A Dendritic Neuron Model with Adaptive Synapses Trained by Differential Evolution Algorithm

by

Zhe Wang

A dissertation
submitted to the Graduate School of Science and Engineering for Education
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Engineering

University of Toyama
Gofuku 3190, Toyama-shi, Toyama 930-8555 Japan

2020
(Submitted June 18, 2020)

Abstract

According to the research of biology, the computational unit of human brain is neuron. It can respond to changes in the environment, and then send information to other neurons. In the human brain, there are about 86 billion neurons, interconnected to form an extremely complex nervous system. Biologically, neurons are usually composed of cell bodies, nuclei, dendrites, and axons. Dendrites are used to receive signals from other neurons, and a neuron has many dendrites.

In recent years, several dendritic computing models considering the functions of dendrites in a neuron have been proposed in the literature. Differential evolution (DE) has been employed to train dendritic neuron model with adaptive synapses (DMAS) in our research. Differential evolution algorithm is an efficient global optimization algorithm. It is also a population-based heuristic search algorithm, and each individual in the population corresponds to a solution vector. The evolution process of differential evolution algorithm is very similar to that of genetic algorithm, including mutation, crossover and selection operations, but the specific definition of these operations is different from that of genetic algorithm.

According to the signal transmission order, a DNM can be divided into four parts: the synaptic layer, dendritic layer, membrane layer, and somatic cell layer. It can be converted to a logic circuit that is easily implemented on hardware by removing useless synapses and dendrites after training. This logic circuit can be designed to solve complex nonlinear problems using only four basic logical devices: comparators, AND (conjunction), OR (disjunction), and NOT (negation). To obtain a faster and better solution, we adopt the most popular DE for DMAS training. We have chosen five classification datasets from the UCI Machine Learning Repository for an experiment. We analyze and discuss the experimental results in terms of the correct rate, convergence rate, ROC curve, and the cross-validation and then compare the results with a dendritic neuron model trained by the backpropagation algorithm (BP-DNM) and a neural network trained by the backpropagation algorithm (BPNN). The analysis results show that the DE-DMAS shows better performance in all aspects.

The remainder is organized as follows: In Chapter 1, we give a brief description of neuron model and differential evolution algorithm. Chapter 2 introduces the backgrounds of this reaserch. The back propagation algorithm is explained in chapter 3. Chapter 4 introduces the structurtrode of our model (DMAS). The learning algorithm (DE) is explained in chapter 5. The experimental method is designed in chapter 6. Chapter 7 presents the analysis and discussion of the experimental results.

The conclusions are drawn in chapter 8.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The human brain consists of billions of neurons, and a single neuron cell is constituted by a cell body, an axon, a cell membrane and a dendrite. Dendrites occupy more than 90 percent of the nerve cell organization and have a pivotal role in a human's learning process. The first artificial neuron was originally proposed by MuCulloch and Pitts in 1943 [1]. This model is an abstract and simplified model that was constructed according to the structure and working principle of a biological neuron membrane based on mathematics and algorithms called threshold logic.

The perceptron is a method for pattern recognition, which was first created by Rosenblatt in 1958 [2, 3]. It was the first artificial neural network model, laying the foundation for the neural network model. However, in Minsky Papert's analysis of Rosenbatt's single-layer perceptron from a mathematical perspective [4], the artificial neural network was criticized with an example of the XOR operation. The problem of how an intelligent system independently learns from an environment is not well solved, and the development of artificial neural networks (ANNs) has deteriorated. In the mid-1980s, scholars began to explore the inner logic of knowledge discovery in depth and discovered that inductive logic, especially incomplete induction logic, is a reasonable way to discover knowledge. Rumelhart et al. surprisingly discovered that the back-propagation error (BP) [5], which was invented by Werbos more than 10 years ago, can effectively solve the learning problems of hidden nodes in multi-layer networks. It is not correct to accept MillsIcy's assertion that there may be no effective learning methods for multi-layer networks. Since then, people's enthusiasm for ANN research has been rekindled.

However, researchers have argued that the use of McCulloch and Pitts's neuron is inadvisable because it disregards the dendritic structure in a real biology neuron. Koch et al. [6, 7] proposed that the interaction between synapses and the action at the turning point of a branch can be approximated as logic operation. In recent years, several dendritic computing models considering the functions of dendrites in a neuron have been proposed in the literature. A dendritic morphological neural network (DMNN) which is based on the traditional morphological neural networks [8, 9] is proposed for solving classification problems [10] and 3D object recognition tasks [11]. A nonlinear dendritic neuron model equipped with binary synapses [11] is demonstrated to be capable of learning temporal features of spike input patterns.

Most recently, a dendritic neuron model (DNM) with nonlinear synapses has been proposed [12–14]. Different from DMNN, DNM only considers a single neuron rather than the network of a couple of neurons, and has shown great information processing capacity [15–19]. The DNM uses a pruning technique derived from an interesting biological phenomenon: in the early stages of neuron triggering, the selective removal of unnecessary synapses and dendrites does not cause neuron cell death [20, 21]. The DNM subtly solves nonlinear problems that cannot be well handled by the Koch model [22,23]. The DNM has four layers in its structure. The input signal is triggered in the synaptic layer and then sequentially received by the dendritic layer. The membrane layer collects the output from each branch of the dendritic layer and sends the results to the somatic cell layer. By the pruning function of the DNM, the precise dendritic structure and morphology are simplified. After training, all mature neurons are approximately replaced by a logic circuit that consists of comparators, AND gates, OR gates and NOT gates.

In this study, we use a dendritic neuron model with adaptive synapses (DMAS). Recent advances in neurobiology have highlighted the importance of dendritic calculation. In 2019, Beaulieu-Laroche and his team [24] discovered that dendrites are always active when the cell body of a neuron is active, which implies that the dendritic synapse has a role in the neural computing process. Based on this biophysical hypothesis, we develop a synaptic adaptable neuron network without parameters that need to be artificially adjusted. All synaptic layer parameters will be trained by the learning algorithm. The effectiveness of adaptive synapses will be proved in section 4.3. Thus, we have to consider additional aspects in the choice of learning algorithms.

With the emergence of various new optimization algorithms, how to train an ANN has been discussed [25]. BP is very effective as an ANN training algorithm and can solve some nonlinear problems [5]. However, BP has certain limitations, for example, falling into a local minimum is easy, the convergence speed is slow, and it is prone to overfitting [26]. Differential evolution (DE) has been employed to train DMAS in our research. DE was first proposed by Storn and Price in 1997 [27]. It is a biological-inspired, population-based global optimization algorithm. Due to its simple concept, easy implementation, fast convergence and excellent robustness, it has been more extensively utilized than other mainstream evolutionary algorithms, such as the genetic algorithm (GA) [28, 29], the evolutionary strategy (ES) [30, 31], and particle swarm optimization (PSO) [32] in recent years. DE is similar to the GA and ES but differs from them because a unique differential evolution operator is referenced in DE. DE has proven to be superior to many algorithms [33–35]. Because of these characteristics and the advantages of DE, it has been recognized by scholars in the field of ANNs [36, 37]. Also DE has been applied in dendrite morphological neural networks [38].

Five realistic classifications problems are considered in our research to validate our model (DE-DMAS): Iris, BUPA liver disorders, breast cancer, glass, and Australian credit approval (ACA). All the datasets is preprocessed as binary-classification problem. These five dataset have undergone preprocessing, including outlier repair to fill in missing values. We compare the experimental results of DE-DMAS, BP-DNM and BPNN for these five datasets. Experimental results show that DMAS outperforms its peers in terms of test accuracy, sensitivity, specificity, receiver operating characteristic (ROC) and cross-validation.

Chapter 2

Backgrounds

The backgrounds consist of two parts. To be specific, the first part introduces the artificial neural network (ANN), and the other part introduces the algorithm.

2.1 Artificial Neural Network

Artificial neural network is defined as a mathematical calculation model that simulates the structure and function of the central nervous system of animals, particularly the brain in machine learning and cognitive science. Many artificial neurons constitute a neural network to estimate or approximate functions. The internal structure of the artificial neural network can be modified according to the external information, thereby establishing an adaptive system and exhibiting the ability of learning. For the allocation of mathematical statistics, the learning method based on mathematical statistics types should optimize the neural network. Considerable available functions can express the local structure space with standard mathematical statistics method. The application of artificial intelligence in artificial identification can be achieved with statistical methods, i.e., mathematical statistics application program. The artificial neural network exhibits simple judgment and decision-making power, which is consistent with human beings. The merit of this method is significantly more than that of formal logic. Besides, the neural network is also adopted to solve different issues (e.g., machine vision and speech recognition), similar to other machine learning methods. The conventional rule-based programming cannot solve such issues.

2.1.1 Background

The idea of the artificial neural network is inspired from the observation of the human central nervous system. The artificial neural network is a network structure similar to a biological neural network composed of simple artificial nodes connected. The mentioned artificial nodes are called neurons. Thus far, there is no unified definition of artificial neural networks. However, it is termed as neurotic for the statistical models with the features as below. First, there is a group of adjustable weights of numerical parameters adjusted by the learning algorithm. Second, it is adopted to estimate the nonlinear function relationship of input data. The strength between neurons is determined by the mentioned adjustable weights. Artificial neural networks are consistent with biological neural networks as they can calculate each part of the function while describing the specific task of each unit. The neural network can be considered a model employed in statistics, cognitive psychology artificial intelligence, and other fields, while the orifical neuron science and computational neuros ciceronian neural networks that control the central nervous system. In the Mgtr software implementation of a neural network, the bio heuristic method has been replaced with a more practical method based on statistics and signal processing in the modern software application of the neural network. Some large-scale software systems consist of a part of the neural network (e.g., artificial neurons). In these systems, adaptive and nonadaptive elements are included. Though the method applied in this system is more suitable for solving practical problems, it is not associated with the traditional connectionist AI. However, they have some similarities in distributed, nonlinear local computing, as well as parallel and adaptability. In the late 1980s, the application sign of neural network model had shifted from highly symbolic artificial intelligence to low symbolic machine learning (from the expert system expressed by condition rules to the power system parameters).

2.1.2 History

A computational model of neural networks based on Mathematics and an algorithm termed as threshold logie was built by Warren Meculloch and Walter Pitts(1943) [1]. They are two different methods in the research of neural networks, i.e., the biological process in the brain and the neuron.

2.1.3 Hebb Learning

Hebb learning, a hypothesis of learning theory based on the mechanism of neuroplasticity, was proposed by psychologist Donald Hebb in the late 1940s. Herb learning is considered a typical unsupervised learning rule, and its late variant refers to the early model of long-term reinforcement. The wring machine was adopted to study this computational model from 1948. MIT Herb Network was simulated by Farley and Wesley A.Clark(1954) [39]]in computers, then termed as calculators. The behavior of abstract neural networks was simulated on an IBM 704 computer by Nathaniel Rocester (1956) [2] et al. The perception machine was created by Frank Rosenblatt [3]. This algorithm can exploit a simple addition and subtraction method to achieve the pattern recognition of the two-layer computer learning network. The circuits not available in the basic perceptron (e.g., XOR circuits) are expressed by Rosenblatt with mathematical symbols. Such loop was not processed by the neural network until Paul Webers (1975) developed a backpropagation algorithm. Two vital problems of neural networks were identified in Marvin Minsky and SIMORE Piper (1969). After the study on machine learning was published, neural network research was suspended. The two problems indicate that the basic perceptron cannot deal with mutual exclusion or circuit, and the computer is not sufficient to deal with the long time computation required by a large-scale neural network. The study on neural network develops slowly when the computer has insufficient computing power. Frank Rosenblatt created the perception machine, i.e., a pattern recognition algorithm, exploiting simple addition and subtraction to realize two-tier computer learning network. Rosenblatt also adopted mathematical symbols to express circuits that are not available in the basic perceptron (e.g., XOR circuits). Such loops cannot be processed by neural networks until Paul Webers (1975) created an back-propagation algorithm.

After Marvin Minsky and Simore Piper (1969) published a study on machine learning, the study on neural networks was stagnated. They identified two key problems in neural networks. First, the basic perceptron cannot handle exclusive or circuits. Second, computers have insufficient capacity to handle the long computational time required by large neural networks. Until computers have more computing power, the research progress of neural networks is hindered.

2.1.4 Progress After 2006

Since 2006, CMOS has been adopted as a computing device for biophysical simulation and neuron morphological computation. In the latest research , it is reported that nanodevices employed in large-scale principal component analysis and convolution neural networks show promising application prospects. If the research is successful, a novel neural computing device based on learning rather than programming will be developed, and it is basically not digital, though in first example may be digital CMOS devices. Jurgen schmidubers research team developed the recurrent neural network and deep feed forward neural network at Swiss AI Lab IDSIA from 2009 d 2012 and awarded eight international competitions on pattern recognition and machine learning. For instance, the two-way and multi-dimensional LSTM of Alex graves and others won the championship in three ICDAR hyphenate recognition competitions in 2009, whereas they did not know the three languages to learn before. Numerous pattern recognition competitions have been won by Dan Ciresan of IDSIA and his colleagues (e.g., IJCNN 2011 Trac Sign Recognition Competition and others). Moreover, IJCNN 2012 trac sign recognition and NYUS Yann Lecun handwritten numeral Problem are the first artificial pattern recognizer to reach or exceed the human level vital benchmark tests. In 1980, Kunihiko Fukushimal invented deep highly nonlinear neural structures like Neocognitron and visual standard structure (inspired by simple and complex cells that were identified in the primary visual cortex by David H, Hubel and Torsten Wiesel), which can be trained with unsupervised learning methods in Je Sinton Laboratory of the University of Toronto.

2.1.5 Categorization

Typical artificial neural networks have three parts:

1. **Architecture Structure:**

   It elucidates the variables in the network and their topological relationships. For instance, variables in a neural network can be weights of neuron connections and activities of the neurons.

2. **Architecture Rule:**

Most neural network models comply with a short time scale dynamic rule to define how neurons change their excitation values in accordance with the activities of other neurons. The general excitation function is determined by the weight of the network (i.e., the parameters of the network).

3. **Learning Rule:**

Learning rules specify how the weights in the network adjust over time. This is generally considered a long-time-scale dynamic rule. On the whole, learning rules depend on the excitation value of neurons. Moreover, it is likely to be determined by the target value provided by the supervisor and the value of the current weight. For instance, a neural network for handwriting recognition has a set of input neurons. The input neuron is activated by the data of the input image. After the excitation values are weighted and passed through a function (determined by the designer of the network), the excitation values of the mentioned neurons are transmitted to other neurons. The mentioned process is repeated till the output neurons are activated. Lastly, the excitation value of the output neuron determines the recognized letter.

