain is to be restricted to be discrete, the continuous domain needs to be discretized, also known as quantization [7]. The quantization of the theoretical domain can be homogeneous or non-homogeneous.

For example, assuming that the range of the continuous domain is [− 3, 3], Table 3.3 gives the method of uniform quantization:

As can be seen from Table 3.3, the theoretical domain is uniformly divided into different levels, except for the two ends, thus achieving a discretization of the continuous domain.

As can be seen from Table 3.4, the theoretical domain is divided non-uniformly into different levels, except for the two ends, thus achieving a discretization of the continuous domain.

Since the continuous theoretical domain can be transformed into the discrete theoretical domain, we generally discuss the problem on the discrete theoretical domain. In order to transform the input variable into a fuzzy set on the theoretical domain, we need to introduce the description method of the membership function. The description method is introduced from both discrete and continuous perspectives.

The membership functions on discrete theoretical domains are generally described numerically. Table 3.5 gives an example of a description of a discrete theoretical domain.

In Table 3.5, the fuzzy sets NB, NS, ZE, PS, and PB denote Negative Big, Negative Small, Zero, Positive Small, and Positive Big, respectively. ZE can be expressed as:

$$ZE = \frac{0.5}{-1} + \frac{1.0}{0} + \frac{0.5}{1} \tag{3.106}$$

For the membership function on the continuous domain, the general method is functional description. That is the membership is usually expressed in the form of a

Table 3.3 Uniform quantization of continuous theoretical domains

Range	[− 3, − 1.4)	[− 1.4, − 0.8)	[− 0.8, − 0.4)	[− 0.4, 0.4)	[0.4, 0.8)	[0.8, 1.4)	[1.4, 3]
Level	− 3	− 2	− 1	0	1	2	3

Table 3.4 Non-uniform quantization of continuous theoretical domains

Range	[− 3, − 2.5)	[− 2.5, − 1.5)	[− 1.5, − 0.5)	[− 0.5, 0.5)	[0.5, 1.5)	[1.5, 2.5)	[2.5, 3]
Level	− 3	− 2	− 1	0	1	2	3

Table 3.5 Numerical description method

Fuzzy set	− 3	− 2	− 1	0	1	2	3
NB	1.0	0.5	0.0	0.0	0.0	0.0	0.0
NS	0.0	0.5	1.0	0.5	0.0	0.0	0.0
ZE	0.0	0.0	0.5	1.0	0.5	0.0	0.0
PS	0.0	0.0	0.0	0.5	1.0	0.5	0.0
PB	0.0	0.0	0.0	0.0	0.0	0.5	1.0

function. The most common forms of functions are Gaussian function, triangular function, trapezoidal function. The expression of the Gaussian membership function is:

$$\mu_A(x) = e^{-\frac{(x-x_0)^2}{2\sigma^2}} \tag{3.107}$$

where x_0 is the mean value of the membership function and σ is the variance of the membership function. By adjusting the magnitude of the mean value and variance, the Gaussian membership function with different shapes can be obtained.

If the input variable is one-dimensional, it is usually converted into a fuzzy set using the single-point fuzzy method. Assuming that x_0 is a clear input membership, it is usually fuzzified into a single-point fuzzy set, denoted as A, whose expression is:

$$\mu_A(x) = \begin{cases} 1, & x = x_0 \\ 0, & x \neq x_0 \end{cases} \tag{3.108}$$

It is easy to see that the single-point fuzzy set only formally converts the clear variable into a fuzzy variable, while it is still an accurate quantity in substance. For example, if the theoretical domain is $\{-3, -2, -1, 0, 1, 2, 3\}$ and the input variable $x_0 = -2$, then its corresponding single-point fuzzy set is $A = (0,1,0,0,0,0,0,0)$; while for $x_0 = 1$, then its corresponding single-point fuzzy set is $A = (0, 0, 0, 0, 1, 0, 0)$.

(2) Fuzzy inference. Fuzzy inference was introduced in the previous section and is omitted here.

(3) Defuzzification. A fuzzy variable can be obtained by fuzzy inference, while for the actual control problem, it must be converted into a clear variable. It is necessary to convert the fuzzy variable into a clear variable. This is the task to be accomplished by the defuzzification. The common methods of defuzzification include: average maximum membership method, weighted average method, maximum membership taking the minimum method, maximum membership taking the maximum method, and median method.

The average maximum membership method is also known as the "mom" method. If the membership function of the fuzzy set C' of the output variable is a single-peaked

function. That is there is only one peak, the maximum value of the membership function is selected as the clear value, i.e.:

$$\mu_{C'}(z_0) \geq \mu_{C'}(z), z \in Z \tag{3.109}$$

where z_0 denotes the determined value after defuzzification. If the membership function of the fuzzy set C' of output variable is not a single-peaked function, i.e., there are multiple peaks, the average of the values of the elements corresponding to these peaks is selected as the clear value.

Example 3.12 Suppose the fuzzy set of output variable z_1 is known as $C_1' = \frac{0.1}{2} + \frac{0.4}{3} + \frac{0.7}{4} + \frac{1.0}{5} + \frac{0.7}{6} + \frac{0.3}{7}$, and the fuzzy set of another output variable z_2 is $C_2' = \frac{0.3}{-4} + \frac{0.8}{-3} + \frac{1.0}{-2} + \frac{1.0}{-1} + \frac{0.8}{0} + \frac{0.3}{1} + \frac{0.1}{2}$, try to use the average maximum membership method to find the corresponding clear variables z_{10} and z_{20} for z_1 and z_2.

Solution According to the average maximum membership method, C_1' is a single-peaked function and it is easy to know that $\mu_{C_1'}(z_1 = 5) = 1.0$ is its maximum C_1'. Thus, $z_{10} = 5$. C_2' is a function with two peaks. It is easy to know that $\mu_{C_2'}(z_2 = -2) = 1.0$ and $\mu_{C_2'}(z_2 = -1) = 1.0$ are is its maximum membership. Thus, $z_{10} = (-2 - 1)/2 = -1.5$.

The weighted average method is also known as the area center of gravity method, sometimes abbreviated as centroid method. This method is a weighted average of the membership in a fuzzy set to obtain clear values. For a discrete membership function, the expression of the weighted average method is:

$$z_0 = \frac{\sum_{i=1}^{n} z_i \mu_{C'}(z_i)}{\sum_{i=1}^{n} \mu_{C'}(z_i)} \tag{3.110}$$

For a continuous membership function, the expression of the weighted average method is:

$$z_0 = df(z) = \frac{\int_a^b z_i \mu_{C'}(z_i) dz}{\int_a^b \mu_{C'}(z_i) dz} \tag{3.111}$$

Example 3.13 Suppose the fuzzy set of the known output variable z_1 is $C_1' = \frac{0.1}{2} + \frac{0.4}{3} + \frac{0.7}{4} + \frac{1.0}{5} + \frac{0.7}{6} + \frac{0.3}{7}$, try to use the weighted average method to find the corresponding clear variable z_{10} of z_1.

Solution It is easy to know that C_1' is a discrete membership function, so the calculation process of the weighted average method is:

$$z_{10} = \frac{0.1 \times 2 + 0.4 \times 3 + 0.7 \times 4 + 1.0 \times 5 + 0.7 \times 6 + 0.3 \times 7}{0.1 + 0.4 + 0.7 + 1.0 + 0.7 + 0.3} = 4.84 \tag{3.112}$$

The method of taking the minimum value of the maximum membership is also noted as "som" method. This method selects the smallest of all points in the fuzzy set with the maximum membership as the result of defuzzification. The taking the maximum value of the maximum membership method is also known as the "lom" method. This method selects the largest of all points in the fuzzy set with the maximum membership as the result of defuzzification. The median method is also known as the area equalization method and is denoted as bisector method. The median method is to select the median of $\mu_{C'}(z)$ as the result of the defuzzification of z.

Example 3.14 Suppose the domain $U = \{-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6\}$, we have the linguistic variables $x, y, z \in U$. The values of the linguistic variables $T(x) = \{NB, NM, NS, NZ, PZ, PS, PM, PB\}$, where *NB* is Negative Big, *NM* is Negative Middle, *NS* is Negative Small, *PS* is Negative Zero, *PZ* is Positive Zero, and *PS* is Negative Small, *PM* is Positive Middle, *PB* is Positive Big. The linguistic variable value $T(y) = T(z) = \{NB, NM, NS, ZE, PS, PM, PB\}$, where ZE is zero (Zero). For the linguistic variables x, the fuzzy sets corresponding to their linguistic variable values are shown in Table 3.6.

For the linguistic variables y, z, the fuzzy sets corresponding to their linguistic variable values are shown in Table 3.7.

In this fuzzy control system, the knowledge base is the fuzzy control rules for the linguistic variables x, y and z, as shown in Table 3.8.

As shown in Table 3.8, the number of language variable values for language variable x is 8. The number of language variable values for language variable y is 7. Thus this knowledge base contains 56 rules. Note that 56 rules are the maximum possible number, and in practice, it is possible that some possible rules do not exist, i.e., the square column in Table 3.8 can be empty. The rules in the table are, in order, R_1: if x is NB and y is NB, then z is NB; R_2: if x is NB and y is NM, then z is NB; …; R_{56}: if x is PB and y is PB, then z is PB.

Solution If the inputs are noted as x_0 and y_0, then according to the above steps of fuzzy control system. We can use single-point fuzzy sets to fuzzify the input

Table 3.6 Membership of the fuzzy set of linguistic variable x

$\mu(x)$	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
NB	1.0	0.8	0.7	0.4	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NM	0.2	0.7	1.0	0.1	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NS	0.1	0.1	0.3	0.7	1.0	0.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0
NZ	0.0	0.0	0.0	0.0	0.1	0.6	1.0	0.0	0.0	0.0	0.0	0.0	0.0
PZ	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.6	0.1	0.0	0.0	0.0	0.0
PS	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.7	1.0	0.7	0.3	0.1	0.0
PM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.7	1.0	0.7	0.3
PB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.4	0.7	0.8	1.0

Table 3.7 Membership of fuzzy sets of linguistic variables y and z

$\mu(y)$, $\mu(z)$	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
NB	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NM	0.3	0.7	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NS	0.0	0.0	0.3	0.7	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0
ZE	0.0	0.0	0.0	0.0	0.3	0.7	1.0	0.7	0.3	0.0	0.0	0.0	0.0
PS	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.7	1.0	0.7	0.3	0.0	0.0
PM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.7	1.0	0.7	0.3
PB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.7	1.0

Table 3.8 Fuzzy control rules for linguistic variables x, y and z

x	NB	NM	NS	ZE	PS	PM	PB
NB	NB	NB	NB	NB	NM	ZE	ZE
NM	NB	NB	NB	NB	NM	ZE	ZE
NS	NM	NM	NM	NM	ZE	PS	PS
NZ	NM	NM	NS	ZE	PS	PM	PM
PZ	NM	NM	NS	ZE	PS	PM	PM
PS	NS	NS	ZE	PM	PM	PM	PM
PM	ZE	ZE	PM	PB	PB	PB	PB
PB	ZE	ZE	PM	PB	PB	PB	PB

variables. In fuzzy inference, the intersection of fuzzy sets is used for fuzzy conditional statements connected by "and". The concatenation of fuzzy sets is used for fuzzy conditional statements connected by "also", the max–min method is used for synthesis operations. The intersection method is used for fuzzy implication relations. The weighted average method is used for the defuzzification operation. The fuzzy set C' of the output variable is:

$$C' = \left(A' \times B'\right)^{\circ} \cup_{i=1}^{56} R_i = \cup_{i=1}^{56} \left(A' \times B'\right)^{\circ}((A_i \times B_i) \to C_i) = \cup_{i=1}^{56} C_i' \quad (3.113)$$

where $C_i' = \left(A' \times B'\right)^{\circ}((A_i \times B_i) \to C_i)$.

For $x_0 = -6$ and $y_0 = -6$, $A' = B' = (1, 0, \ldots, 0)_{1\times 13}$ is obtained by the single-point fuzzy set method. We have $C_1' = \left(A' \times B'\right)^{\circ}((A_1 \times B_1) \to C_1) = \left(A' \times B'\right)^{\circ}((A_{NB} \times B_{NB}) \to C_{NB}) = (1, 0.7, 0.3, 0, \ldots, 0)_{1\times 13}$. Similarly, $C_2', C_3', \ldots, C_{56}'$ can be computed. Finally the output fuzzy set $C' = (1, 0.7, 0.3, 0, \ldots, 0)_{1\times 13}$ is calculated. Using the weighted average method, the clear variable is obtained as:

$$z_0 = \frac{1 \times (-6) + 0.7 \times (-5) + 0.3 \times (-4)}{1 + 0.73 + 0.3} = -5.35 \quad (3.114)$$

Similarly, we can calculate the output variable z_0 when $x_0 \in U$ and $y_0 \in U$ are other combinations in the theoretical domain, as shown in Table 3.9, which provides a user-queryable control table.

In Table 3.9, the output variables are retained only to one decimal place to save space and for display purposes.

The above example of a fuzzy control system can also be implemented using Matlab programming with the following programs:

```
A = xlsread('fuzzycon.xlsx','x');
B = xlsread('fuzzycon.xlsx','yz');
C = B;
R = xlsread('fuzzycon.xlsx','r');
U = -6:1:6;
n = length(U);
X = eye(n);
Y = X;
Z = zeros(n);
for i=1:n
   for j=1:n
      x0 = X(i,:);
      y0 = Y(j,:);
      zi = defuzzyAlsoAnd(A,B,C,R,x0,y0);
      zi = sum(zi.*U)/sum(zi);
      Z(i,j) = roundn(zi, -2);
   end
end
```

Table 3.9 Control table for linguistic variables x, y and z

x	− 6	− 5	− 4	− 3	− 2	− 1	0	1	2	3	4	5	6
− 6	− 5.4	− 5.2	− 5.4	− 5.2	− 5.4	− 5.2	− 4.7	− 4.3	− 2.7	− 2.0	− 1.3	0.0	0.0
− 5	− 5.0	− 5.0	− 5.0	− 5.0	− 5.0	− 5.0	− 3.9	− 3.7	− 2.4	− 1.8	− 1.1	0.2	0.2
− 4	− 4.7	− 4.5	− 4.7	− 4.5	− 4.7	− 4.5	− 3.1	− 2.9	− 1.9	− 1.4	− 0.7	0.6	0.6
− 3	− 4.3	− 4.3	− 4.3	− 4.3	− 4.3	− 4.3	− 2.9	− 2.3	− 1.4	− 0.9	− 0.3	1.0	1.0
− 2	− 4.0	− 4.0	− 3.8	− 3.8	− 3.5	− 3.4	− 2.4	− 1.8	− 0.4	0.0	0.2	1.6	1.6
− 1	− 4.0	− 4.0	− 3.4	− 3.1	− 2.5	− 2.1	− 1.5	− 1.1	0.3	1.9	2.3	2.9	2.9
0	− 3.6	− 3.6	− 2.9	− 2.6	− 1.0	− 0.5	0.0	0.5	1.0	2.6	2.9	3.6	3.6
1	− 2.9	− 2.9	− 2.3	− 1.9	− 0.3	1.1	1.5	2.1	2.5	3.1	3.4	4.0	4.0
2	− 1.8	− 1.8	− 0.6	− 0.3	0.4	1.8	2.4	3.4	3.5	3.8	3.8	4.0	4.0
3	− 1.0	− 1.0	0.3	0.9	1.4	2.3	2.9	4.3	4.3	4.3	4.3	4.3	4.3
4	− 0.6	− 0.6	0.7	1.4	1.9	2.9	3.1	4.5	4.7	4.5	4.7	4.5	4.7
5	− 0.2	− 0.2	1.1	1.8	2.4	3.7	3.9	5.0	5.0	5.0	5.0	5.0	5.0
6	0.0	0.0	1.3	2.0	2.7	4.3	4.7	5.2	5.4	5.2	5.4	5.2	5.4

```
xlswrite('fuzzycon.xlsx', Z, 'result');
```

In the above program, the fuzzy set matrix and fuzzy rule matrix are stored in a file with the suffix "xlsx". Moreover, the calculated output results are also stored in a file. The defuzzyAlsoAnd.m function is used for defuzzification:

```
function zi = defuzzyAlsoAnd(A,B,C,R,x0,y0)
m = size(A,1);
n = size(B,1);
for i=1:m
   for j=1:n
      k = R(i,j);
      Rtmp((i-1)*n+j,:)=fuzzyInference(A(i,:),B(j,:),C(k,:),x0,y0);
   end
end
zi = max(Rtmp);
The fuzzyInference.m function is used for fuzzy inference:
function Capo=fuzzyInference(Ai,Bi,Ci,Aapo,Bapo)
m = length(Ai);
n = length(Bi);
for i1 = 1:m
   for j1 = 1:n
      RAB(i1,j1) = min(Ai(i1), Bi(j1));
   end
end
RABLaShen = reshape(RAB', 1, size(RAB, 1) * size(RAB, 2));
m = length(RABLaShen);
n = length(Ci);
for i1 = 1:m
   for j1 = 1:n
      RABC(i1,j1) = min(RABLaShen(i1), Ci(j1));
   end
end
m = length(Aapo);
n = length(Bapo);
for i1 = 1:m
   for j1 = 1:n
      RAapoBapo(i1,j1) = min(Aapo(i1), Bapo(j1));
   end
end
m = size(RAapoBapo, 1);
n = size(RAapoBapo, 2);
RAapoBapoLaShen = reshape(RAapoBapo', 1, m*n);
n = size(RAapoBapoLaShen, 1);
l = size(RABC, 2);
for i1 = 1:n
```

```
    for j1 = 1:l
      Capo(i1, j1) = max(min([RAapoBapoLaShen(i1,:); RABC(:, j1)']));
    end
end
```

It can be seen that the fuzzy control system is a comprehensive use of fuzzy logic theory.

3.7 Fuzzy Logic Designer

In Matlab, the fuzzy logic toolbox implements various computational operations for fuzzy control systems. This section introduces the Fuzzy Logic Designer tool in Matlab, taking the fuzzy control of a washing machine as an example.

In people's daily life, washing machine is a common home appliance. At present, intelligent washing machine has been developed rapidly [7]. Intelligent washing machine based on fuzzy control has also received more and more attention. In the intelligent washing machine, the design and research of intelligent control system is the basis that the various functional indicators of intelligent washing machine can be achieved [8]. Thus, the design and research of intelligent washing machine based on fuzzy computing has important theoretical significance and strategic value.

Example 3.15 The simulation of fuzzy control system can be used to realize the problem of intelligent control of washing machine by computer. It is assumed that the input variables of the fuzzy control system of the washing machine we want to design are sludge and grease of the clothes, and the output of this system is the washing time. The sludge amount and grease amount can be measured by sensors. The theoretical domain of sludge x is set to $U = [0, 100]$. The theoretical domain of grease is also set to $U = [0, 100]$. There are 3 fuzzy sets of sludge x on the theoretical domain, which are small sludge (SD), middle sludge (MD), and large sludge (LD). The membership function of SD is $SD(x) = (50 - x)/50$, $0 \le x \le 50$. The membership function of LD is $LD(x) = (x - 50)/50$, $50 \le x \le 100$. The membership function of MD is:

$$MD(x) = \begin{cases} x/50, & 0 \le x \le 50 \\ (100 - x)/50, & 50 \le x \le 100 \end{cases} \tag{3.115}$$

There are also 3 fuzzy sets of grease y on the theoretical domain, which are small grease (SG), middle grease (MG), and large grease (LG). the membership function of SG is $SG(y) = (50 - y)/50$ with $0 \le y \le 50$, the membership function of LG is $LG(Y) = (y - 50)/50$, $50 \le y \le 100$, and the membership function of MG is:

$$MG(y) = \begin{cases} y/50, & 0 \le y \le 50 \\ (100 - y)/50, & 50 \le y \le 100 \end{cases} \tag{3.116}$$

The output quantity is the washing time t, whose theoretical domain is [0, 60]. The output variable t has five fuzzy sets on the theoretical domain, which are: very short time (VS), short time (S), medium time (M), long time (L), and very long time (VL). The membership function of VS is $VS(t) = (10 - t)/10, 0 \leq t \leq 10$, and the membership function of S is:

$$S(t) = \begin{cases} t/50, & 0 \leq t \leq 10 \\ (25 - t)/15, & 10 < t \leq 25 \end{cases} \tag{3.117}$$

The membership function of M is:

$$M(t) = \begin{cases} (t - 10)/15, & 10 \leq t \leq 25 \\ (40 - t)/15, & 25 < t \leq 40 \end{cases} \tag{3.118}$$

The membership function of L is:

$$L(t) = \begin{cases} (t - 25)/15, & 25 \leq t \leq 40 \\ (60 - t)/15, & 40 < t \leq 60 \end{cases} \tag{3.119}$$

The membership function of VL is:

$$VL(t) = (t - 40)/20, 40 < t \leq 60 \tag{3.120}$$

The fuzzy rules in the knowledge base are: if more sludge and more grease, the longer the washing time; if the sludge is moderate and the grease is moderate, the washing time is moderate; if the sludge is less and the grease is less, the washing time is shorter. Table 3.10 gives all the fuzzy rules. It is clear that there are 9 fuzzy rules in the knowledge base of the fuzzy control system of the washing machine.

For the input variables $x = 60$, $y = 70$, try to find the output quantity t.

Solution We solve the above problem with the help of Matlab's Fuzzy Logic Designer, which can be opened by clicking Fuzzy Logic Designer from the App in Matlab.

The interface after opening the fuzzy logic designer is shown in Fig. 3.7. Under this interface, there is one input variable (located in the top left of the figure) and one output variable (located in the top right of the figure) by default. Users can also modify the name of the current variable, which is changed to sludge in Fig. 3.7.

Table 3.10 Fuzzy rules for fuzzy control system of washing machine

x	SG	MG	LG
SD	VS	M	L
MD	S	M	L
LD	M	L	VL

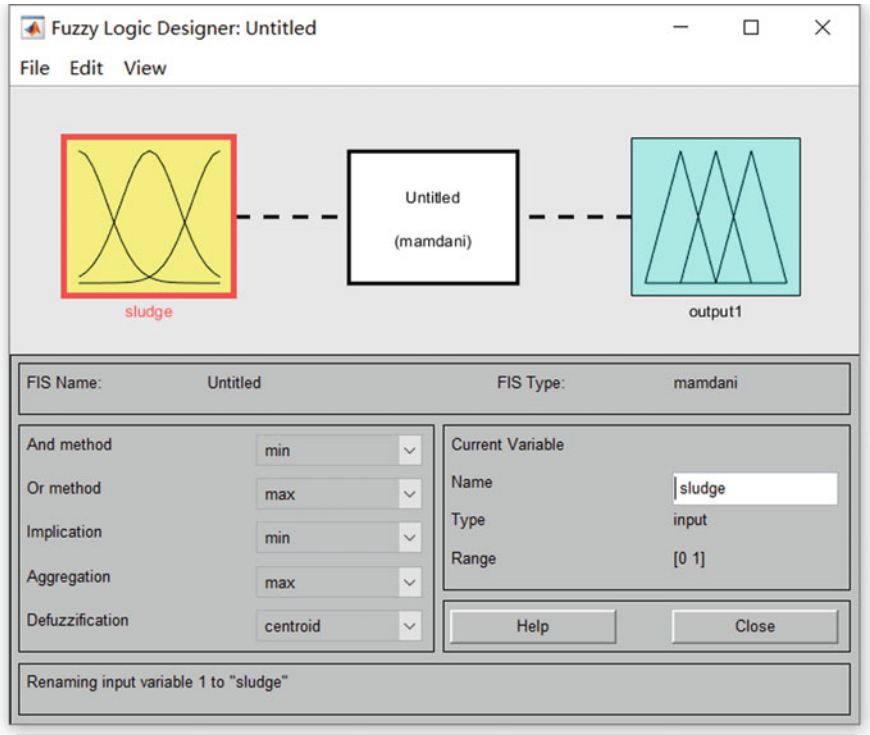

Fig. 3.7 Adding sludge in fuzzy logic designer

Double-clicking the icon of sludge with the mouse will bring up the dialog box shown in Fig. 3.8, in which you can modify the theoretical domain and membership function of sludge. We name the membership functions of sludge as SD, MD and LD, respectively.

Close the membership function dialog box of sludge. We have to add another input variable by selecting Add Variable under the Edit menu of the fuzzy logic designer, and then clicking Input. Name the newly added input variable as grease. Users can follow the same operation as sludge to add the membership function of grease to get the result as shown in Fig. 3.9.

From the bottom left of the Fig. 3.9, we can see that the fuzzy conditional statement connected with "and" uses "min", which is the intersection operation. The fuzzy conditional statement connected with "also" uses "max". These are the same as the fuzzy control system example in the previous section. When solving the fuzzy control problem of the washing machine, these default parameters are also used, but the user can choose other settings as needed. The default defuzzification is "centroid", i.e., weighted average. It is modified to "mom", i.e., the average maximum membership method.

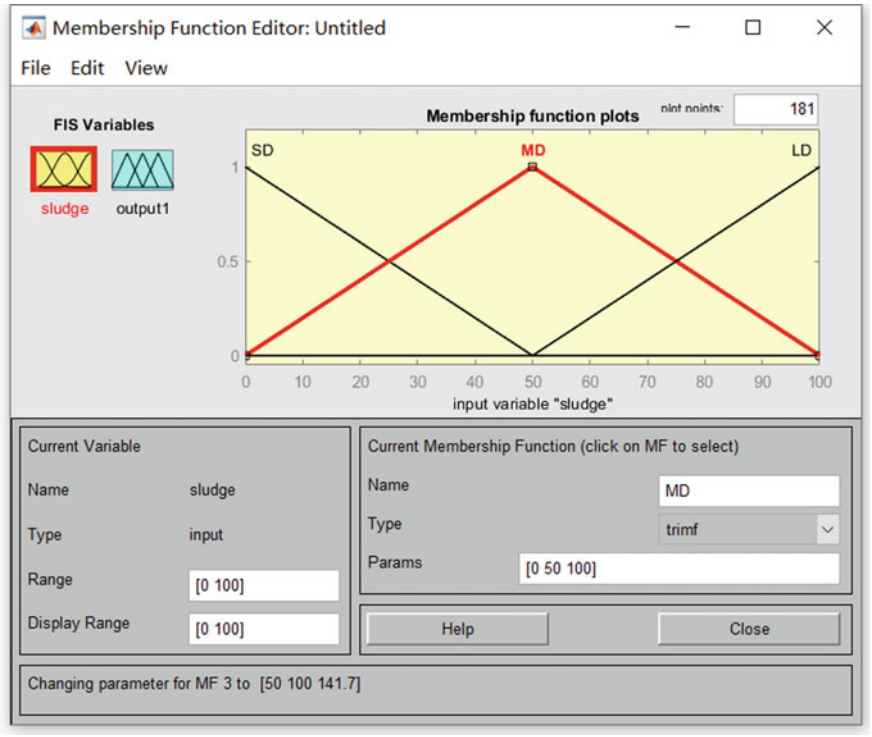

Fig. 3.8 Adding membership function for sludge

Then modify the name of the output variable as washing time. We can open the subordinate function dialog box by double-clicking the icon of the output variable. Then we modify the domain of washing time as [0, 60], add 5 subordinate functions according to the requirements of the example, and then close the subordinate function dialog box to get the result as shown in Fig. 3.10.

Next, we add the rules of fuzzy control. Under the fuzzy controller dialog box, double click the mamdani icon in the middle to open the rule editing dialog box. We can add the 9 fuzzy control rules in the example. This completes the design of the washing machine fuzzy control system. Click the File menu, select Export, and save the file as "washingMachineConrol.fis".

To calculate the output variable t for the input variables $x = 60$, $y = 70$, under the View menu of the fuzzy control designer, click on Rules to open the Rules view, as shown in Fig. 3.11. In this dialog box, at the bottom left, enter [60; 70], and you can see the washing time is about 24.9 at the top right.

The problem of fuzzy control of the washing time of a washing machine can also be programmed as follows:

```
fis=mamfis('Name','washingMachineControl');
var1=fisvar([0,100],'Name','sludge');
```

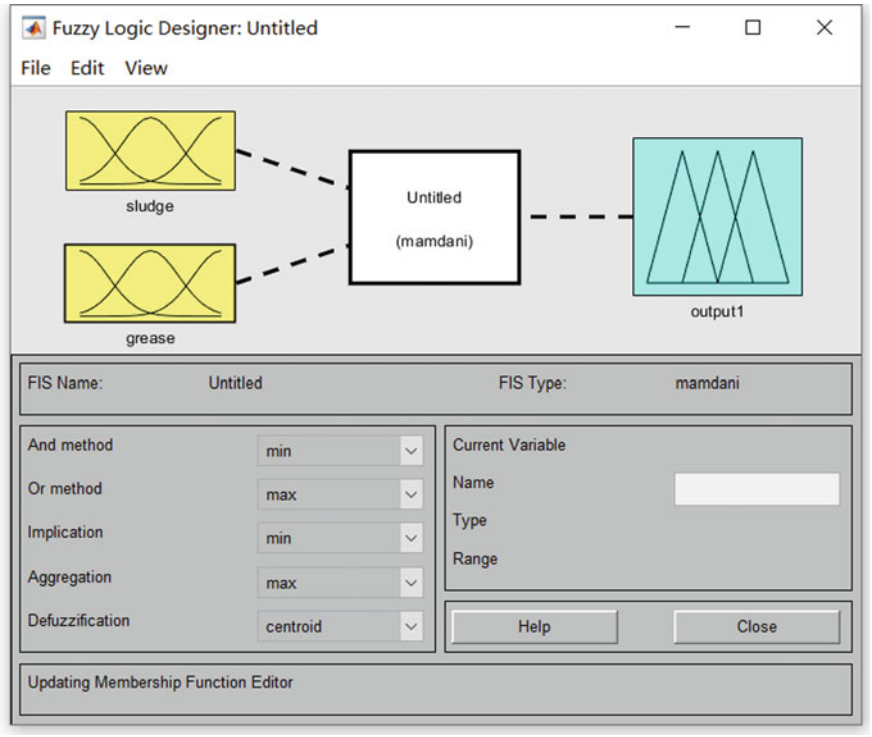

Fig. 3.9 Adding membership function for grease

```
var1=addMF(var1,'trimf', [-50,0,50],'Name','SD');
var1=addMF(var1,'trimf', [0,50,100],'Name','MD');
var1=addMF(var1,'trimf', [50,100,150],'Name','LD');
var2=fisvar([0,100],'Name','grease');
var2=addMF(var2,'trimf', [-50,0,50],'Name','SG');
var2=addMF(var2,'trimf', [0,50,100],'Name','MG');
var2=addMF(var2,'trimf', [50,100,150],'Name','LG');
fis.Inputs = [var1,var2];
var3=fisvar([0,60],'Name','washingtime');
var3=addMF(var3,'trimf', [-10,0,10],'Name','VS');
var3=addMF(var3,'trimf', [0,10,25],'Name','S');
var3=addMF(var3,'trimf', [10,25,40],'Name','M');
var3=addMF(var3,'trimf', [25,40,60],'Name','L');
var3=addMF(var3,'trimf', [40,60,80],'Name','VL');
fis.Outputs=var3;
rulelist=[1,1,1,1,1;
   1,2,3,1,1;
   1,3,4,1,1;
```

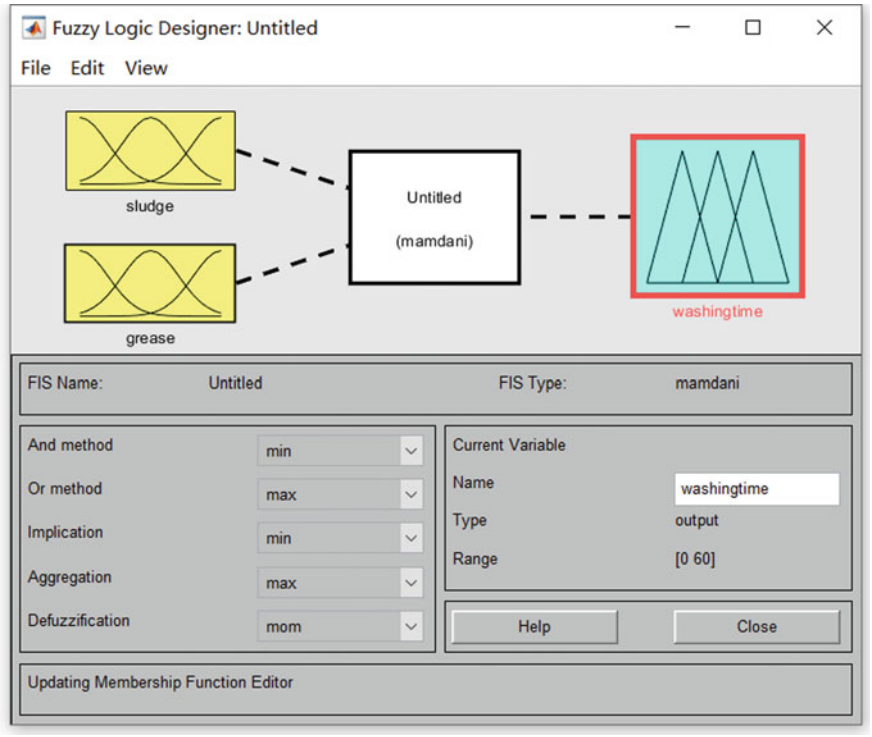

Fig. 3.10 Adding washing time in fuzzy logic designer

```
    2,1,2,1,1;
    2,2,3,1,1;
    2,3,4,1,1;
    3,1,3,1,1;
    3,2,4,1,1;
    3,3,5,1,1];
fuzzyRules=fisrule(rulelist,2);
fuzzyRules=update(fuzzyRules,fis);
fis.Rules=fuzzyRules;
fis.DefuzzificationMethod='mom';
x=60;
y=70;
t=evalfis(fis,[x,y]);
showrule(fis,1:2,'verbose');
figure(1);
plotfis(fis);
figure(2); hold on; box on; grid on;
subplot(2,1,1)
```

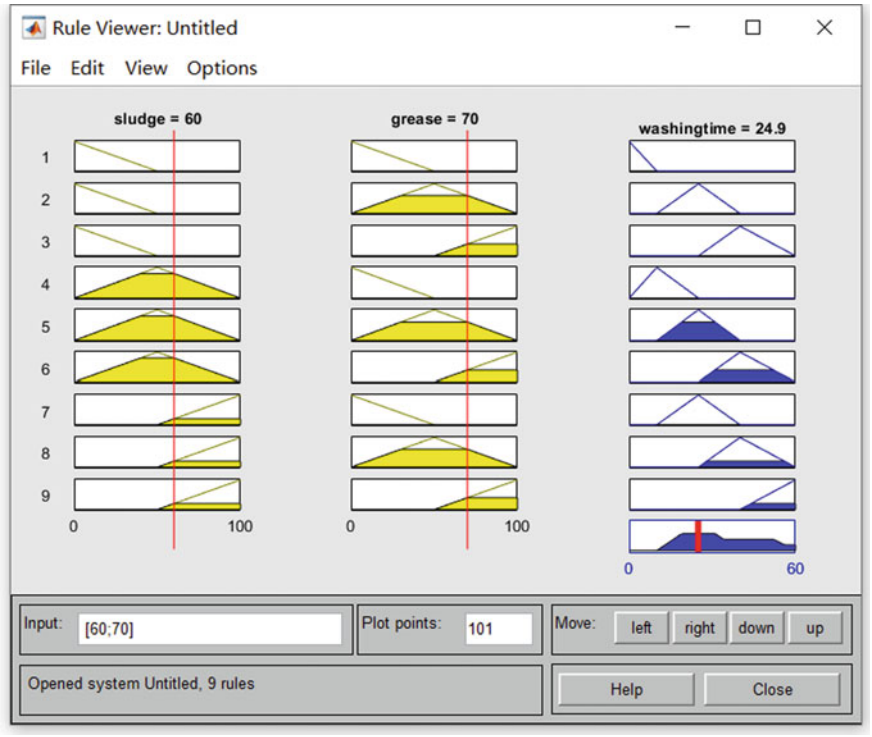

Fig. 3.11 Rule viewer to calculate the output variable

```
plotmf(fis,'input',1)
subplot(2,1,2)
plotmf(fis,'output',1)
hold off;
figure(3);
gensurf(fis)
title('function mapping curve of input and output')
```

After running the above program, the results of the output variable are stored in the variable *t*. The program will draw three figures. The first one is the model graph of the fuzzy control system as shown in Fig. 3.12. The graph shows that the inputs are sludge and grease with 3 membership functions for each input, the washing time is the output with 5 affiliation functions, and the model is of Mandani type with 9 fuzzy rules.

The second figure drawn by the above program is the membership functions of the input and output variables, as shown in Fig. 3.13. It can be seen that the three membership functions of sludge are triangular. The five membership functions of washing time are also triangular.

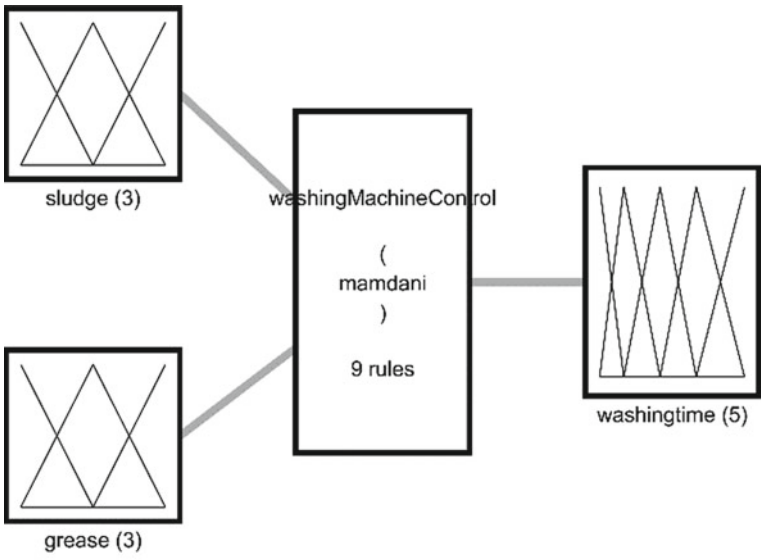

Fig. 3.12 Model of fuzzy control system for washing machine

The third one plotted by the above program is the mapping surface diagram of input and output variables, as shown in Fig. 3.14. It can be seen that the washing time is a function of sludge and grease. It is getting longer as the sludge and grease increase. Since fuzzy logic is used for control, the change of washing time shows a certain slope, which increases the stability of the washing machine control system.

The Example 3.15 shows the usage of fuzzy logic designer and programs to design a controller for washing machine. Users can choose either way to solve their control problems.

