A New Method for Figuring the Number of Hidden Layer Nodes in BP Algorithm

Chang Mu*, Bo-Zhang Qiu, Xiu-Hai Liu
School of Science, Nanjing University of Science and Technology,
Nanjing, Jiangsu 210094, China
*Corresponding author: jt_mc17@sina.com

Abstract: In the field of artificial neural network, BP neural network is a multi-layer feed-forward neural network. Because it is difficult to figure the number of hidden layer nodes in a BP neural network, the theoretical basis and the existing methods for BP network hidden layer nodes are studied. Then based on traditional empirical formulas, we propose a new approach to rapidly figure the quantity of hidden layer nodes in two-layer network. That is, with the assistance of experience formulas, the horizon of unit number in hidden layer can be confirmed and its optimal value will be found in this horizon. Finally, a new formula for figuring the quantity of hidden layer codes is obtained through fitting input dimension, output dimension and the optimal value of hidden layer codes. Under some given input dimension and output dimension, efficiency and precision of BP algorithm may be improved by applying the proposed formula.

***Key words**: Artificial neural network; BP algorithm; Hidden layer nodes.*

*****

I. Introduction

With the coming of information age, the data analysis and its forecast and decision-making have become more and more important. Artificial neural network is initiated through imitation of the structure and characteristics of human brain. Interconnecting a large number of simple processing units (neurons or nodes), we get massively parallel distributed information processing and nonlinear dynamic systems which have several advantages, such as excellent fault-tolerance, powerful learning ability, fast running speed and so on. It contains a lot of networks, among which BP algorithm proposed in 1986 is pretty mature. A group of scientists led by Rumelhart and McCelland have detailedly discussed an error back propagation algorithm error for multi-layer perceptron with nonlinear continuous transformation function in their book "Parallel distributed processing". BP algorithm model has become the most widely used neural network model, and its application field is still expanding.

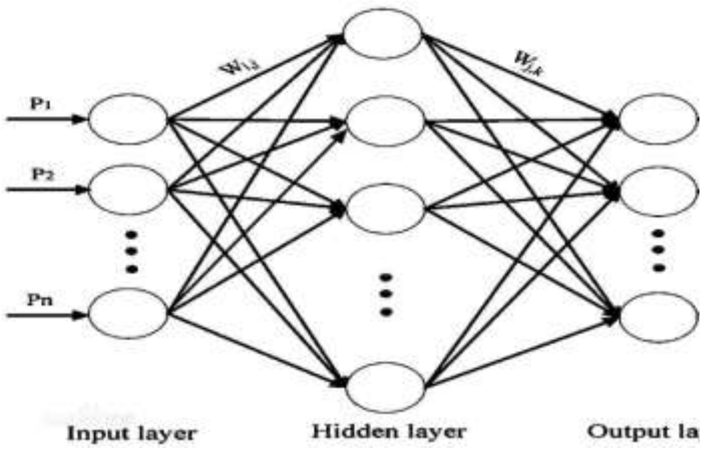
BP algorithm owns many merits including good self-study character and the realization of nonlinear mapping from the input space to the output space. However, there are some disadvantages concerning BP algorithm, for example, network training calling for a large scale of input and output data, slow convergence speed, easy to fall into local minimum point, strict choice of initial weights and difficulty of network structure design, namely, there is no theoretical guidance about the selection concerning quantity of hidden layer codes, etc. In BP network, the choice of hidden layer nodes is very crucial, not only having a great influence on the performance of establishing neural network model, but also being the direct reason of “overfitting” appearing in network training, but in theory there exists no scientific and universal method for determining hidden layer codes [1–3] .

In fact, applying artificial neural network to solve nonlinear problems is a matching process between the optimal solution of practical problem and the balance of network through employing artificial neurons in the network. The performance of network is an important index for judging the quality of network structure, and the selection problem of artificial neural network structure is actually a determining problem about the number of hidden layer neurons. But at present the formulas of the hidden layer nodes in lots of articles are proposed provided that training samples can be obtained without any limitation, and furthermore they are given for the worst situation, which both seldom appear in practice. In this paper, we mainly discuss the selection problem for the quantity of hidden layer neurons in two-layer BP neural network, and through the investigation about the simulation for 6 common functions when input dimension and output dimension are both limited from 1 to 10, a new method for determining the number of neurons in hidden layer of BP algorithm is presented.