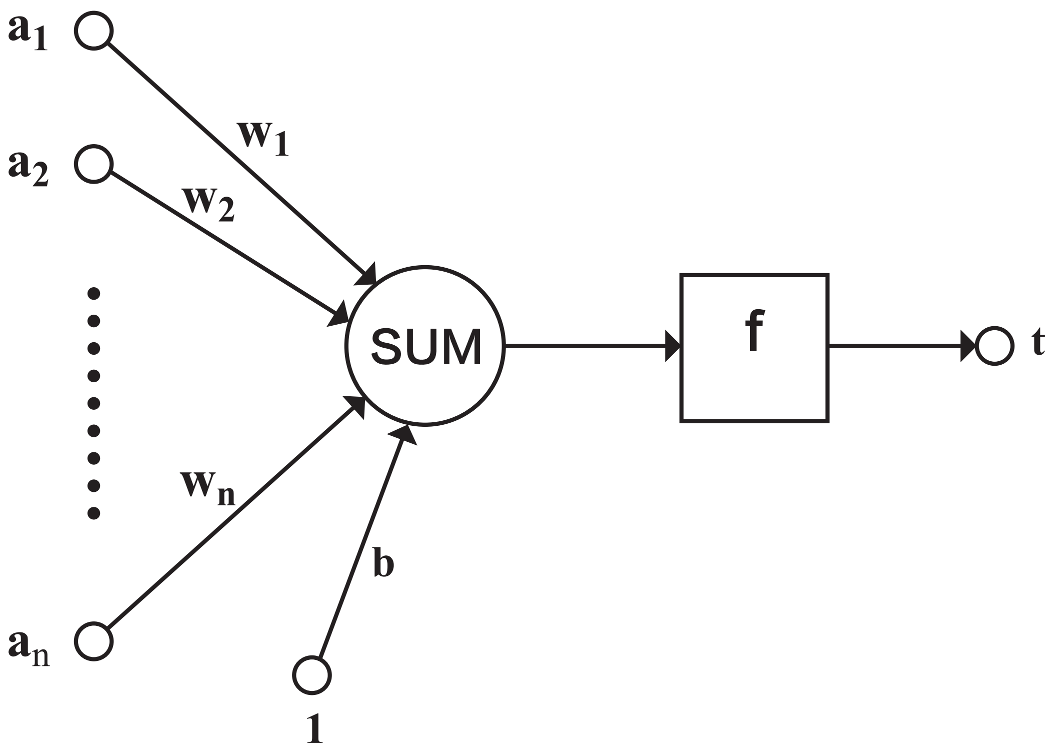

Figure 2.1: Neuron schematic diagram showsl.

2.1.6 Neurons

Neuron schematic diagram is illustrated in Fig.2.1

· $a_1 - a_n$ is the component of the input vector.

· $w_1 - w_n$ is the weight of each synapse in neurons.

· b is biased.

· f denotes a transfer function, usually a non-linear function. On the whole, there are trained (), Tansig (), hardlim (). The following default is hardlim ().

·t denotes the output of neurons.

Mathematical expression: $T = f(\overrightarrow{W'}\overrightarrow{A} + b)$

· $\overrightarrow{W}$ is the weight vector, $\overrightarrow{W'}$ is transposed by $\overrightarrow{W}$

· A is input vectors

· b is biased.

· f is the transfer function.

It can be suggested that the function of a neuron is to obtain the inner product of the input vector and the weight vector, and then obtain a scalar result is obtained a non-linear transfer function.

The function of a single neuron: divide an n-dimensional vector space is divided into two parts with a hyperplane (called judging boundary). Given an input vector, the neuron is capable of determining which side of the hyperplane the vector is located on.

The equation of the hyperplane: $\overrightarrow{W'}\overrightarrow{p} + b = 0$

· $\overrightarrow{W}$ is the weight vector.

$blackdsquare$ b is biased.

· $\overrightarrow{W}$ denotes the vectors on hyperplanes

2.1.7 Single Layer Neural Network

It is the critical form of neuron network, consisting of a limited number of neurons. The input vectors of all neurons refer to the identical vectors. Since each neuron generates a scalar result, the output of a single neuron is a vector whose dimension is equated with the number of neurons. As Fig.2.2 shown

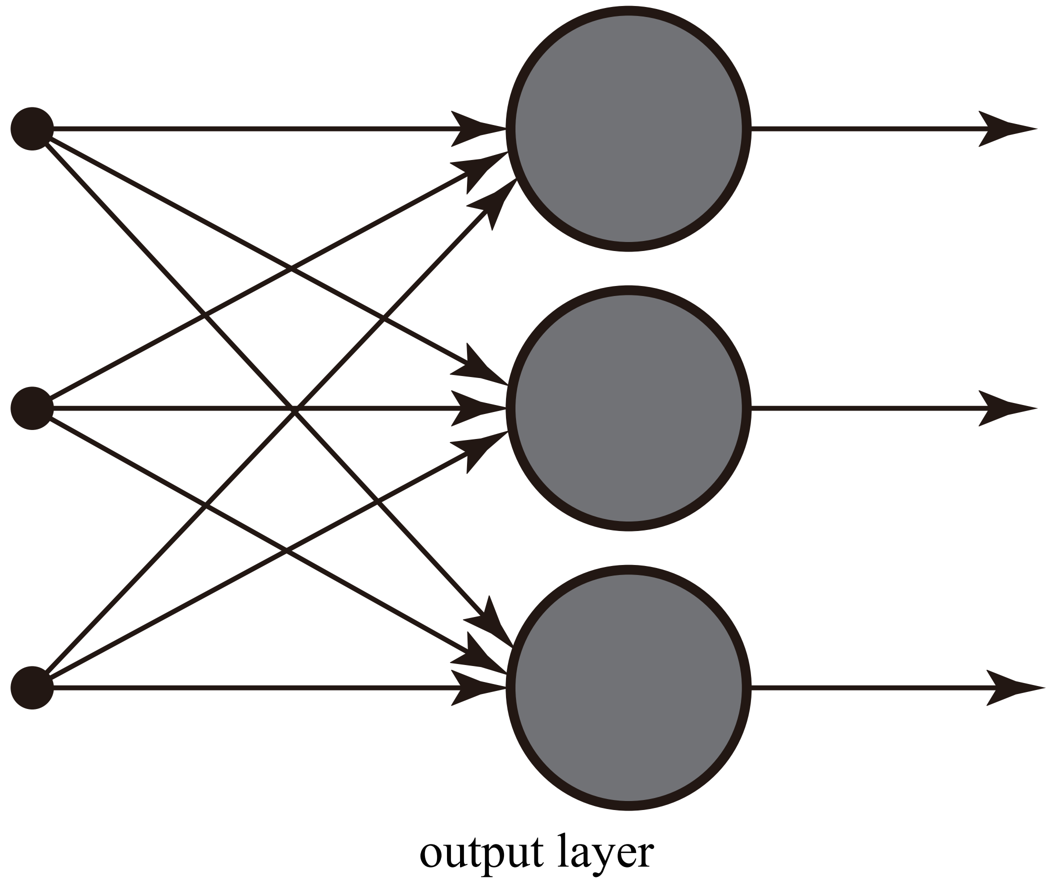

Figure 2.2: Single layer neural network.

2.1.8 Multilayer Neural Network

The practical of artificial neural network

Artificial neural network refers to a system capable of can learning and summarizing, it can learn and summarize by experimentally applying known data. Artificial neural network is able to infer and generate an automatic recognition system by comparing the local conditions (determined by the automatic learning in different scenarios and the complexity of the actual problem to be solved). Different learning methods based on symbol system also exhibit reasoning function, whereas they are built based on logical algorithm, the reason why they can reason is that they require a set of reasoning algorithms.

Artificial Neural Network Model

Overall, an artificial neural network consists of a multilayer neuron structure; each layer of neurons has input (with its input as the output of the previous layer of neurons) and output; each layer (recorded with symbols) (i) is composed of N_i (N_i represents N on the i layer) network neurons, each $N_i$network neuron outputs the corresponding N_i neurons. For its input, the biological names the connection between neurons and their corresponding neuronss are adopted, termed as synapse. In the mathematical model, respective synapse has a weighted value, which we call weight. Subsequently it should be calculated that the potential energy of a neuron in layer i is equaled with each weight multiplied by the output of the corresponding neuron in layer $i-1$. The potential energy obtained by a neuron in layer i is overall summated. Subsequently, the value of the potential energy can be regulated by the activation function (with sigmoid function used commonly) on the neuron, since it can be differentiated and continuous, which is convenient for the Delta rule to seek the output of the neuron. It is noteworthy that the output is a non-linear value.
In other words, the value calculated by the excitation function determines whether to activate the neuron according to the limit value.

Furthermore, we are not interested in whether the output of a neural network is linear.

Basic Structure

A common multilayer feedforward network consists of three parts:

Input Layer: In the input layer, numerous neurons receive considerable non-linear input messages. The input message is termed as the input vector.

Ouput Layer: In the output layer, messages are transmitted, analyzed and weighed in the neural link to generate output results. Output messages are termed as output vectors.

Hidden Layer: Hidden layer covers of considerable neurons and links between input and output layers. A hidden layer can have one or more layers. The number of nodes (neurons) in the hidden layer is unclear, whereas the more the number, the more significant the nonlinearity of the neural network will be. Thus, the robustness of the neural network (the control system maintains some performance characteristics under certain structure, size and other parameter perturbations) is of higher significance. It is customary to select a node 1.2 to 1.5 times the input node. Overall, this type of network is termed as perceptron (for single hidden layer) or multi-layer perceptron (for multi hidden layer). There exist a wide variety of neural network, and this hierarchical structure does not apply to all neural networks.

2.1.9 Learning Process

The process of creating a model by correcting the training samples and learning the weights of respective layer is termed as the automatic learning process. The specific learning methods are different for the different network structure and model. The back-propagation algorithm (back propagation / reverse transfer / reverse propagation, with output by complying with the first differential delta rule to modify the weight) is commonly adopted to verify.

2.1.10 Category

Artificial neural networks fall into the two types below

1. According to the learning strategy, the classification mainly includes:

(1)Mainly supervised learning network.

(2)Unsupervised learning network.

(3)Hybrid learning network.

(4)Associate learning network.

(5)Optimization application network.

2. According to Connectionism, the main categories are:

(1)Feed forward network

(2)Recurrent Network

(3)Reinforcement Network

2.1.11 Theoretical Nature

Computing power

Multilayer Perceptron (MLP) refers to a general function approximator, as proved by Cybenko theorem. However, the proof is not determined by a specific number or weight of neurons. As demonstrated by the work of Hava Siegelmann and Eduardo D. Sontag, a special recursive structure with rational weight (corresponding to the full precision real weight value) is equivalent to a universal Turing machine, consisting of a limited number of neurons and a standard linear relationship. They further suggested that using the weight value of irrational numbers can produce a hyper machine.

Capacity

ANN models have an attribute termed as capacitance, approximately equivalent to their ability to remember (rather than classify) input data correctly. It is related to the parameters and structure of the network. Google used tagging in research to test whether the model can remember all the output. Though the performance of the model on the test set is almost random guess, the model can remember all the input data of the training set, i.e., the label after they are disturbed. With limited sample information in mind, a lower limit is imposed on the number of parameters (weights) required for the model.

Convergence

The model does not always converge to the only solution since it is determined by some factors. First, considerable local minima may exist in the function, relying on the cost function and the model. Second, when the majorization method is far away from the local minimum, the convergence may not be ensured. Third, for a large number of data or parameters, some methods become unrealistic. Generally, we find that convergence ensured by theory cannot be a reliable guide for practical application.

2.2 Algorithm

Algorithms, in mathematics (computer science) and computer science, are the specific computational steps for any series of well-defined [40], commonly employed in computation, data processing, and automatic reasoning. As an effective method, the algorithm is adopted to calculate the function [41], covering a series of well-defined instructions, which can be clearly expressed in a limited time and space.

The instructions in the algorithm present a calculation that, when running, can start from an initial state and initial input (possibly empty), via a series of finite and clearly defined states which ultimately generate an output. Besides, they stop at a final state. The transition from one state to another is not necessarily deterministic. Some algorithms (e.g., randomization algorithms) contain some random inputs.

The concept of formalized algorithms partially stems from the attempts to solve the decision problem proposed by Hilbert and then attempts to define effective computability or effective method. The mentioned attempts consist of the recursive functions proposed by Kurt G¨odel, Jacques Elbron, and Stephen Cole Klein in 1930, 1934, and 1935, respectively, as proposed by Alonzo Church in 1936. λ calculus, 1936 Emile Leon Post's Formulation 1 and Alan Turing's Turing machine proposed in 1937. Even currently, there are common cases where intuitive ideas are difficult to define as formal algorithms. [42]

2.2.1 Feature

1.Input: An algorithm should have zero or more inputs.

2.Output: An algorithm should generate one or more outputs, and the output refers to the result of the algorithm calculation.

3.Clarity: The description of the algorithm should be unambiguous to ensure that the actual execution of the algorithm is precisely by complying with the requirements or expectations, and the practical operational results are usually determined.

4.Finiteness:According to Turing's definition, an algorithm refers to a series of operations that can be simulated with any Turing complete system, while the Turing machine has only a limited number of states, a finite number of input symbols, as well as a finite number of transfer functions (instructions). Several definitions indicate that the algorithm should complete the task in a limited number of steps.

5.Effectiveness: Also known as feasibility. It can be suggested that the operations described in the algorithm can be implemented with a limited number of executions of the basic operations that have been implemented.

2.2.2 Common Design Pattern

Full traversal method and incomplete traversal method: When the solution to the problem refers to a finite discrete solution space, and the correctness and optimality can be verified, the simplest algorithm is to completely traverse all the elements of the solution space and to verify whether the elements are in order. The ideal solution is the most straightforward algorithm. Besides, the implementation is commonly the easiest. However, when the solution space is particularly large, this algorithm can cause an unacceptable amount of computation in engineering. Then, incomplete traversal methods (e.g., various search methods and planning methods) can be adopted to reduce the amount of computation.

1. Divide and conquer: a problem is divided into ideas that are solved separately from each other. One of the benefits of this solution refers to the ease of parallel computing.

2. Dynamic programming method:A method used commonly if the overall optimal solution of a problem covers local optimal solutions.

3. Greedy algorithm: a common approximation solution. A method available when the overall optimal solution of the problem (or cannot be proved to be) covers no local optimal solution, and no requirement is raised for the optimality of the solution.

4. Degenerate method: A method with a problem simplified as a relatively simple or similar, relatively simple model by logical or mathematical reasoning.

2.2.3 Common Implementation Methods

1. Recursive method and iterative method

2. Sequential, Parallel, and Distributed Computing: Sequential computing refers to the sequential execution of a formalized algorithm under a single-threaded serialization of a programming language.

3. Deterministic and non-deterministic algorithms.

4. Exact solution and approximate solution.

2.2.4 Formal Algorithm

An algorithm refers to the essence of a computer's processing of information, since a computer program is essentially an algorithm that introduces the computer the exact steps to perform a specified task (e.g., calculating the employee's salary or printing a student's transcript). Overall, when the algorithm is processing information, it reads data from the input device or the storage address of the data, as well as writing the result to the output device or a storage address for subsequent recall.

2.2.5 Complexity

1. Time complexity:

The time complexity of the algorithm refers to the time resource that the algorithm should consume. Overall, computer algorithms are problematic n. Under the function $f(n)$, the time complexity of the algorithm is also recorded as:

$T(n) = O(f(n))$

Algorithm execution time growth rate are positively correlated and f (n), termed as asymptotic time complexity, referred to as time complexity.