Exercises

(1) Suppose the domain U is the age of a person with the range (0, 100]. There are 3 classes of patterns young A_1, middle-aged A_2 and old A_3 on this domain. For a certain variable $\mu = 35$, please use the principle of maximum membership to determine the class to which μ belongs, where:

$$A_1(\mu) = \begin{cases} 1, & 0 < \mu \leq 20 \\ 1 - 2\left(\frac{\mu-20}{20}\right)^2, & 20 < \mu \leq 30 \\ 2\left(\frac{\mu-40}{20}\right)^2, & 30 < \mu \leq 40 \\ 0, & 40 < \mu \leq 100 \end{cases}$$

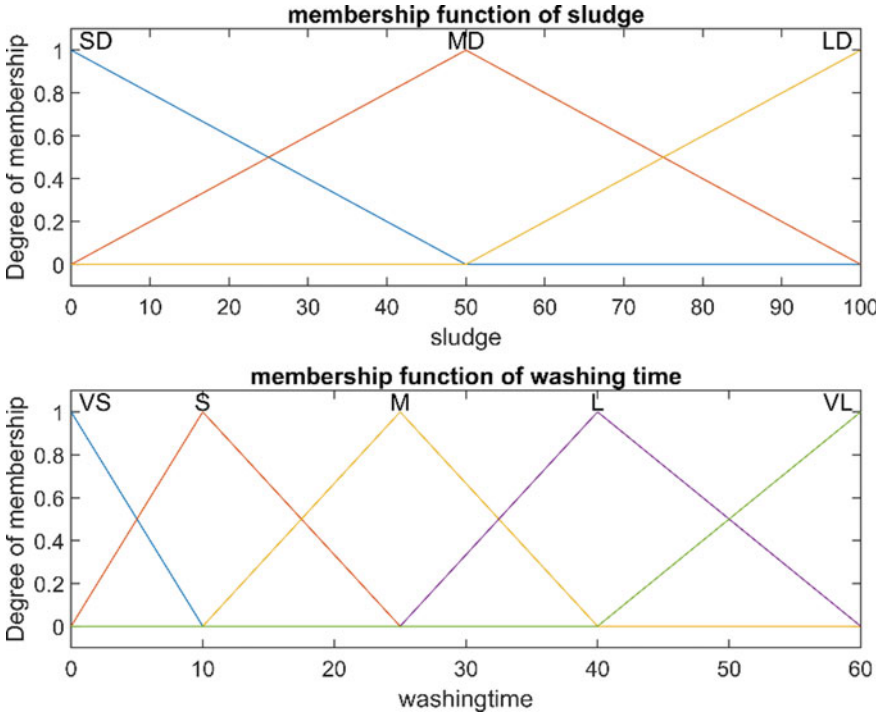

Fig. 3.13 Membership function of sludge and washing time

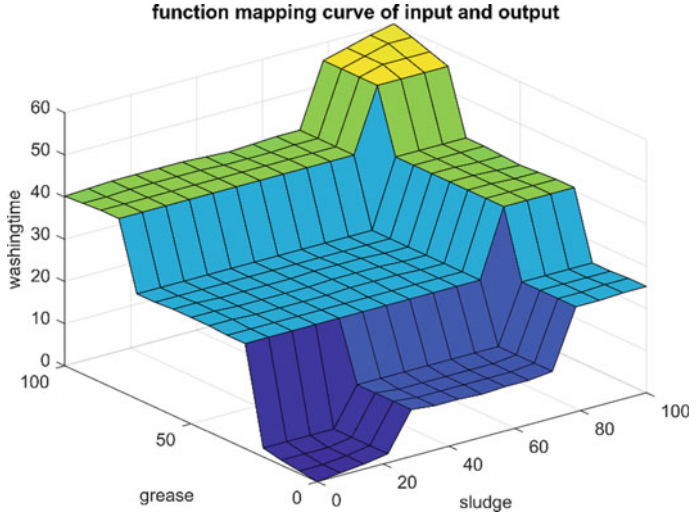

Fig. 3.14 Function mapping curve of input and output

$$A_2(\mu) = \begin{cases} 0, & 0 < \mu \le 20 \\ 2\left(\frac{\mu-20}{20}\right)^2, & 20 < \mu \le 30 \\ 1 - 2\left(\frac{\mu-40}{20}\right)^2, & 30 < \mu \le 40 \\ 1, & 40 < \mu \le 50 \\ 1 - 2\left(\frac{\mu-50}{20}\right)^2, & 50 < \mu \le 60 \\ 2\left(\frac{\mu-70}{20}\right)^2, & 60 < \mu \le 70 \\ 0, & 70 < \mu \le 100 \end{cases}$$

$$A_3(\mu) = \begin{cases} 0, & 0 < \mu \le 50 \\ 2\left(\frac{\mu-50}{20}\right)^2, & 50 < \mu \le 60 \\ 1 - 2\left(\frac{\mu-70}{20}\right)^2, & 60 < \mu \le 70 \\ 1, & 70 < \mu \le 100 \end{cases}$$

(2) Suppose the domain $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ is the quality of tea leaves, where the elements are stripe, color, clarity, soup color, aroma and taste. There are five categories of patterns A_1, A_2, A_3, A_4 and A_5 on this domain. For a certain tea B to be identified, please use the nearest principle to determine the category to which B belongs, where: $A_1 = (0.5, 0.4, 0.3, 0.6, 0.5, 0.4)$, $A_2 = (0.3, 0.2, 0.2, 0.1, 0.2, 0.2)$, $A_3 = (0.2, 0.2, 0.2, 0.1, 0.1, 0.2)$, $A_4 = (0, 0.1, 0.2, 0.1, 0.1, 0.1)$, $A_5 = (0, 0.1, 0.1, 0.1, 0.1, 0.1)$, $B = (0.4, 0.2, 0.1, 0.4, 0.5, 0.6)$.

(3) If we want to use fuzzy logic method to study the effect of the length of queuing time on passenger satisfaction at railroad stations. We can define "time" as the input variable and "satisfaction" as the output variable. The domain of the input time is set to $U = [5, 60]$, and its membership function has three fuzzy sets on the domain, which are short time (ST), medium time (MT), and long time (LT). The membership functions of ST, MT and LT are:

$$ST(t) = \begin{cases} 1, & x < 10 \\ \frac{15-x}{5}, & 10 \le x \le 15 \\ 0, & x > 15 \end{cases}$$

$$MT(t) = \begin{cases} 0, & x < 13 \\ \frac{x-13}{6}, & 13 \le x < 19 \\ 1, & 19 \le x < 27 \\ \frac{35-x}{8}, & 27 \le x < 35 \\ 0, & x \ge 35 \end{cases}$$

$$LT(t) = \begin{cases} 0, & x < 25 \\ \frac{x-25}{17}, & 25 \le x \le 42 \\ 1, & x > 42 \end{cases}$$

The theoretical domain of the output variable time is set to $U = [0, 10]$. The output variable time has three fuzzy sets. They are higher satisfaction (HS), general satisfaction (GS), and poor satisfaction (LS), whose membership functions are:

$$HS(s) = \begin{cases} \frac{5-s}{5}, & 0 \le s \le 5 \\ 0, & s > 5 \end{cases}$$

$$GS(s) = \begin{cases} 0 & s < 3 \\ \frac{s-3}{2}, & 3 \le s \le 5 \\ \frac{s-5}{3}, & 5 \le s \le 8 \\ 0, & s > 8 \end{cases}$$

$$LS(s) = \begin{cases} 0, & s < 7 \\ \frac{s-7}{3}, & 7 \le s \le 10 \end{cases}$$

The fuzzy logic rules are:

(i) if the waiting time is short, the passenger satisfaction is high;
(ii) if the waiting time is medium, the passenger satisfaction is average;
(iii) if the waiting time is long, the passenger satisfaction is poor.

The defuzzification uses the average maximum membership method. Try to find what is the passenger satisfaction when the passenger waiting time is 10. Draw the mapping curve between the input and output variables.

References

1. Zadeh LA (2012) Fuzzy Logic. In: Meyers R (eds) Computational complexity. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-1800-9_73
2. Deng F, Chen W (2020) Intelligent computing and information processing. Beijing Institute of Technology Press, Beijing
3. Wei X, Guo J (2020) Foundations of Computational Intelligence. https://www.icourse163.org/course/NJTU-1207221803. Accessed 17 July 2023
4. Chiu SL (1994) Fuzzy model identification based on cluster estimation. J Intell Fuzzy Syst 2, 267–278
5. Mathur N, Glesk I, Buis A (2016) Comparison of adaptive neuro-fuzzy inference system (ANFIS) and Gaussian processes for machine learning (GPML) algorithms for the prediction of skin temperature in lower limb prostheses. Med Eng Phys 38:1083–1089
6. Sepúlveda R, Castillo O, Melin P et al (2007) Experimental study of intelligent controllers under uncertainty using type-1 and type-2 fuzzy logic. Inf Sci 177(10):2023–2048
7. Zadeh LA (1997) The roles of fuzzy logic and soft computing in the conception, design and deployment of intelligent systems. In: Nwana, HS, Azarmi, N (eds) Software agents and soft computing towards enhancing machine intelligence. Lecture Notes in Computer Science, vol 1198. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-62560-7_45
8. Yager RR, Zadeh LA (2012) An introduction to fuzzy logic applications in intelligent systems. Springer, New York, NY. https://doi.org/10.1007/978-1-4615-3640-6

Chapter 4
Fuzzy Neural Network

Abstract Fuzzy neural network combines fuzzy computing and artificial neural network. Fuzzy neural network inherits the characteristics of both fuzzy logic and neural network such as logical reasoning ability, adaptive ability and learning ability. This chapter first gives an overview of fuzzy neural network including Takagi-Sugeno fuzzy system and expert system. Adaptive network-based fuzzy inference system is introduced to illustrate the usage of fuzzy neural network. Then fuzzy neural network is used to solve time series prediction problem. Interval type-2 fuzzy logic is presented and its performance is studied through time series prediction problem. Fuzzy neural network is then applied to solve clustering problem and suburban commuting prediction problem. Finally, the state-of-the-art research progress of fuzzy computing is presented.

4.1 Overview of Fuzzy Neural Network

The diversity of neural network topology makes it have strong adaptive ability and learning ability, but the disadvantage of neural network is poor interpretability. Neural network is like a black box, unable to explain its theoretical basis for solving problems. Fuzzy logic has a strong logical reasoning ability and expresses knowledge through rules, which makes it more interpretable. However, the rules of fuzzy logic need to be obtained according to the knowledge or experience of experts, resulting in poor adaptive ability. Fuzzy neural network (FNN) is a combination of fuzzy logic and neural network, which aims to form a method with extremely logical reasoning ability, adaptive ability and learning ability.

FNN originated in the 1980s, because fuzzy logic has been successfully applied to the control problems in the home appliance industry, Japanese researchers first started the study of fuzzy neural network [1]. Their purpose is to use the learning ability of neural network to design fuzzy control system. In the 1990s, researchers in the United States also began research in this area, especially the research group led by Zadeh. These researchers promoted the development of FNN, making FNN quickly recognized by the industry.

X. Zhang et al., *Intelligent Information Processing with Matlab*,
https://doi.org/10.1007/978-981-99-6449-9_4

After years of development, the combination of fuzzy logic and neural network can be structurally divided into neuro-fuzzy system, fuzzy-neural system and fuzzy-neural hybrid system.

(1) Neuro-fuzzy system. It refers to the output of the neural network connected to the input of the fuzzy logic, similar to a series structure, using the neural network to adjust the parameters of the fuzzy system, the weight of the neural network can represent the fuzzification function, membership function and defuzzification function and other fuzzy logic calculation required parameters. A neuro-fuzzy system is a kind of fuzzy system.
(2) Fuzzy-neural system. It means that the output of fuzzy logic is connected to the input of neural network, fuzzy logic is used to fuzzify the parameters or weights of neural network, and the neurons in neural network are fuzzy neurons. Fuzzy-neural system is a kind of neural network.
(3) Fuzzy-neural hybrid system. It refers to the mixed use of fuzzy logic and neural network technology, generally the two technologies are independent of each other. The common hybrid system is Adaptive Network-based Fuzzy Inference System (ANFIS).

In terms of structure and function, FNN can be divided into:

(1) Fuzzy systems with learning ability, also known as trainable fuzzy systems. Based on the input sample data, the system can use the learning algorithm of neural network to learn the sample data, obtain the fuzzy rules corresponding to the data, and finally build a fuzzy system.
(2) Fuzzy system based on neural network. The system is a reasoning system of fuzzy logic, and researchers have created a variety of neural network structures to represent fuzzy inference, that is, fuzzy inference is expressed in the form of neural networks.
(3) Fuzzy neural network. It is essentially a neural network. The input, weight and output of the network may be fuzzy values, so as to obtain a neural network that can perform fuzzy computing.

Recalling the Mandani fuzzy system introduced in the last chapter, its fuzzy rule form is: if x is A_i and y is B_i, then z is C_i, where each fuzzy rule is "also" connected. In addition to Mandani fuzzy systems, Takagi-Sugeno fuzzy systems and expert systems are also commonly used fuzzy controllers. Takagi-Sugeno fuzzy system is also abbreviated as T-S fuzzy system, and its fuzzy rules are as follows:

$$R_i : \text{if } e = A_i \text{ and } \Delta e = B_i, \text{ then } z_i = \alpha_i e + \beta_i \Delta e + \gamma_i \tag{4.1}$$

where R_i denotes the i-th rule, e denotes the error, Δe denotes the change rate of the error, A_i and B_i are the given fuzzy sets, α_i, β_i and γ_i are the parameters of the fuzzy systems.

It can be seen that the output variable of the T-S fuzzy system is a linear function of the input variable and is calculated to obtain the clear value, so the system does not have a defuzzification module. Assuming that the T-S fuzzy system contains n rules

and the weight of the i-th rule is denoted as ω_i, the output can be calculated using the weighted summation method or the weighted average method. The expression of the weighted summation method is:

$$\delta = \sum_{i=1}^{n} \omega_i z_i \tag{4.2}$$

where δ represents the total output of the system. The expression for the weighted average method is:

$$\delta = \sum_{i=1}^{n} \omega_i z_i / \sum_{i=1}^{n} \omega_i \tag{4.3}$$

We cannot construct a FNN equivalent to the Mandani fuzzy system, only an approximate construction, but we can construct a FNN equivalent to a T-S fuzzy system or an expert system. The FNN corresponding to the above T-S fuzzy system is shown in Fig. 4.1. From the figure, we can see that the T-S fuzzy system is represented as a neural network structure, where the weights are calculated from the fuzzy set. The T-S fuzzy system in Fig. 4.1 uses "and" operation, i.e., $\min(A_i, B_i)$.

The above T-S fuzzy system consists of one input layer, three hidden layers and one output layer. Since one hidden layer is skipped from the weight ω_i calculation, the T-S fuzzy system can also be said to have two hidden layers. There are four neurons from the input layer to the first hidden layer. The first neuron computes $z_1 = \alpha_1 e + \beta_1 \Delta e + \gamma_1$, the second neuron computes $z_2 = \alpha_2 e + \beta_2 \Delta e + \gamma_2$, the third neuron computes $\omega_1 = \min(A_1, B_1)$, and the fourth neuron computes $\omega_2 = \min(A_2, B_2)$. The second hidden layer has three neurons. The first neuron computes $\omega_1 z_1$, the second neuron computes $\omega_2 z_2$, and the third neuron computes $\omega_1 + \omega_2$. Then $\omega_1 z_1 + \omega_2 z_2$ is computed, and finally the output layer computes δ. It can be seen that the weights between the second layer to the output layer in the T-S type fuzzy neural network are all 1, which is a simplified neural network model.

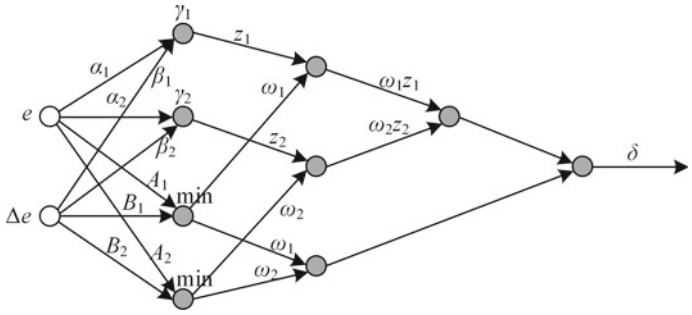

Fig. 4.1 T-S fuzzy system

By properly constructing the topology and activation function of the neural network, we can obtain the neural network model equivalent to the T-S fuzzy system. Then we can use the theory of neural network to analyze the T-S fuzzy controller. For example, for a certain control problem, data are first collected experimentally to form a training set. We then can learn the training dataset by the method of neural network. The trained T-S fuzzy controller can be employed to the associated control problem.

Next, we introduce the neural network model of the fuzzy expert system. The rule form of the fuzzy expert system is as follows:

$$R_i : \text{if } e = A_i \text{ and } \Delta e = B_i, \text{then the control action is } c_i \tag{4.4}$$

where R_i denotes the i-th rule, e denotes the error, Δe denotes the change rate of the error, A_i and B_i is the given fuzzy set, c_i is the triangle fuzzy function. It can be seen that the output variable of the fuzzy expert system is a certain control action. The control action is a clear value, so the system does not have a defuzzification module. Assuming that the fuzzy expert system contains n rules and the weight of the i-th rule is noted as ω_i, the output can be calculated using the weighted average method.

The fuzzy expert system corresponding to the FNN is shown in Fig. 4.2. From the figure, it can be seen that the fuzzy expert system is represented as a neural network structure. In the system, the weights ω_i are calculated from the fuzzy set and the operations used are $\min(A_i, B_i)$. The fuzzy expert system consists of an input layer, two hidden layers and an output layer. There are two neurons from the input layer to the first hidden layer. The first neuron is to calculate the weights $\omega_1 = \min(A_1, B_1)$ and the second neuron is to calculate $\omega_2 = \min(A_2, B_2)$. The second hidden layer has two neurons. The first neuron is to compute $\omega_1 z_1 + \omega_2 z_2$. The second neuron is to compute $\omega_1 + \omega_2$, and the weight of the second neuron is 1. Finally, the output layer computes δ. It can be seen that the weights between the second hidden layer to the output layer in the fuzzy expert neural network are all 1. It is also a simplified neural network model. The fuzzy expert system has a more concise neural network structure compared to the T-S type fuzzy system.

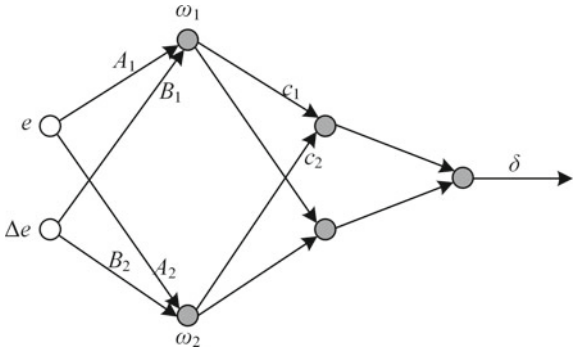

Fig. 4.2 Fuzzy expert system

Example 4.1 Suppose a two-input single-output T-S type fuzzy system needs to be constructed with two input variables, namely x and y. The fuzzy sets of input variables are shown in Table 4.1. We can construct the corresponding input variables according to the parameters in Table 4.1.

The output variable is z. The rules from input to output are as follows:

$$R_1 : \text{if } x = A_1 \text{ and } y = B_1, \text{ then } z_1 = -x + 2y \tag{4.5}$$

$$R_2 : \text{if } x = A_2 \text{ and } y = B_1, \text{ then } z_1 = 8x - 4y + 1 \tag{4.6}$$

$$R_3 : \text{if } x = A_2 \text{ and } y = B_2, \text{ then } z_1 = 3x + 9y + 1 \tag{4.7}$$

$$R_4 : \text{if } x = A_3, \text{ then } z_1 = 5x + 1 \tag{4.8}$$

Try to build a T-S type fuzzy system satisfying the above rules, and compute the value z of give $x = 6$ and $y = 7$.

Solution The example can be solved by two ways. The first way is using fuzzy logic designer. T-S type fuzzy system can be designed in an interactive manner using the fuzzy logic designer. The usage of fuzzy logic designer has been introduced in the last chapter; thus, it is not described here.

The second way is using the following programs to solve the example:

```
fis = sugfis('Name','SugenoExample');
var1 = fisvar([0,10],'Name','X');
var1 = addMF(var1,'trimf', [0,0,4],'Name','x1');
var1 = addMF(var1,'trapmf', [2,4,6,8],'Name','x2');
var1 = addMF(var1,'trimf', [6,10,10],'Name','x3');
var2 = fisvar([0,10],'Name','Y');
var2 = addMF(var2,'trimf', [0,0,7],'Name','y1');
var2 = addMF(var2,'trimf', [3, 10],'Name','y2');
fis.Inputs = [var1,var2];
var3 = fisvar([0,120],'Name','Z');
var3 = addMF(var3,'linear', [- 1,2,0],'Name','z1');
```

Table 4.1 Input variables and their fuzzy sets

Variable	Domain	Fuzzy set	Membership	Parameter
x	[0, 10]	A_1	Triangle	[0, 0, 4]
x	[0, 10]	A_2	Trapezoid	[2, 4, 6, 8]
x	[0, 10]	A_3	Triangle	[6, 10, 10]
y	[0, 10]	B_1	Triangle	[0, 0, 7]
y	[0, 10]	B_2	Triangle	[3, 10, 10]

```
var3 = addMF(var3,'linear', [8,− 4,1],'Name','z2');
var3 = addMF(var3,'linear', [1,3,9],'Name','z3');
var3 = addMF(var3,'linear', [5,0,1],'Name','z4');
fis.Outputs = var3;
rulelist = [1,1,1,1,1;
    2,1,2,1,1;
    2,2,3,1,1;
    3,0,4,1,1];
fuzzyRules = fisrule(rulelist,2);
fuzzyRules = update(fuzzyRules,fis);
fis.Rules = fuzzyRules;
x = 6;
y = 7;
z = evalfis(fis,[x,y]);
showrule(fis,1:2,'verbose');
figure(1);
plotfis(fis);
figure(2);
gensurf(fis)
```

The running results of the above program are shown in Figs. 4.3, 4.4 and 4.5.

As can be seen in Fig. 4.3, the names of the two input variables are defined as X and Y, where X has three fuzzy membership functions and Y has two fuzzy membership functions. The name of this T-S type fuzzy system is "SugenoExample" with four

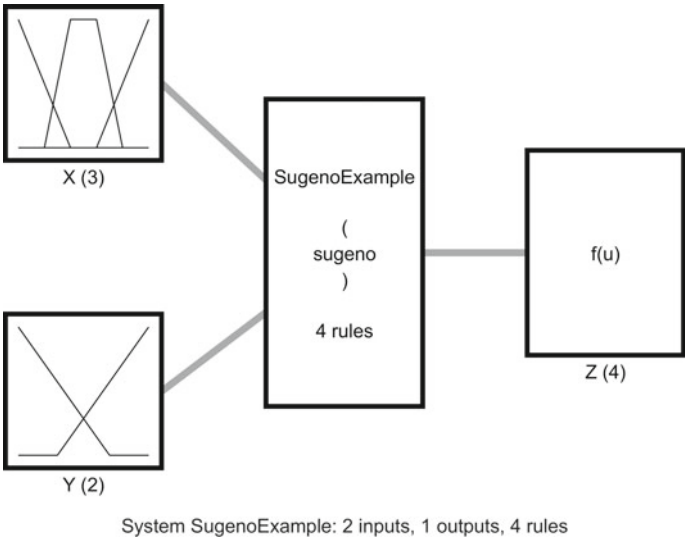

Fig. 4.3 T-S type fuzzy system for Example 4.1

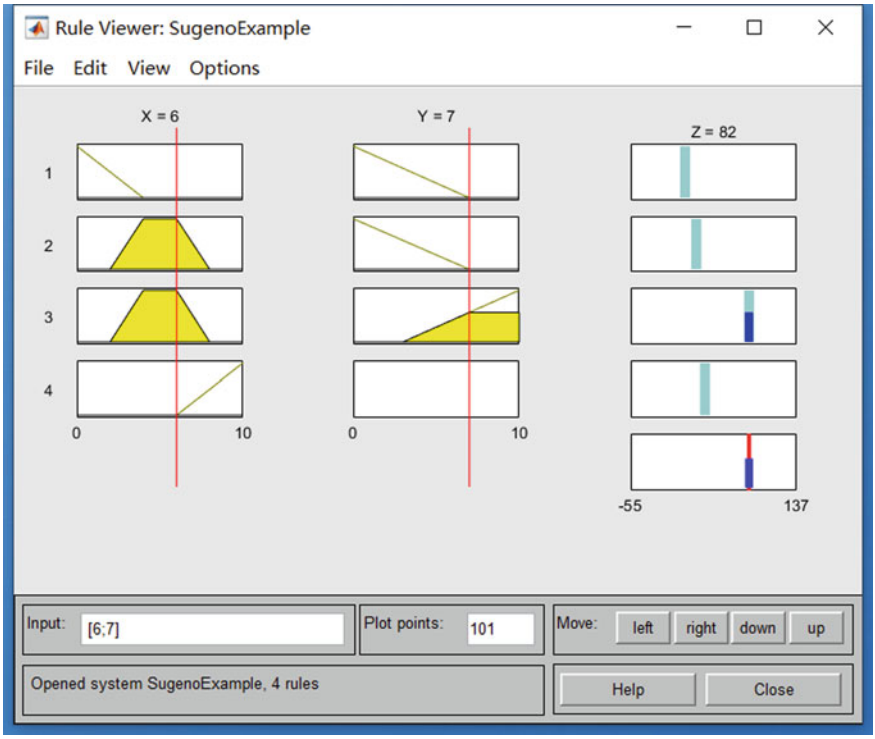

Fig. 4.4 Fuzzy rules for Example 4.1

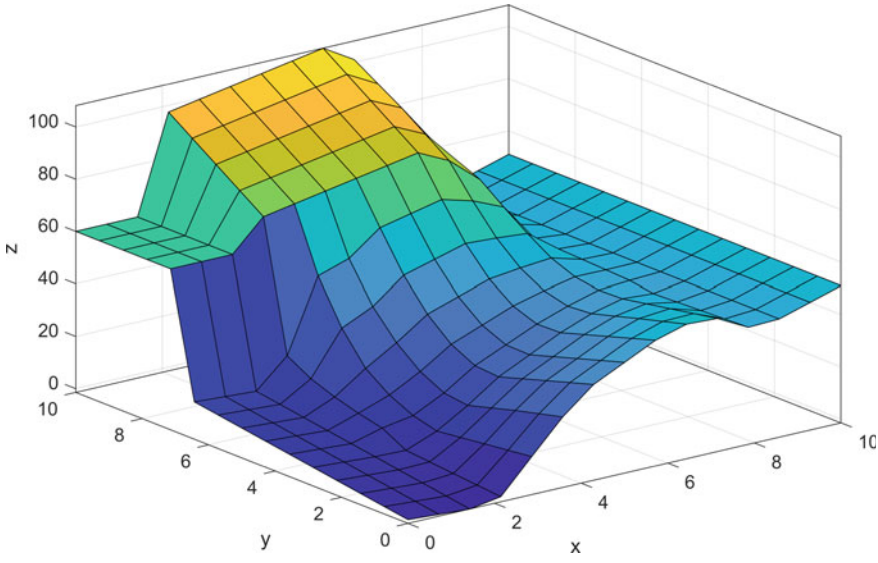

Fig. 4.5 Function mapping curve for Example 4.1

fuzzy rules, while the name of the output variable is defined as Z. The output variable is composed of four functions, each of which is a linear function of the input variables.

The fuzzy rules of the Example 4.1 can be seen in Fig. 4.4, from which the process of fuzzy reasoning by the T-S model can be observed. When the input $x = 6$, $y = 7$, when the output $z = 82$.

Figure 4.5 gives the surface plot of the relationship between two input and output variables. It can be seen from the figure that the mapping curve is not very smooth. There are large fluctuations and drastic changes in some places, which reflects that the system is not perfect. In this case, it is generally necessary to add more fuzzy rules to accumulate more empirical knowledge, so that the unsmooth areas can be eliminated and the whole mapping relationship tends to be continuous and smooth. However, with the increase of fuzzy rules, it makes the maintenance of fuzzy systems more and more complicated and reduces the interpretability of the system. In practical applications, we need to consider the modeling of fuzzy systems from several aspects to achieve a satisfactory balance.

4.2 Adaptive Fuzzy Neural Inference System

In 1993, Jang proposed the Adaptive Network-based Fuzzy Inference System (ANFIS), which combines the learning capability of neural networks with the logical reasoning capability of fuzzy computing [2]. It is a T-S type fuzzy neural network, which has been widely used in control problems because the system is very effective. Compared with the Mandani fuzzy system introduced earlier, the ANFIS fuzzy system outperforms the Mandani fuzzy system when using non-triangular and non-trapezoidal affiliation functions.

We present the ANFIS fuzzy system with two fuzzy rules as an example:

$$R_1 : \text{if } x = A_1 \text{ and } y = B_1, \text{then } z_1 = a_1 x + b_1 y + c_1 \tag{4.9}$$

$$R_2 : \text{if } x = A_2 \text{ and } y = B_2, \text{then } z_2 = a_2 x + b_2 y + c_2 \tag{4.10}$$

where $R_i, i \in \{1, 2\}$ denotes the i-th rule, x and y denotes two linguistic variables, A_i and B_i are the given fuzzy sets, a_i, b_i and c_i are the parameters of the fuzzy system.

This ANFIS fuzzy system is shown in Fig. 4.6, which shows that it is a five-layer neural network structure containing one input layer, four hidden layers and one output layer. In this figure, the first hidden layer and the fourth hidden layer are represented by square blocks, which is because these two layers contain adjustable parameters. While the second hidden layer, the third hidden layer and the output layer are represented by circular blocks, which is because these three layers do not contain adjustable parameters. For the nodes with adjustable parameters represented by square blocks, the learning algorithm of the neural network can be used, and thus determine the final fuzzy system.

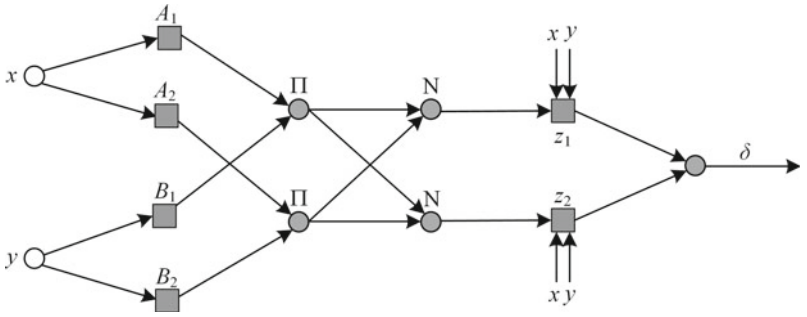

Fig. 4.6 ANFIS fuzzy neural network

In the above ANFIS fuzzy system, the first hidden layer is to fuzzify the input variables. The first two neurons in this layer are to calculate the fuzzy set of variable x. The output value of the first layer is obtained after calculating:

$$o_i^1(x) = \mu_{A_i}(x), i \in \{1, 2\} \tag{4.11}$$

The last two neurons in the first layer are to compute the fuzzy set of variable y, which are computed to obtain the output value of the layer:

$$o_i^1(y) = \mu_{B_i}(y), i \in \{1, 2\} \tag{4.12}$$

where o_i^1 indicates the result obtained by a certain neuron calculation, the number in the right superscript indicates that it is the first hidden layer, and the right subscript indicates the i-th fuzzy rule. It should be noted that there are several forms of membership functions to choose. Different values will be obtained by using different membership functions. The parameters in the chosen membership function are generally called conditional parameters. For example, if a Gaussian-type membership function is used:

$$\mu_{A_i}(x) = \exp\left(-\frac{\|x - d_i\|^2}{\sigma_i^2}\right), \ i \in \{1, 2\} \tag{4.13}$$

where d_i and σ_i are the condition parameters.

The second hidden layer has two neurons. The two neurons perform multiplication operations of the results of the previous layer. Other forms of operations can also be used, here we take multiplication as an example:

$$o_i^2 = \omega_i = \mu_{A_i}(x) \times \mu_{B_i}(x), i \in \{1, 2\} \tag{4.14}$$

where o_i^2 denotes the result obtained from the calculation of a certain neuron. The number in the right superscript indicates that it is the second hidden layer, the right

subscript indicates the i-th fuzzy rule. The result of the multiplication operation is denoted as ω_i.

The third hidden layer has two neurons. The two neurons normalize the result of the previous layer. The normalized value indicates the confidence of a fuzzy rule:

$$o_i^3 = \overline{\omega}_i = \omega_i / (\omega_1 + \omega_2), i \in \{1, 2\} \tag{4.15}$$

where o_i^3 indicates the result obtained from a certain neuron calculation. The number in the right superscript indicates that it is the third hidden layer, the right subscript indicates the i-th fuzzy rule. The result of the normalization operation is denoted as $\overline{\omega}_i$.

The fourth hidden layer has two neurons and is based on the results of the previous layer to calculate the output of each fuzzy rule:

$$o_i^4 = \overline{\omega}_i z_i = \overline{\omega}_i (a_i x + b_i y + c_i), i \in \{1, 2\} \tag{4.16}$$

where o_i^4 denotes the result obtained from the computation of a certain neuron. The number in the right superscript indicates that it is the fourth hidden layer, the right subscript indicates the i-th fuzzy rule. In (4.16), a_i, b_i and c_i are called the conclusion parameters, also known as the posterior parameters. It can be seen that the two neurons in this layer use a linear function.

The fifth layer is the output layer. It fuses all the rules together and calculates the final output:

$$\delta = \sum_{i=1}^{2} \overline{\omega}_i z_i = \frac{\sum_{i=1}^{2} \omega_i z_i}{\sum_{i=1}^{2} \omega_i} \tag{4.17}$$

It can be seen that the output layer yields result equivalent to (4.9) and (4.10), which indicates that the ANFIS fuzzy system can be represented as a fuzzy neural network equivalent to it.

From the above introduction, it can be seen that the ANFIS fuzzy system includes conditional and conclusion parameters, which are determined before they can be used. We can use the BP neural network learning algorithm to learn these parameters, or we can combine the BP neural network learning algorithm with the least square estimation method to learn the parameters. Researchers have found that a mixture of the BP neural network learning algorithm and the least square estimation method is more effective for learning the parameters. The learning algorithms for the conditional and conclusion parameters are not described in detail here. Interested readers can refer to the related materials.

We compare the Mandani fuzzy system and ANFIS fuzzy system in terms of interpretability and accuracy. The fewer the parameters of a fuzzy system, the more interpretable it is; conversely, the more the parameters of a fuzzy system, the better its accuracy. The fewer the rules of a fuzzy system, the more interpretable it is; conversely, the more the rules of a fuzzy system, the better its accuracy. Mandani

fuzzy system uses fuzzy language to describe the problem, thus Mandani fuzzy system is highly interpretable; while the output of ANFIS fuzzy system is clear value, thus ANFIS fuzzy system is highly accurate. It can be seen that to construct a fuzzy system, it is necessary to consider both interpretability and accuracy, which are mutually constrained. Thus, a balance point is preferable to maximize both interpretability and accuracy metrics.

Example 4.2 Suppose there is a single-input, single-output control problem, we have collected 25 samples of this problem as the training set, and another 26 samples as the validation set. Try to construct a T-S type fuzzy system using ANFIS.

Solution For a training set of 25 samples, the programs for constructing the ANFIS fuzzy system are as follows:

```
load('fuzex1trnData.dat');
fis = anfis(fuzex1trnData);
x = fuzex1trnData(:,1);
anfisOutput = evalfis(fis,x);
figure1 = figure(1);
axes1 = axes('Parent',figure1);
hold(axes1,'on');
plot1 = plot(x,fuzex1trnData(:,2),'MarkerSize',8,'LineWidth',2,'LineStyle',
'none');
set(plot1,'DisplayName','Training Data','Marker','*','Color',[1 0 0]);
plot2 = plot(x,anfisOutput,'MarkerSize',8,'LineWidth',2,'LineStyle','none');
set(plot2,'DisplayName','ANFIS Output','Marker','o','Color',[0 0 1]);
xlabel('x');
ylabel('z');
box(axes1,'on');
set(axes1,'FontSize',14);
legend1 = legend(axes1,'show');
set(legend1,'Position',[0.15 0.79 0.25 0.10]);
hold(axes1,'off');
```

As can be seen from the above program, only five lines of code is required to load the training set, construct and evaluate ANFIS. The other codes plot the graph of the results, as shown in Fig. 4.7. In Fig. 4.7, the samples in the training set are indicated by asterisks, and the points predicted by ANFIS are indicated by circle symbols, the mean square error of ANFIS on the training set is 0.2247.

From Fig. 4.7, we can see that the points predicted by ANFIS and the points in the training set differ greatly. We can adjust the parameters of ANFIS to improve its performance. For example, ANFIS has 2 membership functions by default, and we set the number of membership functions to 4. This means we increase the parameters in the fuzzy rules and fuzzy system. We then set the number of training iterations to 50, and the required programs are as follows:

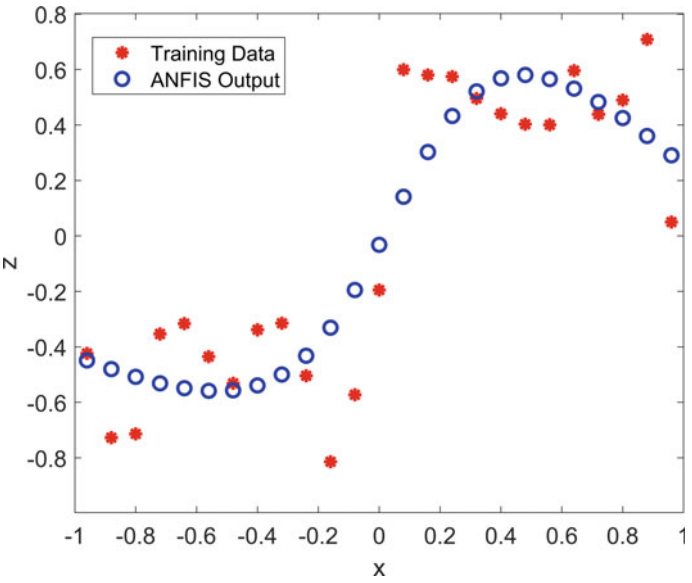

Fig. 4.7 Training samples and output results by ANFIS

```
load('fuzex1trnData.dat');
opt = anfisOptions('InitialFIS',4,'EpochNumber',50);
opt.DisplayANFISInformation = 0;
opt.DisplayErrorValues = 0;
opt.DisplayStepSize = 0;
[fis,trainError] = anfis(fuzex1trnData,opt);
fisRMSE = min(trainError);
x = fuzex1trnData(:,1);
anfisOutput = evalfis(fis,x);
figure1 = figure(1);
axes1 = axes('Parent',figure1);
hold(axes1,'on');
plot1 = plot(x,fuzex1trnData(:,2),'MarkerSize',8,'LineWidth',2,'LineStyle',
'none');
set(plot1,'DisplayName','Training Data','Marker','*','Color',[1 0 0]);
plot2 = plot(x,anfisOutput,'MarkerSize',8,'LineWidth',2,'LineStyle','none');
set(plot2,'DisplayName','ANFIS Output','Marker','o','Color',[0 0 1]);
xlabel('x');
ylabel('z');
box(axes1,'on');
set(axes1,'FontSize',14);
legend1 = legend(axes1,'show');
set(legend1,'Position',[0.15 0.79 0.25 0.10]);
hold(axes1,'off');
```

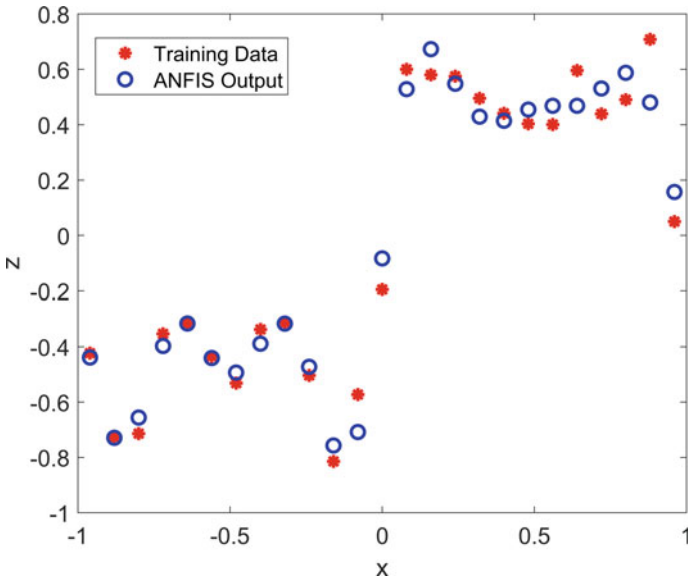

Fig. 4.8 Results of ANFIS prediction after adjusting parameters

After the above program is run, the variable "fisRMSE" stores the mean square error of ANFIS on the training set. The value of "fisRMSE" is 0.0823. We can see a larger reduction in mean square error compared to ANFIS without adjusted parameters. The predicted results are shown in Fig. 4.8. In Fig. 4.8, the samples in the training set are indicated by asterisks, and the points predicted by ANFIS are indicated by circle symbols. It can be seen from the figure that the difference has narrowed between the training samples and the output of ANFIS.