1 BP neural network model

1.1 Basic Theory of BP Neural Network

BP neural network (back-propagation neural network) is a multi-layer forward network, and its structure shows as the following:



The first layer of BP neural network is called input layer, and the last layer is output layer, and every middle layer is called hidden layer. The nonlinear input-output relationship between hidden layer k and the neurons in output layer is denoted as f_k （$k=2,\cdots,m$）, and the connection weight from the j-th neuron of the $(k-1)$th layer to the i-th neuron of the k-th layer is w_{ij}^k. The total inputs of the i-th neuron in the k-th layer are u_i^k, and the output of this neuron is y_i^k. The relationships between the variables are:

$$y_i^k = f_k\left(u_i^k\right), \quad u_i^k = \sum_j w_{ij}^{k-1} y_j^{l-1},$$

$k=2,\cdots,m.$

If the input data of BP neural network is $X=[x_{p1}, x_{p2},\ldots, x_{p_n}]^T$ (assuming there are p_n neurons in input layer), going through every hidden layer node in turn from input layer, the output data $Y=[y_{a_1}^m, y_{a_2}^m,\ldots, y_{a_m}^m]^T$ (assuming there are a_m neurons in output layer) will be obtained. Hence BP neural network can be treated as a nonlinear mapping from input space to output space.

BP learning algorithm minimizes the error according to the reverse learning process, so the objective function is selected as the following:

$$Min\ J = \frac{1}{2}\sum_{j=1}^{a_m}(y_j^m - y_{sj})^2.$$

That is, the sum of the squares of the difference between the expected output y_{sj} and the actual output y_j^m in the neural network is minimized through choosing appropriate neural network weights. Actually, this learning algorithm is to look for the minimum value of the objective function J subject to constraint conditions:

$$y_i^k = f_k\left(u_i^k\right), u_i^k = \sum_j w_{ij}^{k-1} y_j^{l-1},$$

$k=2,\cdots,m.$

Employing "steepest descent method" in nonlinear programming, changing the weight along the negative gradient direction of the objective function, we get the neural network weight correction $\vartriangle w_{ij}^{k-1} = -\varepsilon \frac{\partial J}{\partial w_{ij}^{k-1}}$ ($\varepsilon > 0$), where ε is the learning step size.

Therefore BP learning algorithm can be summarized as follows:

$$\triangle w_{ij}^{k-1} = -\varepsilon\, d_i^k y_j^{-1}$$

$$d_i^m = y_i^m (1 - y_i^m)(y_i^m - y_{si})$$

$$d_i^k = y_i^k (1 - y_i^k) \sum_l w_{li}^k d_l^{k+1}$$

$(k = m-1, \cdots, 2)$[4,5].

Thus, the selection of the error function is a recursive process that begins with the back propagation of output layer, so it is named as back propagation learning algorithm. Through learning plenty of samples, modifying the weight, then constantly reducing the deviation, finally well-pleasing results will be achieved.

The basic idea of the BP algorithm is that the learning process consists of two processes: the forward propagation of the signal and the reverse propagation of the error. Forward propagation is that the input samples come from input layer and are processed by every hidden layer to output layer in the end. If the actual output of output layer does not match the expected output, it turns into the reverse propagation phase of the error. The reverse propagation of the error makes output error going to input layer through every hidden layer by a given form, and the error is distributed to all units of every layer, so the error signal of every layer unit is obtained to be the foundation for the correction of every unit weight. This forward propagation of the signal and every layer weight's adjustment process of error backwards propagation are carried out round and round. In reality, weight's adjustment process is learning and training process of the network. This process continues running until the error of the network output is reduced to an acceptable level, or until the pre-set learning times is reached.[6]

1.2 Improvement of standard BP algorithm

BP algorithm is applied to a three-layer perceptron with a nonlinear transformation function, which can be employed to approximate any nonlinear function with arbitrary precision. This extraordinary advantage makes the multi-layer sensor more and more widely used. However, the standard BP algorithm exposes a lot of inherent defects in practice, such as easy to fall into local minimum point and thus hard to approach the global optimal value. When training new samples it has the tendency to forget old samples. Furthermore more training times make the learning efficiency lower and the convergence speed is slow. For these shortcomings of BP algorithm, many researchers have put forward lots of effective improvement algorithms. A brief introduction about three commonly used methods is presented as follows.