Common time complexity is:

(1). constant order: $O(1)$.

(2). Logarithmic order $O(log(n))$.

(3). Linear order $O(n)$.

(4). Linear logarithmic order $On(log(n))$.

(5). Square order$O(n^2)$.

(6). k secondary order $O(n^k)$.

(7). Index step $O(2^n)$.

As the scale of the problem n as the number of times increases, the -mentioned time complexity increases, and the execution efficiency of the algorithm is lower. The - mentioned time complexity isrising, and the execution efficiency of the algorithm is lower.

2.2.6 Space Complexity

The spatial complexity of the algorithm refers to the space resources that the algorithm should consume. Its calculation and representation methods are consistent with time complexity and overall represented by the asymptotic complexity. Compared with the complexity, the analysis of spatial complexity is significantly simpler.

2.2.7 Implementation

Algorithms can be implemented not only in computer programs, but also in artificial neural networks, circuits, or mechanical devices. The following three algorithms are listed, including global optimization algorithms: genetic algorithm, particle swarm optimization and local optimization algorithms, error back propagation algorithms.

Chapter 3

Back propagation algorithm

3.1 Generalize

Back propagation algorithm (BP algorithm) consists of two stages, incentive propagation and weight updating.

Stage 1: incentive communication

The propagation phase in each iteration consists of two steps: (forward propagation stage) the training input is sent to the network to generate the incentive response;

In the stage of back propagation, the target output corresponding to the training input is calculated to obtain the response error of the output layer and the hidden layer.

Stage 2: weight update

For the weights on each synapse, update as follows:

The gradient of the weight is determined by multiplying the input excitation and the response error.

Multiply the gradient by a proportion and reverse it to the weight.

This proportion (percentage) will affect the speed and effect of the training process, so it becomes a "training factor". The direction of gradient refers to that of error expansion, so it needs to be reversed when updating the weight, to reduce the error caused by the weight.

The first and second stages can be iterated repeatedly till the response of the network to the input reaches a satisfactory predetermined target range.

3.2 Algorithm

Algorithm 1: There layer network algorithm (only one hidden layer)

```
Initialize network weights (usually small random values)
    do
        For each training sample ex
                Prediction = natural net output (network, $ex$)
                Actual = teacher output $ex$
                Calculate the error of the output unit (prediction - actual)
                Calculate $\Delta w_h$ for all hidden layer to output layer weights
                Calculate $\Delta w_i$ for all input layer to hidden layer weights
                Update network weight
    Until all samples are correctly classified or meet other stop criteria
    Return the network
```

The wrong algorithm name refers to the propagation process from the output node to the input node. To be exact, when the network weight can be modified, the algorithm of the network error gradient is the back-propagation algorithm. Generally, the word "back propagation" is employed in a more general sense to refer to the whole process of calculating gradient and using it in random gradient descent method. In the network suitable for back propagation algorithm, it can converge to a satisfactory minimum value efficiently.

BP networks are multi-layer perceptrons (commonly an input layer, a hidden layer, and an output layer). For all useful functions to be applicable to the hidden layer, a multilayer network is activated with a nonlinear activation function: a multilayer network with only a linear activation function is equivalent to a single-layer linear network.

3.3 Derivation

Since the gradient descent method is employed in back propagation, the derivative of the square error function to the network weight should be calculated. Assume that for an output neuron, the square error function satisfies the following equation:

$$E = \frac{1}{2}(t - y)^2, \tag{3.1}$$

where E denotes the square error, t is the target output of the training sample. y is the actual output of the output neuron. Besides,is to coefficient $\frac{1}{2}$ is to counteract the differential exponent. Subsequently, the expression will be multiplied with an arbitrary learning rate, so it is not required to multiply a constant coefficient here.

For respective neuron j, its output o_j is defined as the following equation:

$$o_j = \varphi\left(net_j\right) = \varphi\left(\sum_{i=1}^{n} w_{ji} o_i\right), \tag{3.2}$$

net_j is input to a neuron,it is the weighted sum of the output o_i of the previous neurons. If the neuron is in the first layer behind the input layer, the output o_i of the input layer is expressed as the input x_i of the network. The number of inputs for this neuron refers to n. Variable w_{ji} denotes the neuron i and j weight between. Activation function φ denotes a nonlinear differentiable function generally. Logical functions are commonly act as activation functions:

$$\varphi\left(z\right) = \frac{1}{1+e^{-z}}. \tag{3.3}$$

Its derivative exhibits a good form:

$$\frac{\partial \varphi}{\partial z} = \varphi\left(1-\varphi\right). \tag{3.4}$$

The partial derivative of calculation error to weight w_{ji} is obtained with the chain method twice:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}. \tag{3.5}$$

In the last item on the right, only the weighted sum net_{ji} is determined by w_{ji}, so

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}. \tag{3.6}$$

The derivative of the output of j neuron to its input refer to the partial derivative of the activation function (assuming the use of a logical function here):

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j}\varphi\left(net_j\right) = \varphi\left(net_j\right)\left(1-\varphi\left(net_j\right)\right). \tag{3.7}$$

Thus the reason why the activation function required for back propagation is differentiable is explained. If the neuron is in the output layer,since, $o_j = y$, and

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} frac12\left(t-y\right)^2 = y-t. \tag{3.8}$$

Accordingly the first term can be calculated directly. However if J is any inner layer of the network the derivative of e with respect O_J is hard to find. It is considered as a function of all inputs from neurons j

$$\frac{\partial E\left(o_j\right)}{\partial o_j} = \frac{\partial E\left(net_u, net_v, \cdots, net_w\right)}{\partial o_j}. \tag{3.9}$$

Besides by taking the total differential o_J, a recursive expression of the derivativecan be yielded:

$$\frac{\partial E}{\partial o_j} = \sum_{l\in L}\left(\frac{\partial E}{\partial net_l}\frac{\partial net_l}{\partial o_j}\right) = \sum_{l\in L}\left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l}\frac{\partial net_l}{\partial o_j}\right) = \sum_{l\in L}\left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l}w_{ij}\right). \tag{3.10}$$

For this reason, if all the derivatives of the output o_i about the next layer (the layer closer to the output neuron) are known, the derivatives of o_j can be caclulated and incorporated as:

$$\frac{\partial E}{\partial w_{ji}} = o_i \delta_j, \tag{3.11}$$

Among it:

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \begin{cases} (o_j - t_j)\,\varphi\left(net_j\right)\left(1 - \varphi\left(net_j\right)\right) \\ \left(\sum_{l \in L} \delta_l w_{lj}\right)\varphi\left(net_j\right)\left(1 - \varphi\left(net_j\right)\right), \end{cases} \tag{3.12}$$

To update w_{ji} with gradient descent method, a learning rate α should be selected. The change of weight to be added to the original weight is equeted with the product of learning rate and gradient, multiplied by-1 :

$$\Delta w_{ji} = -\alpha \frac{\partial E}{\partial w_{ji}}. \tag{3.13}$$

The reason for being multiplied by -1 is to update the direction of the minimum value of the error function, rather than the maximum value. For single layer networks, this expression act as delta rule.

3.4 Learning Mode

Three learning modes a re available: online, batch and random. In online and random learning, a weight update is made immediately after respective propagation. In batch learning mode, considerable propagation takes place before weight updating. Online learning is adopted to create a dynamic environment for continuous flow of novel patterns. Both random and batch learning employ a training set of static patterns. Such set of static patterns is used by random learning and batch learning.In order to avoid the situation of situation minimum, random learning adopt the method of random order to pass the data set, which can not only make the weight updated immediately after each propagation, but also make it faster than batch learning.However, because batch learning is given to all types of updates, the local minimum of batch learning is reduced to a more stable level.

3.5 Limit

1. The result is likely to converge to the extreme value. If there is only a minimum, the "Climbing" strategy of gradient descent will certainly operate. However, error surfaces commonly have numerous local minimum and maximum values. If the starting point of gradient descent is exactly between the local maximum value and the local minimum value, the local minimum will be reached following gradient descent maximum.

2. The convergence from back-propagation learning is significantly slow.

3. The convergence of learning in back-propagation cannot be ensured.

4. Adaptive termination conditions can be adopted to ensure a global minimum.

5. Back-propagation learning requires no normalization of input vectors;where as, standardization can enhance performance

Chapter 4

Dendritic Neuron Model with Adaptive Synapses

DMAS is applied in our research. The neuron model includes four layers: the adaptive synaptic layer, dendritic layer, membrane layer, and somatic cell layer. In this section, we detail the structure and principle of these four layers.

4.1 Adaptive Synaptic Layer

The synaptic layer receives and computes the input signal and sends the calculated results to the dendritic layer.Once the input signal breakthrough the threshold, synapese will be fired. To simulate this procress, we design synaptic layer with a sigmoid faction in as the following equation:

$$Y_{i,m} = \frac{1}{1 + e^{-k(w_{im}x_i - q_{im})}}, \tag{4.1}$$

where x_i is the input, and $Y_{i,m}$ is the output of the m-th $(m = 1, 2, , 3, ..., M)$ branch of dendrites. The i in $i = 1, 2, 3...., I$ represents the number of inputs that have been normalized into [0,1] from the dataset. I also represents the number of synapses on each dendrite. k is a tunable parameter which denotes the connection strength between presynaptic and postsynaptic neurons. To reduce the parameters that need to be adjusted in our study, k will be used as the training object. Due to the nature of the sigmoid function, this step has a minimal effect on the function. w_{im} and q_{im} are objects that also need to be trained by the learning algorithm; their values will be set initially within [-2, 2]. Because the synaptic layer works with these three training objects and inputs and no artificial adjustment parameters are needed, this synapse has an adaptive function [43]. The threshold θ_{im} is an important indicator for synapses and is calculated by the following equation.

$$\theta_{im} = \frac{q_{im}}{w_{im}}. \tag{4.2}$$

After the synapse has been activated by the sigmoid function, it can adopt one of 4 different states according to different ranges of w_{im} and q_{im} These states are described as the direct-connecting state (●), opposite-connecting state (▬), constant-1 state (①), and constant-0 state (⓪), as shown in Fig. 4.1. According to the change of the values of w_{im} and q_{im}, the four states are divided into the following six cases:

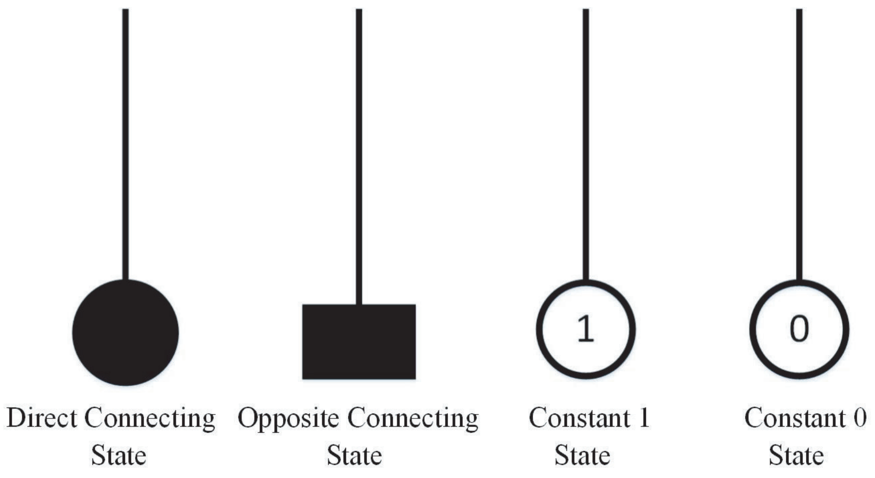

Figure 4.1: The four connecting states.

Case (a): Direct-connecting state, when $w_{im} > q_{im} > 0$. In this state, if the value of the input x_{im} is greater than θ_{im}, the value of the output approximately equals 1; otherwise, it equals 0. For example when $w_{im} = 1.0$ and $q_{im} = 0.5$, the function can be show in Fig. 4.2 (a), where the X-axis represents the value of the input x, and the Y-axis represents the value of the output. Since the range of input is [0, 1], we only need to pay attention to the area between the two dashed lines.

Case (b): Opposite-connecting state, e.g., when $0 > q_{im} > w_{im}$. In this state, if the value of the input x_{im} is less than θ_{im}, the value of the output approximately equals 1; otherwise, it equals 0. A synapse in this state works as a logic NOT operation. For example when $w_{im} = -1.0$ and $q_{im} = -0.5$, the function diagrams as Fig. 4.2 (b).

Case (c1): Constant-1 state when $w_{im} > 0 > q_{im}$. For example when $w_{im} = 1.0$ and $q_{im} = -0.5$, the function diagrams as Fig. 4.2 (c1).

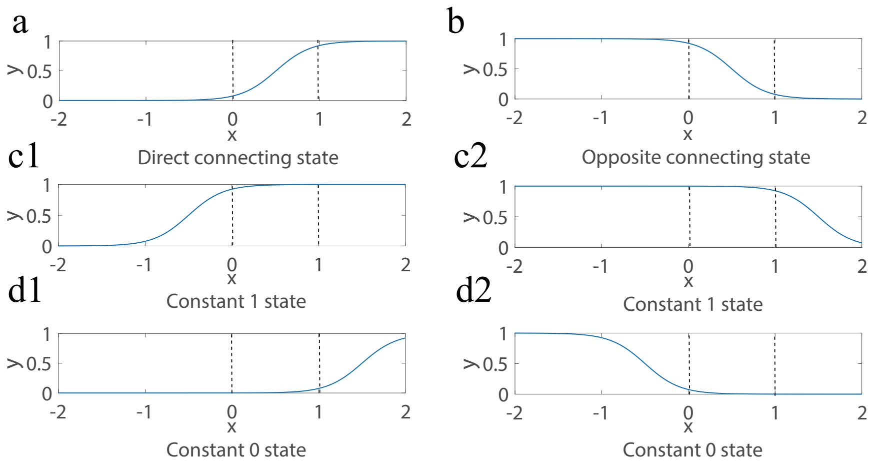

Figure 4.2: The synapse function figures for the four state.

Case (c2): Constant-1 state when $0 > w_{im} > q_{im}$. For example when $w_{im} = -1.0$ and $q_{im} = -1.5$, the function diagrams as Fig. 4.2 (c2). In case (c1) and (c2) regardless of the value of the input $x_{i,m}$, the output remains 1.

Case (d1): Constant-0 state when $0 > q_{im} > w_{im}$. For example when $w_{im} = 1.0$ and $q_{im} = 1.5$, the function diagrams as Fig. 4.2 (d1).

Case (d2): Constant-0 when state $q_{im} > 0 > w_{im}$. For example when $w_{im} = -1.0$ and $q_{im} = 0.5$, the function diagrams as Fig. 4.2 (d2). In case (d1) and (d2), regardless of the value of the input x_{im}, the output remains 0.