Based on the above program, we can also add the validation set. Then we can analyze the mean square error on the training and validation sets with the following programs:

```
load('fuzex1trnData.dat');
load('fuzex1chkData.dat');
opt = anfisOptions('InitialFIS',4,'EpochNumber',50);
opt.DisplayANFISInformation = 0;
opt.DisplayErrorValues = 0;
opt.DisplayStepSize = 0;
opt.ValidationData = fuzex1chkData;
[fis,trainError,stepSize,chkFIS,chkError] = anfis(fuzex1trnData,opt);
fisRMSE = min(trainError);
x = fuzex1trnData(:,1);
anfisOutput = evalfis(fis,x);
figure1 = figure(1);
axes1 = axes('Parent',figure1);
```

```
hold(axes1,'on');
plot1 = plot(x,fuzex1trnData(:,2),'MarkerSize',8,'LineWidth',2,'LineStyle',
'none');
set(plot1,'DisplayName','Training Data','Marker','*','Color',[1 0 0]);
plot2 = plot(x,anfisOutput,'MarkerSize',8,'LineWidth',2,'LineStyle','none');
set(plot2,'DisplayName','ANFIS Output','Marker','o','Color',[0 0 1]);
xlabel('x');
ylabel('z');
box(axes1,'on');
set(axes1,'FontSize',14);
legend1 = legend(axes1,'show');
set(legend1,'Position',[0.15 0.79 0.25 0.10]);
hold(axes1,'off');
figure2 = figure(2);
axes1 = axes('Parent',figure2);
hold(axes1,'on');
epoch = 1:opt.EpochNumber;
[minval,minidx] = min(chkError);
plot1 = plot(epoch,trainError,'LineWidth',2,'LineStyle','none');
set(plot1,'DisplayName','Train','Marker','o','Color',[0 0 1]);
plot2 = plot(epoch,chkError,'LineWidth',2,'LineStyle','none');
set(plot2,'DisplayName','Validation','MarkerSize',8,...
    'Marker','*','Color',[1 0 0]);
plot(minidx,minval,'DisplayName','Best','MarkerSize',25,...
    'Marker','.','LineWidth',3,'LineStyle','none',…
    'Color',[0 0 0]);
ylabel('RMSE');
xlabel('epoch');
box(axes1,'on');
hold(axes1,'off');
set(axes1,'FontSize',14,'XGrid','on','YGrid','on');
legend1 = legend(axes1,'show');
set(legend1,'Position',[0.7 0.55 0.2 0.15]);
```

After the above program is run, the mean square error of ANFIS on the training and validation sets is shown in Fig. 4.9. In Fig. 4.9, the results on the training set are represented by circle symbols, while the results on the validation set are represented by asterisks. The point with the smallest mean square error on the validation set is represented by a solid circle symbol.

From Fig. 4.9, it can be seen that the mean square error of ANFIS on the training set decreases rapidly with the increase of training times (epochs). After 30 epochs, the decreasing trend becomes slower, which indicating that the model tends to be smooth. The mean square error still shows up and down fluctuations. Correspondingly, the mean square error of ANFIS on the validation set decreases first. The curve reaches the minimum mean square error, i.e., the "Best" point in the figure, at the 17-th

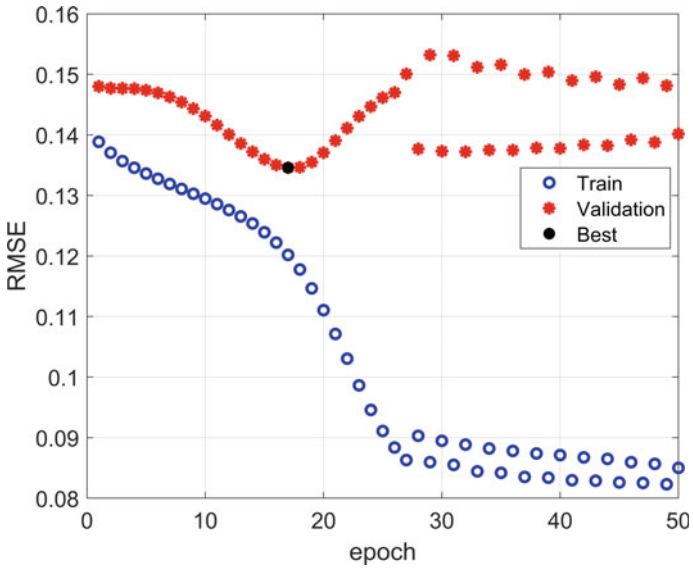

Fig. 4.9 Mean square error of ANFIS on the training and validation sets

iteration. In the subsequent epochs, the mean square error of ANFIS on the validation set gradually increases. Thus, the ANFIS model returned by the above program is the model of the 17-th iteration, which is stored in the variable "chkFIS".

4.3 Time Series Prediction

Time series refers to the quantitative change of a certain thing over time, i.e., the change in quantity is related to time. Time series prediction is to take the quantitative change of a certain thing over time as a sequence, analyze the pattern of change in it, and predict the future trend of such change. Time series prediction is also called time series forecasting. For example, the sales volume forecasting of a certain commodity, the traffic flow forecasting of a certain city, the price forecasting of daily necessities, the financial stock forecasting, etc. These problems are all time series prediction problems.

Time series prediction problems can be solved using regression analysis methods, neural network methods, while in this section we use ANFIS to solve this type of problems.

Example 4.3 The Mackey-Glass (MG) time-delay differential equation is a common test problem in the field of neural networks and fuzzy computing, and its expression is:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{4.18}$$

which t is the independent variable of the problem, τ is the time delay. The problem is an acyclic chaotic sequence, which neither converges nor diverges. Assume that the initial conditions are:

$$x(0) = 1.2 \tag{4.19}$$

$$\tau = 17 \tag{4.20}$$

$$x(t) = 0, t < 0 \tag{4.21}$$

Under the above initial conditions, we use the 4-th order Runge-Kutta method to calculate the numerical solution of this problem. This leads to a set of data for this problem. The dataset contains 1200 points. Try to make predictions based on the available dataset and analyze the results.

Solution It can be seen that the time series prediction problem is one independent variable and one dependent variable. Usually we need to construct a dataset and then do prediction based on the dataset. Assuming that there are already t moments of data, the moment we need to predict is $t + p$. Usually we start from the C-th point in the existing data and take a sample of every interval of D, i.e.:

$$x(t-(C-1)D), \ldots, x(t-D), x(t) \tag{4.22}$$

If we take $C = 4$, $D = p = 6$, a sample with 4 components can be obtained:

$$x(t-18), x(t-12), x(t-6), x(t) \tag{4.23}$$

And the moment to predict at this point is $x(t+6)$. We start from $t = 118$, and end until $t = 1117$. Thus, we can construct 1000 such samples. We use the first 500 of the 1000 samples as the training dataset and the last 500 as the validation dataset.

When constructing ANFIS, we can create an initial system based on the training set and then start training ANFIS. This technique results in a better ANFIS fuzzy system. Similar to the previous section, we can also analyze the root mean squared error (RMSE) on the training set and the RMSE on the validation set, using the following programs:

```
load('mgdata.dat');
time = mgdata(:,1);
x = mgdata(:, 2);
figure(1)
plot(time,x)
```

```
title('Mackey-Glass Chaotic Time Series')
xlabel('Time (sec)')
ylabel('x(t)').
for t = 118:1117.
    Data(t − 117,:) = [x(t − 18) x(t − 12) x(t − 6) x(t) x(t + 6)];
end
trnData = Data(1:500,:);
chkData = Data(501:end,:);
fis = genfis(trnData(:,1:end-1),trnData(:,end),…
    genfisOptions('GridPartition'));
options = anfisOptions('InitialFIS',fis,'ValidationData',chkData);
[fis1,error1,ss,fis2,error2] = anfis(trnData,options);
figure(2);
plot(error1,'-')
hold on
plot(error2,'--')
plot(error1,'o')
plot(error2,'*')
legend('Train error','Valication error')
xlabel('epoch')
ylabel('RMSE')
anfis_output = evalfis(fis2,[trnData(:,1:4); chkData(:,1:4)]);
figure(3);
index = 125:1124;
plot(time(index),[x(index) anfis_output])
xlabel('Time (sec)')
ylabel('x(t)')
figure(4);
diff = x(index) - anfis_output;
plot(time(index),diff)
xlabel('Time (sec)')
ylabel('Prediction Errors')
```

After the above program is run, we obtain four figures. They are Figs. 4.10, 4.11, 4.12 and 4.13.

In Fig. 4.10, the MG time series is given from $t = 0$ to 1200 moments. It can be seen that the system is in a bounded acyclic state.

The RMSE of ANFIS on the training and validation sets is given in Fig. 4.11. It can be seen that the error on both datasets decreases rapidly as the number of epochs increases. The error of ANFIS on the validation set is always smaller than the error on the training set.

The true MG time series and the ANFIS predicted series are given in Fig. 4.12, where the solid line shows the MG time series and the dashed line shows the ANFIS predicted series. It can be seen that the two curves overlap well.

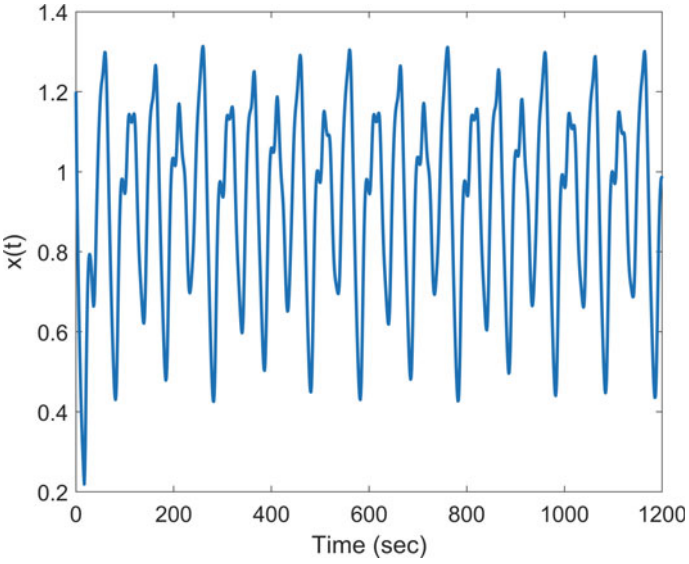

Fig. 4.10 MG time series

Fig. 4.11 RMSE of ANFIS on the training and validation sets

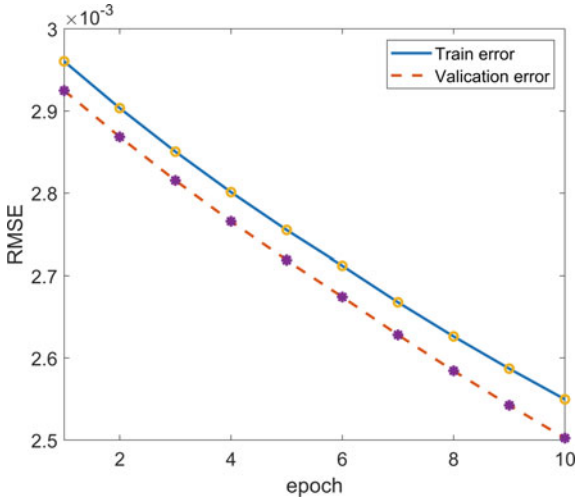

Figure 4.13 gives the ANFIS prediction errors from $t = 0$ to 1200 moments. The RMSE at these points is about 0.033. For the validation set, the RMSE of the model is 0.003. The RMSE values on both training and validation sets are very small. Thus, it can be seen that the obtained ANFIS model is able to solve the MG time series prediction problem.

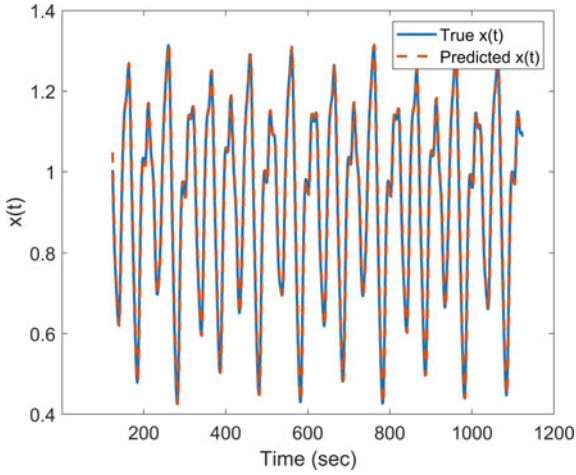

Fig. 4.12 MG time series and ANFIS predicted series

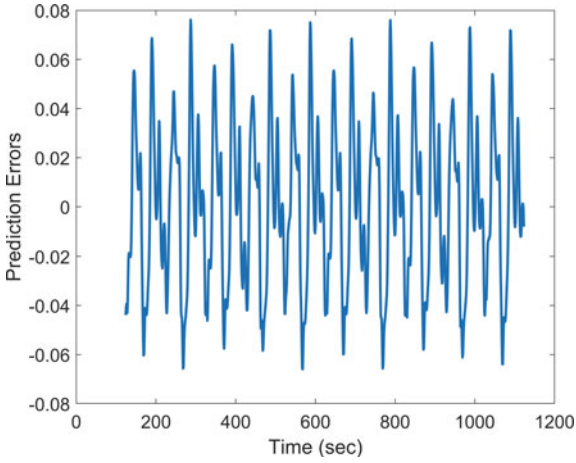

Fig. 4.13 Error of ANFIS prediction

4.4 Interval Type-2 Fuzzy Logic

We know that people often use fuzzy language to describe a concept. Correspondingly, fuzzy logic uses fuzzy theory to explain phenomena and solve problems, so that people can easily understand. For example, the concept of "tall building", if we specify the number of floors 20 as the boundary, buildings with less than 20 floors cannot be called tall buildings, and buildings with more than 20 floors can be called tall buildings. Then this is the classical binary logic, as shown in the left graph in Fig. 4.14. People may think that a building with 18 floors is also a tall building, or even 15 floors is also a tall building. The fuzzy computing theory we introduced earlier uses fuzzy membership to express this situation, as shown in the

middle graph in Fig. 4.14. This way of representing linguistic concepts by a fuzzy set is called a type-1 fuzzy set. Here buildings with floor numbers from 15 to 20 are also tall buildings, except that they belong to the concept of tall buildings to a different extent. This property of a type-1 fuzzy set for describing linguistic concepts is called intra-individual uncertainty.

Zadeh proposed the type-2 fuzzy set in 1975. His starting point is that people do not have the same understanding of the same linguistic concept, also known as inter-individual uncertainty. Take the concept of "tall building" as an example. Suppose a type-1 fuzzy set describes this concept, the membership of an 18-floor building as a tall building is 0.8, but does this membership have to be 0.8? Perhaps someone thinks this membership should be 0.7? A type-2 fuzzy set is to express different views of different individuals. Due to the complexity of expressing type-2 fuzzy sets, researchers nowadays usually use interval type-2 fuzzy sets. In the interval type-2 fuzzy set, the membership is no longer a value but an interval. For example, the membership interval for an 18-floor building belonging to a tall building is [0.7, 0.8], as shown in the right graph in Fig. 4.14. If everyone agrees that the membership degree of the 18-floor building belongs to the tall building is 0.8, then the membership degree interval becomes a value, i.e., the interval type-2 fuzzy set becomes a type-1 fuzzy set. Thus, the interval type-2 fuzzy set is a generalization of the type-1 fuzzy set.

Fuzzy logic based on type-1 fuzzy set is called type-1 fuzzy logic. Similarly, fuzzy logic based on type-2 fuzzy set is called type-2 fuzzy logic [3]. Sometime, type-1 fuzzy logic is called type-I fuzzy logic, and type-2 fuzzy logic is called type-II fuzzy logic. According to the concept of type-2 fuzzy sets, we know that the theories of fuzzy computing introduced earlier are all of type-1, including type-1 fuzzy sets and type-1 fuzzy systems. In 2000, Mendel and his student Liang promoted the study of interval type-2 fuzzy sets, which led to the development of interval type-2 fuzzy computing [4]. Type-2 fuzzy computing is then applied to control and decision-making problems.

The concepts of interval type-2 fuzzy sets are:

(1) upper membership function (UMF). It refers to the upper bound of the membership interval constituted by type-1 fuzzy set.

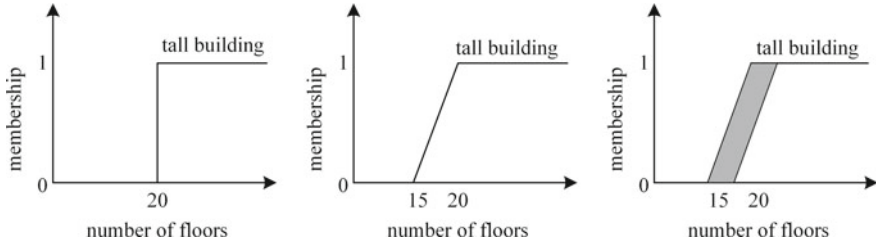

Fig. 4.14 Representations of tall building, left: binary logic, middle: type-1 fuzzy set, right: interval type-2 fuzzy set

(2) lower membership function (LMF). It refers to the lower bound of the membership interval constituted by type-1 fuzzy set.
(3) footprint of uncertainty (FOU). It refers to the area between the upper membership function and the lower membership function.
(4) embedded type-1 fuzzy set (ETS). It refers to a type-1 fuzzy set that is inside the footprint of uncertainty domain.

With the theory of interval type-2 fuzzy sets, we can create fuzzy systems corresponding to them. We can build either type-2 Mandani fuzzy system or type-2 T-S fuzzy system. For type-2 Mandani fuzzy system, both input and output variables can be type-2 fuzzy sets. For T-S type fuzzy system, the input variable can be a type-2 fuzzy set, but the output variable is the same as that of a type-1 T-S type fuzzy system.

Example 4.4 This example takes the MG time series prediction problem as an example. The same training and validation sets are used as in the previous section. The description of the MG time series prediction problem is omitted.

Solution We build a type-2 T-S fuzzy system using the same training and validation sets as in the previous section, with the following programs:

```
load('mgdata.dat');
time = mgdata(:,1);
x = mgdata(:, 2);
figure;
plot(time,x)
title('Mackey-Glass Chaotic Time Series')
xlabel('Time (sec)')
ylabel('x(t)')
C = 4;
for t =118:1117
    Data(t − 117,:) = [x(t − 18) x(t − 12) x(t − 6) x(t) x(t + 6)];
end
trnX = Data(1:500,1:C);
trnY = Data(1:500,C + 1);
vldX = Data(501:end,1:C);
vldY = Data(501:end,C + 1);
fisin = sugfistype2;
numInputs = C;
numInputMFs = 3;
range = [min(x) max(x)];
for I = 1:numInputs
    fisin = addInput(fisin,range,'NumMFs',numInputMFs);
    for j = 1:numInputMFs
        fisin.Inputs(i).MembershipFunctions(j).LowerScale = 1;
        fisin.Inputs(i).MembershipFunctions(j).LowerLag = 0;
    end
```

```
end
numOutputMFs = numInputMFs^numInputs;
fisin = addOutput(fisin,range,'NumMFs',numOutputMFs);
figure;
plotfis(fisin)
options = tunefisOptions;
options.Method = 'particleswarm';
options.OptimizationType = 'learning';
options.NumMaxRules = numInputMFs^numInputs;
options.UseParallel = false;
options.MethodOptions.MaxIterations = 10;
fisout1 = tunefis(fisin,[],trnX,trnY,options);
figure;
plotfis(fisout1)
figure;
gensurf(fisout1,gensurfOptions('InputIndex',1))
evalOptions = evalfisOptions("EmptyOutputFuzzySetMessage","none", ...
    "NoRuleFiredMessage","none","OutOfRangeInputValueMessage","none");
predY = evalfis(fisout1,vldX,evalOptions);
del = predY - vldY;
rmse = sqrt(mean(del.^2));
figure;
plot([predY vldY])
axis([0 length(vldY) min(vldY) − 0.01 max(vldY) + 0.13])
xlabel('t')
ylabel('x(t)')
legend(["predicted value" "true value"],'Location',"northeast")
```

After the above program is run, five figures would be drawn. We only give three figures to show the results, as shown in Figs. 4.15, 4.16 and 4.17.

The type-2 T-S fuzzy system at the initial time is given in Fig. 4.15. It can be seen that the number of fuzzy rules is 0, i.e., there are no fuzzy rules yet.

The type-2 T-S fuzzy system trained using the training set is given in Fig. 4.16, from which it can be seen that the number of fuzzy rules is 68.

The performance of the type-2 T-S fuzzy system on the validation set is given in Fig. 4.17. The RMSE at these points is about 0.071. The result shows that the obtained fuzzy system model is able to solve the MG time series prediction problem.

In conjunction with the previous section, the RMSE on the validation set for the type-1 T-S fuzzy system is 0.003, while the RMSE on the validation set for the type-2 T-S fuzzy system is 0.071. In terms of RMSE metric, the type-2 T-S fuzzy system performs slightly worse than the type-1 T-S fuzzy system. It should be noted that the performance of the model is not intentionally optimized here. Interested readers can make further comparisons.

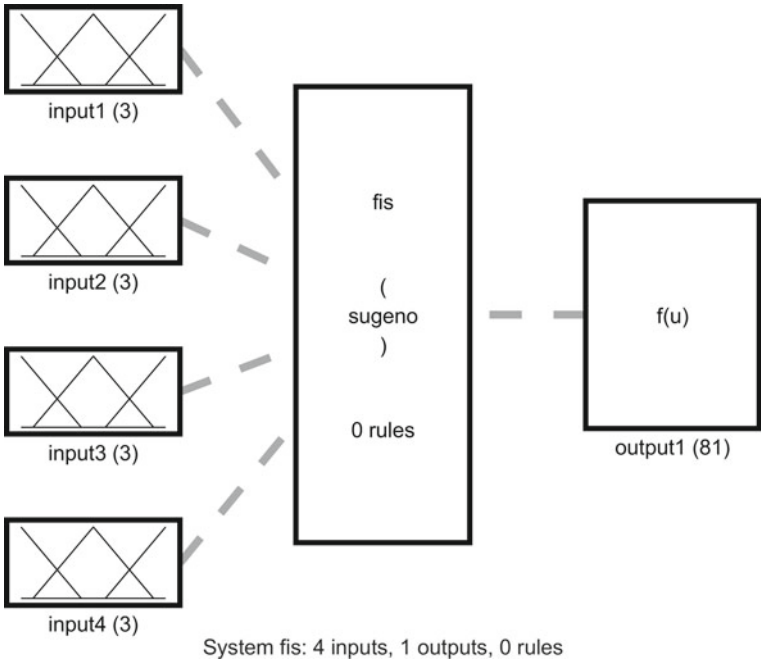

Fig. 4.15 Type-2 T-S fuzzy system at the beginning

4.5 Fuzzy C-means Clustering

Clustering is the basis for many classification and system modeling methods. The purpose of clustering is to identify natural groupings of data from a large amount of data to describe system behavior in a concise form. The best known of the fuzzy clustering methods is the fuzzy c-means clustering (FCM) method. FCM is a data clustering technique that uses membership to indicate the degree to which data points belong to a category. The FCM method was originally proposed by Bezdek in 1981 [5]. It provides a method that shows how to group data points that populate a certain multi-dimensional space into a specific number of distinct clusters.

Initially, the FCM method first randomly selects the locations of the clustering points. These randomly generated clustering centers are likely to be wrong. Then, the FCM method assigns each data point a membership degree belonging to each category. By iteratively updating the cluster centers and membership degrees for each data point, the FCM method iteratively moves the cluster centers to dense locations in the dataset. This iteration is based on minimizing some objective function that represents the distance from any given data point to the cluster center weighted by the membership of that data point. The final FCM method will output the clustering centers it finds.

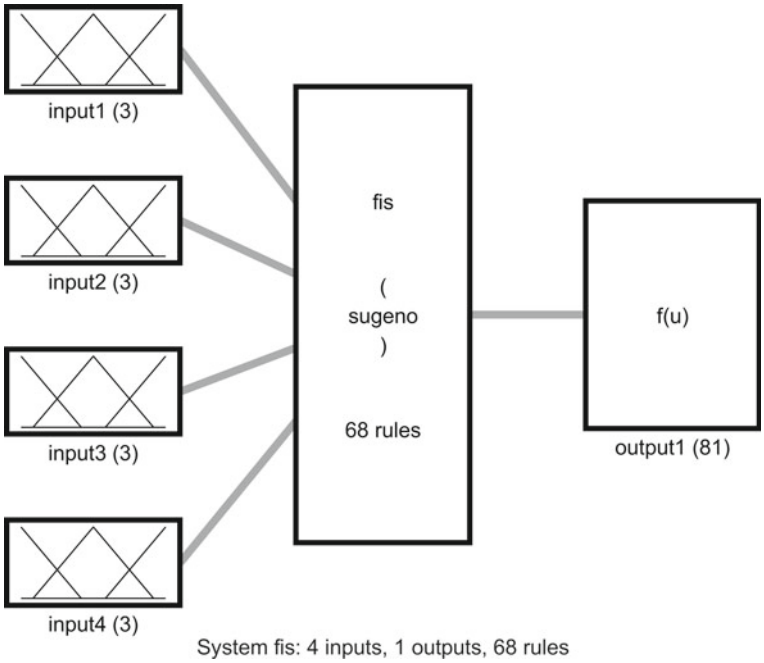

Fig. 4.16 Type-2 T-S fuzzy system after training

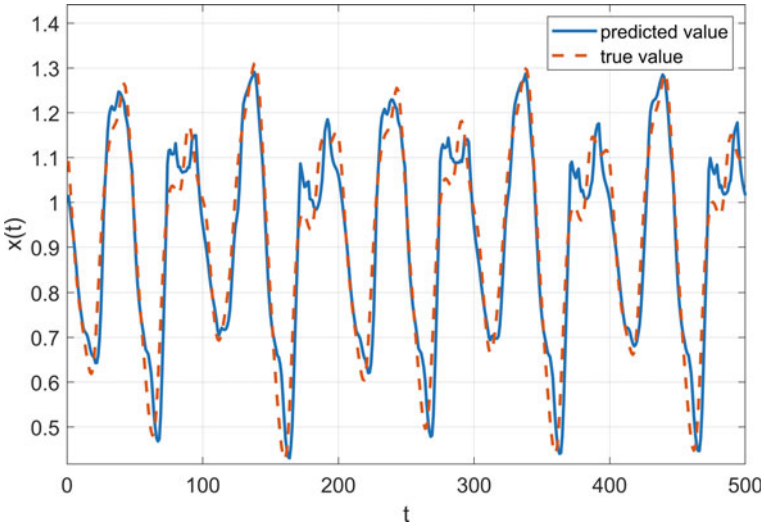

Fig. 4.17 Performance of type-2 T-S fuzzy system on the validation set

Due to the use of fuzzy membership functions, the FCM method is characterized by allowing each sample point to belong to more than one category. The degree to which a sample point belongs to a category is determined by the membership function. Since FCM allows each sample point to belong to more than one category, this makes the boundaries of the categories overlap each other. It is generally represented by the fuzzy separation matrix index, which determines the membership degree of the sample points to different categories. The objective function used in the FCM method is:

$$f_m = \sum_{i=1}^{N}\sum_{j=1}^{M} \mu_{ij}^{m} \left\| x_i - c_j \right\|^2 \tag{4.24}$$

where N is the number of samples in the dataset, M is the number of categories, $m > 1$ is the fuzzy separation matrix index, x_i is the sample point in the dataset, c_j is the center of the j-th category, and μ_{ij} is the membership of the sample point x_i belong to the j-th category.

Based on the objective function in (4.24), the steps of the FCM method are:

Step (1) Randomly initialize the membership μ_{ij} of each sample;
Step (2) Calculation of clustering centers:

$$c_j = \frac{\sum_{i=1}^{N} \mu_{ij}^{m} x_i}{\sum_{i=1}^{N} \mu_{ij}^{m}} \tag{4.25}$$

Step (3) Update the membership of each sample:

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{M} \left(\frac{\left\| x_i - c_j \right\|}{\left\| x_i - c_k \right\|} \right)^{\frac{2}{m-1}}} \tag{4.26}$$

Step (4) Calculate the objective function value according to (4.24);
Step (5) Repeat Steps (2)-(4) until the termination conditions are met.

In Step (5), there are 2 common termination conditions. One termination condition is a predefined maximum number of iterations. For example, if the maximum number of iterations is set to 100, the FCM method stops after Steps (2)–(4) are repeatedly executed 100 times. Another termination condition is the objective function stagnation. For example, if we set a very small number like 10^{-3}, when the absolute value of the difference of the objective function in two consecutive iterations is less than 10^{-3}. It means that the method cannot find a better value of the objective function, so the iteration of the algorithm is terminated.

Example 4.5 Let's take the iris dataset as an example. The dataset contains three types of irises, namely Sentosa iris, Versicolour iris and Virginia iris. There are 50

samples of each iris species. Each sample has 4 attributes, namely sepal length, sepal width, petal length and petal width. Please use the FCM method to perform clustering analysis on this dataset.

Solution The programs for solving this problem using the FCM method are as follows:

```
load('iris.dat');
setosaIndex = iris(:,5) == 1;
versicolorIndex = iris(:,5) == 2;
virginicaIndex = iris(:,5) == 3;
setosa = iris(setosaIndex,:);
versicolor = iris(versicolorIndex,:);
virginica = iris(virginicaIndex,:);
Characteristics = {'sepal length','sepal width','petal length','petal width'};
pairs = [1 2; 1 3; 1 4; 2 3; 2 4; 3 4];
figure1 = figure;
for i =1:6
    x = pairs(i,1);
    y = pairs(i,2);
    subplot1 = subplot(2,3,i,'Parent',figure1);
    hold(subplot1,'on');
    plot(setosa(:,x),setosa(:,y),'Parent',subplot1,…
        'MarkerSize',8,'Marker','.','LineStyle','none');
    plot(versicolor(:,x),versicolor(:,y),'Parent',subplot1,…
        'MarkerSize',8,'Marker','x','LineStyle','none');
    plot(virginica(:,x),virginica(:,y),'Parent',subplot1,…
        'MarkerSize',8,'Marker','square','LineStyle','none');
    xlabel(Characteristics{x});
    ylabel(Characteristics{y});
    box(subplot1,'on');
    hold(subplot1,'off');
    set(subplot1,'FontSize',12);
end
M = 3;
m = 2.0;
maxIter = 100;
minImpr = 1e − 6;
opt = [m maxIter minImpr true];
[centers,U,objFun] = fcm(iris,M,opt);
figure1 = figure;
for i = 1:6
    subplot1 = subplot(2,3,i,'Parent',figrue1);
    x = pairs(i,1);
    y = pairs(i,2);
```

```
    hold(subplot1,'on');
    plot(setosa(:,x),setosa(:,y),'Parent',subplot1,...
        'MarkerSize',8,'Marker','.','LineStyle','none');
    plot(versicolor(:,x),versicolor(:,y),'Parent',subplot1,...
        'MarkerSize',8,'Marker','x','LineStyle','none');
    plot(virginica(:,x),virginica(:,y),'Parent',subplot1,...
        'MarkerSize',8,'Marker','square','LineStyle','none');
    for j = 1:M
        text(centers(j,x),centers(j,y),int2str(j),...
            'FontSize',12,'FontWeight','bold');
    end
    xlabel(Characteristics{x});
    ylabel(Characteristics{y});
    box(subplot1,'on');
    hold(subplot1,'off');
    set(subplot1,'FontSize',12);
end
```

After the above program is run, the results are shown in Figs. 4.18 and 4.19.

As shown in Fig. 4.18, the iris dataset has four attributes. A flat graph is drawn by using two attributes. That is the $C_4^2 = 6$ cases in Fig. 4.18. It can be seen that the overlap of the sample categories for the sepal width and sepal length attributes is

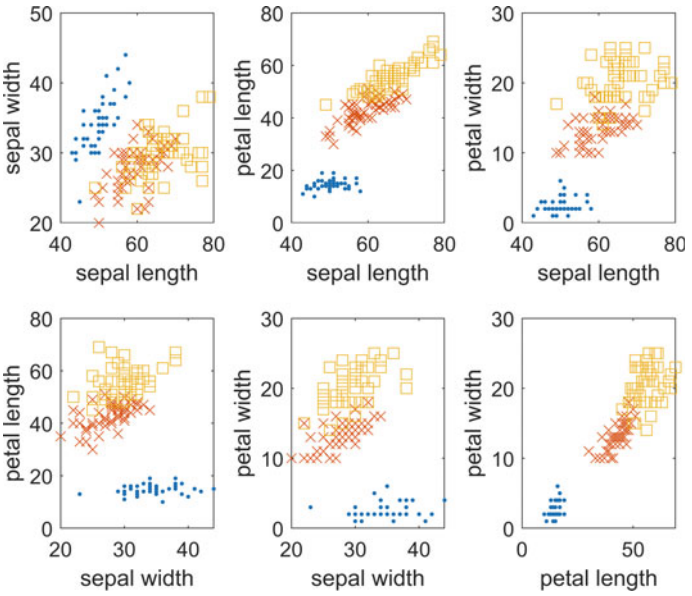

Fig. 4.18 Visualizing the Iris dataset

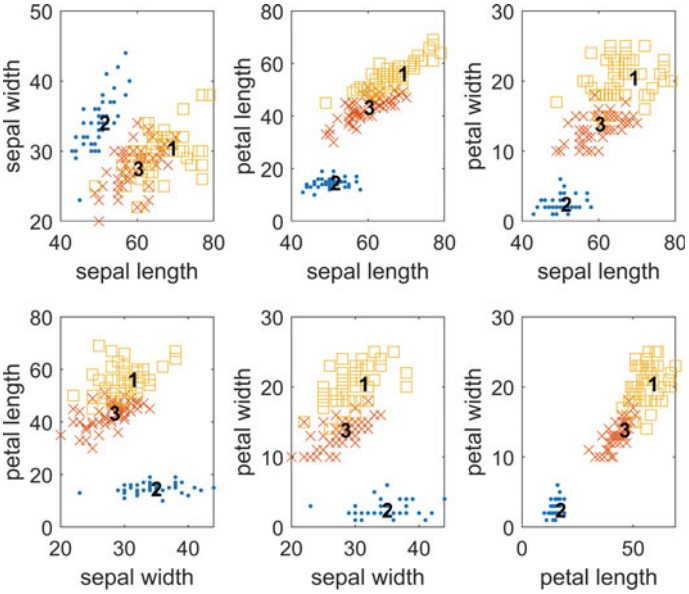

Fig. 4.19 Clustering results of the FCM method on the Iris dataset

more, while the overlap of the sample point categories for the sepal width and petal width attributes is less.

The clustering results of the FCM method are given in Fig. 4.19. The centers of the clusters are represented by numbers. Numbers 1, 2 and 3 indicate category 1, category 2 and category 3, respectively. It can be seen from the figure that the FCM method solves this clustering problem.

The above program also outputs the result of the objective function. After 22 iterations, the FCM method reaches the 10^{-6} minimum threshold. At this point, the method terminates and the objective function value is 6058.69.

The FCM method requires a predetermined number of categories. This is one of the shortcomings of this method. Chiu proposed the subtractive clustering (SC) method in 1994. The starting point of subtractive clustering is that it does not require a predetermined number of categories. It is also fast to estimate the number of categories and calculate the centers of clusters. The steps of the SC method are:

Step (1) calculate the probability that each sample is a cluster center. Assuming that each sample point is a possible cluster center, and that this probability is based on the density of other sample points around the sample point;
Step (2) select the sample points most likely to be cluster centers as temporary cluster centers;
Step (3) remove sample points from the neighborhood near the temporary cluster centers. The size of the neighborhood is determined by a parameter called the category influence range;

Step (4) among the remaining sample points, the one most likely to be the cluster center is selected as the temporary cluster center;
Step (5) Repeat Steps (3) and (4) until some termination condition is met.

Example 4.6 Please perform a clustering analysis of the Iris dataset using the subtractive clustering method.

Solution The programs for solving this problem using the SC method are as follows:

```
load('iris.dat');
setosaIndex = iris(:,5) == 1;
versicolorIndex = iris(:,5) == 2;
virginicaIndex = iris(:,5) == 3;
setosa = iris(setosaIndex,:);
versicolor = iris(versicolorIndex,:);
virginica = iris(virginicaIndex,:);
clusterInfluenceRange = 1;
[centers,sigma] = subclust(iris,clusterInfluenceRange);
Characteristics = {'sepal length','sepal width','petal length','petal width'};
pairs = [1 2; 1 3; 1 4; 2 3; 2 4; 3 4];
figure1 = figure;
for i = 1:6
    subplot1 = subplot(2,3,i,'Parent',figure1);
    x = pairs(i,1);
    y = pairs(i,2);
    hold(subplot1,'on');
    plot(setosa(:,x),setosa(:,y),'Parent',subplot1,...
        'MarkerSize',8,'Marker','.','LineStyle','none');
    plot(versicolor(:,x),versicolor(:,y),'Parent',subplot1,...
        'MarkerSize',8,'Marker','x','LineStyle','none');
    plot(virginica(:,x),virginica(:,y),'Parent',subplot1,...
        'MarkerSize',8,'Marker','square','LineStyle','none');
    for j = 1:size(centers,1)
        text(centers(j,x),centers(j,y),int2str(j),...
            'FontSize',12,'FontWeight','bold');
    end
    xlabel(Characteristics{x});
    ylabel(Characteristics{y});
    box(subplot1,'on');
    hold(subplot1,'off');
    set(subplot1,'FontSize',12);
end
```

After the above program is run, the result is shown in Fig. 4.20.

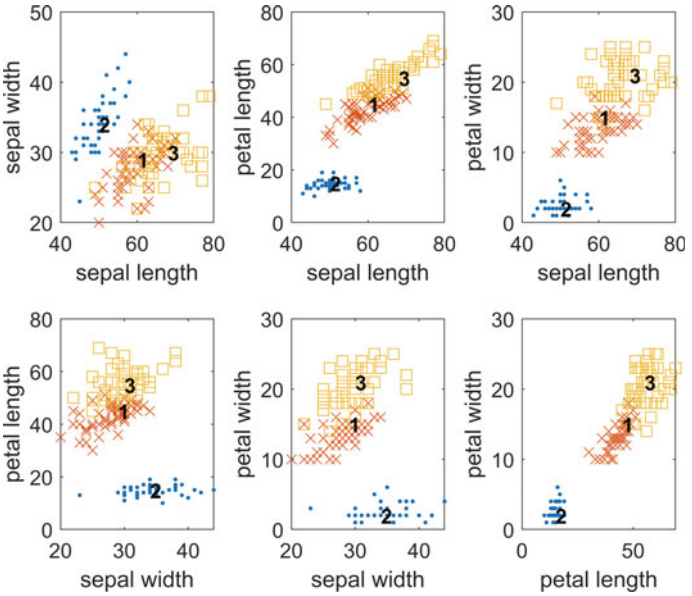

Fig. 4.20 Clustering results of the SC method on the Iris dataset

The clustering results of the SC method are given in Fig. 4.20. The centers of the clusters are represented by numbers. Numbers 1, 2 and 3 indicate category 1, category 2 and category 3, respectively. It can be seen from the figure that the SC method solves this clustering problem.

The advantage of the SC method is that it does not require a predetermined number of categories, but introduces a category influence range parameter. The smaller the value of the category influence range parameter, which is between 0 and 1, the greater the number of categories classified by the method; conversely, the closer its value is to 1, the smaller the number of categories classified by the method.

The SC method can find out the number of categories as well as the cluster centers of the dataset. We can initialize the FCM method with the number of categories and cluster centers to discover more suitable clustering results. Moreover, the FCM and SC methods can be used for the construction of fuzzy inference systems.

In the previous sections, the fuzzy system, by default, uses a grid partitioning method. Grid partitioning method uniformly partitions the range of input variables and generates the membership function on the result of this partition. If the FCM method is used to build the fuzzy system, the fuzzy system uses the clustering centers obtained from the FCM method to generate the membership functions and fuzzy rules. If the SC method is used to build the fuzzy system, the fuzzy system uses the clustering centers obtained by the SC method to generate the membership functions and fuzzy rules.

Example 4.7 Please use the FCM and SC methods to build fuzzy systems to solve MG time series prediction problem.

Solution Based on the FCM method, we build a fuzzy system to solve the MG time series prediction problem with the following programs:

```
load('mgdata.dat');
time = mgdata(:,1);
x = mgdata(:, 2);
figure(1)
plot(time,x)
title('Mackey-Glass Chaotic Time Series')
xlabel('Time (sec)')
ylabel('x(t)')
C = 4;
for t = 118:1117
    Data(t - 117,:) = [x(t-18) x(t - 12) x(t - 6) x(t) x(t + 6)];
end
trnX = Data(1:500,1:C);
trnY = Data(1:500,C + 1);
vldX = Data(501:end,1:C);
vldY = Data(501:end,C + 1);
M = 3;
m = 2.0;
maxIter = 100;
minImpr = 1e - 6;
opt = [m maxIter minImpr true];
[centers,U,objFun] = fcm(trnX,M,opt);
opt2 = genfisOptions('FCMClustering');
opt2.NumClusters = M;
opt2.Exponent = opt(1);
opt2.MaxNumIteration = opt(2);
opt2.MinImprovement = opt(3);
opt2.Verbose = opt(4);
fis = genfis(trnX,trnY,opt2);
options = anfisOptions('InitialFIS',fis,'ValidationData',[vldX,vldY]);
[fis1,error1,ss,fis2,error2] = anfis([trnX,trnY],options);
figure(2);
plot(error1,'-')
hold on
plot(error2,'--')
plot(error1,'o')
plot(error2,'*')
legend('Train error','Valication error')
xlabel('epoch')
```

```
ylabel('RMSE')
evalOptions = evalfisOptions("EmptyOutputFuzzySetMessage","none", ...
    "NoRuleFiredMessage","none","OutOfRangeInputValueMessage","none");
predY = evalfis(fis2,vldX);
diff = vldY - predY;
rmse = sqrt(mean(diff.^2));
figure(3);
plot([predY vldY]);
axis([0 length(vldY) min(vldY)− 0.01 max(vldY) + 0.13])
xlabel('t')
ylabel('x(t)')
legend(["predicted value" "true value"],'Location',"northeast")
```

After the above program is run, we can obtain three figures, which are omitted for the saving of space. The RMSE of the fuzzy system using the FCM method is 0.0182.