1.2.1 Increase momentum

Some mathematicians in 1986 stated the standard BP algorithm adjusts the weight only according to negative gradient direction of the error in time t, but neglects the gradient direction before time t, hence leading to training process concussion and slow convergence speed. In order to improve training speed of the network, a momentum item may be added in the weight adjustment formula. If W represents the weight matrix of a layer, and X represents the input vector of a layer, then the weight adjustment vector including the momentum item can be expressed as:

$$\Delta W(t) = \eta \delta X + \alpha \Delta W(t-1).$$

1.2.2 Adaptive adjustment learning rate

The learning rate η is also called the step size and keeps constant in standard BP algorithm. However, in practice, it is difficult to determine the best learning rate that is appropriate from beginning to end. It can be seen from the error surface that training times may increase in the flat area if η is too small, so η is expected to become bigger. In the region where the error variation changes rapidly, the narrow " Pound " may be missed if η is too large which makes training effect shocking and iteration times increasing. In order to speed up the convergence process, a better idea is to change the learning rate adaptively, so that the rate η increases and decreases in proper situations.

1.2.3 Introduce steepness factor

There are some flat areas on the error surface, and the weight adjustment enters into a flat area because the neuron output enters into a saturation region of the transferring function. If we reduce the net input of the neuron after the adjustment entering into the flat area, letting the output depart from the unsaturated region of the transferring function, hence the shape of the error function can be converted so that the adjustment will be separated from the flat area. [5,7,8]

1.3 Research results on the number of hidden nodes

1.3.1 Empirical formula method

At present, there does not exist a complete theoretical guide concerning the method for determining the number of hidden neurons in artificial neural networks which in operation is usually based on neural network, designer's experience and estimation.

(1) Kolmogorov gave an equality about N_{in} and N_{hid} for the artificial neural network containing only three layers: $N_{hid} = 2N_{in} + 1$;

(2) Jadid and Fairbarin presented the upper bound formula of the hidden neurons:

$$N_{hid} \le N_{train} \big/ \left[R + \left(N_{in} + N_{out}\right)\right],$$

where N_{train} is the number of training samples, and $5 \le R \le 10$;

(3) Hecht-Nieslsen theoretically proved the inequality relation between the number of hidden nodes and the number of input nodes according to Kolmogorov theorem: $N_{hid} \le N_{in} + 1$;

(4) Gao obtained an empirical formula by simulations:

$$N_{hid} = \left(0.43N_{in} * N_{out} + 0.12N_{out}^2 + 2.54N_{in} + 0.77N_{out} + 0.35\right)^{1/2},$$

and the right part of the results retained integer as the value of N_{hid}; [9]

(5) According to some given learning samples, Zhou and other researchers selected different numbers of neurons in hidden layer, training the network model and then comparing different results of the model, and finally proposed an empirical formula:

$$N_{hid} = \left[N_{in} + \left(N_{out} + 1\right) + 1\right]^{1/2}.$$

In fact, once the number of neurons in hidden layer is determined, the number of neurons will never be changed throughout the network adjustment. In this case, the number of neurons may be either too large or too small in the established neural network model, so that it is hard to ensure whether the network structure is optimal or not.

1.3.2 Repeated test method

Most researchers do not advocate to employ empirical formula, and they consider that the most effective way to find the optimal number of hidden neurons is based on repeated tests, so that the number of hidden layer neurons which enables the sample error to achieve the preset default accuracy can be regarded as the optimal number of hidden layer neurons in the network model. Cross-validation is a typically iterative test method, finding effective information from the existing learning samples, and the connection strength between different network neurons will be adjusted according to the information. Applying the cross-validation method, we are able to find the optimal number of hidden layer neurons, so that the predetermined accuracy can be achieved for the network learning samples. The cross-validation method is simple and easy to understand. Firstly, the learning samples are divided into two parts: learning samples and test samples; secondly, the learning samples are networked to adjust the neuron connection strength until the preset accuracy of the network is approached; finally learning for test samples is made. In short, in order to get a better learning performance, both learning samples and test samples should be as much as possible to participate in learning. [10]

1.3.3 Growth Law

The so-called growth method refers to choose a few neurons of hidden layer before the operation of the artificial neural network, and then gradually increases the number of hidden neurons (increasing one or more nerves each time) in learning process for a given practical problem until the performance requirements of the network model are satisfied.