4.2 Dendritic Layer

The outputs of the synaptic layer are calculated by the dendritic layer using multiplication. Because the sigmoid function is employed, the outputs are approximately equal to either 1 or 0. The outputs of the dendritic layer are also approximately equal to either 1 or 0. The dendrites work the same as a logic AND operation. The equation is

$$Z_m = \prod_{i=1}^{I} Y_{im}. \tag{4.3}$$

4.3 Membrane Layer

The membrane accepts the output of the dendritic layer as the input and linearly sums the values. The summation can be approximately simulated with logic OR operation [13]s. The equation is

$$V = \sum_{m=1}^{M} Z_m. \tag{4.4}$$

4.4 Somatic Cell Layer

The somatic cell layer will receive the signal from the membrane. The signal is calculated using the sigmoid function as follows:

$$O = \frac{1}{1 + e^{-k_{soma}(V-\theta_{soma}))}}, \tag{4.5}$$

k_{soma} and θ_{soma} are set to 10 and 0.5, which were suggested to be the most promising setting in our previous papers [44] [45].

4.5 Simplified Model

We pruned the synapses and dendrites to obtain our simplified model. The synapse receiving the input signal is activated and converted into the constant-1, constant-0, direct-connecting or opposite-connecting state. The activated signal is transmitted to the dendrites. These signals are multiplied in the dendrites, enter the membrane, and are received by the soma. When a synapse is converted into the constant-1 state, we will remove this synapse since 1 multiplied by any number is equal to the number itself. When a synapse on a dendrite is converted to the constant-0 state, we will remove this dendrite since 0 multiplied by any number equals 0. An example is shown in Fig. 4.3. In the upper left diagram, dendrite (2) has a constant-0 state synapse (c) and an opposite-connecting state synapse (d). Because synapse (c) is in the constant-0 state, dendrite (2) is removed, including the other synapses, as shown in the lower left diagram. We refer to this step as dendrite pruning. In the lower left diagram, a constant-1 state synapse (a) and a direct-connecting state synapse (b) exist on dendrite (1). Since synapse (a) is in the constant-1 state, this synapse is removed. We refer to this step as synapse pruning. The diagram on the right shows the simplified model after pruning, and only dendrite (1) and synapse (b) remain.

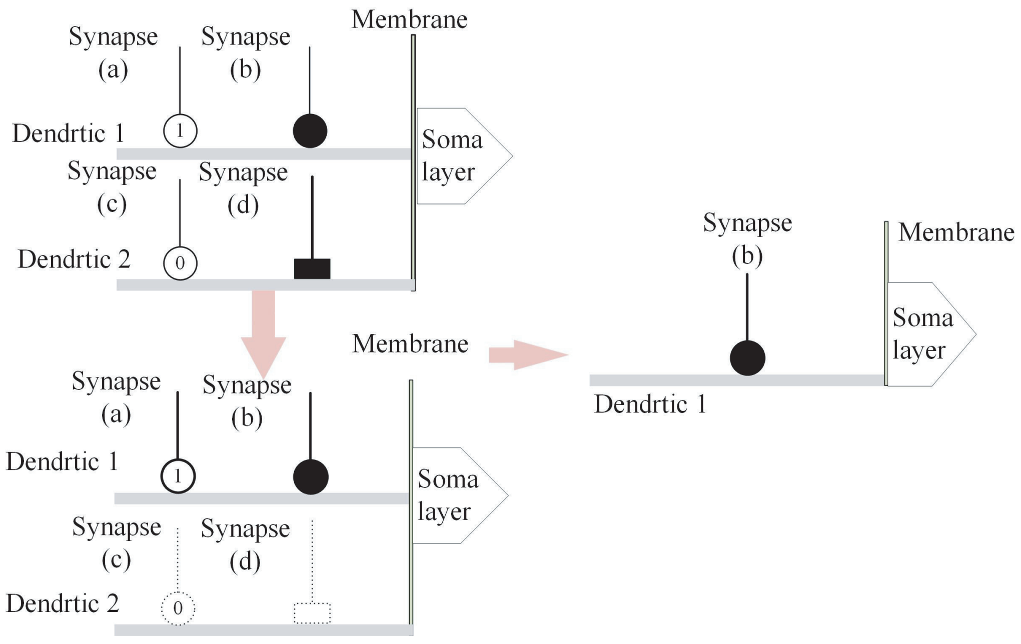

Figure 4.3: Simplified model.

Chapter 5

Learning Algorithm

As previously mentioned, we use the three parameters of the synaptic layer as training objects. This space is a vast search space. We use the DE algorithm as the learning algorithm of DMAS. Since DE is excellent in global optimization [46], it can find the optimal solution faster in the immense search space.

DE demonstrates a fixed number of vectors that are randomly initialized in the search space. The new vectors evolve over time to explore the minimum of the objective function. In the process of evolution, arithmetical operators are combined with the operators of mutation, hybridization and selection. A randomly generated starting population will be evolved to an optimal solution. DE has numerous strategies [47], and we used DE/rand/1/bin in this study. Other strategies include DE/best/1/exp and DE/rand/2/exp, and the preliminary experimental results had suggested that they were slightly inferior than DE/rand/1/bin, because it has the simplest structure [48, 49]. Next, we will explain how DE works.

Step 1: Parameter Setup

Select the population size P and restrict the boundary. Confirm the cross-probability CR, the impact factor F [50] [51], and the termination criterion of the maximum number of generations (G).

Step 2: Initialization of the Population

Set the generation $g = 0$. Initialize a population containing P individuals. The attributes of each individual include weights w, thresholds q and a k, described as D. The number of weights w and thresholds q equals the number of hidden layers (M) multiplied by the number of inputs (I) provided by the dataset. Thus, the population is considered to be a vector matrix of P rows and D ($D = 2 \times M \times I + 1$) columns. Each value of the weights w and thresholds q is initialized as a random real number in the range [-2, 2]. The value of the k is randomly initialized in the range [1, 10].The following equation shows the content of the population.

$$\begin{cases} x_p \to (x_1, x_2, x_3, ..., x_P) \\ x_p = \left\{ w_{(1\times1)p}, q_{(1\times1)p}, ..., w_{(1\times M)p}, q_{(1\times M)p}, ..., w_{(I\times M)p}, q_{(I\times M)p}, k_p \right\}. \end{cases} \tag{5.1}$$

Step 3: Evaluation of the Population

DE can be employed as a training algorithm for DE-DMAS. DE-DMAS can also be regarded as the evaluation function of DE. Therefore, the evaluation becomes a calculation of the mean square error (MSE), which will be formally defined in

Eq.(7.1). In the experiment, the MSE of DMAS as the fitness at each step. Each up-to-date generation will be evaluated after the next mutation, cross and selection operations. In our research, the maximum number of generations is set to 1000. thus, the evaluation function will be run 1001 times.

Step 4: Mutation Operation

The mutation operation produces a mutation operator ($v_{i,g}$). The process of production is shown in the following equation:

$$v_{i,g} = x_{r_1,g} + F(x_{r_2,g} - x_{r_3,g}), \tag{5.2}$$

where $x_{r_1,g}$, $x_{r_2,g}$, and $x_{r_3,g}$ are randomly chosen from the population of this generation. If all individuals are regarded as points in the search space, then the mutation operation can be interpreted as follows: $v_{i,g}$ is a new point after $x_{r_1,g}$ moves in the direction of $x_{r_3,g}$ to $x_{r_2,g}$ by F times the Euclidean distance between $x_{r_2,g}$ and $x_{r_3,g}$.

Step 5: Hybridization Operation

The hybridization operation combines the mutation operator with the target individual, resulting in a new individual. DE involves two methods of hybridization: binomial hybridization and exponential hybridization. Zaharie analyzed the performance of binomial hybridization and exponential hybridization [52] and suggested that exponential hybridization is more affected by population size than binomial hybridization. Binomial hybridization is applied in our research. The following equation shows the hybridization function.

$$u_{j,i,g} = \begin{cases} v_{j,i,g}, \ if \ \ rand_j \leqslant CR \ or \ j = j_{rand} \\ \\ x_{j,i,g}, \ otherwise. \end{cases} \tag{5.3}$$

Generate the random number $rand_j \in [0, 1]$ for each dimension of each individual. If $rand_j$ is less than CR in one dimension, then the target individual $x_{i,g}$ is replaced by the mutation operator $v_{i,g}$ in this dimension; otherwise, it remains the same as the target individual $x_{i,g}$. Before this step, to ensure that the target individual hybridizes in at least one dimension, a random integer $j_{rand} \in \{1, 2, 3, ..., D\}$ is generated. When $j = j_{rand}$, the target individual must hybridize in the j-th dimension.

Step 6: Selection Operation

DE employs the mutation operator and the hybrid operator to generate a son-population and applies a one-to-one selection to compare the son-individuals with the corresponding parent-individuals. The better individuals are saved to the next-generation population. In DE-DMAS, the one-to-one selection operation can be described as follows:

$$x_{i,g+1} = \begin{cases} u_{i,g}, \ if \ \ \mathrm{MSE}_{u_i} \leqslant \mathrm{MSE}_{x_i} \\ \\ x_{i,g}, \ otherwise. \end{cases} \tag{5.4}$$

Since DE employs a one-to-one selection method, the algorithm can ensure that the elitism will not be lost during the evolution process. In addition, one-to-one selection operation has a better ability to maintain population diversity than sequencing or competitive bidding selection [53]. The following Algorithm 2 summarizes the above steps, where two functions *rnd_int* and *rnd_real* return random integer and real numbers in the specified range, respectively.

Algorithm 2: Differential Evolution Algorithm for DMAS.

```
Initial population;
Calculate the fitness of the first generation;
for $g = 1$ $to$ $G$ do
    randomly uniformly select $r_1 \neq r_2 \neq r_3 \neq i$;
    $j_{rand} = rnd\_int(1, D)$;
    for $j = 1$ $to$ $D$ do
        if $rnd\_real_j[0,1) < CR$ $or$ $j == j_{rand}$ then
            $U_{i,g}(j) = X_{r_1,g}(j) + F \times (X_{r_2,g}(j) - X_{r_3,g}(j))$;
        else
            $U_{i,g}(j) = X_{i,g}(j)$;
    for $i$=1 $to$ $P$ do
        Calculate the fitness of the new individual $U_i$;
        if $U_{i,g}$ performs better than $X_{i,g}$ then
            $X_{i,g+1} = U_{i,g}$;
```

Chapter 6

Experimental Design

To achieve the best performance of the proposed method, it is first necessary to confirm the parameters. DE-DMAS has six main parameters. The parameters can be divided into fixed parameters and adjustable parameters. The best adjustable parameters are determined using the Taguchi method for each dataset [54], which is detailed in section 4.2. In section 4.3, we will prove the adaptability of synapses as mentioned above. Finally, DE-DMAS is compared with BP-DNM and BPNN, which are introduced in section 4.4 and section 4.5. The five datasets adopted in our research are introduced in the following subsection.

6.1 Dataset

The five datasets, which are obtained from UCI, are extensively applied in optimization research. The datasets have been standardized by maximum minimization to [0,1] in our research. Their detailed introduction and summary are provided in Table 6.1.

Table 6.1: Dataset introduction.

Dataset Name	No. of Instances	No. of Inputs Attributes	No. of Class 1 Instances	No. of Class 2 InIstances
iris	150	4	50	100
liver	345	6	164	181
cancer	699	9	458	241
glass	214	9	51	163
ACA	690	14	307	383

The iris data was provided by R. A. Fisher in July 1988 [55–57]. The data have three classes: Iris Setosa, Iris Versicolour, and Iris Virginica. Each class has 50 instances. We chose one of the instances as the experimental standard; thus, the data is divided into two categories. The selected 50 instances are divided into one class, and the other 100 instances are divided into another class. Each instance has four attributes: sepal length, sepal width, petal length, and petal width. In our research, we use the class Iris Versicolour as the output, thus, it becomes a binary-classification. Because of the limitations of the single neuron model, the DE-DMAS can only solve binary-classification problem. So we apply the iris dataset as a binary-classification problem.

The liver disorders dataset was provided by Richard S. Forsyth in "None known other than what is shown in the PC/BEAGLE User's Guide". It has been applied in [44, 58]. The dataset has 345 instances. Each instance has six attributes, which include five kinds of blood tests and average daily alcohol consumption. The liver dataset has two classifications: 164 healthy disorders and 181 unhealthy disorders.

The breast cancer data were provided by Dr. William and H. Wolberg in July 1992 [59,60]. It has been applied in [45]. The 699 instances of this data consist of 458 benign instances and 241 malignant instances. Breast cancer data can be divided into two classes. The breast cancer data include 9 attributes, such as clump thickness, uniformity of cell size and shape, and marginal adhesion.

The glass identification database was provided by B. Herman in September 1987. It has been applied in [61]. The glass data include 163 window glass instances and 51 non-window glass instances for a total of 214 instances. The attributes of the glass data include various element contents (Na, Mg, Al, Si, K, Ca, Ba, and Fe) and the refractive index (RI). The instances can be classified by these 9 attributes.

The ACA data indicates whether the applicants are creditworthy. It has been applied [62,63]. The credit history of the applicants classify the data into two classes. These data provide information about 690 applicants. The applicants include 307 people who are creditworthy and 383 people who have no credit. The information that can be considered as the attributes of the ACA dataset consist of 8 categorical records and 6 numerical records.

Table 6.2: Parameter ranges in DE-DMAS.

Dataset	NP	CR	F	M
iris	10, 30, 60	0.3, 0.6, 0.9	0.3, 0.6, 0.9	4, 8, 12
liver	10, 30, 60	0.3, 0.6, 0.9	0.3, 0.6, 0.9	6, 12, 18
cancer	10, 30, 60	0.3, 0.6, 0.9	0.3, 0.6, 0.9	9, 18, 27
glass	10, 30, 60	0.3, 0.6, 0.9	0.3, 0.6, 0.9	9, 18, 27
ACA	10, 30, 60	0.3, 0.6, 0.9	0.3, 0.6, 0.9	16, 32, 48

6.2 Optimal Parameter Settings

Three parameters, F, CR and NP, are mentioned in the DE learning algorithm. The number of hidden layers is an important parameter in DE-DMAS, namely, M. For different datasets, M should be suitably determined. The parameter ranges in DE-DMAS are shown in Table 6.2.

Typically, we need to experiment with all combinations of parameters to obtain the optimal parameters. However, four parameters exist, and each parameter has three choices. Thus, we should perform 81 (3^4) different experiments, which will be time consuming. To ensure the credibility of the experimental results, we should repeat each different experiment 30 times. Because this approach is time consuming, we should reduce the number of different experiments. Taguchi's method is a kind of method to efficiently obtain the optimal parameters [64] [65]. This method is primarily employed using orthogonal arrays. According to the previously mentioned parameter ranges, four parameter trials containing three datasets are available. Thus,

Table 6.3: Orthogonal array for parameters of the iris dataset.