Based on the SC method, we build a fuzzy system to solve the MG time series prediction problem with the following program:

```
load('mgdata.dat');
time = mgdata(:,1);
x = mgdata(:, 2);
figure(1)
plot(time,x)
title('Mackey-Glass Chaotic Time Series')
xlabel('Time (sec)')
ylabel('x(t)')
C = 4;
for t = 118:1117
    Data(t − 117,:) = [x(t − 18) x(t − 12) x(t − 6) x(t) x(t + 6)];
end
trnX = Data(1:500,1:C);
trnY = Data(1:500,C + 1);
vldX = Data(501:end,1:C);
vldY = Data(501:end,C + 1);
clusterInfluenceRange = 1;
opt = [2.0 0.8 0.7 0];
[centers,sigma] = subclust(trnX,clusterInfluenceRange,'Options',opt);
opt2 = genfisOptions('SubtractiveClustering');
opt2.ClusterInfluenceRange = clusterInfluenceRange;
opt2.SquashFactor = opt(1);
opt2.AcceptRatio = opt(2);
opt2.RejectRatio = opt(3);
opt2.Verbose = opt(4);
fis = genfis(trnX,trnY,opt2);
```

```
options = anfisOptions('InitialFIS',fis,'ValidationData',[vldX,vldY]);
[fis1,error1,ss,fis2,error2] = anfis([trnX,trnY],options);
figure(2);
plot(error1,'-')
hold on
plot(error2,'--')
plot(error1,'o')
plot(error2,'*')
legend('Train error','Valication error')
xlabel('epoch')
ylabel('RMSE')
evalOptions = evalfisOptions("EmptyOutputFuzzySetMessage","none", ...
    "NoRuleFiredMessage","none","OutOfRangeInputValueMessage","none");
predY = evalfis(fis2,vldX);
diff = vldY - predY;
rmse = sqrt(mean(diff.^2));
figure(3);
plot([predY vldY]);
axis([0 length(vldY) min(vldY)− 0.01 max(vldY) + 0.13])
xlabel('t')
ylabel('x(t)')
legend(["predicted value" "true value"],'Location',"northeast")
```

After the above program is run, we can obtain three figures, which are omitted here and will not be described. The RMSE of the fuzzy system using the SC method is 0.0961.

From the Example 4.7, it can be seen that the fuzzy system based on grid partitioning has the smallest RMSE on the validation set, followed by the fuzzy system based on the FCM method, and the largest RMSE is the fuzzy system based on the SC method. It should be noted that in this example, we did not adjust the parameters of the methods, so we cannot determine which method is superior based on this result. It is better for the reader to try all of them when solving specific problems and optimize the parameters to get the best performance.

4.6 Suburban Commuting Prediction Problem

This section tries to solve the suburban commuting prediction problem. We introduce the integrated application of fuzzy neural network, fuzzy clustering and ANFIS in the problem.

Example 4.8 The dataset for the suburban commuting prediction problem is demographic and travel data for 100 transportation analysis areas in New Castle County, Delaware (USA). The dataset contains five demographic factors as input variables:

population, number of dwelling units, vehicle ownership, median household income, and total employment. The sample data in this dataset contains one output variable, i.e., the number of automobile trips. We have divided the dataset into a training set and a validation set, where the training set has 75 samples and the validation set has 25 samples.

Solution First, we use the SC method as the clustering method in fuzzy systems as a way to define the membership function and fuzzy rules. We will create a T-S type fuzzy system with the following programs:

```
load('trafficData');
[clusters,sigma] = subclust([datain dataout],0.5);
figure
plot(datain(:,5),dataout(:,1),'o',…
     clusters(:,5),clusters(:,6),"r*");
legend("Data points","Cluster centers","Location","southeast")
xlabel("Total Employment")
ylabel("Number of Trips")
title("Data and Cluster Centers")
opt = genfisOptions("SubtractiveClustering",…
     "ClusterInfluenceRange",0.5);
fis = genfis(datain,dataout,opt);
showrule(fis);
figure
plotmf(fis,"input",1)
fuzout = evalfis(fis,datain);
trnRMSE = norm(fuzout-dataout)/sqrt(length(fuzout));
valfuzout = evalfis(fis,valdatain);
valRMSE = norm(valfuzout-valdataout)/sqrt(length(valfuzout));
figure
plot(valdataout,'o');
hold on.
plot(valfuzout,'*');
hold off
ylabel('Output value')
legend("Validation data","FIS output","Location","northwest")
anfisOpt = anfisOptions('InitialFIS',fis,…
    'EpochNumber',100,'InitialStepSize',0.1,…
    'ValidationData',[valdatain valdataout],…
    'DisplayANFISInformation',0,…
    'DisplayErrorValues',0,…
    'DisplayStepSize',0,…
    'DisplayFinalResults',0);
[fis2,trnErr,stepSize,fis3,valErr] = anfis([datain dataout],anfisOpt);
[minValErr,minValErrIdx] = min(valErr);
```

```
fuzout3 = evalfis(fis3,datain);
trnRMSE3 = norm(fuzout3-dataout)/sqrt(length(fuzout3));
valfuzout3 = evalfis(fis3,valdatain);
valRMSE3 = norm(valfuzout3-valdataout)/sqrt(length(valfuzout3));
figure;
plot(valdataout,'o');
hold on
plot(valfuzout,'*');
plot(valfuzout3,'x');
hold off
ylabel('Output value')
legend("Validation data","Initial FIS: RMSE = " + num2str(valRMSE), ...
    "Tuned FIS: RMSE = "+ num2str(valRMSE3), ...
    "Location","northwest")
figure;
plot(trnErr);
title('Training Error');
xlabel('Number of Epochs');
ylabel('Error');
figure;
plot(valErr);
hold on;
plot(minValErrIdx,minValErr,'*');
title('Validation Error');
xlabel('Number of Epochs');
ylabel('Error');
```

After the above program is run, we can obtain six figures. Three of them are shown in the following and the other figures are omitted for the saving of space.

The clustering results of the SC method are given in Fig. 4.21, where the raw data are represented by circle symbols, while the cluster centers are represented by star symbols. In this figure, the horizontal axis is the total employment of the input variable and the vertical axis is the number of car trips of the output variable.

The RMSE of the fuzzy system using the SC method is 0.5276 on the training set, while the RMSE on the validation set is 0.6179.

We can initialize ANFIS with the obtained fuzzy system in order to be able to obtain a better fuzzy system. The results of ANFIS on the validation set using the SC method are given in Fig. 4.22. In this figure, the original data are represented by circle symbols, the results predicted by the fuzzy system are represented by star symbols, and the results predicted by ANFIS are represented by crosses. As can be seen from the figure, the model of ANFIS shows better performance. The RMSE of ANFIS on the training set is 0.3393, while the RMSE on the validation set it is 0.5834.

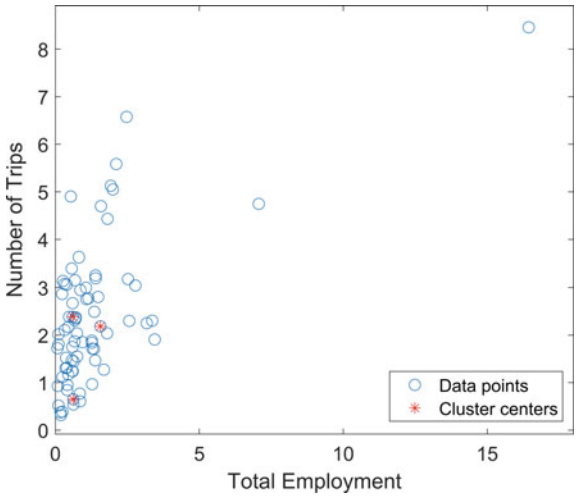

Fig. 4.21 Clustering results of the SC method

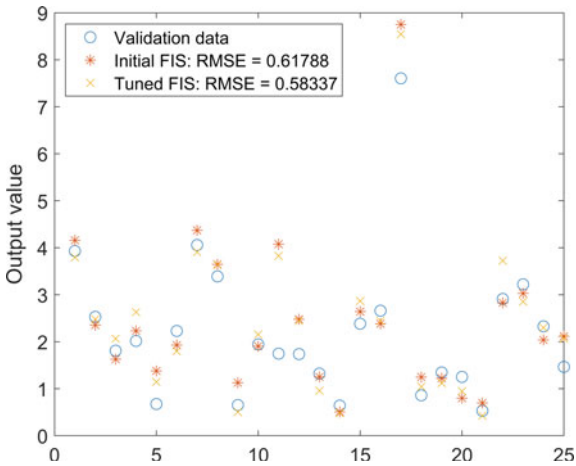

Fig. 4.22 Results of ANFIS on the validation set using the SC method

Figure 4.23 gives the error of the model on the validation set for each epoch. It can be seen from the figure that the error gradually decreases with the number of epochs and reaches its lowest at the 52nd epoch, as shown by the asterisk point in the figure. The error increases again during the subsequent epochs. Even after the 52nd epoch, the error of the model on the training set still decreases, but the error of the model on the validation set increases. This indicates that ANFIS is in an overfitting state when the number of epochs exceeds 52. Thus, we use the result obtained from the 52nd epoch as the final model.

Next, the SC method divides the dataset into 3 categories. We use this number of categories to initialize the FCM method and repeat the above procedure as follows:

load('trafficData');

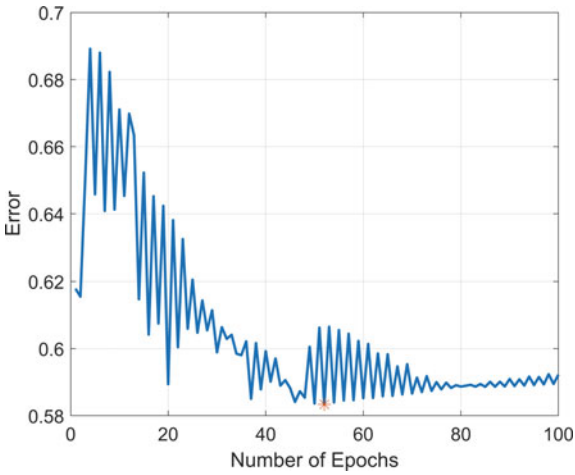

Fig. 4.23 Error curve of ANFIS on the validation set

```
M = 3;
m = 2.0;
maxIter = 100;
minImpr = 1e − 6;
opt = [m maxIter minImpr true];
[centers,U,objFun] = fcm([datain dataout],M,opt);
figure
plot(datain(:,5),dataout(:,1),'o',…
    centers(:,5),centers(:,6),"r*");
legend("Data points","Cluster centers","Location","southeast")
xlabel("Total Employment")
ylabel("Number of Trips")
title("Data and Cluster Centers")
opt = genfisOptions("FCMClustering",…
    "NumClusters",M);
fis = genfis(datain,dataout,opt);
showrule(fis);
figure
plotmf(fis,"input",1)
fuzout = evalfis(fis,datain);
trnRMSE = norm(fuzout-dataout)/sqrt(length(fuzout));
valfuzout = evalfis(fis,valdatain);
valRMSE = norm(valfuzout-valdataout)/sqrt(length(valfuzout));
figure
plot(valdataout,'o');
hold on
plot(valfuzout,'*');
hold off
```

```
ylabel('Output value')
legend("Validation data","FIS output","Location","northwest")
anfisOpt = anfisOptions('InitialFIS',fis,…
    'EpochNumber',100,'InitialStepSize',0.1,…
    'ValidationData',[valdatain valdataout],…
    'DisplayANFISInformation',0,…
    'DisplayErrorValues',0,…
    'DisplayStepSize',0,…
    'DisplayFinalResults',0);
[fis2,trnErr,stepSize,fis3,valErr] = anfis([datain dataout],anfisOpt);
[minValErr,minValErrIdx] = min(valErr);
fuzout3 = evalfis(fis3,datain);
trnRMSE3 = norm(fuzout3-dataout)/sqrt(length(fuzout3));
valfuzout3 = evalfis(fis3,valdatain);
valRMSE3 = norm(valfuzout3-valdataout)/sqrt(length(valfuzout3));
figure;
plot(valdataout,'o');
hold on
plot(valfuzout,'*');
plot(valfuzout3,'x');
hold off
ylabel('Output value')
legend("Validation data","Initial FIS: RMSE = " + num2str(valRMSE), …
    "Tuned FIS: RMSE = "+ num2str(valRMSE3), …
    "Location","northwest")
figure;
plot(trnErr);
title('Training Error');
xlabel('Number of Epochs');
ylabel('Error');
figure;
plot(valErr);
hold on;
plot(minValErrIdx,minValErr,'*');
title('Validation Error');
xlabel('Number of Epochs');
ylabel('Error');
```

After the above program is run, we can obtain six figures, which are omitted for the saving of space.

The RMSE of the T-S type fuzzy system constructed using the FCM method is 0.6858 on the training set and 0.6393 on the validation set. The RMSE of the ANFIS fuzzy system using the FCM method is 0.5799 on the training set and 0.4867 on the validation set.

Finally, we use the grid partitioning method as a clustering method in fuzzy systems. We repeat the above steps with the following programs:

```
load('trafficData');
opt = genfisOptions("GridPartition");
fis = genfis(datain,dataout,opt);
showrule(fis);
figure
plotmf(fis,"input",1)
fuzout = evalfis(fis,datain);
trnRMSE = norm(fuzout-dataout)/sqrt(length(fuzout));
valfuzout = evalfis(fis,valdatain);
valRMSE = norm(valfuzout-valdataout)/sqrt(length(valfuzout));
figure
plot(valdataout,'o');
hold on
plot(valfuzout,'*');
hold off
ylabel('Output value')
legend("Validation data","FIS output","Location","northwest")
anfisOpt = anfisOptions('InitialFIS',fis,…
    'EpochNumber',100,'InitialStepSize',0.1,…
    'ValidationData',[valdatain valdataout],…
    'DisplayANFISInformation',0,…
    'DisplayErrorValues',0,…
    'DisplayStepSize',0,…
    'DisplayFinalResults',0);
[fis2,trnErr,stepSize,fis3,valErr] = anfis([datain dataout],anfisOpt);
[minValErr,minValErrIdx] = min(valErr);
fuzout3 = evalfis(fis3,datain);
trnRMSE3 = norm(fuzout3-dataout)/sqrt(length(fuzout3));
valfuzout3 = evalfis(fis3,valdatain);
valRMSE3 = norm(valfuzout3-valdataout)/sqrt(length(valfuzout3));
figure;
plot(valdataout,'o');
hold on
plot(valfuzout,'*');
plot(valfuzout3,'x');
hold off
ylabel('Output value')
legend("Validation data","Initial FIS: RMSE = " + num2str(valRMSE), …
    "Tuned FIS: RMSE = " + num2str(valRMSE3), …
    "Location","northwest")
figure;
plot(trnErr);
```

```
title('Training Error');
xlabel('Number of Epochs');
ylabel('Error');
figure;
plot(valErr);
hold on;
plot(minValErrIdx,minValErr,'*');
title('Validation Error');
xlabel('Number of Epochs');
ylabel('Error');
```

After the above program is run, the figures are omitted for the saving of space.

The RMSE of the fuzzy system using the grid partitioning method is 2.7851 on the training set and 2.6563 on the validation set. We initialize ANFIS with the obtained fuzzy system. The RMSE of the ANFIS using grid partitioning method is 0.0274 on the training set and 0.8618 on the validation set. The error of the model using the grid partitioning method on the training set is much smaller than the error on the validation set. This indicates that the model has a serious overfitting problem.

In this section, we use the SC method, FCM method and grid partitioning method to construct fuzzy systems. Moreover, we build ANFIS to enhance the performance of fuzzy systems. The RMSE of the fuzzy systems obtained by these three methods on the validation set are 0.5834, 0.4867 and 0.8618, respectively. The order of the three methods obtained from this result is not consistent with the results in the example in the previous section. Thus, users have to try multiple methods for simulation and analysis before choosing the most suitable one when solving their problems.

4.7 Research Progress of Fuzzy Computing

This section reviews the state-of-the-art research progress of fuzzy computing. These researches are classified to three categories. They are fuzzy control system, type-2 fuzzy logic and other fuzzy logic model.

(1) fuzzy control system

Low-frequency power oscillations in the power system have always been a serious risk to stabilizing the power system. A fuzzy logic power system stabilizer (FLPSS) was proposed by Sun et al. to alleviate system oscillation problems [6]. In response to the parameter configuration and rule setting issues of FLPSS, the use of FLPSS based on the grey wolf optimizer (GWO) achieved faster stabilization time and improved the effectiveness of damping oscillation. A comparative study was conducted on the proposed GWO-based FLPSS and traditional stabilizers on a single machine infinite bus system. The simulation results show that GWO-based FLPSS can better suppress low-frequency oscillations and maintain power grid stability.

Power transformers are one of the most expensive and critical electrical devices among all networked devices. Various electrical, mechanical, thermal and chemical stresses can reduce the insulation performance of power transformers. Tarafdar et al. proposed a new method combining fuzzy logic controller with fuzzy clustering method [7]. They used expert systems to diagnose and predict early faults in power transformers. In addition, this expert model was tested and validated using 200 transformer test datasets.

A distributed approximate optimal control scheme based on fuzzy logic nonzero-sum game was proposed by An et al. [8]. for modular robots with human-machine cooperation (HRC) tasks. A modular robot manipulator (MRM) system was established using joint torque feedback technology. Based on the differential game strategy, the optimal control problem of the MRM system facing the HRC task was transformed into a nonzero-sum game problem of multiple subsystems. Using adaptive Dynamic programming algorithm and a new fuzzy logic nonzero-sum game method, a distributed approximate optimal control strategy for HRC tasks was developed to solve the coupled Hamilton Jacobian equations. Experimental results were provided to verify the superiority and effectiveness of this method.

Sierra-Garcia et al. tried to control the pitch angle of the wind turbine [9]. Due to the nonlinearity and complex dynamics of the renewable energy systems, the control was very difficult for floating offshore wind turbines. To address this issue, Sierra-Garcia et al. proposed a hybrid system that combines fuzzy logic and deep learning. Deep learning technology was used to estimate current wind speeds and predict future wind speeds. The effective wind obtained by deep learning was sent to the fuzzy controller. The simulation results show how incorporating effective wind can improve the performance of intelligent controllers against different disturbances.

Laundry is one of people's daily lives. Currently, hand cleaning has been replaced by washing machines. Raja et al. developed a fuzzy logic control system based on people's needs and behaviors [10]. The inputs to this system were the type of clothing, the degree of dirt, and the quality of the fabric load; while the received outputs were washing time, drying time, and temperature. The simulation results indicate that the system provides good washing quality.

(2) type-2 fuzzy logic

In developing countries, 40% of fresh fruits and vegetables are usually transported in non-refrigerated trucks and will rot before use. This causes wholesalers to lose potential profits due to the spoilage of their products. Here, the wholesaler's transportation of fresh goods starts from the warehouse and returns to the warehouse after reducing the quantity at the node (retailer) based on the previously placed order. Due to the changes in freshness of perishable items over time, the selling price of the item depends on its freshness at the time of delivery to the retailer. There are multiple routes connecting retailers and warehouses. Thakur et al. developed a multi-route model for fresh agricultural products considering the freshness of the product, optimal route plan, suitable route, sales revenue, driver wages and fatigue and so on [11]. The goal was to find the best route planning, optimal route between nodes, and vehicle speed for wholesalers to maximize profits, minimize fuel costs, or both. A Virgin discrete

fireworks algorithm was developed for solutions based on type-1 and type-2 fuzzy logic. Numerical results of the trade-off between driver rest, non-rest, continuous driving risks, and profit and green are presented.

A new coupling-based hybrid interval type-2 fuzzy logic controller (MIT2FLC) was studied by Kumar et al. for trajectory tracking problems of robotic arm objects [12]. The main obstacle for these types of plants was the coupling between robot links during operation. In addition, the performance of these devices was also adversely affected by parameter uncertainty, random noise, and external interference. Therefore, a MIT2FLC method with additional degrees of freedom in the membership function was proposed to effectively handle uncertainty problems and provide robust performance. The robustness analysis of the proposed controller under external disturbances, system parameter changes, and random noise was studied.

Due to fierce market competition, transportation companies are facing the need to reduce fleet costs. The fleet management takes action on the driver's driving style to achieve fuel consumption savings. Zdravković et al. developed a model for evaluating driving style using a type-2 fuzzy logic system [13]. The type-2 fuzzy logic system considered engine speed, accelerator pedal position, and acceleration or deceleration parameters. The system output was the driver's score, representing the impact of driving style on fuel consumption. The experimental results show that the method can significantly reduce fuel consumption and improve the energy efficiency of the fleet based on driving style.

The tracking control of steer-by-wire (SbW) system with unknown nonlinear Friction torque and unmeasurable angular velocity was studied by Luo et al. [14]. In order to eliminate the adverse effect of Friction torque on SbW system, an observer based adaptive interval type-2 fuzzy logic system controller was proposed. Interval type-2 fuzzy logic system (IT2-FLS) was used to model the Friction torque, in which the model and parameters were not effectively identified. Compared with type-1 fuzzy logic system, IT2-FLS had better ability to handle uncertainty, so friction modeling based on IT2-FLS had a more satisfactory effect in practical applications. Numerical simulation experiments have verified the effectiveness and superiority of the proposed method and control strategy.

Valdez summarized the parameter adaptive algorithms based on swarm intelligence, and used some techniques to obtain the best results [15]. Valdez analyzed the most popular algorithms, such as ant colony optimization, particle swarm optimization, artificial bee colony optimization, and firefly algorithm. These algorithms used type-2 fuzzy logic for parameter adaptation. These algorithms have proven superior to other swarm intelligence-based optimization methods in some applications.

(3) other fuzzy logic model

In this technological world, e-learning has become a more feasible choice for a range of people, from beginners to experts in specific fields. However, due to some weaknesses in e-learning systems, the development of e-learning systems has not yet provided sufficient adaptability for e-learners. Usually, e-learners have made varying degrees of progress in their respective learning methods. For a period of time, this will

affect the performance of e-learners and provide the same course for all e-learners. Therefore, it is necessary to create an adaptive e-learning environment that provides appropriate e-learning content for all e-learners. Karthika et al. proposed a novel, intelligent, and adaptive e-learning context based on fuzzy logic for programming languages [16]. The dependency relationships between concepts in programming languages were provided using fuzzy cognitive maps. Fuzzy set and fuzzy rules represented the knowledge level of e-learners, and helped to provide appropriate suggestions for previous and subsequent related concepts in fuzzy cognitive maps. The simulation results show that the proposed intelligent e-learning system provides promising results in accurately classifying e-learners.

Homomorphic encryption (HE) is a powerful feature of some cryptographic systems, which allows privacy protection of encrypted text. However, due to limitations in efficiency and availability, HE is not widespread. In the challenge of HE, scheme parameterization is a related multifaceted issue. There is no general optimal choice of parameters, and this choice depends on the circuit and application scenario. Cabrero-Holgueras et al. propose a unified solution to address the aforementioned challenges [17]. Specifically, Cabrero-Holgueras et al. proposed an expert system that combines fuzzy logic with linear programming. The fuzzy logic module received high-level priorities selected by users for the security of the password system. Based on these preferences, the expert system generated a linear programming model. The simulation results show that the optimal parameter selection generated by the expert system can maintain user preferences without experiencing the inherent complexity of analyzing the circuit.

Inspired by the surge in interest in the Internet of Things (IoT), Zahra et al. focused on the dynamics of IoT security. The IoT had brought convenience to people's lives and quickly become a trillion-dollar industry. However, the future of the IoT will depend on how to address its security and privacy issues. Zahra et al. provided a critical analysis of the latest and relevant state-of-the-art methods for IoT security. The authors identified parameters that are crucial to any security situation in the IoT. Zahra et al. proposed a universal lightweight security mechanism for detecting malicious behavior in uncertain IoT environments using fuzzy logic and fog-based methods [18]. The proposed method can produce better accuracy results than existing benchmarks. In addition, the proposed method has extremely low pressure on constrained nodes and supports the heterogeneity and uncertainty of IoT environments.

The deep neural fuzzy system (DNFS) utilizes the effective learning process of deep neural networks and the reasoning ability of fuzzy inference systems. The DNFS has been successfully applied to real-world problems. Talpur et al. provided a comprehensive review of DNFS and divided it into two important parts [19]. The first part was to understand DNFS and its architectural representation, while the second part reviewed the optimization methods of DNFS. This study aims to help researchers understand various ways of developing DNFS models through mixed deep neural networks and fuzzy inference systems. This study shows that the proposed DNFS architecture performs 11.6% better than non-fuzzy models. The study based on optimization method shows that the overall accuracy of DNFS using meta-heuristic algorithms is 21.10% higher than that of DNFS model using gradient-based method. The

study of Talpur et al. suggests using new and improvised meta-heuristic algorithms to implement optimization methods to improve model performance.

Exercises

(1) Try to construct a T-S type fuzzy system by using fuzzy logic designer and observe the relationship surfaces of system model, fuzzy rules, input variables and output variables.
(2) Try to briefly explain the differences and connections between type-1 fuzzy systems and interval type-2 fuzzy systems.
(3) Choose a proportional-integral-derivative (PID) control problem and try to construct a ANFIS fuzzy system to solve this control problem and analyze its performance.

References

1. de Campos Souza PV (2020) Fuzzy neural networks and neuro-fuzzy networks: a review the main techniques and applications used in the literature. Appl Soft Comput 92:106275. https://doi.org/10.1016/j.asoc.2020.106275
2. Jang JSR (1993) ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 23(3):665–685
3. Wu D, Zeng Z, Mo H (2020) Feiyue Wang, Interval type-2 fuzzy sets and systems: overview and outlook. ACTA Automatica Sinica 46(8):1539–1556
4. Mendel JM (2017) Uncertain rule-based fuzzy systems: introduction and new directions, 2nd edn. Springer, Cham, pp 229–234
5. Chiu S (1994) Fuzzy model identification based on cluster estimation. J Intell Fuzzy Syst 2(3):267–278
6. Sun Z, Cao Y, Wen Z et al (2023) A grey wolf optimizer algorithm based fuzzy logic power system stabilizer for single machine infinite bus system. Energy Rep 9:847–853. https://doi.org/10.1016/j.egyr.2023.04.365
7. Tarafdar A, Majumder P, Deb M, Bera UK (2023) Diagnosis and prognosis of incipient faults and insulation status for asset management of power transformer using fuzzy logic controller & fuzzy clustering means. Electr Power Syst Res 220:10925. https://doi.org/10.1016/j.epsr.2023.109256
8. An T, Zhu X, Zhu M et al (2023) Fuzzy logic nonzero-sum game-based distributed approximated optimal control of modular robot manipulators with human-robot collaboration. Neurocomputing 543:126276. https://doi.org/10.1016/j.neucom.2023.126276
9. Sierra-Garcia JE, Santos M (2022) Deep learning and fuzzy logic to implement a hybrid wind turbine pitch control. Neural Comput Applic 34:10503–10517. https://doi.org/10.1007/s00521-021-06323-w
10. Raja K, Ramathilagam S (2021) Washing machine using fuzzy logic controller to provide wash quality. Soft Comput 25:9957–9965. https://doi.org/10.1007/s00500-020-05477-4
11. Thakur K, Maji S, Maity S et al (2023) Multiroute fresh produce green routing models with driver fatigue using Type-2 fuzzy logic-based DFWA. Expert Syst Appl 229:120300. https://doi.org/10.1016/j.eswa.2023.120300
12. Kumar A, Raj R, Kumar A, Verma B (2023) Design of a novel mixed interval type-2 fuzzy logic controller for 2-DOF robot manipulator with payload. Eng Appl Artif Intell 123:106329.https://doi.org/10.1016/j.engappai.2023.106329

13. Zdravković S, Vujanović D, Stokić M et al (2021) Evaluation of professional driver's eco-driving skills based on type-2 fuzzy logic model. Neural Comput Applic 33:11541–11554. https://doi.org/10.1007/s00521-021-05823-z
14. Luo G, Wang Z, Ma B et al (2021) Observer-based interval type-2 fuzzy friction modeling and compensation control for steer-by-wire system. Neural Comput Applic 33:10429–10448. https://doi.org/10.1007/s00521-021-05801-5
15. Valdez F (2020) A review of optimization swarm intelligence-inspired algorithms with type-2 fuzzy logic parameter adaptation. Soft Comput 24:215–226. https://doi.org/10.1007/s00500-019-04290-y
16. Karthika R, Deborah JL, Vijayakumar P (2020) Intelligent e-learning system based on fuzzy logic. Neural Comput Applic 32:7661–7670. https://doi.org/10.1007/s00521-019-04087-y
17. Cabrero-Holgueras J, Pastrana S (2023) Towards automated homomorphic encryption parameter selection with fuzzy logic and linear programming. Expert Syst Appl 229:120460. https://doi.org/10.1016/j.eswa.2023.120460
18. Zahra SR, Chishti MA (2022) A generic and lightweight security mechanism for detecting malicious behavior in the uncertain Internet of Things using fuzzy logic- and fog-based approach. Neural Comput Applic 34:6927–6952. https://doi.org/10.1007/s00521-021-06823-9
19. Talpur N, Abdulkadir SJ, Alhussian H et al (2022) A comprehensive review of deep neuro-fuzzy system architectures and their optimization methods. Neural Comput Applic 34:1837–1875. https://doi.org/10.1007/s00521-021-06807-9

Chapter 5
Evolutionary Computing

Abstract Evolutionary computing mimics the laws of biological evolution. It solves optimization problems through the reproduction of individuals and the competition between individuals. Evolutionary computing is a collection of evolutionary algorithms that follow the survival of the fittest law in species. Evolutionary algorithms are global probability search algorithms based on natural selection, genetic mutation and other biological evolution mechanisms. Evolutionary computing has been used in various fields such as pattern recognition, image processing, economic management, mechanical engineering, electrical engineering, wireless communication, etc. This chapter first introduces an overview of evolutionary computing and simple genetic algorithm. Genetic algorithm is then used to solve travelling salesman problem. Then, this chapter introduces ant colony optimization, particle swarm optimization and differential evolution algorithms. These algorithms have been used to solve both travelling salesman problem and continuous optimization problem.

5.1 Overview of Evolutionary Computing

Evolutionary Computing is an intelligent computing technology that mimics the laws of biological evolution and solves optimization problems through the reproduction of individuals and the competition between individuals. Evolutionary computing aims to achieve "survival of the fittest" in species; accordingly, it aims to reach optimal solution in optimization problems. Evolutionary computing is also called evolutionary computation. Evolutionary computing (EC) is not a specific algorithm, but a collective term for many algorithms. For example, genetic algorithm preceded the name evolutionary computation, and genetic algorithm is a specific algorithm. In general, genetic algorithm is considered to be the earliest evolutionary computing method.

In the 1970s, genetic algorithm (GA) was first proposed by Holland in the United States [1]. In 1975, Holland published the monograph "Adaptation in Natural and Artificial Systems". In the book, he introduced GA and verified that it could solve the NP-hard (Nondeterministic Polynomial-Hard) problems with good results. Since

X. Zhang et al., *Intelligent Information Processing with Matlab*,
https://doi.org/10.1007/978-981-99-6449-9_5

then, many scholars have noticed GA as a method and have continued to derive more effective versions, so the GA proposed by Holland is often referred to as the simple genetic algorithm.

Genetic algorithm is a way to simulate the evolutionary mechanism of biological evolution in nature [2]. Based on Darwin's theory of biological evolution, GA translates the law of survival of the fittest into a strategy for finding the optimal solution. In scientific and practical problems, the function of GA is to find, among all possible solutions, the one that best fits the problem by satisfying the constraints. GA can provide an optimal solution to an optimization problem.

In the 1960s, Fogel in the United States proposed evolutionary programming (EP). In the same period, Rechenberg and Schwefel in Germany proposed evolution strategies (ES). They applied ES to complex engineering problems and achieved good results, thus gaining wide recognition. Methods such as GA, EP, and ES were developed alone for more than a decade. Until the 1980s, these methods did not attract much attention, partly because they were not mature enough by themselves and partly because they were not really applied to practical problems due to the limitations of computer performance.

In the 1990s, Koza in the United States proposed genetic programming (GP) in his monograph. GP uses hierarchical tree structure to express problems. After the branch of GP was proposed, Evolutionary computing began to emerge as a discipline. The four methods of GA, EP, ES and GP influence each other, learn from each other, and gradually evolve new evolutionary methods, which promote the rapid development of EC.

The GA mentioned earlier is able to solve NP-hard problems, which are actually a class of combinatorial optimization problems. When the learning rules of neural networks adjust the parameters of weights, gradient descent method is used to continuously approach the optimal weights through iterations. Subsequently, stochastic gradient descent and batch gradient descent methods are derived. All these methods need to calculate the gradient of the loss function, which is generally used to require the loss function to have continuity and differentiability. In combinatorial optimization problems, the values of the independent variables are often discrete, which makes the gradient-based methods no longer applicable. Gradient-based methods are sometimes referred to as traditional optimization methods, while EC methods such as GA are called modern optimization methods. This is because the gradient descent method dates back to 1847 and was proposed by Cauchy.

The travelling salesman problem (TSP) is a typical combinatorial optimization problem. The TSP problem is to find the shortest distance or optimal path to visit each city once and return to the starting point, given that some cities and their distances from each other are known. Suppose the number of cities in the TSP is N, then the possible paths to visit each city are $(N-1)!$, where the exclamation point denotes the factorial. We know that the factorial function tends to infinity very fast, and finding the optimal path from these possible paths is very difficult.

So far, the TSP has been derived in various forms, such as the multi-traveler problem. The problem is to have multiple travelers traversing some cities together, and the requirement is that all cities are passed through once and return to their

respective starting points to find the shortest path through all cities. In real life, vehicle routing problem (VRP) is such a multi-traveler problem. However, the VRP problem has more constraints, such as the demand of goods, the arrival time of vehicles, the capacity of vehicles, and the distance traveled.

With the development of EC, researchers have created many function optimization problems in order to test the performance of algorithms. These function optimization problems are synthetic problems, which are generally arithmetic and composite of basic elementary functions. For example, the Schwefer function is a composite of N power and sine functions, typically $N = 20$, with the independent variable x taking values in the range $[-500, 500]$. The function is very deceptive in that it has one global minimum and another local minimum at a more distant location. If an optimization method is trapped in a local minimum, it is difficult for the method to escape from the local region and thus cannot find the global minimum. The Schwefer function is:

$$\begin{aligned} \min \ & f(x) = -\sum_{i=1}^{N} x_i \sin\left(\sqrt{|x_i|}\right) \\ s.t. \quad & x_i \in [-500, 500] \\ & N = 20 \end{aligned} \tag{5.1}$$

As shown in Fig. 5.1, the Schwefer function has many minima, which is only the case for $N = 2$. When the number of independent variables increases, finding the global minima becomes more difficult.

For a better view of the minima, the front view of the Schwefel function is given in Fig. 5.2. From the figure, it can be seen that the global minimum is located near $x_1 = 400$, while the second minima point is located near $x_1 = -300$, which is far away from each other, which also indicates that the Schwefel function is very deceptive. If a certain EC method is able to find the global minimum of this function, it is reasonable to assume that this method has good performance.

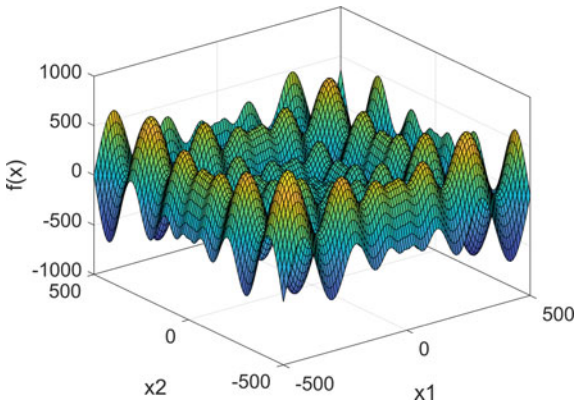

Fig. 5.1 Top view of the Schwefer function

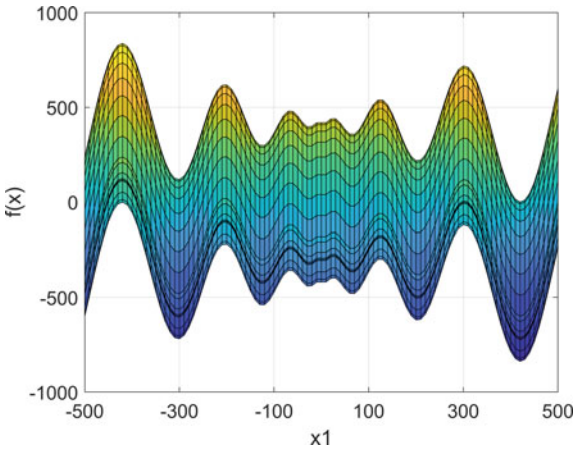

Fig. 5.2 Front view of the Schwefer function

In recent years, researchers have created a number of synthetic functions as benchmark problems to test the performance of algorithms, which are not described here. It should be noted that synthetic function problems and optimization problems in practice have their own advantages and disadvantages. The actual optimization problem is closer to the actual situation and is easily recognized, but has more constraints and is less uniform, open and scalable. On the other hand, the synthetic function has uniformity, openness and scalability. It is easy to calculate the global minimum of a synthetic function, which is convenient for evaluating the optimization effect of the method, but it is different from the optimization problem in practice. The EC method that performs well on the synthetic function may not necessarily achieve satisfactory results on the optimization problem in practice.

5.2 Simple Genetic Algorithm

This section introduces the simple genetic algorithm step by step through an example of an artificial synthesis function. The expression of the synthetic function used is:

$$\begin{aligned} \min \quad & f(x_1, x_2) = x_1^2 + x_2^2 \\ s.t. \quad & x_1 \in \{1, 2, 3, 4, 5, 6, 7\} \\ & x_2 \in \{1, 2, 3, 4, 5, 6, 7\} \end{aligned} \tag{5.2}$$

From Eq. (5.2), it can be seen that the global minimum of the objective function is $(x_1, x_2) = (1, 1)$ and is the only minimum. The steps required for the simple genetic algorithm (SGA) to solve the model in (5.2) include: individual coding, initial population generation, fitness calculation, selection operation, crossover operation, and mutation operation.

(1) Individual coding. In EC, each independent variable is assigned a possible value, and then the combination of the values of these independent variables constitutes a solution to the problem, called an individual. For example, (x1, x2) = (3, 4) is a solution to the problem model (5.2), but it is not a minimal value; it is only a candidate solution to the problem. The SGA does not directly use the values in the range of values of the independent variables, but encodes them in binary notation, thus mimicking the genes of the organism. Considering that the value of the independent variable is a positive integer between 1 and 7, it is possible to represent an independent variable in 3-bit binary. For example, the binary symbol 001 represents a positive integer 1, the binary symbol 010 represents a positive integer 2, and so on, 111 represents a positive integer 7. Here two independent variables can be represented by 6-bit binary symbols.

(2) Initial population generation. A population is a group of individuals, and the SGA uses a population to mimic a population of organisms. Suppose the size of the population is 4, i.e., the population consists of 4 individuals. We can use uniform distribution to randomly generate 0 s and 1 s as binary symbols and form individuals. Suppose the populations produced are: 011101, 101011, 011100 and 111001.

(3) Fitness calculation. According to the law of survival of the fittest, there is competition between individuals, which means that the merits of individuals should be compared. The fitness function can be used to measure an individual, i.e., to assign a value to an individual, so that the fitness value of an individual can be compared to determine the merit of an individual. In SGA, the fitness function is a non-negative function and the maximum value of the function is sought as the optimization objective, which requires a mapping from the objective function to the fitness function. In (5.2), the objective function is to find the minimal value point, and the range of the objective function is greater than 0. Thus, the fitness function can take the inverse of the objective function. By calculation, the fitness values of individuals in the population are shown in Table 5.1.