Experimental runs	**Parameters (Levels)**				**Accuracy (%)**
	branch (3)	**F (3)**	**CR (3)**	**NP (3)**	
1	4	0.3	0.3	10	92.51±4.61
2	8	0.3	0.6	30	94.49±5.46
3	12	0.3	0.9	60	94.96±2.47
4	12	0.6	0.3	30	94.20±4.61
5	4	0.6	0.6	10	**96.74±2.78**
6	8	0.6	0.9	60	93.62±5.32
7	8	0.9	0.3	60	94.20±4.47
8	12	0.9	0.6	10	95.50±4.90
9	4	0.9	0.9	30	94.00±2.74

the $L_9(3^4)$ orthogonal array has been applied in the optimal parameter experiments of the five datasets. The instances of each dataset have been divided into 70% for training and 30% for random testing. The orthogonal experiments of each dataset have been repeated 30 times. The epoch of each orthogonal experiment is set to 1000. The orthogonal experimental result of the iris dataset is shown in Table 6.3. The result of the liver dataset is shown in Table 6.4. The result of the glass dataset is shown in Table 6.5. The result of the cancer dataset is shown in Table 6.6. The result of the ACA dataset is shown in Table 6.7. The last column displays the average correct rate of 30 test experiments. We obtain the most optimal parameters by a comprehensive analysis of the mean and variance. The bold font indicates the optimal combination of parameters The optimal parameters for all datasets are shown in Table 6.8.

Table 6.4: Orthogonal array for parameters of the liver dataset.

Experimental runs	**Parameters (Levels)**				**Accuracy (%)**
	branch (3)	**F (3)**	**CR (3)**	**NP (3)**	
1	6	0.3	0.3	10	70.60±4.81
2	12	0.3	0.6	30	**73.59±6.66**
3	18	0.3	0.9	60	72.17±3.86
4	18	0.6	0.3	30	66.92±6.80
5	6	0.6	0.6	10	71.85±4.75
6	12	0.6	0.9	60	68.33±6.91
7	12	0.9	0.3	60	66.28±7.36
8	18	0.9	0.6	10	68.58±7.74
9	6	0.9	0.9	30	69.61±4.88

6.3 Proof Adaptability of Synapses

In order to demonstrate the adaptability of the synaptic layer in DE-DMAS, we carried out a confront analysis. We removed the hyperparameter k from the population of DE and set it to 1, 5, 10, respectively. The five datasets were randomly divided into two parts, 70% for training and 30% for testing. The parameters except k were set as the same as these in Table 6.8.Then we did 30 independent experiments for them. We recorded all test accuracy results and compared the results in terms of mean and standard deviation, as show in Table 6.9. From it, we found that the adaptive k which was learned by DE generally performed better than these fixed values. Additionally, Friedman test [66] gave the statistical analysis results for the accuracies. In this case the lower value of Friedman test, the better performance. The result of Friedman test shows in Table 6.10. Based on the above results, it is

Table 6.5: Orthogonal array for parameters of the glass dataset.

Experimental runs	**Parameters (Levels)**				**Accuracy (%)**
	branch (3)	**F (3)**	**CR (3)**	**NP (3)**	
1	9	0.3	0.3	10	93.75±2.65
2	18	0.3	0.6	30	94.37±4.13
3	27	0.3	0.9	60	**94.42±2.51**
4	27	0.6	0.3	30	94.06±4.44
5	9	0.6	0.6	10	93.12±4.29
6	18	0.6	0.9	60	93.02±4.07
7	18	0.9	0.3	60	93.54±3.66
8	27	0.9	0.6	10	92.81±4.03
9	9	0.9	0.9	30	94.01±2.66

evident that the adaptive synapse is beneficial for DNM.

6.4 Comparison with BPNN

In this section, we compared DE-DMAS with the most popular model BPNN. To make ne comparison relatively fair, the number of adjusted weights and thresholds (D_{BPNN} and $D_{DE-OMAS}$ shown in Eq.6.1 and Eq.6.2, respectively) in both models should be arranged nearly the same because these numbers generally determine the size of the model and the computational complexity although the two models have different architectures.

$$\begin{aligned} D_{BPNN} = (Input \times Hidden) + (Hidden \times Output) \\ +Hiddenbias + Outputbias \end{aligned} \tag{6.1}$$

$$D_{DE-DMAS} = (Input \times Hidden) + 1 \tag{6.2}$$

Table 6.6: Orthogonal array for parameters of the cancer dataset.

Experimental runs	**Parameters (Levels)**				**Accuracy (%)**
	branch (3)	**F (3)**	**CR (3)**	**NP (3)**	
1	9	0.3	0.3	10	**96.12±1.17**
2	18	0.3	0.6	30	96.12±1.65
3	27	0.3	0.9	60	95.80±1.26
4	27	0.6	0.3	30	95.39±1.97
5	9	0.6	0.6	10	96.09±1.64
6	18	0.6	0.9	60	96.19±1.90
7	18	0.9	0.3	60	96.20±2.00
8	27	0.9	0.6	10	95.81±1.55
9	9	0.9	0.9	30	95.84±1.31

However, the larger the number of weights is, the more computing resources are occupied. In our research, to demonstrate the excellent performance of DE-DMAS for five datasets, the structure of DE-DMAS should be set smaller than that of BPNN. Because the input and output of each dataset are fixed, they are given the same number of weights by adjusting their number of hidden layers. In the previous section, we have configured this parameter (the number of hidden layers) for DE-DMAS. We configure the BPNN with the number of hidden layers according to the above principles. The structures of BPNN and DE-DMAS for the five datasets are shown in the following Table 6.11. The learning rate set to be 0.1 according to the experience.

Table 6.7: Orthogonal array for parameters of the ACA dataset.

Experimental runs	**Parameters (Levels)**				**Accuracy (%)**
	branch (3)	**F (3)**	**CR (3)**	**NP (3)**	
1	16	0.3	0.3	10	84.41±5.00
2	32	0.3	0.6	30	85.81±2.41
3	48	0.3	0.9	60	85.28±2.28
4	48	0.6	0.3	30	85.18±1.67
5	16	0.6	0.6	10	85.58±2.14
6	32	0.6	0.9	60	85.70±1.64
7	32	0.9	0.3	60	84.89±2.26
8	48	0.9	0.6	10	84.41±5.73
9	16	0.9	0.9	30	**86.18±1.98**

Table 6.8: Parameters for DE-DMAS

Dataset	**NP**	**CR**	**F**	**M**
iris	0.6	0.6	10	4
liver	0.3	0.6	30	12
cancer	0.3	0.3	10	9
glass	0.3	0.9	60	27
ACA	0.9	0.9	30	16

6.5 Comparison with BP-DNM

In order to compare BP-DNM and DE-DMAS fairly, the three common parameters of k_{soma}, θ_{soma} and No.of hidden layers (M) set to be the same. According to the experience, the learning rate is set to 0.01, and the value of k is set to 3.BP-DNM is also a single neuron model with synaptic nonlinearities. It has been proven to have outstanding performance in the liver [44], cancer [45] and ACA [63]. We will show the performs of BP-DNM when it has the same structure as DE-DMAS. We will use multiple objective methods to demonstrate the performances of DE-DMAS

Table 6.9: Demonstrate on the adaptability of synapse in term of test accuracy

Datase	**k=1**	**k=5**	**k=10**	**Apative**
iris	80.81±6.66	94.15±2.83	95.11±2.50	96.74±2.78
liver	69.39±3.57	69.81±4.12	69.39±5.56	73.59±6.66
cancer	92.76±3.35	92.97±3.38	93.59±2.11	96.12±1.17
glass	91.04±2.84	93.33±3.23	85.48±2.02	94.42±2.51
ACA	85.44±2.99	85.56±1.70	85.86±2.53	86.18±1.98

Table 6.10: Demonstrate on the adaptability of synapse by Friedman test

Datase	**k=1**	**k=5**	**k=10**	**Apative**
iris	4	2.33	2	1.67
liver	2.77	2.73	2.55	1.95
cancer	3.05	2.95	2.73	1.27
glass	2.55	1.98	3.93	1.53
ACA	2.48	2.73	2.52	2.27

and BP-DNM on the five datasets and make a discusstion.

Table 6.11: Structures of DE-DMAS and BPNN for the five dataset

Dataset	**Method**	**No. of inputs**	**No. of branches**	**No. of outputs**	**No. of adjusted weights**
Iris	**DE-DMAS**	4	4	1	33
	BPNN	4	28	1	169
Liver	**DE-DMAS**	6	12	1	145
	BPNN	6	18	1	145
Cancer	**DE-DMAS**	9	9	1	163
	BPNN	9	18	1	199
Glass	**DE-DMAS**	9	27	1	487
	BPNN	27	45	1	496
ACA	**DE-DMAS**	14	16	1	449
	BPNN	14	30	1	481

Chapter 7

Experimental Result Analysis

The comparison experiment of DE-DMAS vs BPNN and DE-DMAS vs BP-DNM is set up as follows: 1) The instances of the five datasets are divided into 30% for testing and 70% for training randomly. 2) The number of iterations is set to 1000. 3) All experiments are run using Matlab 2018a.

7.1 Convergenc Comparison

We use the value of the mean square error (MSE) to represent the degree of convergence. The smaller the value is, the better the convergence. We calculate the value of MSE after each iteration in the DE-DMAS , BP-DNM and BPNN training process and record it. We employ the following equation to calculate the value of the MSE:

$$MSE = \frac{1}{2N}\sum_{i=1}^{N}(O_i - T_i)^2, \tag{7.1}$$

where N represents the number of training instances. O_i and T_i represent the output and the teacher signal of the $i-th$ training instance.

We perform 30 training sessions for DE-DMAS , BP-DNM and BPNN. We randomly select 70% of the instances as the input for each training. We draw two graphs to analyze the convergence effect of DE-DMAS and BP-DNM and BPNN for the five training datasets. In the first figure, the ordinate represents the mean value of the MSE for 30 training sessions, and the abscissa represents the number of iterations. A total of 1000 MSE values are recorded from the start of initialization for the DE-DMAS , BP-DNM and BPNN. We can evaluate the speed of convergence by the degree of the curve drop. In Fig.7.1, we note that curve of DE-DMAS is falling faster than the comparators. These figure shows that DE-DMAS has an advantage in convergence speed. We record the value of MSE in the final iteration for 30 times in the training sessions. We use a box-plot [67] to represent the value of the MSE, as shown in Fig. 7.2. In this figure, the ordinate represents the value of the MSE. The horizontal line from the top to the bottom of each box represents the maximum, 3/4 median, median, 1/4 median and minimum. The 1/4 median, median, and 3/4 median represent the value of MSE at 25%, 50%, and 75%, respectively, after sorting. The + sign represents an outlier, which is a value that exceeds twice the standard deviation. The lines corresponding to the maximum, 3/4 median line, median, 1/4

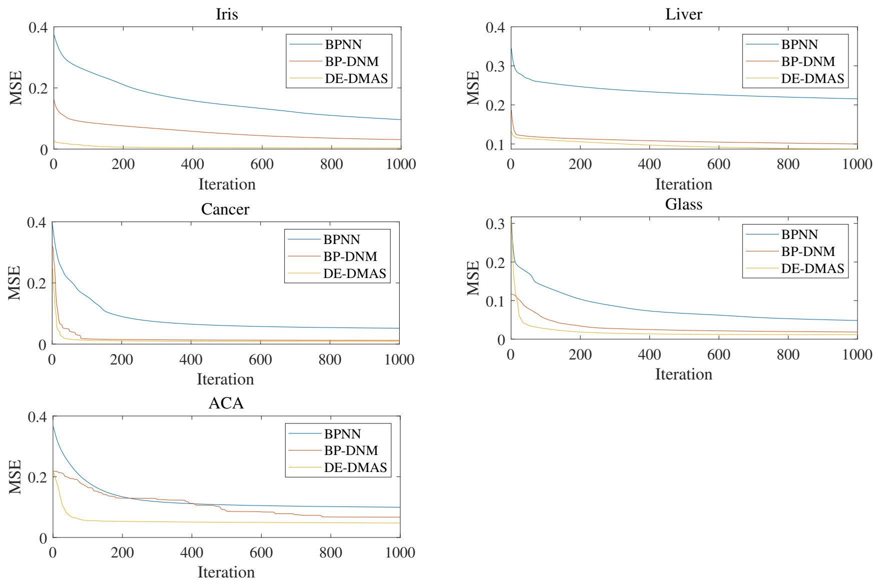

Figure 7.1: Convergence graphs for the five datasets0.

median and minimum for DE-DMAS are below those of BP-DNM and BPNN for the five datasets. Many outliers exist in the boxes of BPNN and BP-DNM. The outliers above the maximum represent falling into a local minimum during training. On the conversely, there is no outlier above the maximum in the boxes of DE-DMAS. The results show that the convergence effect of DE-DMAS is better than that of BP-DNM and BPNN.

7.2 Accuracy Comparison

We compare DE-DMAS , BP-DNM and BPNN in terms of the test accuracy, sensitivity, specificity, and receiver operating characteristic (ROC) curve [68], which is a method for objectively analyzing the performance of classifiers. To draw the ROC curve, we collected the output (O) of the five datasets tested by DE-DMAS ,

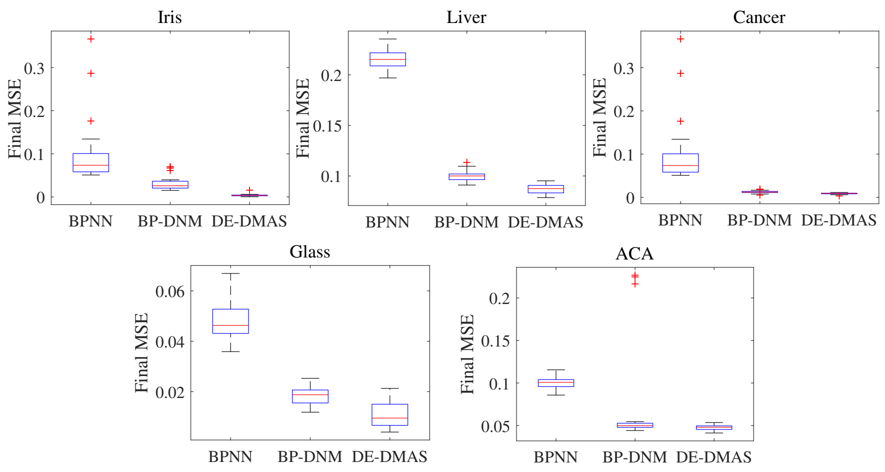

Figure 7.2: Box-plots for the fianl MSE .

BP-DNM and BPNN. (T) represents the corresponding teacher signals We convert the output O from a real number to an integer of 0 or 1. For a two-category problem, the instances are divided into positive and negative classes. The actual classification has four situations [69]:

(1) If an instance is in the positive class and is predicted to be in the positive class, then it is a true classification (true positive (TP)).

(2) If an instance is in the positive class but is predicted to be in the negative class, then it is a false negative classification (false negative (FN)).

(3) If an instance is in the negative class but is predicted to be in the positive class, then it is a false positive classification (false positive (FP)).

(4) If an instance is in the negative class and is predicted to be in the negative class, then it is a true negative classification (true negative (TN)).

The true positive rate (TPR), which represents the proportion of actual positive instances in the positive class predicted by the classifier to all positive instances, equals the sensitivity. The false positive rate (FPR), which represents the proportion of actual negative instances in the positive class predicted by the classifier to all negative instances, equals 1-specificity. The ROC curve is drawn with the FPR (1-specificity) as the x-axis and the TPR (sensitivity) as the y-axis. The AUC is the area under the ROC curve. The value of AUC is between 0.0 and 1.0 since the ROC curve is drawn in an square area. The greater the values of the sensitivity, specificity and AUC are, the better the performance of the classifier. These terms are defined in Table 7.1. We calculate the accuracy, sensitivity, specificity, and AUC based on these terms using the following equation:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP}, \tag{7.2}$$

$$Sensitivity = TP/(TP + FN), \tag{7.3}$$

$$Specificity = TN/(TN + FP), \tag{7.4}$$

Table 7.1: Description of terms.