In Table 5.1, we give the number of each individual. For the individual numbered 1, the programs to calculate its fitness value are as follows:

```
x1 = bin2dec('011');
x2 = bin2dec('101');
fx = x1^2 + x2^2;
fitx = 1/fx;
```

Table 5.1 Fitness values of individuals in the population

Number	Individual	(x_1, x_2)	$f(x_1, x_2)$	Fitness
1	011101	(3, 5)	34	0.0294
2	101011	(5, 3)	34	0.0294
3	011100	(3, 4)	25	0.0400
4	111001	(7, 1)	50	0.0200

The rest of the individuals are calculated in a similar way and are not described in detail here.

(4) Selection operation

The selection operation refers to the selection of individuals with high fitness from the population for reproduction. In general, individuals with high fitness have a higher probability of being selected; conversely, individuals with low fitness have a lower probability of being selected. The SGA uses the roulette wheel method for the selection operation as follows:

First, the relative fitness of each individual is calculated, which is the probability of an individual being selected. That is the value of the fitness of an individual divided by the sum of the fitness of all individuals. Take the above population as an example, as shown in Table 5.2. The Matlab programs used are as follows:

```
fitx = [0.0294, 0.0294, 0.04, 0.02];
prob = fitx/sum(fitx);
probcum = cumsum(prob);
```

Next, the four individuals are lined up according to their numbers from smallest to largest, and each selected probability value corresponds to an interval as shown in the last column of Table 5.2, while the sum of all probability values is 1. We use a uniform distribution to generate random numbers between 0 and 1, generally to generate the same number of random numbers as the population size. Assume that the four random numbers generated are: 0.2, 0.8, 0.4, and 0.7, as shown in Table 3.3.

In Table 5.3, the first random number is 0.2, and the probability of the first individual being selected is 0.2475, so 0.2 falls in the region corresponding to the first individual, i.e., the first individual is selected. The second random number is 0.8, which falls in the area corresponding to the third individual, i.e., the third individual is selected. Similarly, the third and fourth random numbers are selected for the second and third individuals, respectively. It can be seen that the fourth individual is not selected because its probability of being selected is too low.

(5) Crossover operation

The crossover operation mimics the genetic crossover of chromosomes in an individual organism. The SGA uses a single-point crossover operator, which requires two individuals to participate in the operation. For example, we pair the first and

Table 5.2 Calculating the probability of an individual being selected

Number	Individual	Fitness	Probability	Interval
1	011101	0.0294	0.2475	[0, 0.2475)
2	101011	0.0294	0.2475	[0.2475, 0.4949)
3	011100	0.0400	0.3367	[0.4949, 0.8316)
4	111001	0.0200	0.1684	[0.8316, 1]

Table 5.3 Selection of individuals using random numbers

Number	Probability	Random number	Count	Selection result
1	0.2475	0.2	1	011101
2	0.2475	0.8	1	011100
3	0.3367	0.4	2	101011
4	0.1684	0.7	0	011100

second selected individuals and then randomly select the position for the crossover operation. An individual has 6 binary bits, so there are 5 possible positions for the crossover. We still use a uniform distribution to produce random numbers, assuming the position of the crossover is 2. Swapping the binary bits behind the crossover position gives us the individual after the crossover operation, as shown in Fig. 5.3.

We then pair the third and fourth selected individuals. By passing the random number generator, we assume that the crossover point is 1, and we exchange the binary bits following the crossover point. We obtain the two individuals after the crossover operation. It can be seen that four individuals are obtained after the crossover operation, which is the same size as the initial population.

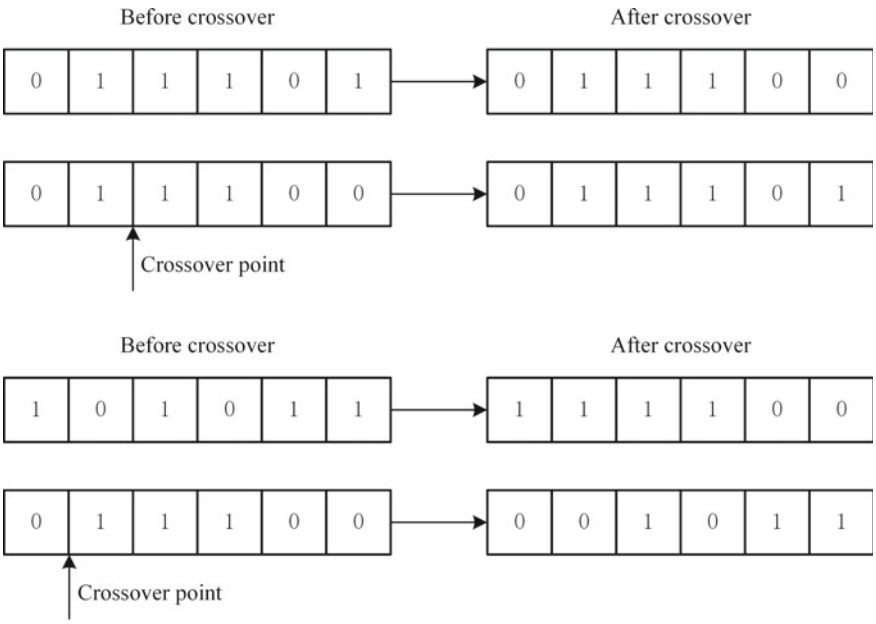

Fig. 5.3 Single-point crossover operation

(6) Mutation operation

The mutation operation mimics the genetic mutation of chromosomes in an individual organism. The SGA uses a basic bitwise mutation operator that mutates one of the individual's binary symbols, while the mutation positions are chosen randomly. For example, the specific steps of the mutation operation are given in Table 5.4.

As can be seen in Table 5.4, the mutation operation first requires determining the mutation point location, which is generally determined randomly using a certain probability to determine the location to be mutated. Next, the original binary bit of the mutation site is inverted to obtain the mutated result, which is a new individual. The mutation operation eventually results in a new population.

(7) The new population replaces the old one, then return to step (3), and repeat steps (3) to (6) until some termination condition is satisfied, then the SGA is finished

For the simple genetic algorithm, the last individual with the highest fitness in the population is output as the optimal solution of the model (5.2). The flowchart of the SGA is given in Fig. 5.4. After the previous step-by-step examples, the reader has a preliminary understanding of the genetic algorithm. The SGA provides a prototype and basis for solving optimization problems imitating biological evolution, and also provides a framework for an EC approach. Based on this, researchers have analyzed and discussed in depth the operation of each step of the genetic algorithm, and have developed many proven improvement techniques.

It is important to note that the SGA contains hyperparameters. For example, in step (2), the size of the initial population needs to be given in advance; in step (6), the probability of the mutation operation also needs to be predefined. The size of the population is generally denoted by "popsize" or Np. The probability of the mutation operation is generally denoted by p_m. In addition, in some improved crossover operations, which also contain hyperparameters, the probability during the crossover operation is denoted by p_c. These hyperparameters can affect the performance of genetic algorithms, just as the learning rate can affect the performance of neural networks. The study of genetic algorithm hyperparameters and, furthermore, the study of EC methods hyperparameters is a hot topic of research. Without causing confusion, such hyperparameters can be directly referred to as parameters.

Table 5.4 Basic bitwise mutation operation

Number	Crossover result	Basic bit position	Mutation result
1	011100	3	010100
2	011101	5	011111
3	111100	2	101100
4	001011	6	001010

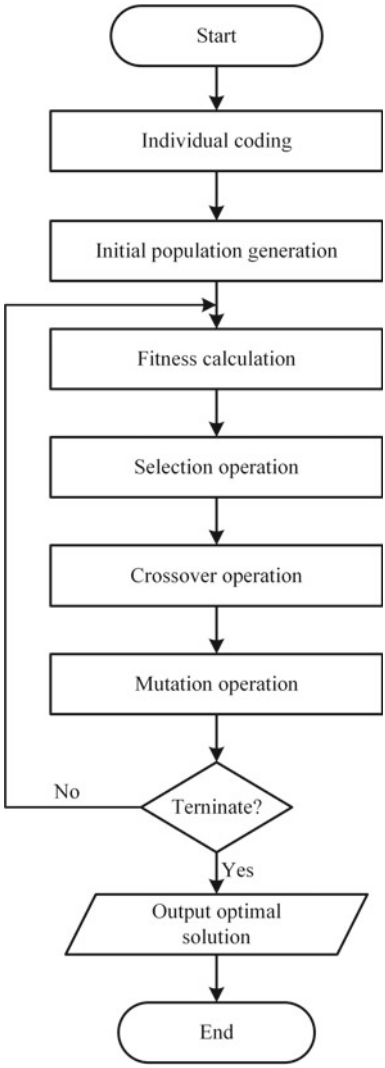

Fig. 5.4 Flowchart of the simple genetic algorithm

5.3 Genetic Algorithm for Travelling Salesman Problem

The concept of the TSP was introduced earlier, which is to find a shortest path that traverses all cities once and only once. An arrangement of all cities is a traversal path. We can number the cities and represent them with independent variables, which take different values to indicate different cities to be traversed, such that the independent variables are discrete and positive integers. Suppose there are n cities, the cities are numbered 1, 2, …, n.

The geographical location of cities is fixed, so the distance between cities is fixed. We can assume that the travel cost between two cities is known and fixed. The objective of the TSP is to find an ordered arrangement of visits to all cities such that the travel cost is minimized. We can take the distance between cities can be directly as the objective function, this is because the travel cost is generally proportional to the distance. We need to build a list of city locations and distances between cities.

Let's take the simplest TSP as an example so that we can concentrate on the introduction of genetic algorithm. From a graph theory perspective, cities can be viewed as nodes in a graph, and the paths between cities can be viewed as connecting lines between nodes in the graph, called edges. If the direction of travel between cities is not considered, then the connecting line between two cities is undirected. The TSP then boils down to the problem of finding a Hamiltonian loop in an empowered undirected graph such that the total weight is minimized.

The weighted undirected graph is denoted as $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes and E is the union of edges. The distance between the nodes is denoted as C. Then C is an n-th order symmetric matrix, and c_{ij} is denoted as a component of C. The mathematical expression of the TSP is:

$$\begin{aligned} \min \quad & f(x) = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \\ & \sum_{j\neq i} x_{ij} = 1, i \in V \\ & \sum_{i\neq j} x_{ij} = 1, j \in V \\ s.t. \quad & \sum_{i\in S} \sum_{j\in \overline{S}} x_{ij} \geq 1, S \subset V, S \neq \phi \\ & x_{ij} \in \{0, 1\}, i, j \in V \end{aligned} \tag{5.3}$$

From the above equation, it can be seen that $f(x)$ is the objective function, which is the traversal cost of a path. The first constraint indicates that each city must go out once; similarly, the second constraint indicates that each city can only go in once. Together, these two constraints mean that each city passes through once and only once. The third constraint is the elimination of subloops in the path. The last constraint is the range of values of the independent variable x_{ij}, which indicates whether the route from city i to city j is selected. If it is necessary to go from city i to city j, then $x_{ij} = 1$; otherwise $x_{ij} = 0$.

Suppose the traveler wants to visit some cities in the United States, which is also an example that comes with Matlab. It should be noted that the map used here is an abbreviated version and not a complete map of the United States, which can reduce the difficulty of the problem.

Example 5.1 Suppose there are $n = 40$ cities selected within the map boundary. The traveler needs to traverse all cities once and only once and return to the initial city location. The geographic locations of these 40 cities are known and the distances

between the cities have been given. Please use the genetic algorithm solver to solve the TSP and draw a graph to analyze the results.

Solution First, let's configure the basic data for the TSP, the programs used are as follows:

```
load('usborder.mat','x','y','xx','yy');
cities = 40;
locations = zeros(cities,2);
rng(1);
n = 1;
while (n <= cities)
    xp = rand*1.5;
    yp = rand;
    if inpolygon(xp,yp,xx,yy)
        locations(n,1) = xp;
        locations(n,2) = yp;
        n = n + 1;
    end
end
distances = zeros(cities);
for count1 = 1:cities
    for count2 = 1:count1
        x1 = locations(count1,1);
        y1 = locations(count1,2);
        x2 = locations(count2,1);
        y2 = locations(count2,2);
        distances(count1,count2) = sqrt((x1 - x2)^2 + (y1 - y2)^2);
        distances(count2,count1) = distances(count1,count2);
    end
end
figure1 = figure(1);
axes1 = axes('Parent',figure1);
hold(axes1,'on');
box(axes1,'on');
grid(axes1,'on');
plot(x, y, 'Color','black', 'LineWidth',2);
plot(locations(:,1),locations(:,2),'bo','LineWidth',2,'LineStyle','none');
hold(axes1,'off');
set(axes1,'FontSize',14);
```

The result of the above program after running is shown in Fig. 5.5, where the circle symbols indicate the location of the city.

Next, let's configure the operation steps of GA. According to the previous section, we are going to encode the individuals, and here the integer encoding is used instead of binary encoding, and the specific programs are as follows:

Fig. 5.5 City locations of the TSP of Example 5.1

```
function pop = create_permutations(NVARS,FitnessFcn,options)
totalPopulationSize = sum(options.PopulationSize);
n = NVARS;
pop = cell(totalPopulationSize,1);
for i = 1:totalPopulationSize
    pop{i} = randperm(n);
end
```

The number of cities is denoted as "NVARS". The *i*-th individual is denoted as pop{i}. An individual is an arrangement of integers from 1 to NVARS. It can be seen that an individual is a traversal route of all cities. The crossover operation is realized by:

```
function xoverKids = crossover_permutation(parents,options,NVARS, ...
    FitnessFcn,thisScore,thisPopulation)
nKids = length(parents)/2;
xoverKids = cell(nKids,1);
index = 1;
for i = 1:nKids
    parent = thisPopulation{parents(index)};
    index = index + 2;
    p1 = ceil((length(parent) - 1) * rand);
    p2 = p1 + ceil((length(parent) - p1 - 1) * rand);
    child = parent;
    child(p1:p2) = fliplr(child(p1:p2));
    xoverKids{i} = child;
end
```

From the above program, it can be seen that the crossover operation is not a single-point crossover, but a special crossover operation. This operation randomly

selects two gene positions and inverts the values between the two gene positions. The mutation operation is realized by:

```
function mutationChildren = mutate_permutation(parents ,options,NVARS, ...
    FitnessFcn, state, thisScore,thisPopulation,mutationRate)
mutationChildren = cell(length(parents),1);
for i = 1:length(parents)
    parent = thisPopulation{parents(i)};
    p = ceil(length(parent) * rand(1,2));
    child = parent;
    child(p(1)) = parent(p(2));
    child(p(2)) = parent(p(1));
    mutationChildren{i} = child;
end
```

From the above program, it is clear that the mutation operation is not a basic bitwise mutation; while it is a position swap method. The operation is to randomly select two mutated genes and then swap the values of the two positions. Crossover and mutation operations use "FitnessFcn" as the objective function as follows:

```
function scores = traveling_salesman_fitness(x,distances)
scores = zeros(size(x,1),1);
for j = 1:size(x,1)
    p = x{j};
    f = distances(p(end),p(1));
    for i = 2:length(p)
       f = f + distances(p(i − 1),p(i));
    end
    scores(j) = f;
end
```

From the above program, it can be seen that the objective function is to calculate the total distance of the path.

Finally, we will configure the parameters of the GA and draw a graph of the solution results. The programs for configuring GA are as follows:

```
FitnessFcn = @(x) traveling_salesman_fitness(x,distances);
my_plot = @(options,state,flag) traveling_salesman_plot(options, ...
    state,flag,locations);
options = optimoptions(@ga, 'PopulationType', 'custom',
'InitialPopulationRange', ...
    [1;cities]);
options = optimoptions(options,'CreationFcn',@create_permutations, ...
    'CrossoverFcn',@crossover_permutation, ...
    'MutationFcn',@mutate_permutation, ...
    'PlotFcn', my_plot, ...
    'MaxGenerations',500,'PopulationSize',50, ...
```

```
    'MaxStallGenerations',200,'UseVectorized',true);
numberOfVariables = cities;
[x,fval,reason,output] = ga(FitnessFcn, ...
    numberOfVariables,[],[],[],[],[],[],[],options);
```

To draw the search process of GA, the following program is used:

```
function state = traveling_salesman_plot(options,state,flag,locations)
persistent x y xx yy
if strcmpi(flag,'init')
  load('usborder.mat','x','y','xx','yy');
end
plot(x,y,'Color','black', 'LineWidth',2);
axis([− 0.1 1.5 − 0.2 1.2]);
hold on;
box on;
grid on;
[unused,i] = min(state.Score);
genotype = state.Population{i};
plot(locations(:,1),locations(:,2),'bo','LineWidth',2,'LineStyle','none');
plot(locations(genotype,1),locations(genotype,2),'LineWidth',2);
hold('off');
```

The results of the above program after running are shown in Fig. 5.6. The total length of the distance of the best path output by the genetic algorithm is about 5.05. As can be seen in the figure, the line of the two cities directly to the top right of the middle falls outside the map boundary. The path shown in the figure becomes infeasible if the route that restricts the traveler to travel must be inside the map boundary. This shows that the real-life traveler problem has more constraints, and our case here is a simplified problem.

Fig. 5.6 Optimal route found by genetic algorithm

The careful reader may notice that the GA presented in this section differs in many ways from the simple genetic algorithm in the previous section, and also does not present the selection operation and the calculation of the fitness function. The selection operation is performed using the roulette wheel method implemented in Matlab as follows:

```
function parents = selectionroulette(expectation,nParents,options)
expectation = expectation(:,1);
wheel = cumsum(expectation) / nParents;
parents = zeros(1,nParents);
for i = 1:nParents
    r = rand;
    for j = 1:length(wheel)
        if(r < wheel(j))
            parents(i) = j;
            break;
        end
    end
end
```

In the above programs, the expectation refers to the fitness value of all individuals, not the objective function value. The fitness function uses the objective function, but the selection operation does not use the fitness function value directly; it must be transformed to fit the selection operation. In Matlab, the default mapping method is the sorting-based fitness scaling method with the following programs:

```
function expectation = fitscalingrank(scores,nParents)
scores = scores(:);
[~,i] = sort(scores);
expectation = zeros(size(scores));
expectation(i) = 1 ./ ((1:length(scores)) .^ 0.5);
expectation = nParents * expectation ./ sum(expectation);
```

In the above procedure, scores refer to the objective function value, while expectation is the adapted value after conversion. The mapping method of converting the objective function value to the fitness value can affect the performance of GA. When using the scaling method, if the scaled values vary too much, individuals with high scaling values are likely to reproduce faster than those with low scaling values, i.e., individuals with high scaling values have a higher probability of being selected, which can limit the search range of GA. On the other hand, if the scaled value is too small, the probability of all individuals being selected tends to be the same, which reduces the convergence speed of GA and leads to a longer number of iterations and computation time.

The ranking-based fitness scaling method scales each individual according to its ranking in the population and does not directly use the original objective function value for scaling. For example, for the minimization optimization problem, the rank of the individual with the lowest objective function value is denoted as 1, the rank of

the individual with the second lowest objective function value is denoted as 2, and so on. The rank of the individual with the largest objective function value is denoted as "popsize". The scaled fitness value for the individual whose rank is i is:

$$fit(i) = \frac{1}{\sqrt{i}} \times popsize \tag{5.4}$$

where $fit(i)$ denotes the fitness value of the individual ranked as i. From (5.4), it can be seen that the fitness value is a multiple of $1/\sqrt{i}$. The ranking-based fitness scaling method can avoid the uneven dispersion of the objective function values. The individual ranked 1 has the largest scaled multiplier, while the remaining individuals have the same scaling multiplier.

Besides the above operation methods, we can find other selection operations, crossover operations and mutation operations. They are not described here.

5.4 Ant Colony Optimization Algorithm

EC is a collective name for a series of methods that mimic the theory of biological evolution to solve optimization problems. For example, GA mimic operations such as crossover and mutation of an organism's genes. A set of genes can form chromosomes, and a certain number of chromosomes make up a population, hence, genetic algorithms are simulations of the microcosm of biological evolutionary theory. Around the 1990s, researchers kept proposing new methods to mimic biological evolutionary theory, called Swarm Intelligence (SI), which mimics a population of organisms, e.g., ant swarm, bee swarm, bird swarm, etc. In contrast to GA, the SI approach is a simulation of the macrocosm of the organism. There is no unified criterion whether SI methods belong to the category of EC. We do not discuss the affiliation between them here. For simplicity, we group SI methods into the chapter of EC and introduce SI methods starting from this section.

The ant colony optimization (ACO) algorithm was proposed by Dorigo in his Ph.D. thesis in 1991 [3]. Ant Colony Optimization is also known as ant colony algorithm or ant algorithm. The starting point of ACO is that a colony of ants with slight intelligence exhibits intelligent behavior during foraging by cooperating and communicating with each other, and this phenomenon provides new insights for solving optimization problems.

The ACO algorithm was inspired by the double-bridge experiment by Deneubourg and his colleagues [4]. The double-bridge experiment consists of a colony of ants, two bridges, and a food source, i.e., the ants and the food source are located at opposite ends, with two bridges in parallel in the middle. If the two bridges were of the same length, i.e., the distance from the ant nest to the food source was the same, the double bridge experiment revealed that the ants would gather on one bridge after a period of time. If one bridge was shorter than the other, i.e., the lengths of the two bridges were significantly different, the experiment found that the ants would

congregate on the shorter bridge after a period of time. The double-bridge experiment showed that ants make random choices when they encounter different paths on their way to find food sources. The ants leave pheromones on the path as they move, and the pheromones change the information in the environment, allowing individual ants to communicate with each other. Initially, ants make random choices, but as pheromones accumulate and change, ants adjust the probability of choice according to the size of the pheromone. In the experiment, it was shown that the ants chose the bridge with the shorter distance.

The ACO algorithm was first used to solve TSP. This section will also use the TSP from the previous section to introduce the ACO algorithm.

Example 5.2 Suppose there are $n = 40$ cities within the selected map boundary, the traveler needs to traverse all cities once and only once and return to the initial city location. The geographic locations of these 40 cities are known and the distances between the cities are given. Write a program for the ACO algorithm to solve the problem and draw a graph to analyze the results.

Solution This example is the same as the previous section, so the background of the problem is not presented. The programs are as follows:

```
popsize = 50;
Alpha = 1;
Beta = 5;
Rho = 0.1;
NC_max = 200;
Q = 100;
D = distances;
for i = 1:cities
    D(i, i) = eps;
end
Eta = 1./D;
Tau = ones(cities,cities);
Tabu = zeros(popsize,cities);
R_best = zeros(NC_max,cities);
L_best = inf.*ones(NC_max,1);
L_ave = zeros(NC_max,1);
NC = 1;
while NC <= NC_max
    Randpos = [];
    for i = 1:(ceil(popsize/cities))
        Randpos = [Randpos,randperm(cities)];
    end
    Tabu(:,1) = (Randpos(1,1:popsize))';
    for j = 2:cities
        for i = 1:popsize
            visited = Tabu(i,1:(j - 1));
```

```
            J = zeros(1,(cities − j + 1));
            P = J;
            Jc = 1;
            for k = 1:cities
                if length(find(visited == k)) == 0
                   J(Jc) = k;
                   Jc = Jc + 1;
                end
            end
            for k = 1:length(J)
              P(k) = (Tau(visited(end),J(k))^Alpha)*(Eta(visited(end),J(k))^Beta);
            end
            P = P/(sum(P));
            Pcum = cumsum(P);
            Select = find(Pcum >= rand);
            to_visit = J(Select(1));
            Tabu(i,j) = to_visit;
        end
    end
    if NC >= 2
        Tabu(1,:) = R_best(NC − 1,:);
    end
    L = zeros(popsize,1);
    for i = 1:popsize
        R = Tabu(i,:);
        for j = 1:(cities − 1)
           L(i) = L(i) + D(R(j),R(j + 1));
        end
        L(i) = L(i)+D(R(1),R(cities));
    end
    L_best(NC) = min(L);
    Pos = find(L == L_best(NC));
    R_best(NC,:) = Tabu(pos(1),:);
    L_ave(NC) = mean(L);
    NC = NC + 1;
    Delta_Tau = zeros(cities,cities);
    for i = 1:popsize
       for j = 1:(cities − 1)
         Delta_Tau(Tabu(i,j),Tabu(i,j + 1)) = Delta_Tau(Tabu(i,j),Tabu(i,j + 1))
+ Q/L(i);
        end
        Delta_Tau(Tabu(i,cities),Tabu(i,1)) = Delta_Tau(Tabu(i,cities),Tabu(i,1))
+ Q/L(i);
    end
    Tau = (1-Rho).*Tau+Delta_Tau;
```

```
        Tabu = zeros(popsize,cities);
    end
    Pos = find(L_best == min(L_best));
    Shortest_Route = R_best(Pos(1),:);
    Shortest_Length = L_best(Pos(1));
    figure1 = figure(2);
    axes1 = axes('Parent', figure1);
    hold(axes1,'on');
    box(axes1,'on');
    grid(axes1,'on');
    plot(x, y, 'Color','black', 'LineWidth',2);
    plot(locations(:,1),locations(:,2),...
        'bo','LineWidth',2,'LineStyle','none');
    plot(locations(Shortest_Route,1), ...
        locations(Shortest_Route,2),'LineWidth',2);
    plot([locations(Shortest_Route(1),1), ...
        locations(Shortest_Route(end),1)], ...
        [locations(Shortest_Route(1),2), ...
        locations(Shortest_Route(end),2)],'LineWidth',2);
    hold(axes1,'off');
    set(axes1,'FontSize',14);
    figure1 = figure(3);
    axes1 = axes('Parent', figure1);
    hold(axes1,'on');
    box(axes1,'on');
    grid(axes1,'on');
    plot(L_best, '-', 'LineWidth',2);
    plot(L_ave, ':', 'LineWidth',2);
    legend('shortest distance','average distance');
    xlabel('number of iterations');
    ylabel('objective function');
    hold off;
```

After running the above program, we obtain two figures as shown in Figs. 5.7 and 5.8. In Fig. 5.7, it gives the optimal travel route found by the ACO algorithm. Figure 5.8 gives the convergence process of the ACO algorithm, including the shortest path distance and the average path distance for each generation during the iteration.

The total distance length of the optimal route output by the ACO algorithm is about 5.13, which is greater than the result given by the GA. This indicates that the optimal path given by the ACO is slightly worse than the optimal path given by the GA. Note that the simulation results in this section alone do not allow us to conclude that the performance of the ACO is inferior to that of the GA. This is because the comparison of two algorithms requires certain criteria and cannot be concluded based on a single simulation result. The comparison criteria for EC methods are more complex and will not be described in detail here.

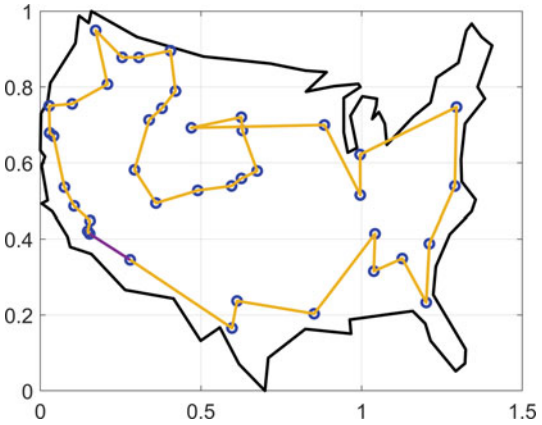

Fig. 5.7 Optimal route found by the ACO algorithm

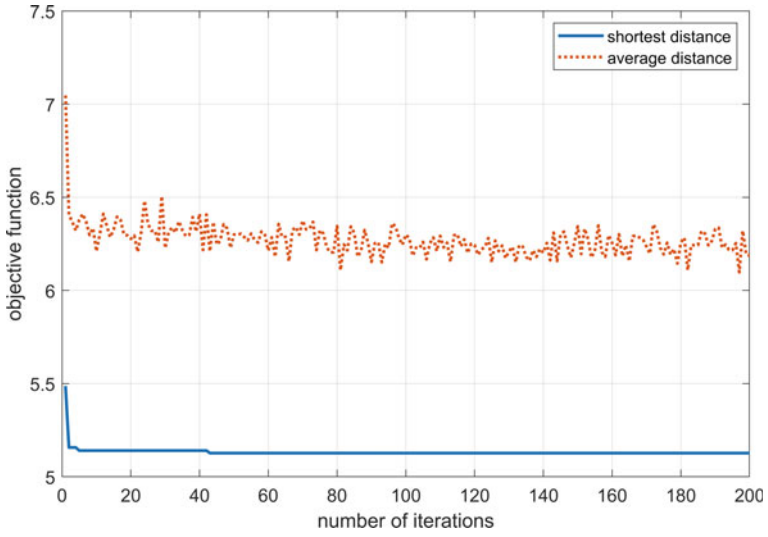

Fig. 5.8 Convergence process of the ACO algorithm

From the Example 5.2, we have a preliminary understanding of the ACO algorithm. The flowchart of the ACO algorithm is given in Fig. 5.9. From the figure, we can see that the first step of the ACO algorithm is to configure the initial parameters for ACO. The initial operating environment include the ant population size, which is recorded as "popsize". The search space of the TSP is discrete, so the movement of ants from one city to the next is the process of state transfer, which is the second step of the algorithm. After the ants move, the pheromone needs to be updated, which is the third step of the algorithm. In the fourth step, the termination condition is

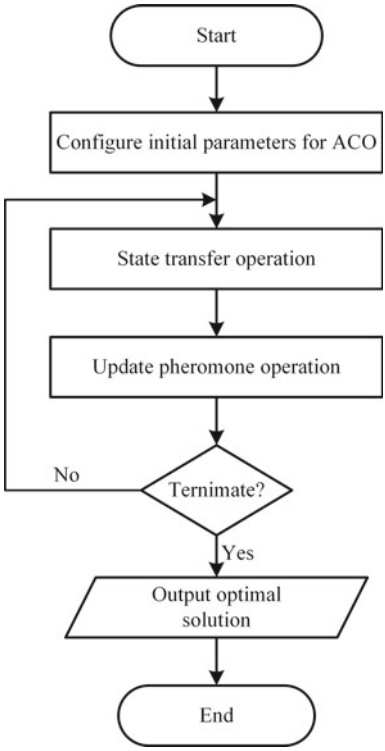

Fig. 5.9 Flowchart of the ACO algorithm

judged, and if not, the second and third steps are repeatedly executed; otherwise, the algorithm is terminated and the optimal solution is output. It can be seen that the second and third steps are the core part of the algorithm.

In the second step, the state transfer operation of ants is based on a certain probability to choose the next city. In making the choice, the ant can only go to cities that it has not visited, thus satisfying that each city is visited. The ant has to try to choose the route with high pheromone concentration and thus walk according to the residual pheromone on the route. The ant also has to try to choose the more favorable city within its visibility range, i.e., each ant needs to know its visibility to locally determine a favorable walking route. As can be seen, state transfer of ants requires a certain probability of making a choice. The mathematical expression for the transfer probability is:

$$p_{ij}^{k}(t) = \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{s \in allowed_k} \tau_{is}^{\alpha}(t)\eta_{is}^{\beta}(t)}, j \in allowed_k \tag{5.5}$$

where k denotes the k-th ant, i and j denote cities, τ_{ij} denotes the pheromone on the route from city i to city j, η_{ij} denotes the heuristic information from city i to city j, α denotes the importance of the pheromone, β denotes the importance of the heuristic

information, and $allowed_k$ denotes the set of cities that the k-th ant is allowed to visit. Equation (5.5) shows that the state transfer of the ant relies on the pheromone of the route and also relies on the number of cities it can see.

There are also some tricks to update the pheromone. The ants update the pheromone after the end of the trip. The original pheromone will volatilize after each iteration. Each ant will make a recommendation for its walking route. This recommendation is based on the consumption cost of the ant during the whole trip. The mathematical expression of the pheromone update is:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{popsize} \Delta\tau_{ij}^{k} \tag{5.6}$$

where ρ is the pheromone volatility parameter between 0 and 1, $\Delta\tau_{ij}^{k}$ indicating the recommendation of all ants for the route from city i to city j. The mathematical expression of $\Delta\tau_{ij}^{k}$ is:

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k, & ij \in l_k \\ 0, & ij \notin l_k \end{cases} \tag{5.7}$$

where Q is a constant, L_k denotes the length of the kth ant's walking path, and l_k denotes the k-th ant's walking path. From (5.6), it can be seen that the shorter the length of the path an ant walks, the higher the pheromone it leaves on the route through the city. Thus, more ants will choose the route with the shorter path in the subsequent iterations.

This section presents only the simplest version of the ACO algorithm. It performs poorly when faced with larger traveler problems, thus researchers have proposed a number of improvement techniques to improve the performance of the ACO algorithm.

5.5 Particle Swarm Optimization Algorithm

The particle swarm optimization (PSO) algorithm was proposed by Eberhart and Kennedy in 1995 [5]. The PSO algorithm aims to solve optimization problems by using inter-individual collaboration and information sharing. The PSO algorithm was inspired by the behavior of a flock of birds during travel. The researchers found that the flock of birds would suddenly change direction during travel and that the changes were synchronized in a group. Moreover, the birds would often gather together and disperse at the same time when there was a situation. Thus, the researchers speculated that there was some underlying rule at work that made the flock of birds exhibit this behavior. Specifically, the rules of flock behavior are:

(1) Birds are to avoid collisions with their neighbors;
(2) The bird should remain similar in speed and consistent in-flight attitude to the birds around it;
(3) The birds should move closer to the center of the group they identify and not too far away from the center.

The PSO algorithm is a method of group collaboration designed formally by simulating the predatory behavior of a flock of birds. It should be noted that although the method was proposed by researchers observing the behavior of flocks of birds, the name of the method uses particle swarms and does not use flocks of birds. Particle flocks can be seen as a more generalized concept, where the particles can be birds or other species.

The flow chart of the PSO algorithm is shown in Fig. 5.10. As shown in the figure, the PSO algorithm first needs to initialize the parameters such as position and velocity of the particles. Second, there should be a fitness function to calculate the fitness of each particle. Each particle is able to search for what it thinks is the global optimum and the local optimum particle. Then, each particle updates its own velocity and position. When all particles complete the above steps, the particle swarm has completed a generation of search and foraging behavior. These steps continue until the algorithm terminates when a certain condition is met, and finally the optimal solution is output.

GA and ACO algorithms were first used to solve the TSP. TSP is an optimization problem in which the independent variables are discrete. Unlike the GA and ACO algorithms, the PSO algorithm was first used to solve optimization problems in which the independent variables are continuous. In the following, we describe the details of the PSO algorithm.

Step (1) Initialize the velocity and position of the particles. Initially, the position of each particle in the particle swarm is randomly placed in the search space. The moving velocity of each particle is generally set to the distance from the particle's location to the farthest boundary. The velocity is set so that the particle can search the entire search space. This provides a guarantee for finding the global minimum. Suppose the size of the particle swarm is "popsize". The position of particle i is denoted as Xi, and the moving velocity of particle i is denoted as Vi.

Step (2) Calculate the fitness of the particles. In the PSO algorithm, the objective function of the optimization problem can be used as the fitness function. With the fitness function, the fitness of each particle can be calculated, denoted as f(Xi).

Step (3) Find local best and global best particles. Each particle searches for its own perceived local best and global best particles. For the minimization problem, the global best particle is the location with the smallest fitness. To find the local best particle, it is necessary to determine for each particle the range in which it can be seen. For example, each particle is able to observe the positions of k particles around it, i.e., the size of the local neighborhood is k. Euclidean distance is generally used to measure the proximity between particles so that the particles contained in the local neighborhood can be determined. The local best position of particle i is denoted as pbesti, and the global optimal position of all particles is denoted as gbest.

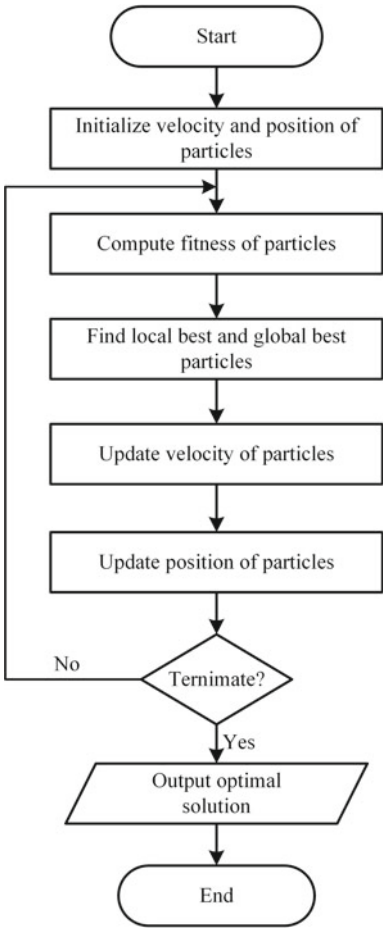

Fig. 5.10 Flow chart of the PSO algorithm

Step (4) Update the velocity of particles. Each particle determines the velocity of the movement based on itself and the surrounding particles with the following expression:

$$V_i(t+1) = V_i(t) + c_1 r_1 (pbest_i - X_i(t)) + c_2 r_2 (gbest - X_i(t)) \tag{5.8}$$

where t denotes the number of iterations, c_1 and c_2 are learning factors, also known as acceleration factors. r_1 and r_2 are random numbers between 0 and 1, which are randomly generated at each iteration and can adjust the particle's moving speed. In addition, after the particles calculate their velocities based on (5.8), it is possible that they become particularly large. It is necessary to set the velocity range of the particle flight, denoted as $[-V_{max}, V_{max}]$.

Step (5) Update the position of particles. Each particle is adjusted according to its own position and moving velocity. Its mathematical expression is:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{5.9}$$

Step (6) Determine whether to terminate the search, if the termination condition is not met, return to Step (2) to continue the execution; otherwise, go to Step (7).

Step (7) Output the global best solution gbest.

The basic steps of the PSO algorithm are introduced. It is easy to see that the most critical parts of the method are Steps (3) and (4). In particular, Step (4) determines the next position of the particle. The above presented is only the original version of the PSO algorithm, which sometimes does not work well. The PSO algorithm with inertia factor was proposed by Yuhui Shi et al. in 1998. The velocity expression used is:

$$V_i(t+1) = \omega V_i(t) + c_1 r_1(pbest_i - X_i(t)) + c_2 r_2(gbest - X_i(t)) \tag{5.10}$$

which ω is called the inertia factor and generally takes a non-negative value. As can be seen from (5.10), a larger ω value makes the particle's velocity large and can search a larger area; conversely, a smaller ω value makes the particle's velocity small and more suitable for local search. The linear decreasing strategy of the inertia factor is a frequently used technique. Its expression is:

$$\omega(t) = \omega_{min} + (\omega_{max} - \omega_{min})\frac{(t_{max} - t)}{t_{max}} \tag{5.11}$$

where ω_{min} and ω_{max} are the minimum and maximum values of the inertia factor, t is the current generation, and t_{max} is the maximum number of iterations of the PSO algorithm.

Due to the introduction of the inertia factor, the PSO algorithm using (5.10) and (5.11) has achieved better results and gained recognition to the extent that (5.10) and (5.11) are often used as the standard PSO algorithm.

Example 5.3 The sphere function is a single-peaked function, also known as the DeJong function, whose expression is:

$$\begin{aligned} \min\ & f(x_1, x_2) = x_1^2 + x_2^2 \\ s.t.\quad & x_1 \in [-100, 100] \\ & x_2 \in [-100, 100] \end{aligned} \tag{5.12}$$

There are only two independent variables in the above equation. The spherical function is scalable and the number of independent variables can be adjusted as needed. Please use the PSO algorithm to solve the problem and draw a graph to analyze the results.

Solution In this example, we solve the problem by two ways. One is using toolbox in Matlab; the other is write program by ourselves to solve the problem. The independent variables of this example are continuous. The objective function can be used as the

fitness function. It is easy to see that (0, 0) is the minimal value point of the sphere function with the following programs:

```
function fx = functionSphere(x)
fx = sum(x.^2, 2);
```

In the above program, x can be a single particle, or be a swarm of particles. Each row of x can be a feasible solution of sphere function. We can use the toolbox of Matlab to solve the problem, and the programs are as follows:

```
rng(1);
fun = @functionSphere;
nvars = 2;
lb = [− 100, − 100];
ub = [100, 100];
options = optimoptions('particleswarm','SwarmSize',50);
[x, fval, exitflag, output] = particleswarm(fun, nvars, lb, ub, options);
```

After running the above program, we obtain the best solution of the PSO algorithm, which is approximately equal to (0, 0). The objective function value is also approximately equal to 0.