Teacher signal	Real output		
	Positive (1)	Negative (0)	Row total
Positive (1)	TP	FN	TP + FN
Negative (0)	FP	TN	FP+ TN
Column total	TP + F	FN + TN	N = TP + TN + FP + FN

$$AUC(\%) = \frac{1}{2}(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}) \times 100. \tag{7.5}$$

We plot the ROC curves of the five datasets to compare DE-DMAS with BP-DNM and BPNN , as shown in the Fig.7.3. The DE-DMAS curves are above the BP-DNM and BPNN curves. The sensitivity, specificity, and AUC, which can be determined from the numerical values, are shown in Table 7.2. The test accuracy is the average of 30 experiments, which we represent by the mean and variance in Table 7.2. DE-DMAS exhibits higher values than BP-DNM and BPNN for these four assessment levels. All test results prove the superiority of DE-DMAS.

7.3 Cross-validation

So as to promote the comparison performance, four different train-to-test ratios will be applied and the four multi-fold cross-validation ($K\times$ CV) methods include tenfold CV (90–10%, $\times$ 10), fivefold CV (80–20%, $\times$ 5), fourfold CV (75–25%, $\times$ 4) and twofold CV(50-50%, $\times$ 2). The train-to-test ratio means the ratio between

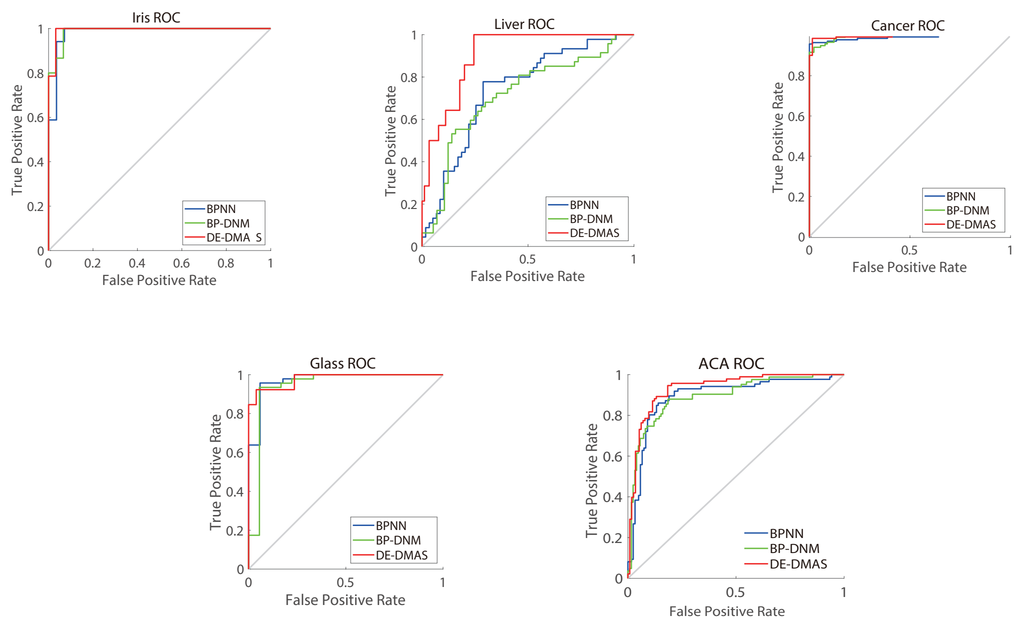

Figure 7.3: ROC analysis for the five datasets .

sample size of training and testing. With $K\times$ CV ($K = 2, 4, 5, 10$), the whole dataset will be divided into K in random and mutually superior subset.The subset roughly equal sample sizes. In K *times* CV, this method will be used to train the subset and measure the test error on the test subset. Duplicate this process for a total of K trials, each time using a different subset for testing. By performing tests on all trajectories of the experiment, the performance of the model is evaluated by the average value of the squared error. Compared with the single-fold validation method, $K \times$ CV has an advantage of minimizing the correlation deviation of random sampling of training samples, but its disadvantage lies in that it may need too much computation since the model has to be trained K times. We select four kinds of BPNN with different learning rate and No.of branch as Table 7.3. BPNN1, BPNN2, BPNN3, BPNN4. BP-DNM

Table 7.2: Rate results for the five datasets

Dataset	**Method**	**Test accuracy**	**Sensitivity**	**Specificity**	**AUC**
Iris	**DE-DMAS**	96.74±2.78	100	96.88	98.44
	BP-DNM	91.78 ±5.87	91.16	86.67	95.65
	BPNN	85.93 ±10.89	90.00	91.43	90.71
Liver	**DE-DMAS**	73.59±6.66	66.67	83.87	75.27
	BP-DNM	68.62±5.24	52.50	.79.69	66.09
	BPNN	59.94±6.33	57.89	72.73	65.31
Cancer	**DE-DMAS**	96.12±1.17	96.45	98.55	97.50
	BP-DNM	96.33±1.43	97.04	94.67	95.85
	BPNN	93.76±11.06	95.89	96.88	96.38
Glass	**DE-DMAS**	94.42±2.51	98.57	92.85	96..67
	BP-DNM	91.87±3.88	84.62	96.08	90.35
	BPNN	92.50±3.34	97.92	81.25	89.58
ACA	**DE-DMAS**	86.18±1.98	88.04	86.09	87.07
	BP-DNM	83.66±9.26	85.54	81.45	83.50
	BPNN	85.57±2.28	87.10	78.07	82.58

and DE-DMAS used the above four types of training-to-test ratios. Five datasets were applied to each type of training-to-test ratios for each model, 30 independent experiments were performed. Finally compare the mean and standard deviation of their test accuracy results. Table 7.4 shows the cross-validation results of iris dataset. Table 7.5 shows the cross-validation results of liver dataset. Table 7.6 shows the cross-validation results of cancer dataset. Table 7.7 shows the cross-validation results of glass dataset. Table 7.8 shows the cross-validation results of ACA dataset. Bold fonts in these tables indicates the top two according to the standard deviation and the mean. DE-DMAS only in the CV5 of the cancer dataset is not bold font. So we can conclude that DE-DMAS is excellent [45].

Table 7.3: No.of Model

No.of BPNN	**Learning rate**	**No.of branch**
BPNN1	0.1	30
BPNN2	0.08	60
BPNN3	0.08	30
BPNN4	0.06	60

Table 7.4: Cross-validation for iris

Model	**CV10**	**CV5**	**CV4**	**CV2**
DE-DMAS	**94.44±5.59**	**94.56±5.90**	**95.00±3.61**	**93.38±5.09**
BP-DNM	93.33±10.79	93.22±6.64	91.93±4.98	86.80±9.16
BPNN1	90.67±7.50	91.42±5.62	89.06±9.41	91.29±7.182
BPNN2	**93.55±10.72**	**93.35±7.92**	**92.80±7.82**	**91.82±7.97**
BPNN3	91.10±9.85	88.77±9.20	90.27±9.89	86.71±13.43
BPNN4	90.44±8.69	89.89±9.07	88.80±9.59	89.07±9.72

7.4 Simplified Model

As previously mentioned, we remove the useless dendrites and synapses by the pruning function. The entire simplification process for the iris dataset is shown in Fig.7.4. First, initialize the structure of neurons with four dendrites, as shown in Fig.7.4 (A). Synapses on these dendrites receive the inputs X_1, X_2, X_3, and X_4. The synapses are activated and converted to the direct-connecting state (●), opposite-connecting state (▬), constant-1 state(①) or constant-0 state (⓪) after learning. Second, remove all useless dendrites by the dendrite pruning function; that is, if at least one synapse on a dendrite is in the constant-0 state, remove the dendrites In Fig.7.4 (B), we denote removed dendrite-1, dendrite-3 and dendrites-4 with the

Table 7.5: Cross-validation for liver

Model	**CV10**	**CV5**	**CV4**	**CV2**
DE-DMAS	**70.20±7.98**	**71.93±4.7**9	**71.40±0.40**	**70.12±3.10**
BP-DNM	58.82±7.93	**69.42±5.68**	**68.88±5.57**	**66.44±4.81**
BPNN1	58.76±9.45	59.18±5.72	59.43±7.34	60.44±4.30
BPNN2	**63.14±8.20**	58.72±6.11	61.47±6.72	58.83±5.05
BPNN3	58.00±8.33	58.55±5.80	61.47±6.72	59.37±8.50
BPNN4	60.19±8.94	61.11±6.09	60.92±8.45	60.71±5.56

Table 7.6: Cross-validation for cancer

Model	**CV10**	**CV5**	**CV4**	**CV2**
DE-DMAS	**96.38±2.36**	96.05±1.74	**96.11±1.12**	**95.90±0.70**
BP-DNM	95.71±2.17	96.02±1.84	95.75±1.64	95.70±1.06
BPNN1	91.48±11.65	92.26±11.80	**96.29±1.67**	94.29±1.08
BPNN2	**96.57±2.51**	**96.28±1.54**	94.33±1.52	93.71±2.44
BPNN3	93.45±12.00	95.52±1.45	95.73±1.35	93.53±11.54
BPNN4	94.05±12.53	**96.30±1.34**	96.10±1.75	**96.07±1.04**

symbol ✖ . After dendrite pruning, only dendrite-2 remains, as shown in Fig.7.4 (C). Then, remove all unnecessary synapses by the synapse pruning function; that is, remove the synapses with the constant-1 state. Fig.7.4 (D) shows the simplified structure for the iris dataset. This structure is simplified from 4 layers of dendrites and 4 inputs to only dendrite-1 and the 2 inputs X_3 and X_4.

Liver dataset has 12 layers of dendrites and 6 inputs shows in Fig.7.5(A). In Fig.7.5 (B), we denote removed (dendrite-1, 3, 4, 5, 6, 7, 8, 9, 10, 12) with the symbol ✖ . After dendrite pruning, dendrite-2, 11 remains, as shown in Fig.7.5 (C). After synapse pruning, the X_1, X_3, X_4, and X_6 have been remianed. Fig.7.5 (D) shows the simplified structure for the liver dataset. This structure is simplified from 12 layers

Table 7.7: Cross-validation for glass

Model	**CV10**	**CV5**	**CV4**	**CV2**
DE-DMAS	**95.24±3.75**	**96.05±1.74**	**96.01±1.36**	**95.70±0.70**
BP-DNM1	**92.53 ±4.95**	92.40±3.23	**91.58±3.16**	91.12±2.77
BPNN1	91.59±5.55	90.39±4.44	91.05±3.19	91.53±2.22
BPNN2	91.90±4.36	**92.56±4.12**	90.99±4.12	90.31±5.24
BPNN3	91.75±4.83	89.61±4.56	91.42±4.28	89.50±3.85
BPNN4	90.00±6.77	92.02±4.08	90.00±5.36	**91.78±3.72**

Table 7.8: Cross-validation for ACA

Model	**CV10**	**CV5**	**CV4**	**CV2**
DE-DMAS	**84.98±3.95**	**86.79±2.70**	**85.59±2.81**	**85.78±1.39**
BP-DNM	83.09 ±8.52	82.00±10.62	83.06±10.51	82.07±10.36
BPNN1	**85.94±4.34**	85.20±2.66	**85.04±2.63**	85.39±1.61
BPNN2	84.12±3.97	85.11±2.60	83.51±3.41	**86.13±1.28**
BPNN3	83.51±4.64	85.48±2.79	84.00±3.49	85.35±1.18
BPNN4	84.16±4.48	**86.11±2.15**	83.51±33.30	85.09±1.55

of dendrites and 6 inputs to only dendrite-2, 11 and the 4 input X_1, X_3, X_4, and X_6.

Cancer dataset has 9 layers of dendrites and 9 inputs shows in Fig.7.6(A). In Fig.7.6 (B), we denote removed (dendrite-1, 2, 3, 4, 6, 7, 9) with the symbol ✖ . After dendrite pruning, dendrite-5, 8 remains, as shown in Fig.7.6 (C). After synapse pruning, the X_1, X_3, X_4, and X_6 have been remianed. Fig.7.6 (D) shows the simplified structure for the liver dataset. This structure is simplified from 12 layers of dendrites and 6 inputs to only dendrite-2, 11 and the 4 input X_1, X_2, X_3, X_5, and X_6.

Glass dataset has 27 layers of dendrites and 9 inputs shows in Fig.7.7(A). In Fig.7.7 (B), we denote removed (dendrite-1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27) with the symbol ✖ . After dendrite pruning, dendrite-6, 24 remains, as shown in Fig.7.7(C). After synapse pruning, the X_1, X_3, X_4, and X_8 have been remianed. Fig.7.7 (D) shows the simplified structure for the liver dataset. This structure is simplified from 12 layers of dendrites and 6 inputs to only dendrite-6, 24 and the 4 input X_1, X_3, X_4, and X_8.

ACA dataset has 16 layers of dendrites and 14 inputs shows in Fig.7.8(A). In Fig.7.8 (B), we denote removed (dendrite-1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 14, 15, 16) with the symbol ✖ . After dendrite pruning, dendrite-10, 13 remains, as shown in Fig.7.8 (C). After synapse pruning, the X_3, X_7, X_8, X_{10}, X_{12} and X_{13} have been remianed. Fig.7.8 (D) shows the simplified structure for the liver dataset. This structure is simplified from 16 layers of dendrites and 6 inputs to only dendrite-10, 13 and the 6 inputs X_3, X_7, X_8, X_{10}, X_{12} and X_{13}.

As shown in these figures, we have obtained the final simplified models of the five datasets. After simplifying the models, the structures of the models have been reduced by more than 90%, which indicates that we can use simpler logic to solve the real problem. The problem used to be solved with more than hundreds of logic components but can now be solved by only a few dozen simple logic components, such as comparators, AND gates, OR gates and NOT gates. This change substantially reduces the labor and time costs. Logic circuits of the five datasets have been drawn in Fig. 7.9. The value of θ in the comparator in the figure is the value of θ in the

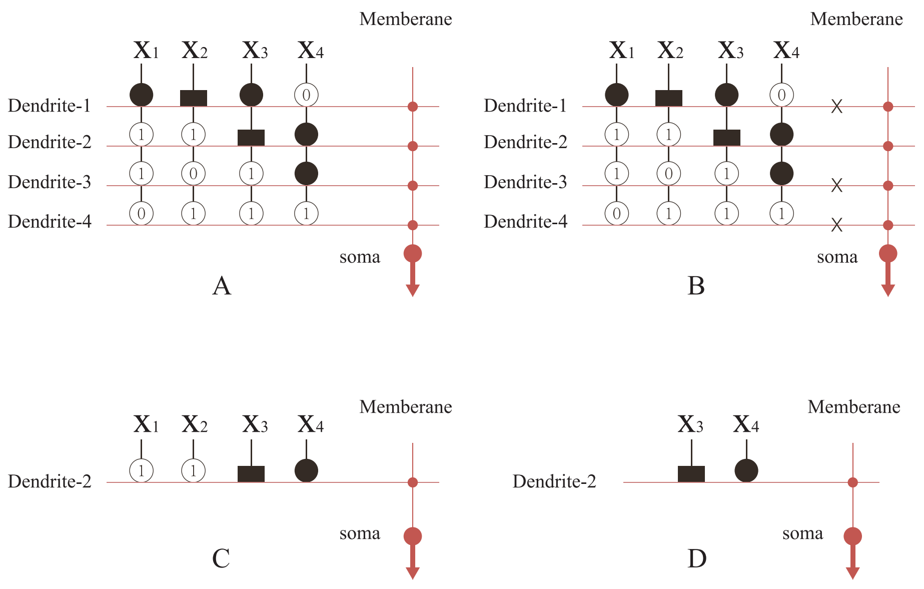

Figure 7.4: Structure simplification process for the iris dataset.

corresponding synapse after training. The standardized input calculated by these logic components to get the expected output as 0 or 1.