Except using the toolbox of Matlab to solve the problem, we can also write our own programs for the PSO algorithm:

```
function [xbest, fbest, cvgef] = PSO(fhd, xdim, xlb, xub, maxFEs)
popsize = 40;
omiga = 1/(2*log(2)) * ones(1, popsize);
maxomiga = 0.9;
minomiga = 0.4;
c1 = 0.5 + log(2);
K = 3;
popu = zeros(popsize, xdim); fpopu = zeros(popsize, 1);
tfun = 0;
for inp = 1:popsize
    popu(inp, :) = xlb + (xub - xlb).*rand(1, xdim);
    t1 = tic;
    fpopu(inp, 1) = feval(fhd, popu(inp, :));
    tfun = tfun + toc(t1);
end
xbestidx = find(fpopu == min(fpopu));
if ~isempty(xbestidx)
    xbestidx = xbestidx(end);
    fbest = fpopu(xbestidx);
    xbest = popu(xbestidx, :);
else
    fbest = inf;
```

```
    xbest = inf * ones(1, xdim);
end
vmin = − 2*(xub-xlb); vmax = 2*(xub-xlb);
vel = repmat(vmin, popsize, 1) + rand(popsize, xdim).*(repmat(vmax-vmin,
popsize, 1));
pbest = popu; fpbest = fpopu;
neighbor = neighborSelection(popu, fpopu, K);
cvgef = nan * ones(1, maxFEs);
ieval = popsize;
igen = 1; imprFlag = 1;
idxf1 = 1; idxf2 = ieval; idxstat = 1; isidxstatupdated = − 1;
while ieval < maxFEs
    cvgef(idxf1:idxf2) = fbest;
    isidxstatupdated = − 1;
    if imprFlag < 0
        neighbor = neighborSelection(popu, fpopu, K);
        imprFlag = 1;
    end
    [nbest, fnbest] = update_nbest(pbest, fpbest, neighbor);
    popunew = nan * ones(popsize, xdim); fpopunew = nan * ones(popsize, 1);
    velnew = nan * ones(popsize, xdim);
      omiga = inertiaWeightAdjustment(omiga, minomiga, maxomiga, ieval,
maxFEs);
    for inp = 1:popsize
        veltmp = c1 * rand(1,xdim).*(pbest(inp, :) - popu(inp, :)) + …
            c1 * rand(1,xdim).*(nbest(inp, :) - popu(inp, :));
        velnew(inp, :) = omiga(1,inp) * vel(inp, :) + veltmp;
        for ix = 1:xdim
    if velnew(inp, ix) < vmin(1, ix)
            velnew(inp, ix) = vmin(1, ix) + rand(1) * (vmax(1,ix)-vmin(1,ix));
            elseif velnew(inp, ix) > vmax(1, ix)
             velnew(inp, ix) = vmax(1, ix) + rand(1) * (vmax(1,ix)-vmin(1,ix));
            end
        end
        popunew(inp, :) = popu(inp, :) + velnew(inp, :);
        for ix = 1:xdim
            if popunew(inp, ix) < xlb(ix)
                popunew(inp, ix) = xlb(ix) + rand(1)*(xub(ix)-xlb(ix));
            elseif popunew(inp, ix) > xub(ix)
                popunew(inp, ix) = xub(ix) + rand(1)*(xub(ix)-xlb(ix));
            end
        end
    end % for inp
    t1 = tic;
    for inp = 1:popsize
```

```
            fpopunew(inp, 1) = feval(fhd, popunew(inp, :));
            ieval = ieval + 1;
        end
        tfun = tfun + toc(t1);
        popu = popunew; fpopu = fpopunew;
        vel = velnew;
        [pbest, fpbest] = update_pbest(popu, fpopu, pbest, fpbest);
        igen = igen + 1;
        [fbesttmp, idx] = min(fpopu);
        if fbesttmp(1) < fbest
            fbest = fbesttmp(1);
            xbest = popu(idx(1), :);
            imprFlag = 1;
        else
            imprFlag = − 1;
        end
        idxf1 = idxf2 + 1;
        idxf2 = ieval;
        idxstat = idxstat + 1;
        isidxstatupdated = 1;
    end
    if isidxstatupdated < 0
        idxf1 = idxf2 + 1;
        idxf2 = maxFEs;
        idxstat = idxstat + 1;
    end
    if idxf2 > maxFEs
        idxf2 = maxFEs;
    end
    cvgef(idxf1:idxf2) = fbest;
    end
```

The above program is the main program of the PSO algorithm. When we compute the local best and global best particles, we use the following programs:

```
function [nbest, fnbest] = update_nbest(popu, fpopu, neighbor)
[np, xdim] = size(popu);
nbest = nan * ones(np, xdim); fnbest = nan * ones(np, 1);
for irow = 1:np
    nidx = neighbor(irow, :) > 0.5;
    ftmp = fpopu(nidx); xtmp = popu(nidx, :);
    [fnbest(irow), bestidx] = min(ftmp);
    nbest(irow, :) = xtmp(bestidx, :);
end
end
```

When calculating local optimum, the local neighborhood is computed by the following programs:

```
function neighbor = neighborSelection(popu, fpopu, K)
np = size(popu, 1);
neighbor = eye(np,np);
for irow = 1:np
  nidx = randperm(np);
  nidx(nidx == irow) = [];
  neighbor(irow, nidx(1:K)) = 1;
end
end
```

After movement, each particle has to record its local best. The programs are:

```
function [pbest, fpbest] = update_pbest(popu, fpopu, pbest, fpbest)
for irow = 1:length(fpbest)
   if fpopu(irow) < fpbest(irow)
      pbest(irow, :) = popu(irow, :);
      fpbest(irow) = fpopu(irow);
   end
end
end
```

The linear decreasing strategy of the inertia factor is realized by:

```
function omiga = inertiaWeightAdjustment(omiga, minomiga, maxomiga, ieval,
maxFEs)
for irow = 1:length(omiga)
    omiga(irow) = ((maxFEs − ieval) * (maxomiga - minomiga)) / (maxFEs −
1) + minomiga;
end
end
```

If we want to use a custom PSO algorithm to solve the sphere function problem, the programs are as follows:

```
rng(1);
fun = @functionSphere;
nvars = 2;
lb = [− 100, − 100];
ub = [100, 100];
maxFEs = 3000;
[xbest, fbest, cvgef] = PSO(fun, nvars, lb, ub, maxFEs);
```

After running the above program, we see that the optimal solution output by the algorithm is approximately equal to (0, 0), and the corresponding objective function value is close to 0. This is the same result obtained using the toolbox in Matlab.

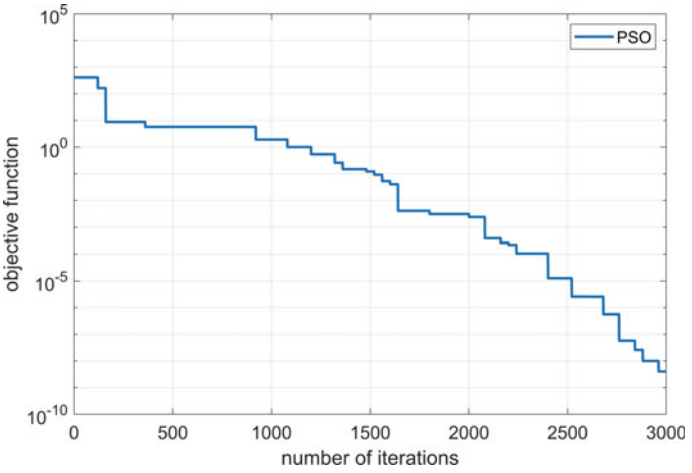

Fig. 5.11 Convergence curve of the PSO algorithm on Example 5.3

The advantage of using a custom program is the manipulability. For example, the "cvgef" variable returned in the above custom program, which records the optimal fitness value of the algorithm for each generation, can be used to plot the convergence curve of the algorithm, as shown in Fig. 5.11. In this figure, where the vertical axis is on a logarithmic scale, making it easier to observe the convergence process of the algorithm.

The programs for Fig. 5.11 are as follows:

```
figure1 = figure(1);
axes1 = axes('Parent',figure1);
box(axes1,'on');
grid(axes1,'on');
plot(1:maxFEs, cvgef, 'LineWidth',2);
set(axes1,'FontSize',14,'YMinorTick','on','YScale','log');
set(axes1,'FontSize',14);
xlabel('number of iterations');
ylabel('objective function');
```

As shown in Fig. 5.11, the PSO algorithm finds the optimal fitness value is about 10^{-9} after 3000 iterations. We know that the global minimum of the sphere function is 0, and the optimal value of the PSO algorithm output is about 10^{-9}. The difference between the two is less than 10^{-6}. When using computer simulation, since the floating-point accuracy is limited, we generally consider that less than 10^{-6} is equivalent to the algorithm finding the global minimum.

Although the PSO algorithm was proposed for continuous optimization problems, it has been modified by researchers to handle discrete optimization problems as well. For example, we can use the PSO algorithm to solve the TSP problem.

Example 5.4 Suppose there are $n = 40$ cities within the map boundary, the traveler needs to traverse all cities once and only once and return to the initial city location. The geographic locations of these 40 cities are known and the distances between the cities are given. Write a program for the PSO algorithm to solve the TSP problem, and draw a graph to analyze the results.

Solution This example has been used earlier, so the background of the problem will not be introduced. The specific programs of the PSO algorithm are as follows:

```
load('usborder.mat','x','y','xx','yy');
cities = 40;
locations = zeros(cities,2);
rng(1);
n = 1;
while (n <= cities)
    xp = rand*1.5;
    yp = rand;
    if inpolygon(xp,yp,xx,yy)
        locations(n,1) = xp;
        locations(n,2) = yp;
        n = n+1;
    end
end
distances = zeros(cities);
for count1 = 1:cities
    for count2 = 1:count1
        x1 = locations(count1,1);
        y1 = locations(count1,2);
        x2 = locations(count2,1);
        y2 = locations(count2,2);
        distances(count1,count2) = sqrt((x1 - x2)^2+(y1 - y2)^2);
        distances(count2,count1) = distances(count1,count2);
    end
end
popsize = 50;
IterNum = 200;
c1 = 0.5;
c2 = 0.1;
omiga = 1/(2*log(2)) * ones(1, popsize);
popu = zeros(popsize,cities);
v = zeros(popsize,cities);
iter = 1;
fpopu = zeros(popsize,1);
pbest = zeros(popsize,cities);
pbest_fit = zeros(popsize,1);
```

```
gbest = zeros(IterNum,cities);
gbest_fit = zeros(popsize,1);
avgl = zeros(IterNum,1);
maxomiga = 0.9;
minomiga = 0.4;
for i = 1:popsize
    popu(i,:) = randperm(cities);
    v(i,:) = randperm(cities);
end
for i = 1:popsize
    for j = 1:cities-1
        fpopu(i) = fpopu(i) + distances(popu(i,j),popu(i,j+1));
    end
    fpopu(i) = fpopu(i) + distances(popu(i,end),popu(i,1));
end
pbest_fit = fpopu;
pbest = popu;
[gbest_fit(1),min_index] = min(fpopu);
gbest(1,:) = popu(min_index);
avgl(1) = mean(fpopu);
while iter <IterNum
    iter = iter +1;
    omiga = maxomiga-(maxomiga-minomiga)*(iter/IterNum)^2;
    change1 = positionChange(pbest,popu);
    change1 = changeVelocity(c1,change1);
    change2 = positionChange(repmat(gbest(iter-1,:),popsize,1),popu);
    change2 = changeVelocity(c2,change2);
    v = originalVelocity(omiga, v);
    for i = 1:popsize
        for j = 1:cities
            if change1(i,j) ~= 0
                v(i,j) = change1(i,j);
            end
            if change2(i,j) ~= 0
                v(i,j) = change2(i,j);
            end
        end
    end
    popu = updatePosition(popu,v);
    fpopu = zeros(popsize,1);
    for i = 1:popsize
        for j = 1:cities-1
            fpopu(i) = fpopu(i) + distances(popu(i,j),popu(i,j+1));
        end
        fpopu(i) = fpopu(i) + distances(popu(i,end),popu(i,1));
```

```
        end
        for i = 1:popsize
            if fpopu(i) < pbest_fit(i)
                pbest_fit(i) = fpopu(i);
                pbest(i,:) = popu(i,:);
            end
        end
        [minvalue,min_index] = min(fpopu);
        if minvalue <gbest_fit(iter-1)
            gbest_fit(iter) = minvalue;
            gbest(iter,:) = popu(min_index,:);
        else
            gbest_fit(iter) = gbest_fit(iter-1);
        gbest(iter,:) = gbest(iter-1,:);
        end
        avgl(iter) = mean(fpopu);
    end
    [bestRouteLen,index] = min(gbest_fit);
    bestRoute = gbest(index,:);
    figure1 = figure1 (2);
    axes1 = axes('Parent',figure1);
    hold(axes1,'on');
    box(axes1,'on');
    grid(axes1,'on');
    plot(x, y, 'Color','black', 'LineWidth',2);
    plot(locations(:,1),locations(:,2),...
        'bo','LineWidth',2,'LineStyle','none');
    plot(locations(bestRoute,1), ...
        locations(bestRoute,2),'LineWidth',2);
    plot([locations(bestRoute(1),1), ...
        locations(bestRoute(end),1)], ...
        [locations(bestRoute(1),2), ...
        locations(bestRoute(end),2)],'LineWidth',2);
    hold(axes1,'off');
    set(axes1,'FontSize',14);
    figure1 = figure1 (3);
    axes1 = axes('Parent',figure1);
    hold(axes1,'on');
    box(axes1,'on');
    grid(axes1,'on');
    plot(gbest_fit', '-', 'LineWidth',2);
    plot(avgl', ':', 'LineWidth',2);
    legend('shortest distance','average distance');
    xlabel('number of iterations');
    ylabel('objective function');
    hold off;
```

Fig. 5.12 Optimal route found the PSO algorithm for the TSP

After the above program is run, we obtain the optimal value and the optimal solution found by the PSO algorithm, as shown in Fig. 5.12.

It can be seen from Fig. 5.12 that the optimal route output by the PSO algorithm is not good. The route in the figure has large straight-line segments. In fact, the total distance length of the route output by the PSO is about 7.43, which is larger than the results given by the GA and the ACO algorithms.

The convergence curve of the PSO algorithm is given in Fig. 5.13. The solid line in the figure indicates the shortest distance of the particle swarm at each generation, while the dashed line indicates the average distance of the particle swarm at each generation. It can be seen that after 200 generations, the shortest distance gradually decreases and level off, indicating that the PSO algorithm is close to convergence. However, compared with the GA and ACO algorithms, the output of the PSO algorithm is not good, which to a certain extent indicates that the PSO algorithm is not very suitable for discrete optimization problems. The performance of the PSO algorithm needs to be further improved.

5.6 Differential Evolution Algorithm

This section introduces the differential evolution (DE) algorithm and summarizes the research related to the DE algorithm.

The DE algorithm is derived from the genetic annealing algorithm proposed and published by Storn and Price in 1997 [6]. The DE algorithm is simple to implement programmatically, easy to use, and fast to compute. The conventional DE algorithm can be implemented in Matlab with less than 70 lines of programming code. The DE algorithm was proposed in the mid-1990s and has been extensively studied in the last few decades. The DE algorithm paradigm has also proven to be very powerful.

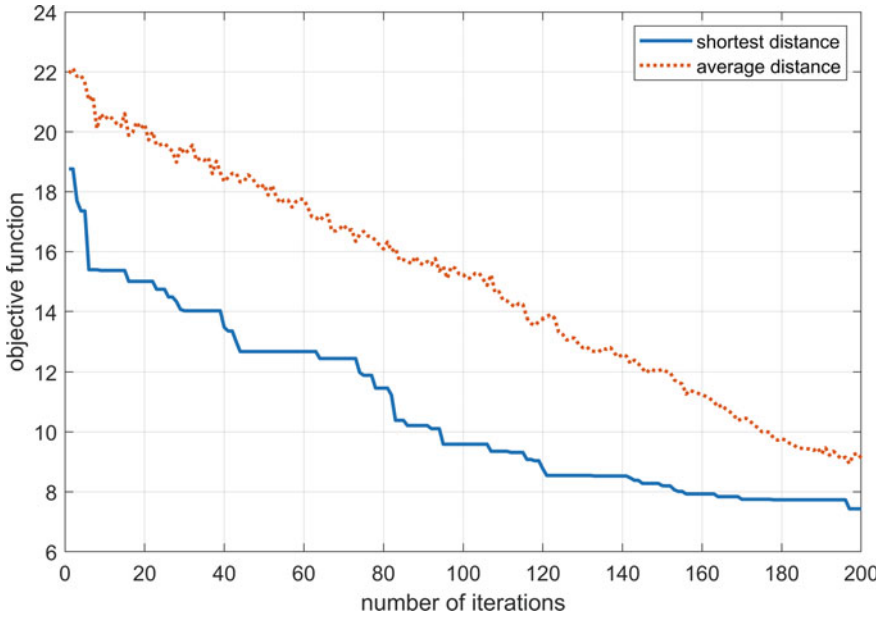

Fig. 5.13 Convergence curve of the PSO algorithm for solving the TSP

Next, let us describe the DE algorithm in detail. In general, the DE algorithm consists of four steps: initialization operation, mutation operation, crossover operation and survivor selection operation, as shown in Fig. 5.14.

The initialization operation is to randomly generate a population of *Np* individuals, where *Np* is a user-predefined population size. Each component of an individual is created randomly between the lower and upper bounds of the search space. We initialize the overall population and denote as $G_1 = \{\mathbf{x}_{1,1}, \mathbf{x}_{2,1}, .., \mathbf{x}_{Np,1}\}$, where the first subscript i denotes the i-th individual in the population, while the second subscript 1 denotes the population of the first generation. Mathematically, a single $\mathbf{x}_{i,1}$ is created in the following way:

$$x_{j,i,1} = x_{j,min} + \left(x_{j,max} - x_{j,min}\right) \times rand(0, 1),\ j = 1, 2, \ldots, D \tag{5.13}$$

where $rand(0, 1)$ generates a random number that is uniformly distributed between 0 and 1.

After initialization operation, mutation, crossover and survivor selection operations are performed sequentially until the algorithm terminates. The current generation is denoted as G_g. In the mutation operation, a population consisting of *Np* mutation vectors, denoted as $V = \{\mathbf{v}_{1,g}, \mathbf{v}_{2,g}, \ldots, \mathbf{v}_{Np,g}\}$, will be created. To generate the vector $\mathbf{v}_{i,g}$ after the mutation operation, three mutually distinct individuals are selected from G_g. Assume that the selected individuals are denoted as $\mathbf{x}_{r1,g}$, $\mathbf{x}_{r2,g}$ and $\mathbf{x}_{r3,g}$, where $r1 \neq r2 \neq r3 \neq i$. Mathematically, $\mathbf{v}_{i,g}$ is generated as follows:

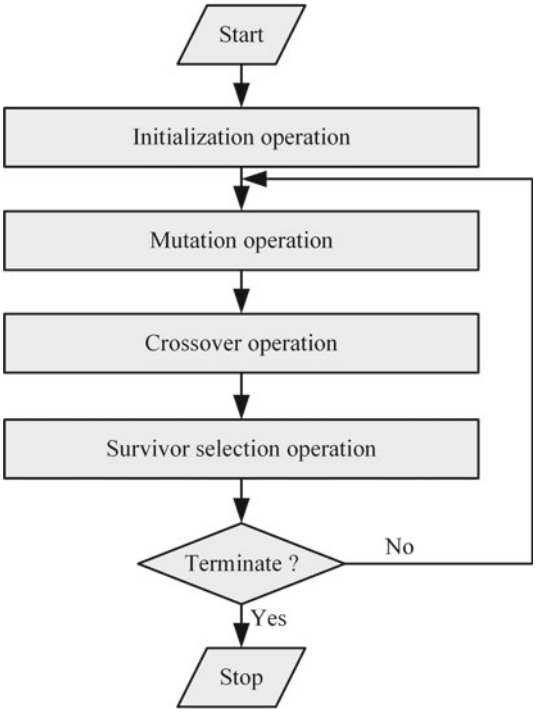

Fig. 5.14 Flow chart of the DE algorithm

$$v_{j,i,g} = x_{j,r1,g} + F \times \left(x_{j,r2,g} - x_{j,r3,g}\right),\ j = 1, 2, \ldots, D \tag{5.14}$$

or expressed in vector form as:

$$\mathbf{v}_{i,g} = \mathbf{x}_{r1,g} + F \times \left(\mathbf{x}_{r2,g} - \mathbf{x}_{r3,g}\right) \tag{5.15}$$

where F is called the scale factor and it is a user-predefined parameter. $\mathbf{x}_{i,g}$, $\mathbf{x}_{r1,g}$ and $\left(\mathbf{x}_{r2,g} - \mathbf{x}_{r3,g}\right)$ are called the target vector, the base vector, and the differential vector, respectively. A common way to reset an out-of-bounds component $v_{\mathrm{j,i,g}}$ is to reinitialize it randomly. This ensures that each individual is a feasible solution.

In the crossover operation, a population of Np trial vectors is created, denoted as U $= \{\mathbf{u}_{1,g}, \mathbf{u}_{2,g}, \ldots, \mathbf{u}_{Np,g}\}$. To generate the vector $\mathbf{u}_{i,g}$ after the crossover operation, the crossover probability Cr must be defined by the user. each component of $\mathbf{u}_{i,g}$ is taken from the i-th variation vector $\mathbf{v}_{i,g}$ with probability Cr or from the i-th individual $\mathbf{x}_{i,g}$ with probability 1-Cr. Then, to ensure that at least one component of $\mathbf{u}_{i,g}$ is inherited from $\mathbf{v}_{i,g}$, a component indicator k ($1 \leq k \leq D$) is chosen at random and the component k chosen in $\mathbf{u}_{i,g}$ is set equal to the component of $\mathbf{v}_{i,g}$ at the same position. Mathematically, $\mathbf{u}_{i,g}$ is generated by:

$$u_{j,i,g} = \begin{cases} v_{j,i,g} \text{ if } rand(0,1) \leq Cr \text{ or } j = k \\ x_{j,i,g} \text{ otherwise} \end{cases} \tag{5.16}$$

After generating all the trial vectors, their function values are calculated and then the survivor selection operation is performed.

In the survivor selection operation, a new population containing Np individuals will be generated, denoted as $G_{g+1} = \{\mathbf{x}_{1,g+1}, \mathbf{x}_{2,g+1}, \ldots, \mathbf{x}_{Np,g+1}\}$. For generating $\mathbf{x}_{i,g+1}$, greedy selection is performed between $\mathbf{u}_{i,g}$ and $\mathbf{x}_{i,g}$. Mathematically, the selection is given by:

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g}, \text{ if } f\left(\mathbf{u}_{i,g}\right) \leq f\left(\mathbf{x}_{i,g}\right) \\ \mathbf{x}_{i,g}, \text{ otherwise} \end{cases} \tag{5.17}$$

Obviously, there is Np parallel competition between the parent individual and the trial vector in this operation.

After the population G_{g+1} is produced, the mutation, crossover and survivor selection operations described above are repeated until some termination condition is reached. In the DE algorithm, a cycle of mutation, crossover and survivor selection operations is called a generation. Researchers have implemented the classical DE algorithm using several programming languages, and the relevant source code can be downloaded for free from the website of Storn.

Next, we present the work related to the DE algorithm. Since 2000, many variants of the DE algorithm have been proposed by researchers in order to obtain better performance on optimization problems. Several variants of the DE algorithm have been studied analytically by Storn and Price.

The method presented earlier in this section is known as the classical DE algorithm, or the common DE algorithm, because it is the first published version and the most commonly used version. Later, researchers proposed many different variants to improve the performance of this classical DE algorithm, and these variants can be called DE algorithm families.

The DE algorithm families can be written in the form of DE/x/y/z. The "x" represents the string that denotes the base vector to be mutated and the base vector to be changed. For example, x is "rand" for the randomly selected base vector; x is "best" means that the base vector is the individual with the best function in the current population. "y" is the number of differential vectors used in the variation operator. "z" indicates the method of crossover operation. For example, if a binomial crossover operator is used, "z" is "bin"; if an exponential crossover is used, it is "exp"; if an either/or crossover is used, it is "either/or". Using this expression, the classical DE algorithm can be expressed as: DE/rand/1/bin. In the DE algorithm family of Storn and Price, the mutation strategy variants differ in how they generate new solutions, as follows:

"DE/rand/1" stands for the following equation:

$$\mathbf{v}_{i,g} = \mathbf{x}_{r1,g} + F \times \left(\mathbf{x}_{r2,g} - \mathbf{x}_{r3,g}\right) \tag{5.18}$$

"DE/best/1" stands for the following equation:

$$\mathbf{v}_{i,g} = \mathbf{x}_{best,g} + F \times \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right) \tag{5.19}$$

"DE/target-to-best/1" stands for the following equation:

$$\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F \times \left(\mathbf{x}_{best,g} - \mathbf{x}_{i,g}\right) + F \times \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right) \tag{5.20}$$

"DE/best/2" stands for the following equation:

$$\mathbf{v}_{i,g} = \mathbf{x}_{best,g} + F \times \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right) + F \times \left(\mathbf{x}_{r3,g} - \mathbf{x}_{r4,g}\right) \tag{5.21}$$

"DE/rand/2" stands for the following equation:

$$\mathbf{v}_{i,g} = \mathbf{x}_{r1,g} + F \times \left(\mathbf{x}_{r2,g} - \mathbf{x}_{r3,g}\right) + F \times \left(\mathbf{x}_{r4,g} - \mathbf{x}_{r5,g}\right) \tag{5.22}$$

Note that some scholars also refer to DE/target-to-best/1 as DE/current-to-best/1. For consistency, this section uses the name DE/target-to-best/1 used by its original author. exponential crossover and either/or crossover are less popular than binomial crossover. In this section, only binomial crossover is considered. It is found that DE/rand/1/bin is more robust in solving multimodal functions, but converges slower; DE/best/1/bin, DE/target-to-best/1/bin and DE/rand/2/bin converge faster, but are less reliable for highly multimodal functions. From our experimental results, DE/rand/2/bin is similar to DE/rand/1/bin, but slightly slower.

In this section, we provide an overview of the DE algorithm. At the beginning, DE was found to be effective for solving global optimization problems. The effectiveness and efficiency of the DE paradigm is due to the use of the differential vector, which is the difference between two decision vectors that can be explored in the search space after the initialization phase and used in the subsequent evolution phase to obtain good solutions. Another key feature of DE is the parallel search.

In addition to creating variation strategies for various DE algorithms, the researchers noticed that DE is sensitive to its parameters F and Cr, and therefore introduced techniques to control these parameters. Subsequently, researchers have also tried to control the variation of Np, studied distributed or parallel DE algorithms, and implemented collections of parameters, mutation strategies, and crossover operators. Meanwhile, many practitioners tried to implement DE algorithms on finite resource devices, discrete optimization, and multi-objective optimization problems.

Based on these observations, we observe that the paradigm of DE has been comprehensively studied since 2000. Topics worthy of further research in the future include large-scale optimization, practical applications, and convergence analysis. Note that large-scale optimization is also a hot topic in the field of EC, as it is observed that algorithms that perform well in low-dimensional problems (e.g., $D < 50$) may not scale well to high-dimensional problems (e.g., $D > 100$).

In the mutation operation of DE, the random mutation is caused by the differential vector of two different individuals chosen at random. Apparently, the differential

vector may be the direction of descent leading to the improvement of fitness. The differential vector plays an important role in the evolutionary process. However, the number of possible differential vectors is large; therefore, the probability of choosing the direction of descent is low. On the other hand, it is observed that the optimal fitness found so far by the DE algorithm may not be improved every generation. This observation inspired us to create a mutation operator capable of finding good directions to guide the search process of DE. A good direction is a vector that, when applied to a basis vector, can lead to an improvement in fitness. Readers can discover mutation methods to improve the performance of the DE algorithm from this direction.

The classical DE algorithm can solve continuous optimization problems. We can use the DE algorithm to solve the sphere function problem in the previous section.

Example 5.5 The sphere function is a single-peaked function. There are only two independent variables in the problem. Please use the DE algorithm to solve the problem and draw a graph to analyze the results.

Solution In this example, we use the same problem setting as in Example 5.3. The specific programs of the DE algorithm are as follows:

```
rng(1);
fhd = @functionSphere;
D = 2;
xlb = [− 100, − 100];
xub = [100, 100];
MFE = 3000;
ps = 50;
F = 0.5;
CR = 0.5;
me = round(MFE/ps);
if length(xlb) == 1
    xlb = repmat(xlb,1,D);
    xub = repmat(xub,1,D);
end
xlb = repmat(xlb,ps,1);
xub = repmat(xub,ps,1);
popu = xlb+(xub-xlb).*rand(ps,D);
fpopu = feval(fhd,popu);
fitcount = ps;
[gbestval,gbestid] = min(fpopu);
gbest = popu(gbestid,:);
cvgef = gbestval;
for i = 2:me
      popuNew = popu;
      fpopuNew = fpopu;
      for ips = 1:ps
```

```
            rips = randperm(ps − 1);
            rips = rips(1:3);
            tmp1 = rips> = ips;
            rips = rips+tmp1;
            vips = popu(rips(1),:) + F*(popu(rips(2),:)-popu(rips(3),:));
            tmp1 = xlb(ips,:);
            tmp2 = xub(ips,:);
            vips = ((vips >= tmp1)&(vips <= tmp2)).*vips...
                + (vips<tmp1).*(tmp1+0.25.*(tmp2-tmp1).*rand(1,D))...
                + (vips>tmp2).*(tmp2-0.25.*(tmp2-tmp1).*rand(1,D));
            uips = zeros(1,D);
            for jcol = 1:D
                if rand <= CR
                    uips(1,jcol) = vips(jcol);
                else
                    uips(1,jcol) = popu(ips,jcol);
                end
            end
            jrand = ceil(D*rand);
            uips(1,jrand) = vips(jrand);
            fuips = feval(fhd,uips);
            fitcount = fitcount+1;
            if fuips <= fpopu(ips)
                popuNew(ips,:) = uips;
                fpopuNew(ips) = fuips;
            end
        end
        popu = popuNew;
        fpopu = fpopuNew;
        [gbestval,gbestid] = min(fpopu);
        gbest = popu(gbestid,:);
        cvgef = [cvgef,gbestval];
    end
    figure1 = figure1(1);
    axes1 = axes('Parent',figure1);
    box(axes1,'on');
    grid(axes1,'on');
    plot(ps*(1:me), cvgef, 'LineWidth',2);
    set(axes1,'FontSize',14,'YMinorTick','on','YScale','log');
    set(axes1,'FontSize',14);
    xlabel('number of iterations');
    ylabel('objective function');
```

After running the above program, we see that the optimal solution output by the algorithm is approximately equal to (0, 0) and the corresponding objective function

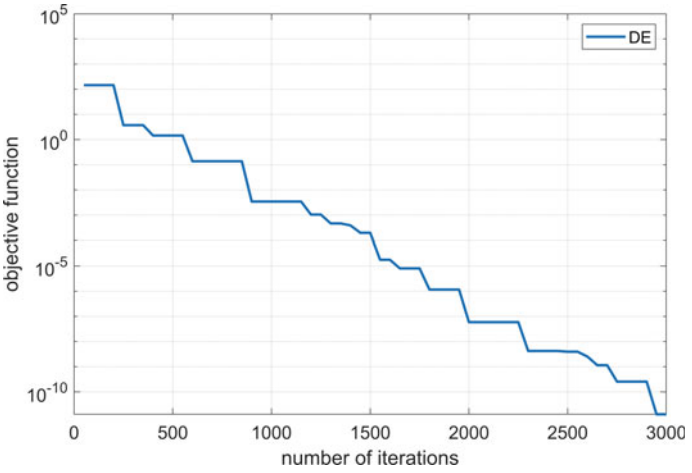

Fig. 5.15 Convergence curves of the DE algorithm on Example 5.5

value is close to 0. This is the same result obtained using the PSO algorithm. The results indicates that both the DE and the PSO algorithms are able to solve the problem.

The convergence curve of the DE algorithm for solving the Example 5.5 is shown in Fig. 5.15. Compared with the PSO algorithm, the convergence curve of the DE algorithm is closer to the lower left corner, which indicates that the DE algorithm converges faster than the PSO algorithm.

Although the DE algorithm is suitable for solving continuous optimization problems, with simple modifications, it can also be used for discrete optimization problems. We use the DE algorithm to solve the TSP in the previous section.

Example 5.6 Suppose there are $n = 40$ cities within the map boundary, the traveler needs to traverse all cities once and only once and return to the initial city location. Write a program for the DE algorithm to solve the TSP problem, and draw a graph to analyze the results.

Solution This example has been used earlier, so the background of the problem will not be introduced. The specific programs of the DE algorithm are as follows:

```
load('usborder.mat','x','y','xx','yy');
cities = 40;
locations = zeros(cities,2);
rng(1);
n = 1;
while (n <= cities)
    xp = rand*1.5;
    yp = rand;
    if inpolygon(xp,yp,xx,yy)
```

```
        locations(n,1) = xp;
        locations(n,2) = yp;
        n = n + 1;
    end
end
distances = zeros(cities);
for count1 = 1:cities
    for count2 = 1:count1
        x1 = locations(count1,1);
        y1 = locations(count1,2);
        x2 = locations(count2,1);
        y2 = locations(count2,2);
        distances(count1,count2) = sqrt((x1 - x2)^2 + (y1 - y2)^2);
        distances(count2,count1) = distances(count1,count2);
    end
end
IterNum = 200;
iter = 1;
popsize = 50;
F = 0.5;
Cr = 0.1;
popu = zeros(popsize,cities);
fpopu = zeros(popsize,1);
gbest = zeros(IterNum,cities);
gbest_fit = zeros(popsize,1);
avgl = zeros(IterNum,1);
rng('shuffle');
for i = 1:popsize
    popu(i,:) = randperm(cities);
end
popurt = popu;
for i = 1:popsize
    for j = 1:cities-1
        fpopu(i) = fpopu(i) + distances(popurt(i,j),popurt(i,j + 1));
    end
    fpopu(i) = fpopu(i) + distances(popurt(i,end),popurt(i,1));
end
[gbest_fit(1),min_index] = min(fpopu);
gbest(1,:) = popurt(min_index);
avgl(1) = mean(fpopu);
while iter <IterNum
    iter = iter + 1;
    popuNew = popu;
    popurtNew = popurt;
    fpopuNew = zeros(size(fpopu));
```

```
for i = 1:popsize
    r1 = randi([1,popsize],1,1);
    while (fpopu(r1)<fpopu(i))
        r1 = randi([1,popsize],1,1);
    end
    r2 = randi([1,popsize],1,1);
    while(r2 == r1)||(r2 == i)
        r2 = randi([1,popsize],1,1);
    end
    r3 = randi([1,popsize],1,1);
    while(r3 == i)||(r3 == r2)||(r3 == r1)
        r3 = randi([1,popsize],1,1);
    end
    jrand = randi([1,cities],1,1);
    r = rand;
    for j = 1:cities
        if (r <= Cr) || (j == jrand)
                popuNew(i,j) = popu(r1,j) + F*(popu(r2,j)-popu(r3,j));
                if popuNew(i,j)< 1
                    popuNew(i,j) = randi([1,cities],1,1);
                end
                if popuNew(i,j)> cities
                    popuNew(i,j) = randi([1,cities],1,1);
            end
        else
            popuNew(i,j) = popu(i,j);
        end
    end
    [tmp1,idx1] = sort(popuNew(i,:));
    popurtNew(i,:) = idx1;
    for j = 1:cities-1
       fpopuNew(i) = fpopuNew(i) + distances(popurtNew(i,j),popurtNew(i,j
+ 1));
    end
 fpopuNew(i) = fpopuNew(i) + distances(popurtNew(i,end),popurtNew(i,1));
        if fpopuNew(i)<fpopu(i)
        popu(i,:) = popuNew(i,:);
        popurt(i,:) = popurtNew(i,:);
        fpopu(i) = fpopuNew(i);
    end
end
[minvalue,min_index] = min(fpopu);
if minvalue <gbest_fit(iter − 1)
    gbest_fit(iter) = minvalue;
    gbest(iter,:) = popurt(min_index,:);
```

```
        else
            gbest_fit(iter) = gbest_fit(iter-1);
            gbest(iter,:) = gbest(iter-1,:);
        end
        avgl(iter) = mean(fpopu);
    end % while
    [bestRouteLen,index] = min(gbest_fit);
    bestRoute = gbest(index,:);
    figure1 = figure(2);
    axes1 = axes('Parent',figure1);
    hold(axes1,'on');
    box(axes1,'on');
    grid(axes1,'on');
    plot(x, y, 'Color','black', 'LineWidth',2);
    plot(locations(:,1),locations(:,2),...
        'bo','LineWidth',2,'LineStyle','none');
    plot(locations(bestRoute,1), ...
        locations(bestRoute,2),'LineWidth',2);
    plot([locations(bestRoute(1),1), ...
        locations(bestRoute(end),1)], ...
        [locations(bestRoute(1),2), ...
        locations(bestRoute(end),2)],'LineWidth',2);
    hold(axes1,'off');
    set(axes1,'FontSize',14);
    figure1 = figure(3);
    axes1 = axes('Parent',figure1);
    hold(axes1,'on');
    box(axes1,'on');
    grid(axes1,'on');
    plot(gbest_fit', '-', 'LineWidth',2);
    plot(avgl', ':', 'LineWidth',2);
    legend('shortest distance','average distance');
    xlabel('number of iterations');
    ylabel('objective function');
    hold off;
```

After the above program is run, the optimal value and optimal solution of the DE algorithm are output, as shown in Fig. 5.16.

It can be seen from Fig. 5.16 that the optimal route output by the DE is not good. The route in the figure has long straight-line segments. In fact, the total distance length of the optimal route output by the DE algorithm is about 9.92, which is greater than the results given by the GA, ACO and PSO algorithms.

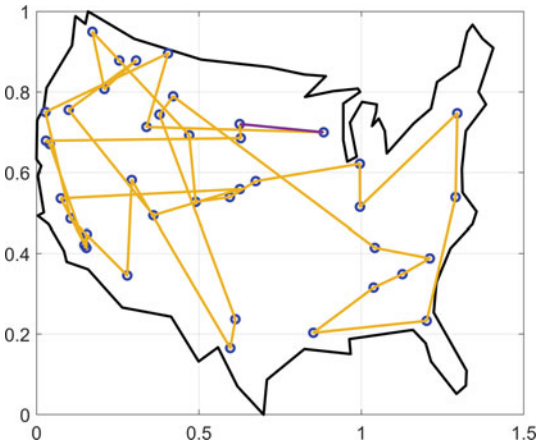

Fig. 5.16 Optimal route of the DE algorithm for Example 5.6

The convergence curve of the DE algorithm for Example 5.6 is given in Fig. 5.17. The solid line in the figure indicates the shortest route distance of the population at each generation, while the dashed line indicates the average route distance of the population at each generation. In Fig. 5.17, the number of iterations means the number of generations. It can be seen that after 200 generations, the shortest route distance gradually decreases and level off, indicating that the DE algorithm is close to convergence. However, compared with the GA, ACO and PSO algorithms, the output of the DE algorithm is not good, which to a certain extent indicates that the DE algorithm is not well suited for discrete optimization problems. The DE algorithm needs further improvement to better solve the TSP.

Exercises

(1) Readers choose a set of optimization test functions, then select no less than two evolutionary computing methods. Readers have to write programs, adjust the parameters of the evolutionary computing methods to solve the optimization test function, and compare the performance of the chosen methods in terms of the number of convergence iterations, the optimal solution output by the methods, and the stability of the methods over multiple runs.