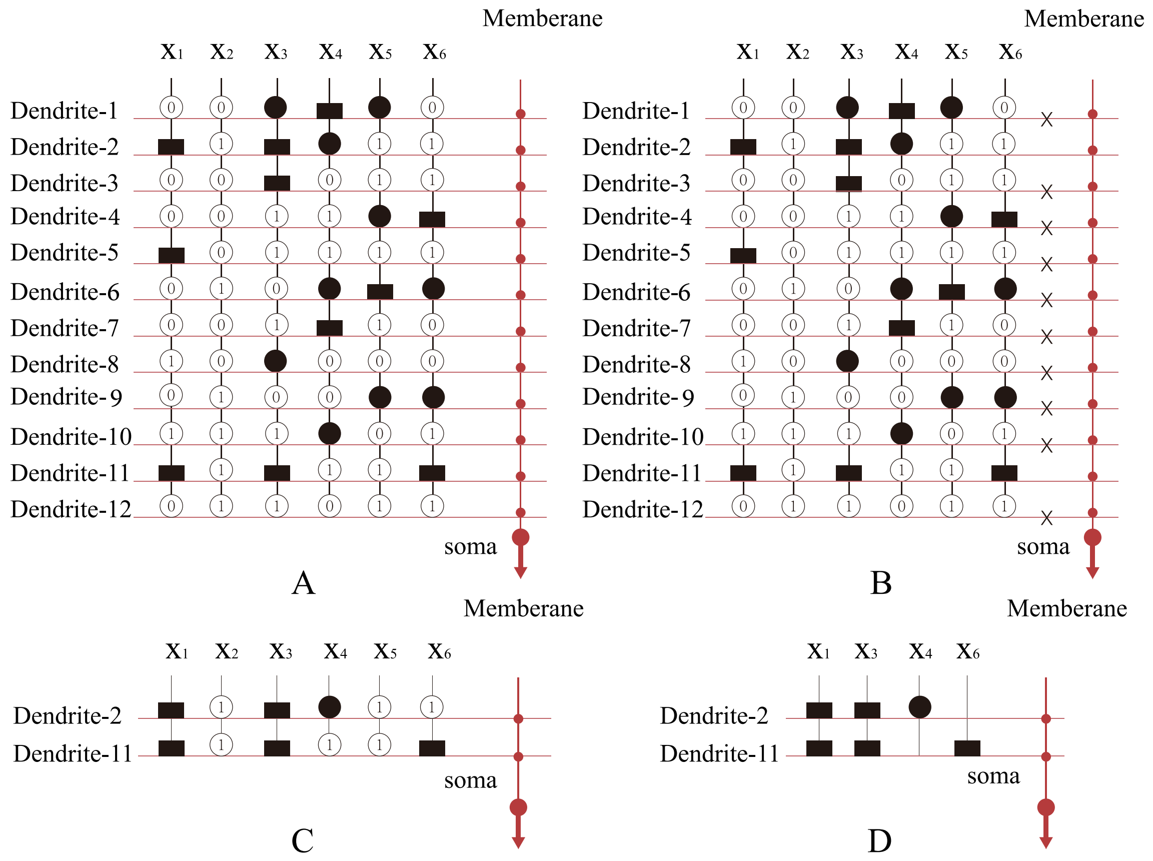

Figure 7.5: Structure simplification process for the liver dataset.

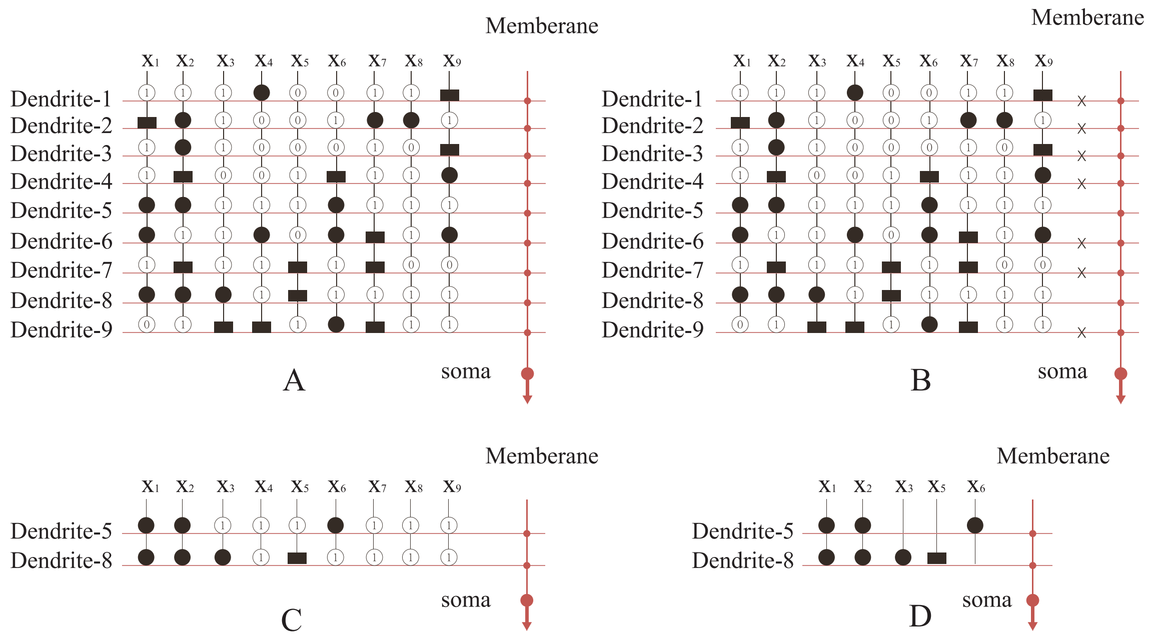

Figure 7.6: Structure simplification process for the cancer dataset.

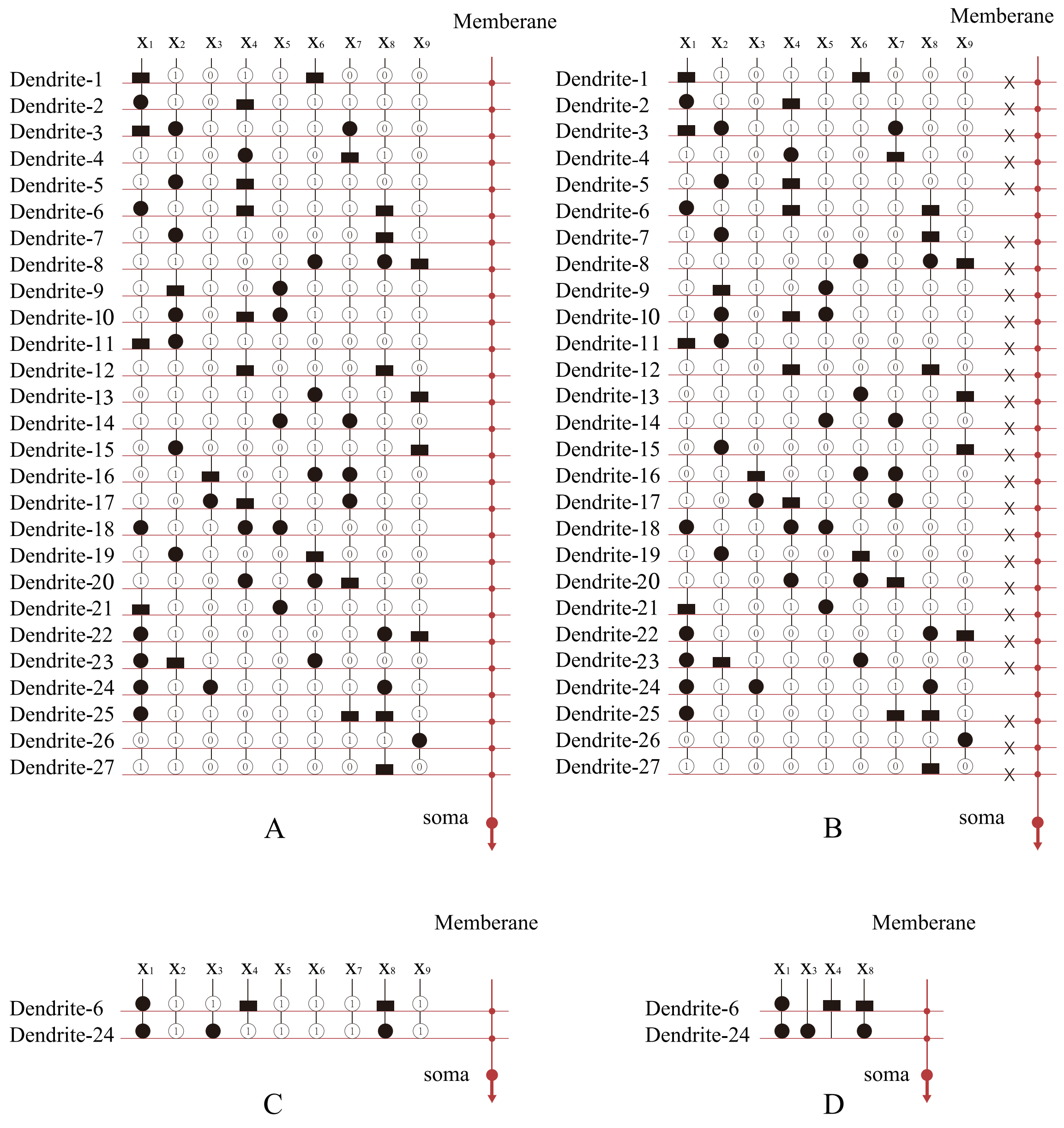

Figure 7.7: Structure simplification process for the glass dataset.

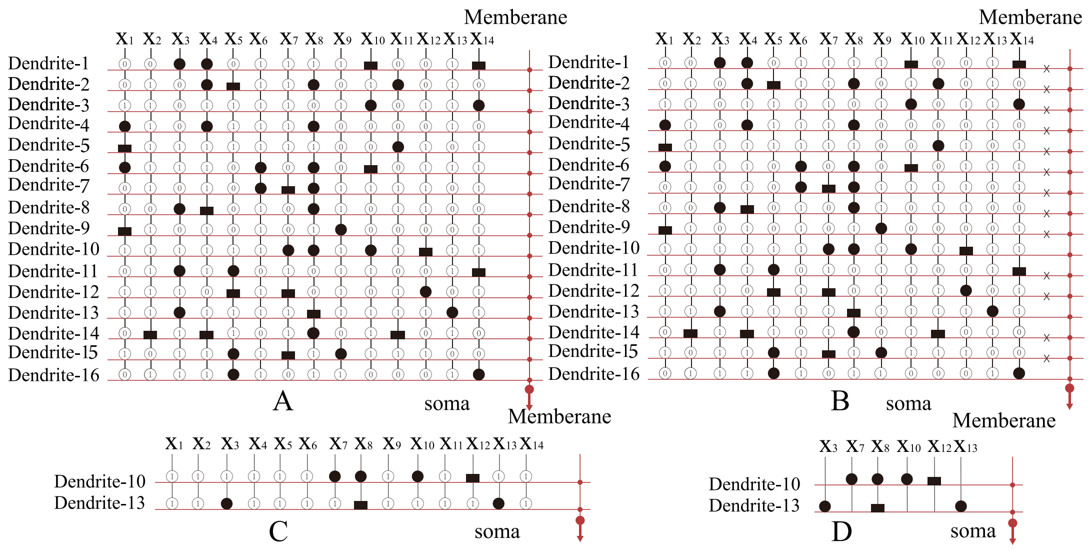

Figure 7.8: Structure simplification process for the ACA dataset.

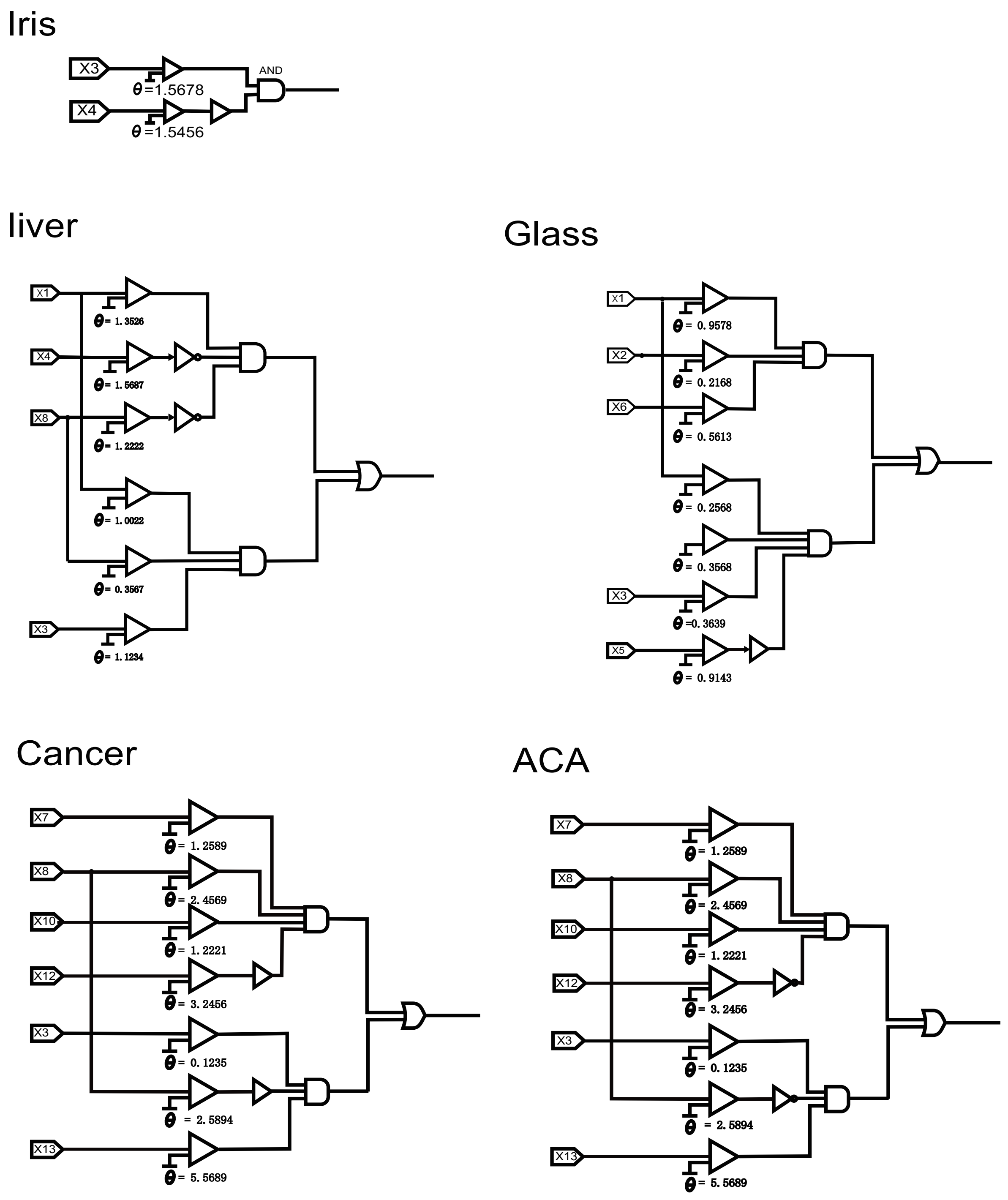

Figure 7.9: Logic circuit of iris.