(2) Suppose there is a traveler who wants to visit 31 provincial capitals across the country, and the traveler needs to choose the route to be taken. The coordinates of the 34 cities in the country are given in Table 5.5 as follows:

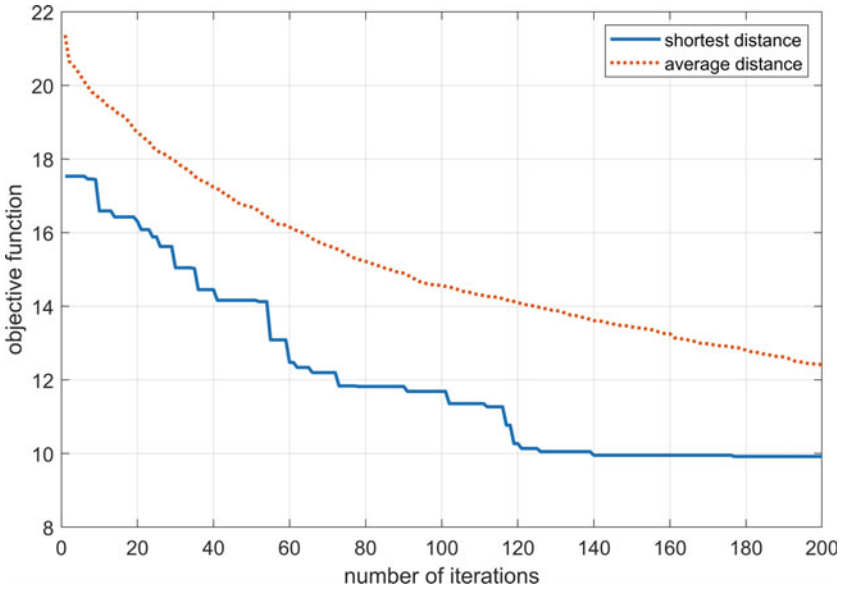

Fig. 5.17 Convergence curve of the DE algorithm for Example 5.6

Table 5.5 The coordinates of the 34 cities

Number	City	Coordinate	Number	City	Coordinate
1	Beijing	(116.4, 39.9)	18	Changsha	(113.0, 28.2)
2	Tianjin	(117.2, 39.1)	19	Guangzhou	(113.3, 23.1)
3	Shijiazhuang	(114.5, 38.0)	20	Nanning	(108.3, 22.8)
4	Taiyuan	(112.5, 37.9)	21	Haikou	(110.3, 20.0)
5	Hohhot	(111.7, 40.8)	22	Chongqing	(106.5, 29.5)
6	Shenyang	(123.4, 41.8)	23	Chengdu	(104.1, 30.7)
7	Changchun	(125.3, 43.9)	24	Guiyang	(106.7, 26.6)
8	Harbin	(126.6, 45.8)	25	Kunming	(102.7, 25.0)
9	Shanghai	(121.5, 31.2)	26	Lhasa	(91.1, 29.7)
10	Nanjing	(118.8, 32.0)	27	Xi'an	(108.9, 34.3)
11	Hangzhou	(120.2, 30.3)	28	Lanzhou	(103.8, 36.1)
12	Hefei	(117.3, 31.9)	29	Xining	(101.8, 36.6)
13	Fuzhou	(119.3, 26.1)	30	Yinchuan	(106.3, 38.5)
14	Nanchang	(115.9, 28.7)	31	Urumqi	(87.6, 43.8)
15	Jinan	(117.0, 36.7)	32	Taipei	(121.5, 25.0)
16	Zhenzhou	(113.7, 34.8)	33	Hong Kong	(114.2, 22.3)
17	Wuhan	(114.3, 30.6)	34	Macao	(113.5, 22.2)

References

1. Katoch S, Chauhan SS, Kumar V (2021) A review on genetic algorithm: past, present, and future. Multimedia Tools Appl 80:8091–8126. https://doi.org/10.1007/s11042-020-10139-6
2. Sivanandam SN, Deepa SN (2010) Introduction to genetic algorithms. Springer Berlin, Heidelberghttps://doi.org/10.1007/978-3-540-73190-0
3. Deneubourg JL, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. J Insect Behav 3:159–168
4. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66
5. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of the 1995 IEEE international conference on neural networks, Perth, WA, Australia, 27 Nov–1 Dec 1995, pp 1942–1948
6. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359

Chapter 6
Testing and Evaluation of Evolutionary Computing

Abstract Evolutionary computing is a collection of evolutionary algorithms. Different algorithms have different properties. For example, genetic algorithm is suitable for discrete optimization problems; while differential evolution algorithm is suitable for continuous optimization problems. The pros and cons of such algorithms have to be studies and tested on well-known optimization problems. This chapter presents a test set of traveling salesman problem and a test set of continuous optimization problem. The evaluation metrics are introduced to compare and analyze evolutionary algorithms. Then, this chapter introduces two recent evolutionary computing methods. They are artificial bee colony algorithm and fireworks algorithm. Finally, the state-of-the-art research progress of evolutionary computing are presented.

6.1 Test Set of Traveling Salesman Problem

The traveling salesman problem (TSP) is a typical discrete optimization problem that has been studied extensively by researchers in recent years. In the previous chapter, we gave a case study of using evolutionary computing (EC) methods to solve the TSP. In this section we will give a test set of traveling salesman problems (TSPs). The test set contains seven problems with increasing difficulty. The seven problems are well suited for testing the performance of optimization methods.

Example 6.1 The TSP is a classical combinatorial optimization problem and a NP-hard problem. It has always been a hot problem of interest in academia and industry. We collected 3000 cities from China. The latitude and longitude coordinate data of the cities are used to generate seven TSPs, as shown in Table 6.1. As can be seen from the table, the scale of the problem gradually increases from 100 to 3000 cities. The test set is called TSPCN.

Figure 6.1 gives the city distribution of the 1-st instance of the test set. In Fig. 6.1, the horizontal axis is east longitude and the vertical axis is north latitude. As can be seen from the figure, the city distribution shows a scattered nature, with some cities being far away and others being close.

X. Zhang et al., *Intelligent Information Processing with Matlab*,
https://doi.org/10.1007/978-981-99-6449-9_6

Table 6.1 List of Chinese TSPs

Number	Name	Scale
1	TSPCNProblem1	100
2	TSPCNProblem2	500
3	TSPCNProblem3	1000
4	TSPCNProblem4	1500
5	TSPCNProblem5	2000
6	TSPCNProblem6	2500
7	TSPCNProblem7	3000

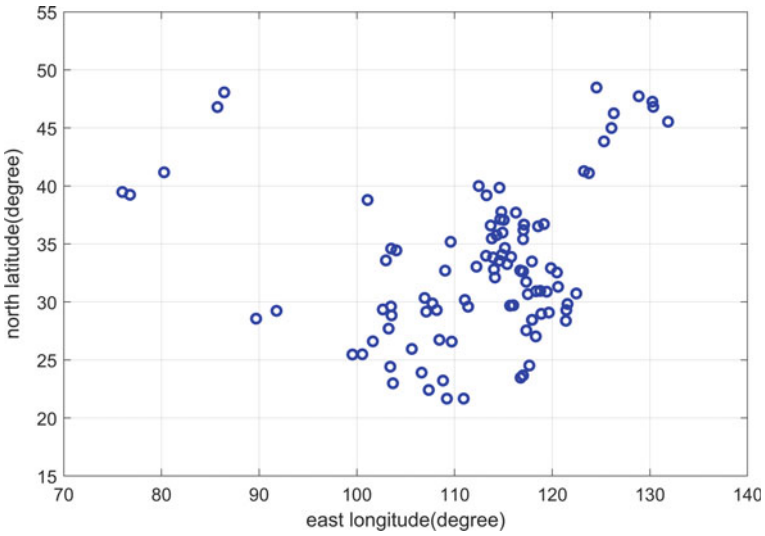

Fig. 6.1 City distribution map of TSPCNProblem1

Figure 6.2 gives the city distribution for the 7-th instance in the test set. As can be seen from the figure, the distribution of cities shows unevenness, with some cities being far away and others being close. In particular, the cities in the lower right are more densely distributed, while those in the upper left are more scattered.

Figures 6.1 and 6.2 give the city distribution maps for the 1-st and 7-th instances. We do not give the city distribution maps for the remaining instances because the distribution characteristics of the remaining cities are between these two instances. The cities of TSPCNProblem1 are a subset of cities of TSPCNProblem7; while TSPCNProblem7 contains more cities than TSPCNProblem1.

We stored the north latitude and east longitude coordinate data as "csv" files with corresponding names. We could solve the above 7 TSP instances using the EC methods. Due to the stochastic nature of the EC method, we need to independently repeat the optimization method used 31 times, saving the optimal route and the shortest route distance as "csv" files. In each file, the data are split by commas.

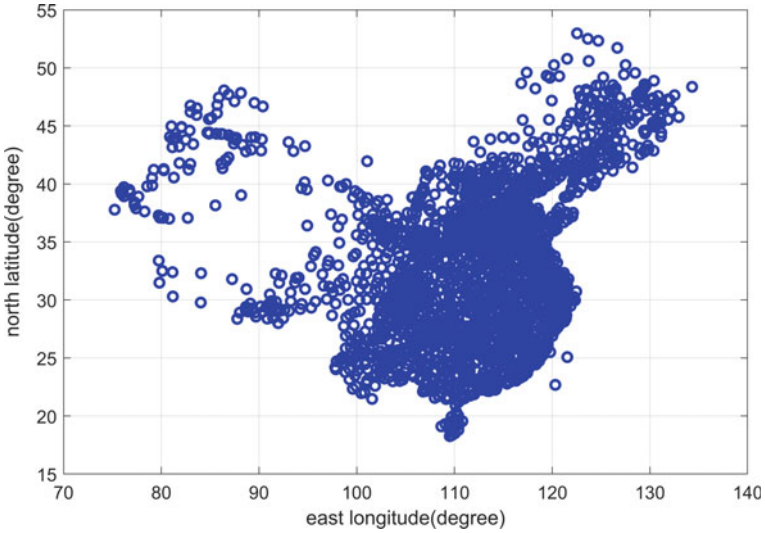

Fig. 6.2 City distribution map of TSPCNProblem7

Solution We use the GA as a baseline method and use it to solve 7 TSP instances. Specifically, each TSP instance needs to be run 31 times independently. We can get 31 traversal routes and their route lengths. The traversal route for the G to solve the 1-st instance is shown in Fig. 6.3.

As can be seen in Fig. 6.3, the traversal route found by the GA for the 1-st instance has overlapping paths. This indicates that the GA did not find the optimal solution for

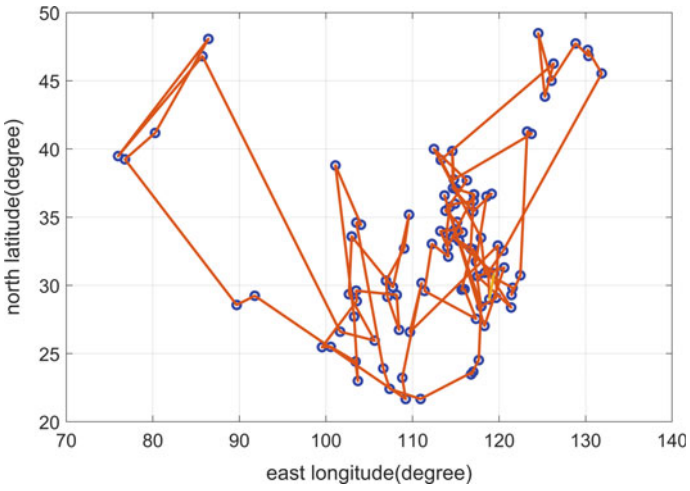

Fig. 6.3 Traversal route found by the GA for TSPCNProblem1

the instance. Interested readers can further find better traversal routes by adjusting the hyperparameters of the GA.

The results for all instances of the GA to solve the TSPCN test set are in Table 6.2. This table gives the lengths of the best route found by running the GA after 31 independent runs. In Table 6.2, columns 2 through 8 are the results from the 1-st instance to the 7-th instance. Each row from row 2 onward is the length of the best route obtained by running the GA independently.

The results of all instances of the GA solving the TSPCN test set are summarized in Table 6.3. The table counts the minimum (min), mean, maximum (max), median(med) and standard deviation(std) of the route length for 31 independent runs of the GA to solve all instances.

As seen in Table 6.3, the std of the GA gradually increases from instance 1 to instance 7, which indicates that the GA is somewhat unstable as the problem size increases. One reason is that we use the same termination condition for different instances. As the problem size increases, the maximum number of iterations can be increased. Here we only introduce the use of the GA to solve the TSPCN test set without adjusting and optimizing the hyperparameters, thus, the GA does not find the optimal solution for the instances.

6.2 Test Set of Continuous Optimization Problem

With the development of EC, researchers have created many function optimization problems in order to test the performance of EC methods. Examples include the Schwefel function used in the previous sections, the sphere function, etc. The independent variables of these functions are continuous, and the corresponding functions are continuous optimization problems. When verifying and comparing the performance of EC methods, researchers use test functions. This section describes the commonly used continuous test functions. If not specified, all test functions in this section are continuous test functions, and the objective is to minimize the problems.

In the early days of EC, there were not many test functions, and researchers only needed to test the performance of methods on a few functions. As scientific research evolved and progressed, more and more functions were tested and their mathematical expressions became more and more complex. Today, researchers usually need to test the performance of methods on a dozen or even more functions.

The international conference on evolutionary computation (CEC) is a prestigious conference in the field of EC. Since 2005, Suganthan and his colleagues have been running a standard test function competition at CEC. The competition aims to lead the development of EC methods on continuous optimization problems and develops test functions of different difficulties. As shown in Table 6.4, they held competitions for unconstrained continuous optimization test functions in 2005, 2010, 2013, 2014, 2015, 2016, 2017, 2018 and 2020, respectively. In particular, in recent years, Liang et al. have been organizing this competition at CEC every year as the main members of this competition [1].

Table 6.2 Route length found by GA on TSPCN test set

Run	Problem1	Problem2	Problem3	Problem4	Problem5	Problem6	Problem7
1	458.90	4711.89	10,986.70	17,615.64	24,292.58	30,872.35	37,870.12
2	506.78	4599.83	10,854.65	17,389.26	24,057.87	31,042.54	37,756.79
3	475.60	4500.51	10,854.67	17,679.53	24,183.69	30,792.08	37,742.41
4	489.69	4548.82	10,845.60	17,364.05	24,306.60	30,906.32	37,415.95
5	450.43	4523.72	10,662.04	17,508.73	24,241.97	31,005.62	37,709.33
6	494.82	4550.75	10,937.31	17,526.22	24,557.65	30,802.00	37,354.17
7	485.05	4571.54	10,869.33	17,533.28	24,118.64	30,848.09	37,739.57
8	488.49	4536.62	10,810.00	17,486.75	24,399.72	31,019.17	37,426.61
9	482.37	4577.21	10,611.92	17,503.16	24,422.68	30,898.99	37,656.89
10	473.74	4615.64	10,939.66	17,437.89	24,101.27	31,094.86	37,789.40
11	487.50	4525.43	10,996.00	17,447.90	24,345.42	30,702.66	37,790.37
12	522.07	4468.64	10,734.58	17,531.16	23,952.89	30,898.23	37,596.78
13	505.94	4424.91	10,863.40	17,538.20	24,278.33	30,852.89	38,020.12
14	460.03	4453.66	10,770.62	17,420.88	24,092.93	30,698.39	37,375.31
15	496.16	4415.28	10,801.63	17,365.03	24,151.60	30,589.59	37,387.39
16	493.15	4601.98	11,022.97	17,499.47	24,202.65	30,880.22	37,854.88
17	489.93	4608.33	10,874.44	17,332.55	24,386.07	30,980.10	37,806.99
18	459.96	4585.26	10,785.87	17,396.98	24,057.18	30,996.18	37,859.06
19	465.90	4597.85	10,849.19	17,488.53	24,368.66	30,817.79	37,827.11
20	457.42	4362.73	10,954.89	17,268.97	24,308.97	30,761.75	37,554.47
21	479.00	4567.75	10,899.19	17,479.26	24,033.67	30,809.49	37,818.98
22	548.00	4608.57	10,957.55	17,708.66	24,332.32	30,812.02	37,633.35
23	499.92	4611.04	10,821.21	17,591.53	24,379.46	31,051.46	37,385.24
24	531.51	4576.09	10,946.50	17,326.66	23,992.26	30,915.60	37,720.52
25	459.74	4529.45	10,951.31	17,424.42	24,283.34	30,849.92	37,440.99
26	504.12	4535.64	10,832.67	17,427.46	24,411.96	30,743.79	37,468.66
27	522.09	4453.06	10,810.52	17,307.97	24,196.57	30,742.30	37,660.40
28	474.26	4542.23	10,784.33	17,477.88	24,518.23	30,836.35	37,754.02
29	513.08	4571.82	11,058.16	17,358.74	24,226.62	30,798.61	37,554.08
30	531.99	4604.87	10,908.74	17,512.34	24,230.52	30,644.38	37,530.43
31	503.32	4482.99	10,877.93	17,596.28	24,486.42	31,005.07	37,762.21

As can be seen in Table 6.4, researchers have also organized test functions with constraints, multi-objectives and real-world application problems. These test functions form the standard for testing and evaluating EC methods.

We use the test function used in CEC in 2020 as an example, denoted as CEC2020. The continuous single-objective optimization function is modeled as:

Table 6.3 Summary of results found by GA on TSPCN test set

Problem	Min.	Mean	Max.	Med.	Std.
1	450.43	490.68	548.00	489.69	24.72
2	4362.73	4544.00	4711.89	4550.75	72.49
3	10,611.92	10,866.89	11,058.16	10,863.40	99.46
4	17,268.97	17,469.21	17,708.66	17,479.26	105.63
5	23,952.89	24,255.44	24,557.65	24,278.33	156.58
6	30,589.59	30,860.28	31,094.86	30,849.92	123.99
7	37,354.17	37,653.63	38,020.12	37,709.33	180.13

Table 6.4 Test function competitions held in CEC

Year	Problem type	Member
2005	Single-objective test functions	Suganthan, Hansen, Liang et al.
2006	Constrained single-objective test functions	Liang, Runarsson, Mezura-Montes et al.
2007	Multi-objective test functions	Huang, Qin, Deb et al.
2008	Large-scale single-objective test functions	Tang, Yao, Suganthan et al.
2009	Bound constrained single-objective test functions	Zhang, Zhou, Zhao et al.
2010	Single-objective test functions	Mallipeddi, Suganthan
2011	Single-objective application test functions	Das, Suganthan
2013	Single-objective test functions	Liang, Qu, Suganthan et al.
2014	Single-objective test functions	Liang, Qu, Suganthan
2015	Single-objective test functions	Chen, Liu, Zhang et al.
2016	Single-objective test functions	Chen, Liu, Zhang et al.
2017	Single-objective test functions	Chen, Liu, Zhang et al.
2018	Single-objective test functions	Chen, Liu, Zhang et al.
2020	Single-objective test functions	Kumar, Wu, Ali et al.

$$\begin{aligned} &\min\ f(\mathbf{x}) = f(x_1, x_2, \ldots, x_D) \\ &s.t. \qquad x_i \in [-100, 100] \end{aligned} \tag{6.1}$$

where D denotes the number of independent variables, $\mathbf{x}$ denotes the vector consisting of the independent variable x_i, and f denotes the objective function. In order to avoid the minimum to be located in the center of the feasible space, researchers use translation to transform the minimum to the center of the feasible space. For example, $\mathbf{x}_0$ is used to denote the location of the translation. According to the knowledge of geometry, coordinate transformations include translation, rotation and scaling transformations. These three transformations can be used when constructing synthetic test functions, so that an EC method can be tested for translation invariance, rotation

invariance and scaling invariance. The rotation transformation changes the shape of the objective function, while the scaling transformation allows the range of the independent variables to be mapped to the same interval, e.g., [− 100, 100]. All three transformations can be implemented by operations such as matrix product, and they are not described here.

The CEC2020 test function set includes 10 synthetic functions, as shown in Table 6.5. As can be seen from the table, only the first function is a single-peaked function, the rest are multi-peaked functions. The hybrid and composition functions are greatly increase the complexity of the problem, which brings a great challenge to EC methods.

It should be noted that the competitions in recent years use fixed test functions. For example, the test functions in CEC2020 are from the CEC competitions in 2014 and 2017. It may be that the hybrid and composition functions are too complex and the current EC methods have not yet solved these problems. Thus, the test set has not been changed and updated since 2020.

The functions in the CEC2020 test set are built on 12 base functions, as follows:

$$f_{b1}(\mathbf{x}) = x_1^2 + 10^6 \sum_{i=2}^{D} x_i^2 \tag{6.2}$$

where f_{b1} denotes the first base function. It is generally referred to as the Bent Cigar function. The second base function is:

$$f_{b2}(\mathbf{x}) = \sum_{i=1}^{D} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right) \tag{6.3}$$

Table 6.5 Test set in CEC2020

Number	Function name	Property
1	Bent Cigar function	Single-peaked
2	Schwefel's function	Multi-peaked
3	Lunacek bi-Rastrigin function	Multi-peaked
4	Expanded Rosenbrock and Griewank function	Multi-peaked
5	Hybrid function 1	Multi-peaked, non-separable
6	Hybrid function 2	Multi-peaked, non-separable
7	Hybrid function 3	Multi-peaked, non-separable
8	Composition function 1	Multi-peaked, non-separable, asymmetrical
9	Composition function 2	Multi-peaked, non-separable, asymmetrical
10	Composition function 3	Multi-peaked, non-separable, asymmetrical

The second base function is also known as the Rastrigin function. The third base function is:

$$f_{b3}(\mathbf{x}) = \sum_{i=1}^{D} \left(10^6\right)^{\frac{i-1}{D}} x_i^2 \tag{6.4}$$

The third base function is also known as the highly conditional elliptic function. The fourth base function is:

$$f_{b4}(\mathbf{x}) = \left| \left(\sum_{i=1}^{D} x_i^2 \right)^2 - \left(\sum_{i=1}^{D} x_i \right)^2 \right|^{\frac{1}{2}} + \left(0.5 \sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i \right) / D + 0.5 \tag{6.5}$$

The fourth base function is also known as the HGBat function. The fifth base function is:

$$f_{b5}(\mathbf{x}) = \sum_{i=1}^{D} \left(100 \left(x_i^2 - x_{i+1} \right)^2 + (x_i - 1)^2 \right) \tag{6.6}$$

The fifth base function is also known as the Rosenbrock function. The sixth base function is:

$$f_{b6}(\mathbf{x}) = \sum_{i=1}^{D} \left(\frac{x_i^2}{4000} \right) - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1 \tag{6.7}$$

The sixth base function is also known as the Griewank function. The seventh base function is:

$$f_{b7}(\mathbf{x}) = -20exp\left(-0.2 \sqrt{\sum_{i=1}^{D} x_i^2 / D} \right) - \exp\left(\sum_{i=1}^{D} \cos(2\pi x_i) / D \right) + 20 + e \tag{6.8}$$

The seventh base function is also known as the Ackley function, where e is the natural constant. The eighth base function is:

$$f_{b8}(\mathbf{x}) = \left| \sum_{i=1}^{D} x_i^2 - D \right|^{\frac{1}{4}} + \left(0.5 \sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i \right) / D + 0.5 \tag{6.9}$$

The eighth base function is also known as the Happycat function. The ninth base function is:

$$f_{b9}(\mathbf{x}) = 10^6 x_1^2 + \sum_{i=1}^{D} x_i^2 \tag{6.10}$$

The ninth base function is also known as the Discus function. The tenth base function is:

$$f_{b10}(\mathbf{x}) = \min\left\{\sum_{i=1}^{D}(\hat{x}_i - \mu_0)^2, dD + s\sum_{i=1}^{D}(\hat{x}_i - \mu_1)^2\right\} + 10\left(D - \sum_{i=1}^{D} cos(2\pi\hat{z}_i)\right) \tag{6.11}$$

where (6.11) contains the following parameters:

$$\mu_0 = 2.5 \tag{6.12}$$

$$\mu_1 = -\sqrt{(\mu_0^2 - 1)/s} \tag{6.13}$$

$$s = 1 - 1/\left(2\sqrt{D + 20} - 8.2\right) \tag{6.14}$$

$$\hat{x}_i = 2sgn(x_i)y_i + \mu_0 \tag{6.15}$$

$$\mathbf{y} = 10(\mathbf{x} - \mathbf{x}_0)/100 \tag{6.16}$$

$$\hat{z}_i = 2sgn(x_i)y_i \tag{6.17}$$

The tenth base function is also known as the Lunacek bi-Rastrigin function. The eleventh base function is:

$$f_{b11}(\mathbf{x}) = 418.9829D - \sum_{i=1}^{D} g(z_i) \tag{6.18}$$

Among them:

$$z_i = x_i + 420.968746 \tag{6.19}$$

$$g(z_i) = \begin{cases} z_i sin\left(|z_i|^{\frac{1}{2}}\right) & |z_i \le 500| \\ (500 - mod(z_i, 500))sin\left(\sqrt{|500 - mod(z_i, 500)|}\right) - \frac{(z_i - 500)^2}{10000D} & z_i > 500 \\ (mod(|z_i|, 500) - 500)sin\left(\sqrt{|500 - mod(|z_i|, 500)|}\right) - \frac{(z_i - 500)^2}{10000D} & z_i < -500 \end{cases} \tag{6.20}$$

The eleventh base function is also known as the modified Schwefel function. The twelfth base function is:

$$f_{b12}(\mathbf{x}) = g(x_1, x_2) + g(x_2, x_3) + \ldots + g(x_{D-1}, x_D) + g(x_D, x_1) \tag{6.21}$$

Among them:

$$g(x_i, x_j) = 0.5 + \left(sin^2\left(\sqrt{x^2 + y^2}\right) - 0.5\right) / \left(\left(1 + 0.001\left(x^2 + y^2\right)\right)^2\right) \tag{6.22}$$

The twelfth base function is also known as the extended Schaffer function.

The functions in the CEC2020 test set are built on the twelve base functions mentioned above. For example, the first function in the CEC2020 test set is:

$$f_1(\mathbf{x}) = f_{b1}(M(\mathbf{x} - \mathbf{x}_0)) + f_0 \tag{6.23}$$

where M is a matrix, whose role is to perform a rotation transformation on the independent variable. $\times 0$ is a vector, whose role is to perform a translation transformation on the independent variable. f_0 is a real number whose role is to perform a translation transformation on the objective function value. It can be seen that the first function, although single-peaked, makes the problem considerably more difficult after the translational and rotation transformations.

The other functions in the CEC2020 test set are not introduced here. Interested readers should consult the relevant references.

6.3 Evaluation of Continuous Optimization Problems

To verify the performance of an EC method, it needs to be tested on some test functions, such as the CEC2020 test set. Multiple methods have to be tested in a fair environment in order to be able to compare method performance and draw meaningful conclusions. In this section, we continue to use the CEC2020 test set as an example to introduce the testing and evaluation EC methods.

In the CEC2020 test set, all 10 problems can set the number of independent variables, i.e., the value of D. It is common to set $D = 10$ or $D = 20$. Moreover, the seventh function can also set $D = 15$; except the seventh function, other functions can also set $D = 5$ or $D = 15$. it can be seen that this test set can only support the case of a small number of independent variables. For a larger number of independent variables, it is necessary to use other test sets. In this test set, the range of each independent variable is the same. The range of independent variables is set to $[-100, 100]$.

The EC methods introduces randomness in the process of evolutionary iteration, so that the same method may obtain different results when run independently. In other words, a method may converge to different solutions in different runs. Then, for a test function or problem, the EC methods need to be run independently several times to obtain the average performance of the method in solving the problem. The recommended number of independent runs for the CEC2020 test set is 30.

The termination condition is set to a fixed number of function valuations, i.e., the maximum number of functions that can be valued, denoted as *MFE*. For problems with a small number of independent variables, the *MFE* can be set to a small value; while for problems with a large number of independent variables, the *MFE* can be set to a large value. the recommended setting for the CEC2020 test set is $MFE = 50{,}000D$. For example, for $D = 10$ or $D = 20$, the *MFE* can be set to 1,000,000 or 10,000,000, respectively.

The optimal solutions of the functions in the CEC2020 test set are all within the range of values of the independent variables. If the optimal solution is noted as $\mathbf{x}^*$ and its corresponding optimal value is $f* = (\mathbf{x}^*)$, then $\mathbf{x}^* \in [-100, 100]^D$. The optimal values of the functions in the test set are known, which makes it convenient to check whether the problem is solved. The CEC2020 test set recommends two conditions for the algorithm to terminate. One is to consume the given number of evaluations, i.e., to reach the *MFE*; the other is that the error from the optimal value is less than 10^{-8}, i.e., $|f - f^*| < 10^{-8}$. It should be noted that we consider the method to have found the optimal solution for this test function when the error from the optimal solution is less than 10^{-8}, i.e., the error is considered to be 0 at this point.

In addition, researchers found that the initial population has an impact on the final results of the EC methods. To eliminate this effect from initialization, the CEC2020 test set recommends the use of a uniformly distributed random initialization method.

In the optimization search process, we have to record the convergence process of the tested method. For example, at some appropriate number of iterations, the value of the current optimal function is recorded, and the expression is:

$$t = \left\lfloor D^{\frac{k-15}{5}} MFE \right\rfloor, \ k = 0, 1, 2, \ldots, 15 \tag{6.24}$$

where t denotes the current number of iterations, $\lfloor \cdot \rfloor$ indicating rounding down. It can be seen that in the process of finding the best, we have to record 16 function values. And the last one, i.e., at $k = 15$, we record the best solution found by the algorithm. This solution will be used to reflect the performance of the algorithm. It should be noted that the optimal value of each function in the CEC2020 test set is known, so we use the error between the best function value found by the EC method and the optimal value, i.e., f- f^*, in presenting the final results.

Example 6.2 Suppose we use the PSO algorithm to solve for the functions in the CEC2020 test set with $D = 10$, 15 and 20. Please analyze the effect of the PSO algorithm.

Solution The CEC2020 test set has been introduced in Sect. 6.2. The PSO algorithm has been introduced in Sect. 5.5. In this example, we use the PSO algorithm on the CEC2020 test set, and the results are given in Table 6.6.

Table 6.6 gives the error between the best solution found by the PSO algorithm and the optimal value of the corresponding function. In Table 6.6, the second column shows the minimum error (min) for each function; the third column shows the

Table 6.6 Results of the PSO algorithm on the CEC2020 test set with $D = 10$

f	Min.	Max.	Med.	Mean.	Std.
f_1	3.91	9682.58	3191.66	3853.33	3150.56
f_2	6.89	373.26	146.58	132.15	89.17
f_3	2.61	20.56	14.40	13.45	4.35
f_4	0.48	1.67	0.87	0.89	0.25
f_5	70.36	4181.66	1124.23	1437.18	1179.66
f_6	0.04	135.20	0.71	10.64	31.65
f_7	0.00	142.95	16.78	32.48	47.40
f_8	0.00	104.16	101.92	95.64	24.08
f_9	100.00	345.19	338.28	330.47	42.95
f_{10}	398.08	446.18	398.51	409.40	19.87

maximum error (max) for each function; the fourth column shows the median error (med) for each function; the fifth column shows the mean error for each function; and the sixth column shows the standard deviation (std) of the error for each function. From these results, it is clear that the PSO algorithm is able to find near-optimal solutions when solving problems f_4, f_6, f_7 and f_8. However, the PSO algorithm does not find the optimal solution in every run. The standard deviation of the method is particularly large when solving problems f_1 and f_5, which indicates that the method is not suitable for solving similar problems or that the performance of the algorithm is not stable on such problems.

Similarly, Tables 6.7 and 6.8 give the cases where the PSO algorithm solves the CEC2020 test set for $D = 15$ and $D = 20$, respectively. As can be seen from the tables, the PSO algorithm in dimension $D = 20$ for solving function f_1 finds a better value than $D = 15$; while for function f_5 and function f_7, the PSO algorithm finds a better value than $D = 20$ in dimension $D = 15$. It can be seen that a consistent law of variation cannot be derived from the results of these two dimensions. A specific analysis of the same function with different D values is required.

According to the time slots defined in (6.24), we can record the 16 error values of the PSO algorithm. After 30 independent runs, we can calculate the median of the 16 error values from the 30 times, which can reflect the convergence process of the PSO algorithm, as shown in Fig. 6.4.

As can be seen in Fig. 6.4, from the initial to 2000 function valuations, the PSO algorithm consistently finds a better solution at each iteration, so the error decreases very quickly. After this, the convergence curve tends to flatten out. As seen in Table 6.6, the PSO algorithm does not find a global optimal solution for f_1. This indicates that the algorithm cannot find a better solution and is likely to be trapped in a local optimal solution. Although we set the number of function valuations to 1,000,000, the number of valuations for the PSO algorithm to converge to a locally optimal solution is actually much smaller than the number of termination conditions.

Table 6.7 Results of the PSO on the CEC2020 test set with $D = 15$

f	Min.	Max.	Med.	Mean.	Std.
f_1	165.19	20,519.06	3798.11	5926.09	5587.57
f_2	7.00	495.26	186.90	228.08	112.13
f_3	1.56	25.61	18.96	15.67	7.83
f_4	0.45	1.65	1.00	1.04	0.28
f_5	59.78	641.68	252.08	252.51	127.86
f_6	0.43	266.50	28.11	51.53	62.48
f_7	4.47	384.25	126.45	160.65	98.37
f_8	100.00	617.62	100.97	130.91	103.64
f_9	337.61	399.24	393.10	391.34	10.46
f_{10}	400.00	400.00	400.00	400.00	0.00

Table 6.8 Results of the PSO algorithm on the CEC2020 test set with $D = 20$

f	Min.	Max.	Med.	Mean.	Std.
f_1	0.23	788.90	99.36	185.49	214.92
f_2	18.41	744.52	207.02	257.77	177.68
f_3	5.25	33.44	24.14	22.04	8.55
f_4	0.68	3.33	1.42	1.57	0.52
f_5	414.93	89,974.94	10,494.36	15,520.28	18,114.02
f_6	0.47	128.15	4.32	16.11	36.79
f_7	102.63	3742.68	839.09	1256.37	1082.77
f_8	52.77	1169.85	101.27	272.52	355.79
f_9	100.00	457.38	425.89	416.58	59.58
f_{10}	400.64	459.80	416.06	417.11	9.41

Thus, we could terminate the algorithm after a small number of function evaluations when solve real-world application problems.

The CEC2020 test set also gives a way to calculate the complexity of the algorithm. This method is performed only on the function f_7. Since this function f_7 only supports $D = 10$, 15 and 20, the computational complexity of the PSO algorithm is shown in Table 6.9.

In Table 6.9, the second column shows the running time T_0, which is the time consumed to run the following programs:

```
t0 = tic;
x = 0.55;
for i = 1:1000000
    x = x + x;
    x = x / 2;
    x = x * x;
```

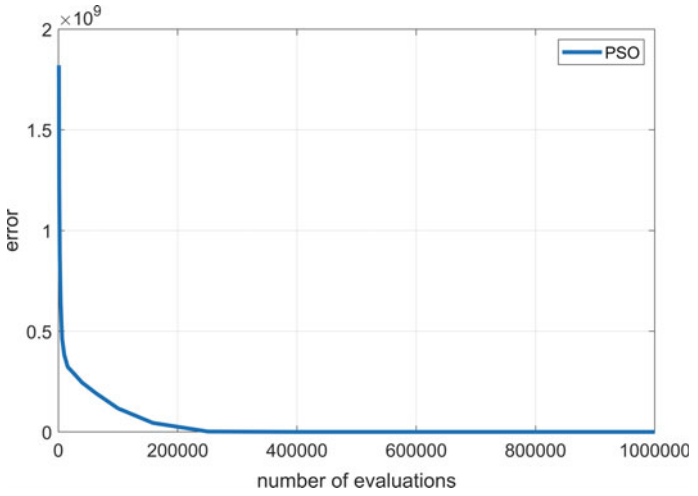

Fig. 6.4 Convergence curve of the PSO algorithm on f_1 with $D = 20$

Table 6.9 Complexity of the PSO algorithm

D	T_0	T_1	T_2	Complexity
10	0.0154	0.1740	0.2884	7.4286
15	0.0152	0.2421	0.3979	10.2780
20	0.0168	0.3180	0.5040	11.0433

```
    x = sqrt(x);
    x = log(x);
    x = exp(x);
    x = x / (x + 2);
end
T0 = toc(t0);
```

It can be seen that T_0 is the calculation of some basic elementary function. The running time T_1 in the third column of Table 6.9 is the time spent running the following programs:

```
t0 = tic;
D = 20;
rng('shuffle');
xlb = - 100;
xub = 100;
fhd = str2func('cec20_func');
fid = 7;
x = xlb + rand(200,000,D)*(xub-xlb);
fx = feval(fhd, x', fid);
T1 = toc(t0);
```

The third to last row and second to last row of the above program are the evaluation of the function f_7. They are programmed in a matrix fashion; i.e., a number of candidate solutions are first created to form a matrix. Each row of the matrix is a candidate solution of the problem. All the candidate solutions are evaluated at once. In addition, it is also possible if the EC method uses a vector or single variable programming approach. It is important to note that the computer runtime is faster using the matrix programming approach. Thus, readers may obtain different complexity by using different programming approach.

The forth column is the running time T_2. The associated time is computed by the following programs:

```
T2 = zeros(1,5);
for jrun = 1:length(T2)
    t0 = tic;
    FuncOpt = [100,1100,700,1900,1700,1600,2100,2200,2400,2500];
    VTR = 1e-8;
    D = 20;
    MFE = 200,000;
    rng('shuffle');
    xlb = −100;
    xub = 100;
    popsize = 100;
    fhd = str2func('cec20_func');
    fid = 7;
    [tmp1,tmp2,tmp3,tmp4] = PSO(fhd,D,popsize,MFE,xlb,xub,FuncOpt(fid),
VTR,fid);
    T2(jrun) = toc(t0);
end
T2 = mean(T2);
```

The above program is the PSO algorithm executed 5 times independently to solve the function f_7. Then the results after 5 times are averaged to obtain T_2.

The last column in Table 6.9 shows the computational complexity of the PSO method, which is computed by:

$$c^{PSO} = \frac{(T_2 - T_1)}{T_0} \tag{6.25}$$

where c^{PSO} denotes the computational complexity of the PSO algorithm. If other EC methods are used, it is only necessary to change the PSO algorithm to other methods. Thus, we can compute complexity of any EC method based on (6.25).

In addition to the above computational complexity calculation methods, the complexity of an EC method can be analyzed theoretically. Taking the PSO algorithm as an example. Suppose that the complexity of the random number generator is not considered, and only the addition, subtraction and multiplication computations of the PSO algorithm are considered. The population size of the PSO algorithm is

Np, the dimension is *D*, and the number of generations of evolution is *Ng*. From Eq. (5.8), it can be seen that 4*D* addition, subtraction and 4*D* multiplication operations are required for the velocity update of each particle. The position update for each particle requires *D* addition operations from Eq. (5.9). Therefore, the number of arithmetic operations required for each particle update is about 9*D*. In each generation, the PSO algorithm needs to update the position for each particle. At the end of each generation, the PSO algorithm requires about 9*DNp* arithmetic operations. And after all evolved generations, the PSO algorithm requires about 9*DNpNg* arithmetic operations. Therefore, the computational complexity of the PSO algorithm is about O(*DNpNg*) in theory.

Readers can do their own theoretical analysis of the computational complexity of other EC methods such as GA and ACO. Most EC methods have a theoretical computational complexity of at least O(*DNpNg*). Clearly, this conclusion may be controversial. At least it is a scheme to analyze the complexity of EC methods.

Example 6.3 Suppose we use the DE algorithm to solve for the functions in the CEC2020 test set with $D = 10$, 15 and 20. Please analyze the effect of the DE algorithm.

Solution The CEC2020 test set has been introduced in Sect. 6.2. The DE algorithm has been introduced in Sect. 5.6. The classical DE algorithm can solve continuous optimization problems. In this example, we use the DE algorithm on the CEC2020 test set, and the results are given in Tables 6.10, 6.11 and 6.12.

Table 6.10 shows the results on CEC2020 test set with $D = 10$. Table 6.11 shows the results on CEC2020 test set with $D = 15$. Table 6.12 shows the results on CEC2020 test set with $D = 20$. As can be seen from the tables, the DE algorithm is able to find the optimal solution for the function f_1 in different dimensional cases. In addition, although the optimal solution of function f_7 cannot be found, the DE algorithm performs better than the PSO algorithm on this function.

Table 6.10 Results of the DE algorithm on the CEC2020 test set with $D = 10$

f	Min.	Max.	Med.	Mean.	Std.
f_1	0.00	0.00	0.00	0.00	0.00
f_2	76.20	268.16	196.98	192.15	52.98
f_3	14.19	19.39	17.37	17.15	1.32
f_4	0.87	1.51	1.22	1.22	0.15
f_5	0.11	3.45	1.45	1.54	0.94
f_6	0.02	0.31	0.05	0.09	0.09
f_7	0.00	0.01	0.00	0.00	0.00
f_8	100.00	100.00	100.00	100.00	0.00
f_9	100.00	340.62	336.14	312.69	70.92
f_{10}	397.74	445.80	398.15	416.34	22.41

Table 6.11 Results of the DE algorithm on the CEC2020 test set with $D = 15$

f	Min.	Max.	Med.	Mean.	Std.
f_1	0.00	0.00	0.00	0.00	0.00
f_2	157.24	609.03	375.16	363.04	118.15
f_3	20.93	30.06	27.01	26.84	1.86
f_4	1.99	2.81	2.47	2.45	0.21
f_5	50.26	107.77	75.69	75.11	12.46
f_6	0.31	7.41	0.68	0.84	1.23
f_7	0.35	1.59	0.66	0.69	0.25
f_8	100.00	100.00	100.00	100.00	0.00
f_9	389.68	390.40	389.68	389.72	0.14
f_{10}	400.00	400.00	400.00	400.00	0.00

Table 6.12 Results of the DE algorithm on the CEC2020 test set with $D = 20$

f	Min.	Max.	Med.	Mean.	Std.
f_1	0.00	0.00	0.00	0.00	0.00
f_2	430.91	795.42	582.21	592.41	91.65
f_3	33.90	45.42	40.33	40.33	2.22
f_4	3.26	4.68	4.13	4.10	0.38
f_5	273.60	588.14	387.97	384.52	67.62
f_6	0.20	0.63	0.50	0.45	0.12
f_7	0.53	1.73	1.01	1.01	0.21
f_8	100.00	100.00	100.00	100.00	0.00
f_9	436.02	458.85	448.68	447.37	5.96
f_{10}	413.66	413.66	413.66	413.66	0.00

From Tables 6.10, 6.11 and 6.12, it can be seen that the standard deviation of the DE algorithm on each function is not large. This indicates that the DE algorithm has good stability, i.e., the optimal solutions found do not differ much when run independently for many times. Even in solving some problems, it is only necessary to run the method once to find the appropriate solution without repeatedly running it many times.