Chapter 8

Conclusion

The dendrite has a pivotal role in the computing process, and the computation ability of a single neuron is undervalued. Although a single neuron has limited ability and cannot surpass a large-scale neural network, we expect to apply DMAS to deep neuron networks in the future. In further research, we plan to adjust DMAS to enable its adaptation to the deep learning structure. We believe that this model has considerable potential in electronic design areas, such as VLSI and biomedical science.DMAS has further access to the real biological neuron with a self-pruning ability. This function can eliminate branches from the dendrite morphology depending on the continuum values. It hence reduces the computational load by evolving and simplifying the dendritic structure without affecting the computational result. Simplified dendritic structure can be implemented in the logic circuit with comparator, OR gate, AND gate, and NOT gate. It makes it possible to solve real problems with less cost.

A self-adaptive synapse is firstly proposed in the paper. Its utility is proved by the Friedman test in section 4.3. The ability of adaptive synapse not only has stronger robustness, but also reduces a parameter in DNM and improves the performance of DNM.

However, there are still many deficiencies in this study:

1. The algorithm is too simple, and some high-performance algorithms will be added in the future work, such as PSO and other relatively new algorithms

2. There is not enough experimental demonstration and only one comparison object. Some comparisons with other algorithms will be added in the future work.

3. Single neuron model can not solve the problem of multi classification.

Membrane play an import role in the neuron system. with many kinds of Ion channel and feature leads to differences in the electrical properties of different membranes. In the process of computing we always assumed that the membranes have a same ability. However, in real word the membranes are diversity. Our research let the membranes ability join in the train process and optimize it with DE algorithm. This work can make our neuron model more real neuron like and improved the computing accuracy.

DE-DMAS neuron model is a real neural based neuron model. Different from the previous model, the difference of cell membrane has been first considered and included in this model. Even though DE-DMAS has advantage, this model also has limited generalizability due to the single model structure in some aspects, such as due to one model only have two phases, a single model cannot deal with multiple classification problem. In our future research, we will try to build an DE-DMAS based neuron network include enormous amount neuron so that we can deal with the multiple classification. Furthermore, we can expand it in a brain like network based on this research. and apply this in variable area.

To improve the calculation ability of the dendritic neuron model (DNM), a dendritic neuron model with adaptive synapses trained by differential evolution algorithm (DE-DMAS) is proposed, which shows enhanced performance in the simulation based on UCI datasets. A comparison with the classic BPNN and BP-DNM is carried out in terms of the test accuracy, sensitivity, specificity, and ROC and cross-validation. DE-DMAS shows its superiority in all the results and DE-DMAS as a single neuron model is found to substantially outperform BPNN and BP-DNM.

Bibliography

[1] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[2] N. Rochester, J. Holland, L. Haibt, and W. Duda, “Tests on a cell assembly theory of the action of the brain, using a large digital computer,” *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 80–93, 1956.

[3] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[4] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry.* MIT press, 2017.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[6] C. Koch, “Computation and the single neuron,” *Nature*, vol. 385, no. 6613, p. 207, 1997.

[7] C. Koch and I. Segev, “The role of single neurons in information processing,” *Nature Neuroscience*, vol. 3, no. 11s, p. 1171, 2000.

[8] J. L. Davidson and F. Hummer, “Morphology neural networks: An introduction with applications,” *Circuits, Systems and Signal Processing*, vol. 12, no. 2, pp. 177–210, 1993.

[9] G. X. Ritter and P. Sussner, “An introduction to morphological neural networks,” in *Proceedings of 13th International Conference on Pattern Recognition*, vol. 4. IEEE, 1996, pp. 709–717.

[10] H. Sossa and E. Guevara, “Efficient training for dendrite morphological neural networks,” *Neurocomputing*, vol. 131, pp. 132–142, 2014.

[11] ——, “Modified dendrite morphological neural network applied to 3d object recognition,” in *Mexican Conference on Pattern Recognition.* Springer, 2013, pp. 314–324.

[12] Y. Todo, H. Tamura, K. Yamashita, and Z. Tang, “Unsupervised learnable neuron model with nonlinear interaction on dendrites,” *Neural Networks*, vol. 60, pp. 96–103, 2014.

[13] J. Ji, S. Gao, J. Cheng, Z. Tang, and Y. Todo, "An approximate logic neuron model with a dendritic structure," *Neurocomputing*, vol. 173, pp. 1775–1783, 2016.

[14] Y. Tang, J. Ji, Y. Zhu, S. Gao, Z. Tang, and Y. Todo, "A differential evolution-oriented pruning neural network model for bankruptcy prediction," *Complexity*, vol. 2019, Article ID 8682124, 2019.

[15] X. Qian, Y. Wang, S. Cao, Y. Todo, and S. Gao, "Mr2dnm: A novel mutual information-based dendritic neuron model," *Computational Intelligence and Neuroscience*, vol. 2019, Article ID 7362931, 2019.

[16] T. Zhou, S. Gao, J. Wang, C. Chu, Y. Todo, and Z. Tang, "Financial time series prediction using a dendritic neuron model," *Knowledge-Based Systems*, vol. 105, pp. 214–224, 2016.

[17] W. Chen, J. Sun, S. Gao, J. Cheng, J. Wang, and Y. Todo, "Using a single dendritic neuron to forecast tourist arrivals to japan," *IEICE Transactions on Information and Systems*, vol. 100, no. 1, pp. 190–202, 2017.

[18] J. Ji, S. Song, Y. Tang, S. Gao, Z. Tang, and Y. Todo, "Approximate logic neuron model trained by states of matter search algorithm," *Knowledge-Based Systems*, vol. 163, pp. 120–130, 2018.

[19] Y. Yu, Y. Wang, S. Gao, and Z. Tang, "Statistical modeling and prediction for tourism economy using dendritic neural network," *Computational Intelligence and Neuroscience*, vol. 2017, Article ID 7436948, 2017.

[20] L. Luo and D. D. O'Leary, "Axon retraction and degeneration in development and disease," *Annual Review of Neuroscience*, vol. 28, pp. 127–156, 2005.

[21] P. Hagmann, O. Sporns, N. Madan, L. Cammoun, R. Pienaar, V. J. Wedeen, R. Meuli, J.-P. Thiran, and P. Grant, "White matter maturation reshapes structural connectivity in the late developing human brain," *Proceedings of the National Academy of Sciences*, vol. 107, no. 44, pp. 19 067–19 072, 2010.

[22] C. Koch, T. Poggio, and V. Torre, "Retinal ganglion cells: a functional interpretation of dendritic morphology," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 298, no. 1090, pp. 227–263, 1982.

[23] ——, "Nonlinear interactions in a dendritic tree: localization, timing, and role in information processing," *Proceedings of the National Academy of Sciences*, vol. 80, no. 9, pp. 2799–2802, 1983.

[24] L. Beaulieu-Laroche, E. H. Toloza, N. J. Brown, and M. T. Harnett, "Widespread and highly correlated somato-dendritic activity in cortical layer 5 neurons," *Neuron*, vol. 103, no. 2, pp. 235–241, 2019.

[25] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neural model with effective learning algorithms for classification, approximation, and prediction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, pp. 601–614, 2019.

[26] X. Wang, Z. Tang, H. Tamura, M. Ishii, and W. Sun, "An improved backpropagation algorithm to avoid the local minima problem," *Neurocomputing*, vol. 56, pp. 455–460, 2004.

[27] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[28] K.-S. Tang, K.-F. Man, S. Kwong, and Q. He, "Genetic algorithms and their applications," *IEEE Signal Processing Magazine*, vol. 13, no. 6, pp. 22–37, 1996.

[29] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms." in *International Joint Conferences on Artificial Intelligence (IJCAI)*, vol. 89, 1989, pp. 762–767.

[30] S. Gao, S. Song, J. Cheng, Y. Todo, and M. Zhou, "Incorporation of solvent effect into multi-objective evolutionary algorithm for improved protein structure

prediction," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 4, pp. 1365–1378, 2018.

[31] S. Song, S. Gao, X. Chen, D. Jia, X. Qian, and Y. Todo, "AIMOES: Archive information assisted multi-objective evolutionary strategy for ab initio protein structure prediction," *Knowledge-Based Systems*, vol. 146, pp. 58–72, 2018.

[32] J. Kennedy, "Particle swarm optimization," *Encyclopedia of Machine Learning*, pp. 760–766, 2010.

[33] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proceedings of the 2004 Congress on Evolutionary Computation*, vol. 2. IEEE, 2004, pp. 1980–1987.

[34] S. Gao, Y. Yu, Y. Wang, J. Wang, J. Cheng, and M. Zhou, "Chaotic local search-based differential evolution algorithms for optimization," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, 2019, doi: 10.1109/TSMC.2019.2956121.

[35] J. Sun, S. Gao, H. Dai, J. Cheng, M. Zhou, and J. Wang, "Bi-objective elite differential evolution for multivalued logic networks," *IEEE Transactions on Cybernetics*, vol. 50, no. 1, pp. 233–246, 2020.

[36] B. Subudhi and D. Jena, “A differential evolution based neural network approach to nonlinear system identification,” *Applied Soft Computing*, vol. 11, no. 1, pp. 861–871, 2011.

[37] E. Bas, “The training of multiplicative neuron model based artificial neural networks with differential evolution algorithm for forecasting,” *Journal of Artificial Intelligence and Soft Computing Research*, vol. 6, no. 1, pp. 5–11, 2016.

[38] F. Arce, E. Zamora, H. Sossa, and R. Barrón, “Differential evolution training algorithm for dendrite morphological neural networks,” *Applied Soft Computing*, vol. 68, pp. 303–313, 2018.

[39] B. Farley and W. Clark, “Simulation of self-organizing systems by digital computer,” *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 76–84, 1954.

[40] R. L. Rivest and C. Stein, “Thomas h. cormen, charles e. leiserson.”

[41] F. Clautiaux, S. Hanafi, R. Macedo, M.-É. Voge, and C. Alves, “Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints,” *European Journal of Operational Research*, vol. 258, no. 2, pp. 467–477, 2017.

[42] B. Engquist and W. Schmid, *Mathematics unlimited-2001 and beyond.* Springer, 2017.

[43] M. J. Tyre and E. Von Hippel, "The situated nature of adaptive learning in organizations," *Organization Science*, vol. 8, no. 1, pp. 71–83, 1997.

[44] Z. Sha, L. Hu, Y. Todo, J. Ji, S. Gao, and Z. Tang, "A breast cancer classifier using a neuron model with dendritic nonlinearity," *IEICE Transactions on Information and Systems*, vol. 98, no. 7, pp. 1365–1376, 2015.

[45] T. Jiang, S. Gao, D. Wang, J. Ji, Y. Todo, and Z. Tang, "A neuron model with synaptic nonlinearities in a dendritic tree for liver disorders," *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 12, no. 1, pp. 105–115, 2017.

[46] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "A comparative study of differential evolution variants for global optimization," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation.* ACM, 2006, pp. 485–492.

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

[48] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2010.

[49] Y. Yu, S. Gao, Y. Wang, and Y. Todo, "Global optimum-based search differential evolution," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 2, pp. 379–394, 2019.

[50] R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, vol. 10, no. 10, pp. 293–298, 2002.

[51] J. Ronkkonen, S. Kukkonen, and K. V. Price, "Real-parameter optimization with differential evolution," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1. IEEE, 2005, pp. 506–513.

[52] D. Zaharie, "Influence of crossover on the behavior of differential evolution algorithms," *Applied Soft Computing*, vol. 9, no. 3, pp. 1126–1138, 2009.

[53] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[54] R. Jugulum, S. Taguchi *et al.*, *Computer-based robust engineering: essentials for DFSS.* ASQ Quality Press, 2004.

[55] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[56] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification and scene analysis.* Wiley New York, 1973, vol. 3.

[57] B. V. Dasarathy, "Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 1, pp. 67–71, 1980.

[58] J. McDermott and R. S. Forsyth, "Diagnosing a disorder in a classification benchmark," *Pattern Recognition Letters*, vol. 73, pp. 41–43, 2016.

[59] O. L. Mangasarian and W. H. Wolberg, "Cancer diagnosis via linear programming," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 1990.

[60] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology." *Proceedings of the National Academy of Sciences*, vol. 87, no. 23, pp. 9193–9196, 1990.

[61] I. W. Evett and J. S. Ernest, “Rule induction in forensic science. central research establishment. home office forensic science service. aldermaston,” *Reading, Berkshire RG7 4PN*, 1987.

[62] J. R. Quinlan, “Simplifying decision trees,” *International journal of man-machine studies*, vol. 27, no. 3, pp. 221–234, 1987.

[63] Y. Tang, J. Ji, S. Gao, H. Dai, Y. Yu, and Y. Todo, “A pruning neural network model in credit classification analysis,” *Computational intelligence and neuroscience*, vol. 2018, Article ID 9390410, 2018.

[64] J. F. Khaw, B. Lim, and L. E. Lim, “Optimal design of neural networks using the taguchi method,” *Neurocomputing*, vol. 7, no. 3, pp. 225–245, 1995.

[65] W. Yang and Y. Tarng, “Design optimization of cutting parameters for turning operations based on the taguchi method,” *Journal of Materials Processing Technology*, vol. 84, no. 1-3, pp. 122–129, 1998.

[66] M. Friedman, “The use of ranks to avoid the assumption of normality implicit in the analysis of variance,” *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, 1937.

[67] C. H. Yu, “Exploratory data analysis,” *Methods*, vol. 2, pp. 131–160, 1977.

[68] N. R. Cook, “Statistical evaluation of prognostic versus diagnostic models: beyond the roc curve,” *Clinical Chemistry*, vol. 54, no. 1, pp. 17–23, 2008.

[69] S. Ma, H. Qiu, S. Hu, Y. Pei, W. Yang, D. Yang, and M. Cao, “Quantitative assessment of landslide susceptibility on the loess plateau in china,” *Physical Geography*, pp. 1–28, 2019.