6.4 Artificial Bee Colony Algorithm

In the Chap. 5, we introduced the GA, ACO, PSO and DE algorithms, all of which are EC methods proposed before 2000. In this chapter, we will introduce the EC methods proposed after 2000. In this section, we want to introduce the artificial bee colony (ABC) algorithm and summarize the research related to the ABC algorithm.

Swarm intelligence (SI) is the urgent collective behavior of decentralized, self-organizing systems. The ACO, ABC and PSO algorithms are all commonly used SI methods. Obviously, the name of each algorithm indicates the source of inspiration for the algorithm. SI systems usually consist of a group of individuals, and simulate the behavior of a group of individuals. For example, the ACO algorithm simulates a group of ants, and the individuals are ants; the PSO algorithm is a group of particles, and the individuals are some kinds of particles; the ABC algorithm is a group of bees, and the individuals are bees.

Next, we present the classical ABC algorithm, which was proposed by Karaboga in 2005 and simulates the foraging behavior of a honeybee colony. It is one of the most prominent methods in the field of bee-inspired SI methods.

The ABC algorithm contains three groups of artificial bees. They are employed bees, onlooker bees and scout bees. As other SI methods, the ABC algorithm has an initialization phase. After initialization, one cycle of the ABC algorithm includes the employed bee phase, the onlooker bee phase and the scout bee phase, which are described in detail below.

In the initialization phase of the ABC algorithm, a colony (population) of Np solutions is created randomly, where Np denotes the population size. For the ABC algorithm, the solution of the problem to be solved is compared to a food source, which attracts bees to collect nectar and make honey. Accordingly, the value of the fitness corresponding to the solution corresponds to the amount of nectar from one food source. As with the minimization problem study, a small value of the function implies a large amount of nectar.

In the employed bee phase, Np hired bees are sent out to search for food sources with a 1:1 ratio of hired bees to food sources. Each hired bee flies out of the hive to search for a food source. This has the advantage that the colony size is fixed and conforms to the population configuration of the EC method. In the ABC algorithm, a hired bee searches for more nectar around the relevant food source, and this behavior is achieved by the following equation:

$$v_{i,j} = \begin{cases} x_{i,j} + \varphi\left(x_{i,j} - x_{r1,j}\right), & \text{if } j = j1 \\ x_{i,j}, & \text{otherwise} \end{cases} \tag{6.26}$$

where $v_{i,j}$, $x_{i,j}$ and $x_{r1,j}$ is the j-th element of $\mathbf{v}_i$, $\mathbf{x}_i$, and $\mathbf{x}_{r1}$, respectively; $\varphi \in [-1, 1]$ is a random number. $j1 \in [1, D]$ and $r1 \in [1, Np]$ are random integers. $\mathbf{v}_i$ is the newly generated candidate solution. After evaluating $\mathbf{v}_i$, a greedy selection is performed between $\mathbf{v}_i$ and $\mathbf{x}_i$ and the winner is stored as the new $\mathbf{x}_i$.

Algorithm 6.1 Pseudocode for the ABC algorithm

Number	Pseudo-code of the ABC algorithm
1	/* initialization phase*/
2	For $i = 1$ to N_p
3	For j = 1 to D
4	$x_{i,j} = x_{j,min} + rand(0, 1) \times \left(x_{j,max} - x_{j,min}\right)$;
5	Evaluate $\mathbf{x_i}$, compute its fitness fit_i ($i = 1, 2, \ldots, N_p$)
6	Set $iter = N_p$, **limit** $= [limit_1, limit_2, \ldots, limit_{Np}] = [0, 0, \ldots, 0]$
7	While $iter <$ MFE
8	/* employed bee phase*/
9	For $i = 1$ to N_p
10	$j1 = randInt(1, D)$;
11	Do $r1 = randInt(1, N_p)$; while($r1 == i$);
12	$\varphi_{i,j1} = rand(-1, 1)$;
13	For $j = 1$ to D
14	If $j == j1$, $v_{i,j} = x_{i,j} + \varphi_{i,j1}\left(x_{i,j} - x_{r1,j}\right)$;
15	Else $v_{i,j} = x_{i,j}$;
16	Evaluate$\mathbf{v}_i$, compute its fitness, set $iter = iter + 1$;
17	If $f(\mathbf{v}_i) < f(\mathbf{x}_i)$
18	Replace $\mathbf{x}_i$ by $\mathbf{v}_i$;
19	Replace $f(\mathbf{x}_i)$ by $f(\mathbf{v}_i)$, replace $fit_{\mathbf{x}_i}$ by $fit_{\mathbf{v}_i}$;
20	$limit_i = 0$;
21	Else $limit_i = limit_i + 1$;
22	/* onlooker bee phase */
23	Compute probability p_i of food sources;
24	For $i = 1$ to N_p
25	Use roulette wheel method to choose $\mathbf{x}_{r1}$ based on p_i;
26	$j1 = randInt(1, D)$;
27	Do $r2 = randInt(1, N_p)$; While($r2 == r1$);
28	$\varphi_{i,j1} = rand(-1, 1)$;
29	For $j = 1$ to D
30	If $j == j1$, $v_{i,j} = x_{r1,j} + \varphi_{i,j1}\left(x_{r1,j} - x_{r2,j}\right)$;
31	Else $v_{i,j} = x_{r1,j}$;
32	Evaluate $\mathbf{v}_i$, compute its fitness, set $iter = iter + 1$;
33	If $f(\mathbf{v}_i) < f(\mathbf{x}_{r1})$
34	Replace $\mathbf{x}_{r1}$ by $\mathbf{v}_i$;
35	Replace $f(\mathbf{x}_{r1})$ by $f(\mathbf{v}_i)$, replace $fit_{\mathbf{x}_{r1}}$ by $fit_{\mathbf{v}_i}$;
36	$limit_{r1} = 0$;

(continued)

(continued)

Number	Pseudo-code of the ABC algorithm
37	Else $limit_{r1} = limit_{r1} + 1$;
38	/* scout bee phase */
39	For $i = 1$ to N_p
40	If $limit_i > limit$
41	For $j = 1$ to D
42	$v_{i,j} = x_{j,min} + rand(0, 1) \times (x_{j,max} - x_{j,min})$;
43	Evaluate $\mathbf{x}_i$, compute its fitness, set $iter = iter + 1$;
44	Replace $\mathbf{x}_i$ by $\mathbf{v}_i$;
45	Replace $f(\mathbf{x}_i)$ by $f(\mathbf{v}_i)$, replace $fit_{\mathbf{x}_i}$ by $fit_{\mathbf{v}_i}$;
46	$limit_i = 0$;

In the onlooker bee phase, a total of *Np* onlooker bees was dispatched. This number was equal to the colony size in the employed bee phase. However, unlike the hiring bee stage, the onlooker bees selected their food source based on their nectar amount. A food source with a high fitness value attracted more onlooker bees; while a food source with a low fitness value was less attractive or even unattractive to the onlooker bees. This suggests that a good food source (high amount of nectar) can attract multiple onlooker bees, while a bad food source (low amount of nectar) can barely attract an onlooker bee. In the ABC algorithm, this behavior is achieved by first calculating the probability value of each solution (food source); then the solution is selected using a roulette wheel method. The probability value of solution $\mathbf{x}_i$ is calculated by the following equation:

$$p_i = \frac{fit_i}{\sum_{j=1}^{N_p} fit_j} \tag{6.27}$$

where the value of the fit_i for $\mathbf{x}_i$ is calculated by:

$$fit_i = \begin{cases} 1/(1 + f(\boldsymbol{x}_i)), & \text{if } f(\boldsymbol{x}_i) \geq 0 \\ 1 + |f(\boldsymbol{x}_i)|, & \text{otherwise} \end{cases} \tag{6.28}$$

After selecting a solution, the onlooker bee uses (6.26) to modify that solution. A greedy selection is then performed between the newly generated solution and the old one. The winner is stored as the new $\mathbf{x}_i$.

If a food source has been searched for a long time, its nectar amount will decrease and it may be abandoned by the bees. In this case, the bee must fly out to find a new food source. In the ABC algorithm, a predefined parameter called *limit* is introduced to determine whether a food source should be abandoned or not. If a solution cannot be further improved in a limited time, then it should be abandoned and the employed

bee becomes a scout bee. The discarded solution is replaced by a randomly generated solution. This is the scout bee phase of the ABC algorithm.

The pseudo-code of the ABC algorithm is shown in Algorithm 6.1. In the algorithm, *rand*(Min, Max) generates a uniformly distributed random number between Min and Max; *randInt*(Min, Max) generates a uniformly distributed random integer between Min and Max. Researchers have implemented the ABC algorithm in several programming languages. The source code can be downloaded for free from Karaboga's personal website.

The performance of the ABC algorithm has been compared with other heuristic algorithms such as the PSO, GA and DE algorithms. Simulation results show that the ABC algorithm outperforms the above algorithms for some problems. In addition, the ABC algorithm is also effective for large-scale problems.

Due to the success of the ABC algorithm, many researchers have tried to improve its performance in recent years. Similar to the functional of the PSO algorithm, Diwold et al. proposed to use the global optimal solution found so far to generate new candidate solutions, called ABCgBest. Inspired by the PSO algorithm, Zhu et al. proposed the GABC (Gbest-guided ABC) algorithm, which uses the information of the global optimal solution to guide the search. Inspired by the DE algorithm, Gao et al. proposed an improved ABC algorithm in which the bees search only around the optimal solution of the previous iteration. Banharnsakun et al. proposed the "best-so-far" ABC (BABC), which biases the direction of the solution toward the best position to date. The above works attempt to speed up the search process of the ABC algorithm by using the global optimal solution. However, the use of global optimal solutions increases the exploitation pressure of the algorithm and has a high risk of premature convergence.

Since the ABC algorithm does not use crossover operators like the GA or DE algorithms, it is not very efficient to propagate good information between solutions. Recently, some modifications have been made to the original ABC algorithm by combining it with crossover operators like the GA, DE or Hooke-Jeeves pattern search algorithms to remedy this situation. In addition, the ABC algorithm has been combined with the agent model and the finite element method in order to search the global optimum more efficiently.

The ABC algorithm has been applied to practical problems such as training of neural networks, clustering, image segmentation, structural design, and inverse electromagnetic field problems. In recent years, the ABC algorithm has been extended to solve multi-objective design optimization problems. A comprehensive review and discussion of the ABC algorithm has also been conducted by researchers.

The ABC algorithm simulates the foraging behavior of honeybees. It is one of the most prominent approaches in the field of bee-inspired methods. We observe that the paradigm of the ABC algorithm involves two algorithm parameters (i.e., *Np* and limit). Compared with the paradigm of the DE algorithm, the ABC algorithm contains a smaller number of parameters. Thus, the ABC algorithm is also popular among researchers, and interested readers can discover more variant methods based on it.

Example 6.4 Suppose we use the ABC algorithm to solve for the functions in the CEC2020 test set with $D = 10$, 15 and 20. Please analyze the effect of the ABC algorithm.

Solution The CEC2020 test set has been introduced in Sect. 6.2. The ABC algorithm can solve continuous optimization problems, and we use it to solve the functions in the CEC2020 test set. The results on the test set are shown in Tables 6.13, 6.14 and 6.15, where Table 6.13 shows the results for functions with $D = 10$, Table 6.14 shows the results for functions with $D = 15$, and Table 6.15 shows the results for functions with $D = 20$.

From Tables 6.13, 6.13 and 6.15, it can be seen that the ABC algorithm has worse results than the DE algorithm and the PSO algorithm on function f_5 and function f_7. While the ABC algorithm has better results than the PSO algorithm on function f_1, and the ABC algorithm has better results than the DE algorithm and the PSO

Table 6.13 Results of the ABC algorithm on CEC2020 test set with $D = 10$

f	Min.	Max.	Med.	Mean	Std.
f_1	3.07	71.95	20.61	25.66	17.58
f_2	8.85	46.99	24.01	23.93	9.05
f_3	9.59	16.60	14.52	14.44	1.48
f_4	0.18	1.07	0.62	0.61	0.23
f_5	1475.77	68,732.30	14,463.95	19,220.44	16,100.46
f_6	0.44	1.95	1.11	1.15	0.39
f_7	305.01	5684.78	1173.59	1321.70	1053.18
f_8	7.52	33.80	25.19	24.39	6.27
f_9	31.27	100.00	54.79	61.81	20.93
f_{10}	116.24	181.37	135.48	138.26	17.15

Table 6.14 Results of the ABC algorithm on CEC2020 test set with $D = 15$

f	Min.	Max.	Med.	Mean	Std.
f_1	49.73	514.58	230.36	241.34	131.24
f_2	4.15	123.33	22.07	28.92	24.30
f_3	15.43	19.72	18.35	18.23	1.05
f_4	0.28	0.83	0.60	0.61	0.15
f_5	4687.85	109,004.95	38,418.16	40,191.92	25,590.43
f_6	6.85	38.04	18.14	19.84	8.04
f_7	1713.00	26,334.04	6997.12	9080.88	6217.21
f_8	4.84	51.07	29.38	29.92	8.39
f_9	94.09	131.20	116.01	116.47	7.51
f_{10}	176.74	400.00	251.13	269.38	72.77

Table 6.15 Results of the ABC algorithm on CEC2020 test set with $D = 20$

f	Min.	Max.	Med.	Mean	Std.
f_1	0.52	20.28	5.33	6.39	4.51
f_2	2.35	13.14	6.71	7.29	2.60
f_3	22.03	24.34	23.15	23.16	0.67
f_4	0.36	1.06	0.83	0.77	0.19
f_5	40,078.49	151,126.43	84,534.70	87,954.17	29,422.54
f_6	0.36	0.99	0.73	0.71	0.14
f_7	2319.70	41,424.00	20,244.38	19,670.52	9740.16
f_8	53.53	100.00	79.22	80.46	11.18
f_9	70.67	104.01	102.53	100.52	7.50
f_{10}	399.49	400.16	399.83	399.83	0.17

algorithm on functions f_8, f_9 and f_{10}. This indicates that each of the three EC methods has its own advantages and disadvantages. From this perspective, readers can try to fuse these three algorithms, for example, using an integrated learning approach, to obtain an improved algorithm with better performance on the CEC2020 test set.

6.5 Fireworks Algorithm

Fireworks Algorithm (FWA) is a SI optimization method [2]. The method is inspired by the explosion of fireworks in the night sky. It generates sparks by simulating the explosion of fireworks to illuminate a part of the night sky. The FWA was proposed by Tan of Peking University in 2010 [3].

The flow of the FWA is shown in Fig. 6.5. In the fireworks algorithm, we need to generate N random locations of fireworks in the search space. One firework corresponds to a feasible solution of the problem. Based on the fitness, we can assign resources to each firework and thus control the explosion behavior of the fireworks. Each firework is assigned a blast radius and the number of sparks it can produce. Then, each firework explodes, producing the corresponding number of sparks. A Gaussian mutation operation is then applied to the generated sparks for a better search. The FWA also has a selection operation to choose N new fireworks locations from the three sets of feasible solutions: fireworks, exploding sparks, and Gaussian variant sparks. The above steps are cycled until the algorithm terminates. The above steps are the flow of the FWA, and we next describe in detail how each step is computed.

Assigning a radius to each firework that can explode, the blast radius of firework i is calculated as follows:

$$R_i = \overline{R}\frac{f_i - y_{min} + \varepsilon}{\sum_{j=1}^{N}\left(f_j - y_{min} + \varepsilon\right)} \tag{6.29}$$

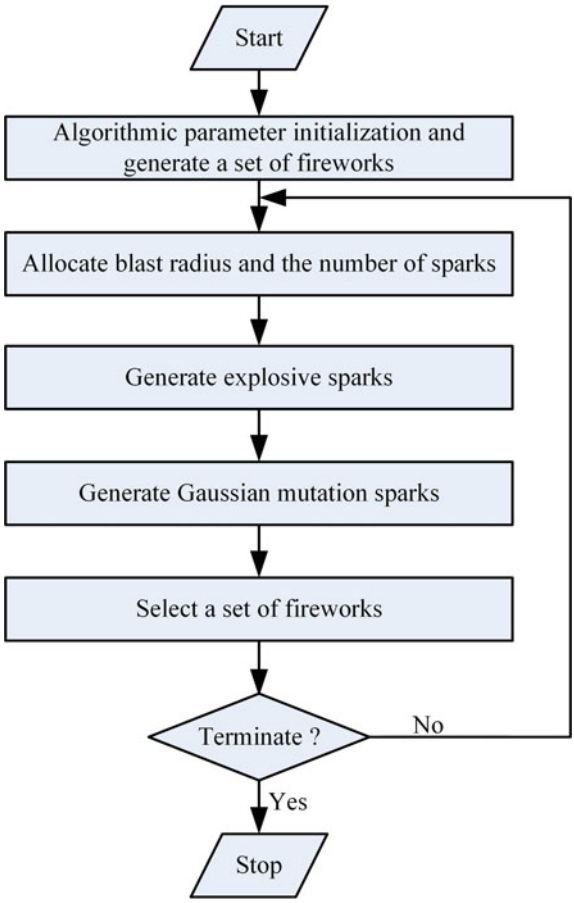

Fig. 6.5 Flow chart of the fireworks algorithm

where $\overline{R}$ is the average blast radius, f_i is the adaptation of the i-th firework, y_{max} is the minimum value of the firework adaptation, and ε is the smallest value that the computer can represent, which avoids numerical problems.

Assigning the number of sparks that can be produced to each firework, the number of sparks that can be produced by firework i is calculated as follows:

$$S_i = M \frac{y_{max} - f_i + \varepsilon}{\sum_{j=1}^{N} \left(y_{max} - f_j + \varepsilon \right)} \tag{6.30}$$

where M is a constant that represents the total number of sparks and is the maximum value of the fireworks adaptation. Besides (6.30), the number of sparks is additionally limited by:

$$S_i = \begin{cases} round(aM), \ if \ S_i < aM \\ round(bM), \ if \ S_i > bM \\ round(S_i), \quad otherwise \end{cases}, 0 < a < b < 1 \tag{6.31}$$

where a and b are constants.

Every firework has the potential to produce sparks. The formula for the explosive sparks produced by firework i is as follows:

$$\boldsymbol{x}_i = \boldsymbol{x}_i + R_i rand(-1, 1) \tag{6.32}$$

where R_i is the blast radius of the firework $\boldsymbol{x}_i$, which $rand(-1, 1)$ is a uniformly distributed random number between 1 and 1. It should be noted that in (6.32), a number of dimensions are randomly selected from the D-dimensional search space for each search, and not all dimensions are updated.

A number of sparks are selected for mutation from the exploding sparks. The equation for the generation of Gaussian mutation sparks by spark i is as follows:

$$\boldsymbol{x}_i = \boldsymbol{x}_i N(1, 1) \tag{6.33}$$

where $N(1, 1)$ is a Gaussian distributed random number with mean 1 and variance 1. Note that N sparks are randomly selected from the exploded sparks for mutation. An operation called Gaussian mutation of sparks, where the positions are scaled by multiplying the original position by a Gaussian distributed random number. Gaussian mutation is also performed by randomly selecting a number of dimensions from the D-dimensional search space, and not all dimensions are updated.

If an exploding spark or a Gaussian variant spark gets a position that is beyond the upper and lower bounds of the search space, it needs to be transformed into the feasible region by the following equation:

$$\boldsymbol{x}_i = \boldsymbol{x}_{lb} + \boldsymbol{x}_i \% (\boldsymbol{x}_{ub} - \boldsymbol{x}_{lb}) \tag{6.34}$$

where $\boldsymbol{x}_{lb}$ is the lower bound of $\boldsymbol{x}_i$, the $\boldsymbol{x}_{ub}$ is upper bound of $\boldsymbol{x}_i$, and % indicates the remainder operation.

The selection phase of the FWA is to select N new fireworks positions from three sets of feasible solutions: fireworks, exploding sparks, and Gaussian variant sparks. The probability of fireworks and sparks being selected is calculated as follows:

$$p(\boldsymbol{x}_i) = \frac{R(\boldsymbol{x}_i)}{\sum R(\boldsymbol{x}_j)} \tag{6.35}$$

where $\boldsymbol{x}_i$ denotes a firework, explosive spark, or Gaussian variant spark, and $R(\boldsymbol{x}_i)$ denotes the sum distances of $\boldsymbol{x}_i$ to other fireworks or sparks. Euclidean distances are generally used here.

In the standard FWA, the default value of N is 5, the default value of $\overline{R}$ is 40, the default value of M is 50, the default values of a and b are 0.04 and 0.8, respectively, and the number of Gaussian variant sparks is 5. It can be seen that the FWA contains six hyperparameters. The number of its hyperparameters is greater than that of the GA, the PSO algorithm, the DE algorithm, and the ABC algorithm.

In recent years, researchers have proposed many optimization algorithms for SI. In addition to the FWA (2010), the SI methods include the brain storm optimization (BSO) algorithm (2011) [4], and the pigeon-inspired optimization (PIO) algorithm (2014) [5]. The FWA is a SI optimization method that simulates the physics of lighting fireworks. The BSO algorithm simulates the human group behavior and proposes a SI optimization method. The PIO algorithm is a SI optimization method proposed by simulating the biological phenomenon of pigeon population.

6.6 Research Progress of Evolutionary Computing

This section reviews the state-of-the-art research progress of evolutionary computing. These researches are classified to seven categories. They are genetic algorithm, ant colony optimization, particle swarm optimization, differential evolution, artificial bee colony, fireworks algorithm and other EC methods [6, 7].

(1) genetic algorithm

With the exponential increase in the amount of data generated and processed daily in machine learning and decision-making systems, data preprocessing has become a key factor in building reliable and high-performance machine learning models. One of the functions of preprocessing is to use feature selection method to reduce variables; However, the processing time required for these methods is a major drawback. Mehanović et al. aimed to alleviate this problem by migrating the algorithm to a MapReduce implementation suitable for parallelization on a large number of commodity hardware units [8]. Hadoop was an open-source MapReduce library used as a framework for implementing parallel genetic algorithms. The feature selection method was applied to four datasets. The experimental results show that genetic algorithm allows feature selection with enhanced randomness. Its parallelization reduces the overall data preprocessing and allows a larger population, which in turn leads to better feature selection. In practice, it has been shown that this implementation is superior to existing feature selection methods.

Liang et al. proposed an image encryption algorithm based on Fibonacci Q-matrix and genetic algorithm [9]. A new four-layer encryption framework with diffusion perturbation diffusion optimization was adopted. The experimental results and security analysis indicate that the algorithm not only has high security, but also has a certain degree of robustness and real-time performance, which is suitable for practical applications.

The clustering of mixed numerical and categorical attributes has attracted many researchers due to its necessity in many practical applications. A key issue in clustering mixed data is selecting appropriate distance metrics for each attribute type. In addition, some current clustering methods are sensitive to initial solutions and are prone to falling into local optima. Therefore, Nguyen et al. proposed a possibility weighted fuzzy c-means based on local search genetic algorithm (LSGA-PWFCM) for clustering mixed data [10]. The possibility weighted fuzzy c-means used object clustering similarity measure to calculate the distance between two mixed attributes. Genetic algorithm was used to find a set of optimal parameters and initial clustering centroids for the possibility weighted fuzzy c-means algorithm. In order to avoid local optima, a variable neighborhood based on local search was embedded in the genetic algorithm. Based on some common datasets, the proposed LSGA-PWFCM algorithm was compared with other benchmark algorithms. The experimental results show that the LSGA-PWFCM method outperforms other algorithms on most test datasets.

With the improvement of quality parameters, a series of Internet of Vehicles (IoV) services have emerged. However, this field still faces some limitations, including resource constraints and time response requirements. Abbasi et al. proposed an algorithm that used genetic algorithm for fault and cost management during resource allocation to services [11]. The main concept was to use genetic algorithm to select resources for services. In the first step, the proposed method determined the priority of the service and allocated resources based on these priorities. In the second step, the proposed method ensured load balancing of the message transmission path and avoided message failures. The performance of this method was evaluated using various parameters and showed to be superior to other evolutionary computing methods. In addition, the proposed method provides acceptable performance in terms of service response time.

(2) ant colony optimization

Luo et al. proposed an improved ant colony optimization algorithm to solve the problems of local optimization, slow convergence rate and low search efficiency [12]. The initial pheromone with unequal distribution was constructed to avoid blind search in early planning. The proposed algorithm used pseudo-random state transition rules to select paths, calculated state transition probabilities based on the current optimal solution and number of iterations, and adaptively adjusted the proportion of determined or randomly selected paths. The results show that compared with other ant colony optimization algorithms, the proposed algorithm improves global optimal search ability and convergence rate under different robot mobile simulation environments.

Laser engraving is an important tool for automatic drawing and three-dimension (3D) printers. When laser engraving tasks become large and complex, the engraving process will be very time-consuming. In order to improve the time and energy efficiency of laser engraving, trajectory optimization of laser engraving was studied by Wu et al. [13]. By transforming grayscale into halftone image, the trajectory of laser engraving robot was modeled as a large-scale TSP. In order to solve the TSP,

a new two-layer ant colony optimization algorithm was proposed that combined k-means, top-level ant colony system, and low-level ant colony system. The experimental results show that compared with traditional engraving methods, the proposed method can reduce laser engraving time by about 50%.

In order to improve the accuracy and stability of ACO algorithm, a dynamic induced clustering ACO algorithm based on coevolutionary chain was proposed by Yu et al. [14]. First, the distribution of pheromone left by ants in small data clustering was divided according to the density to induce subsequent ants to choose, so as to balance convergence rate and solution accuracy. Second, the coevolutionary chain increased the diversity and stability of the algorithm through population coevolution and link dimensionality reduction. Simulation experiments show that the roposed ACO algorithm can effectively balance convergence rate and solution accuracy.

(3) particle swarm optimization

PSO is one of the most concerned meta-heuristic algorithms, which has remarkable performance in solving various optimization problems. However, PSO algorithm faces two main problems, namely slow convergence rate and local optimal capture. Shami et al. proposed a new idea called velocity pausing, in which the particles were supported by the third move option [15]. The velocity pausing allowed the particles to move at the same speed as the last iteration. In order to avoid premature convergence, velocity pausing particle swarm optimization (VPPSO) modified the first term of the velocity equation. In addition, the population of VPPSO was divided into two groups to maintain diversity. The performance of VPPSO was verified on 43 benchmark functions and 4 practical engineering problems. According to Wilcoxon rank-sum and Friedman test, VPPSO can significantly outperform the seven prominent algorithms in most test functions in low dimensional and high-dimensional situations.

Kiruthiga et al. proposed a new optimized deep learning (DL) network design for time series load forecasting [16]. Firstly, the super parameters of DL were optimized using Levy flight particle swarm optimization (LF-PSO) technology; Then, the optimized DL model was used for load forecasting. The experimental and measured results indicate that the proposed method is efficient for short-term load forecasting.

(4) differential evolution

Resource allocation is very important in wireless communication systems. Zhang et al. investigated the spectrum allocation problem in a cellular network operating on an orthogonal frequency-division multiple access (OFDMA) system [17]. Network utility and fairness among all linked users were used to measure service quality in cellular network. The spectrum allocation was represented as a maximization optimization model. The authors proposed an adaptive differential evolution algorithm based on fluctuation, denoted as WADE. The WADE algorithm adjusted the algorithmic parameters through wave propagation. The simulation results show that the WADE algorithm is more effective than other algorithms for allocating spectrum resources in OFDMA system.

Fusion is a state-of-the-art technology for observing behavioral patterns from time series data. The fusion model may become single model constraints due to feature limitations. Kumar et al. proposed a three-stage fusion model to process time series data [18]. In the first stage of integration, stock market inputs were combined with established technical indicators of the stock market. In the second stage, autoregressive integrated moving average and long short-term memory were combined to observe the linear and nonlinear characteristics of the final stock dataset. In the third stage, an improved artificial bee colony using differential evolution algorithm was studied for hyperparameter selection in the proposed model for stock market prediction. The experiments on the historical datasets show that the proposed fusion model outperforms the benchmark model.

(5) artificial bee colony

The IoT provides humanity with a beautiful and intelligent landscape. The connection of various sensors and devices in the IoT will lead to significant energy consumption. Therefore, it is imperative to study energy-saving methods. For wireless sensors in an IoT, achieving sustainable operation through energy-saving methods is crucial due to limited batteries. Zhang et al. studied an IoT network that included wireless sensors and base stations [19]. Wireless power transmission technology was used for providing battery charging, and charging vehicles were responsible for power supply. The electricity and data transmission in the IoT network was represented as a minimization optimization problem. A three-stage restart artificial bee colony method was proposed by Zhang et al. for handling optimization problem. The experimental results show that the proposed method can be used to minimize consumption in the studied IoT network.

Breast cancer is the most common cancer among women, and if not diagnosed early, it can lead to death. Early diagnosis plays a crucial role in reducing global mortality rates. Computer assisted diagnosis (CAD) can overcome the shortcomings of manual methods. The CAD system based on artificial neural network (ANN) is optimized by the meta-heuristic algorithms. Stephan et al. combined the employee bee stage of ABC with the bubble net attack method of whale optimization. Stephan et al. proposed a hybrid algorithm of artificial bee colony and whale optimization (HAW) [20]. The HAW algorithm was used for feature selection and parameter optimization of ANN. Simulation results show that the HAW using elastic back-propagation learning achieved higher accuracy than other methods.

The ABC algorithm has a drawback of imbalanced search behavior. Alrosan et al. introduced a new ABC algorithm, dentoed as MeanABC [21]. The MeanABC algorithm was based on the mean information of the previous best solution and achievd a balance in search behavior by modifying the search equation. The experimental results show that compared with other ABC variants, the MeanABC algorithm enhances the performance of the original ABC in terms of faster convergence rate, better solution quality and better robustness.

Satoh et al. utilized the ABC algorithm to solve the design problem of a discrete-time stable unknown input estimator (UIE) based on parameter optimization [22]. First, a stability assurance design method for UIE was provided by Satoh et al.

Next, a new objective function was developed, which combines waveform-based and norm-based performance standards to allow direct evaluation of the adverse impact of interference on system performance. Finally, the proposed design method was compared with the previous design method, using an objective function based on estimated disturbances.

(6) fireworks algorithm

As the main components of large industrial rolling equipment, rolling bearings have complex working conditions and are prone to malfunctions. The analysis of the initial weak signal can be suitable for identifying the suboptimal health status of industrial rolling equipment. Luo et al. proposed an offline suboptimal health recognition algorithm based on refined composite multi-scale discrete entropy and extreme learning machine optimized by improved FWA [23]. First, FWA was improved by Cauchy mutation and adaptive dynamic explosion radius factor coefficients. Second, the initial vibration signal was processed by the improved parameter optimized maximum correlation kurtosis deconvolution. Finally, extreme learning machine was combined with deep belief network. The number of hidden nodes was optimized using the improved FWA. The simulation results show that the proposed algorithm has higher suboptimal health recognition accuracy and has good industrial application prospects.

In recent years, multimodal multi-objective optimization problems (MMOPs) have received increasing attention. Their goal is to find a Pareto front and as many equivalent Pareto optimal solutions as possible. Although some EC methods have been proposed, they mainly focus on the convergence rate in the decision space and ignore the diversity of solutions. Han et al. proposed a new multi-objective FWA [24]. Han et al. extended the latest single target FWA to handle MMOPs. Then, they incorporated adaptive strategies and special file guidance to update the location of the sparks. The experimental results show that the proposed algorithm outperforms comparative algorithms in solving MMOPs and imbalanced distance minimization problems in CEC2019.

(7) other EC methods

In brain storm optimization (BSO) algorithm, convergence operations use clustering strategies to group populations into multiple clusters, and divergence operations use this clustering information to generate new individuals. However, this mechanism is inefficient in regulating exploration and mining searches. Ma et al. analyzed the main factors that affected the performance of BSO [25]. They proposed an orthogonal learning framework to improve the learning mechanism of BSO. The experimental results show that the proposed method is effective in optimizing complex functions. It not only outperforms previous versions of the BSO algorithm, but also several well-known orthogonal design-based algorithms.

Classification is one of the most classic problems in machine learning. Evolutionary classification model is one of the methods to solve classification problems. In recent years, the FWA and BSO algorithms have been used to implement evolutionary

classification models, and have achieved the desired results. However, existing evolutionary classification models still have some shortcomings. The limited dataset makes the experimental results not convincing enough. More importantly, the structure of the evolutionary classification model is closely related to the dimensions of the dataset, which may lead to poor classification performance. Therefore, Xue et al. modified the structure of evolutionary classification models to improve classification performance [26]. First, they introduced the concept of feature selection, and used different feature subsets to construct the evolutionary classification model. Then, the evolutionary classification model was implemented using the BSO algorithm. The simulation results show that it is feasible to optimize the structure of evolutionary classification model by introducing feature selection. In addition, the proposed method has better classification performance than the original method.

For multiple unmanned aerial vehicles (UAV) performing aerial search and attack missions, there is a trade-off between maximizing total benefits and minimizing consumption while constraining effectiveness. Duan et al. proposed a dynamic discrete pigeon-inspired optimization algorithm to handle cooperative search-attack mission planning for UAV [27]. The proposed algorithm integrated centralized task allocation and distributed path generation aspects of the problem. A solution acceptance strategy was proposed to avoid frequent task switching. Bayesian formulas were used to construct and update probability maps to guide subsequent search movements. A response threshold sigmoid model was used for target allocation during the attack process. Numerical experiments have shown that the proposed method can provide feasible solutions for multiple UAV.

Exercises

(1) The TSPCN test set include seven TSP problems. The test set has been introduced in Sect. 6.1. Try to use GA, PSO, DE, ABC or other EC methods to solve the TSPCN test set, and analyze the solutions obtained by different EC methods.
(2) Section 6.5 introduces the FWA algorithm. Try to use the FWA algorithm to solve the CEC2020 test set, and analyze the performance of the FWA algorithm.

References

1. Yue CT, Price KV, Suganthan PN, Liang JJ, Ali MZ, Qu BY, Award NH, Biswas PP (2020) Problem Definitions and evaluation Criteria for the CEC 2020 special session and competition on single objective bound constrained numerical optimization. Technical Report 201911, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China, Nanyang Technological University, Singapore
2. Tan Y (2015) Fireworks algorithm: a novel swarm intelligence optimization method. Springer, Berlin
3. Li J, Tan Y (2020) A comprehensive review of the fireworks algorithm. ACM Comput Survey 52:1–28

4. Shi Y (2011) Brain storm optimization algorithm. In: Tan Y, Shi Y, Chai Y, Wang G (eds) Advances in swarm intelligence. ICSI 2011. Lecture notes in computer science, vol 6728. Springer, Berlin, Heidelberg, pp 303–309
5. Duan H, Qiao P (2014) Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning. Int J Intell Comput Cybern 7:24–37
6. Cheng S, Qin Q, Chen J et al (2016) Brain storm optimization algorithm: a review. Artif Intell Rev 46:445–458
7. Duan H, Huo M, Shi Y (2020) Limit-cycle-based mutant multiobjective pigeon-inspired optimization. IEEE Trans Evol Comput 24(5):948–959
8. Mehanović D, Kečo D, Kevrić J et al (2021) Feature selection using cloud-based parallel genetic algorithm for intrusion detection data classification. Neural Comput Applic 33:11861–11873. https://doi.org/10.1007/s00521-021-05871-5
9. Liang Z, Qin Q, Zhou C (2022) An image encryption algorithm based on Fibonacci Q-matrix and genetic algorithm. Neural Comput Applic 34:19313–19341. https://doi.org/10.1007/s00521-022-07493-x
10. Nguyen TPQ, Kuo RJ, Le MD et al (2022) Local search genetic algorithm-based possibilistic weighted fuzzy c-means for clustering mixed numerical and categorical data. Neural Comput Applic 34:18059–18074. https://doi.org/10.1007/s00521-022-07411-1
11. Abbasi S, Rahmani AM, Balador A, Sahafi A (2023) A fault-tolerant adaptive genetic algorithm for service scheduling in internet of vehicles. Appl Soft Comput 143:110413. https://doi.org/10.1016/j.asoc.2023.110413
12. Luo Q, Wang H, Zheng Y et al (2020) Research on path planning of mobile robot based on improved ant colony algorithm. Neural Comput Applic 32:1555–1566. https://doi.org/10.1007/s00521-019-04172-2
13. Wu Z, Wu J, Zhao M et al (2021) Two-layered ant colony system to improve engraving robot's efficiency based on a large-scale TSP model. Neural Comput Applic 33:6939–6949. https://doi.org/10.1007/s00521-020-05468-4
14. Yu J, You X, Liu S (2022) Dynamically induced clustering ant colony algorithm based on a coevolutionary chain. Knowl-Based Syst 251:109231. https://doi.org/10.1016/j.knosys.2022.109231
15. Shami TM, Mirjalili S, Al-Eryani Y et al (2023) Velocity pausing particle swarm optimization: a novel variant for global optimization. Neural Comput Applic 35:9193–9223. https://doi.org/10.1007/s00521-022-08179-0
16. Kiruthiga D, Manikandan V (2023) Levy flight-particle swarm optimization-assisted BiLSTM + dropout deep learning model for short-term load forecasting. Neural Comput Applic 35:2679–2700. https://doi.org/10.1007/s00521-022-07751-y
17. Zhang X, Zhang X, Wu Z (2019) Spectrum allocation by wave based adaptive differential evolution algorithm. Ad Hoc Netw 94:101969
18. Kumar R, Kumar P, Kumar Y (2022) Three stage fusion for effective time series forecasting using Bi-LSTM-ARIMA and improved DE-ABC algorithm. Neural Comput Applic 34:18421–18437. https://doi.org/10.1007/s00521-022-07431-x
19. Zhang X, Zhang X, Han L (2019) An energy efficient internet of things network using restart artificial bee colony and wireless power transfer. IEEE Access 7:12686–12695
20. Stephan P, Stephan T, Kannan R et al (2021) A hybrid artificial bee colony with whale optimization algorithm for improved breast cancer diagnosis. Neural Comput Applic 33:13667–13691. https://doi.org/10.1007/s00521-021-05997-6
21. Alrosan A, Alomoush W, Norwawi N et al (2021) An improved artificial bee colony algorithm based on mean best-guided approach for continuous optimization problems and real brain MRI images segmentation. Neural Comput Applic 33:1671–1697. https://doi.org/10.1007/s00521-020-05118-9
22. Satoh T, Nishizawa S, Nagase J et al (2023) Artificial bee colony algorithm-based design of discrete-time stable unknown input estimator. Inf Sci 634:621–649. https://doi.org/10.1016/j.ins.2023.03.130

23. Luo H, He C, Zhou J, Zhang L (2021) Rolling bearing sub-health recognition via extreme learning machine based on deep belief network optimized by improved fireworks. IEEE Access 9:42013–42026
24. Han S, Zhu K, Zhou M et al (2022) A novel multiobjective fireworks algorithm and its applications to imbalanced distance minimization problems. IEEE/CAA J Automatica Sinica 9(8):1476–1489
25. Ma L, Cheng S, Shi Y (2021) Enhancing learning efficiency of brain storm optimization via orthogonal learning design. IEEE Trans Syst Man Cybern Syst 51(1):6723–6742
26. Xue Y, Zhao Y, Slowik A (2021) Classification based on brain storm optimization with feature selection. IEEE Access 9:16582–16590
27. Duan H, Zhao J, Deng Y, Shi Y, Ding X (2021) Dynamic discrete pigeon-inspired optimization for multi-UAV cooperative search-attack mission planning. IEEE Trans Aerosp Electron Syst 57(1):706–720