1.3.4 Global minimum method

In 1989, Mezard from the perspective of the complexity of space, increasing the number of hidden nodes will reduce the practical problem of optimization of the dimension of space, if the appropriate increase in neurons and connectivity, reasonable to determine the network learning parameters, the network error may be out Local minima and converge to global minima. Based on this idea, he proposed the Tiling algorithm to construct the main neuron module and the auxiliary neuron module on each layer to achieve the desired mapping:

$$R^n \to B^m = \{0,1\}^m \quad \text{or} \quad \{-1,1\}^m$$

Frean in 1990 gave the upstar algorithm to add neurons to reduce the network learning error in the way the network at the global minimum point of convergence, to balance the data. In 1992, Chang et al. established a three-layer general network model. When the network into a local minimum, add a hidden layer of neurons to ensure that after adding neurons to the network learning error than adding neurons before the school learning error is small. The neurons are added repeatedly until the network error converges to zero.

1.3.5 Cascade-Conelation algorithm

Fahhnan's CC algorithm is an optimization algorithm that can construct the optimal network structure in the network learning process. Before the network begins to learn, the number of hidden neurons is set to zero; then the network is trained to add a hidden layer neuron until the network learning error does not fall again. The intensity of the interaction associated with the newly added neurons can be determined by the covariance of the actual output value of the neuron and the network learning error is determined. Follow this step to learn more until you meet the requirements of the network model default accuracy. It is not difficult to find that the CC algorithm is a greedy algorithm that seeks to make each neuron contribute to the convergence of the network to the preset accuracy, but there is a benefit that at any one time, only one neuron's intensity is adjusted , And do not need feedback.

1.3.6 Modular growth method

In the process of online learning, each time not add a neuron, but to contain multiple neurons of the module. Littmann et al. constructed a modular approach to a neural network that only applies to single output neurons, which adjusts the connection strength between neurons by introducing an error evaluation function. Lu and other proposed PMSN structure, each increase is a filter module SM (SievingModel), do not have to re-adjust the parameters have been learned, so the network learning speed quickly.

II. BP algorithm test network structure

2.1 Determination of the number of hidden layers

In the BP algorithm, although the hidden layer structure design involves network weight, incentive function, learning rate and other factors, but the direct impact of network performance is the hidden layer structure of the layers and the number of nodes. From the existing research results, we can see that the more complex the number of hidden layers, the more complex the mapping expressed by the network, not only increase the computational complexity, but also lead to data distortion, and according to Kolmogorov theorem, when the hidden layer is greater than or equal to 1 , The BP network has the ability to approach all nonlinear maps, that is, when the hidden layer nodes can be set freely according to the need, then with two layers of forward neural network can achieve arbitrary precision approximation of any continuous function, so Experiment from a simple and practical point of view, choose a hidden layer of the network structure. [11]

Kolmogorov theorem: Given any $\varepsilon > 0$, there is a three-layer BP neural network for any continuous function f, the input layer has p_1 neurons, the middle layer has 2 p_1+1 neurons, and the output layer has p_m neurons. It can be approximated by f in any ε square error accuracy. The theorem proves not only the existence of the mapping network, but also the structure of the mapping network. That is, there is always a three-layer forward neural network with a structure of $p_1 \times (2p_1+1) \times p_m$ that can accurately approximate any continuous function f. [10]

2.2 Determination of the number of hidden nodes

It is known that a two-layer BP network with infinite hidden layer nodes can realize any non-linear mapping from input to output, but for a limited number of input-to-output mappings, there is no need for infinite hidden nodes. How to choose the number of hidden nodes. If the number of hidden nodes is too large, the generalization ability of the network will be worse after the study. If the number of hidden nodes is too small, the learning time will be too long, the network fault tolerance will be poor, the local minimum will be, the error will not be the smallest, and do not produce enough number of connection weights to meet a number of sample learning. Due to the complexity of the problem, so far has not yet found a good analytical formula, the number of hidden nodes is often based on the empirical formula derived from the empirical test and their own to determine, if the requirements of the approximation of the variable function changes, The number of nodes in the hidden layer should be more; if the specified approximation accuracy is high, the number of hidden layer units should be more; if the beginning is to put less hidden layer unit , According to the subsequent study of the situation gradually increased, or the beginning to join enough hidden nodes, through learning to not work on the connection and hidden layer nodes deleted. [12]

III. A fitting experiment about number formula of hidden layer nodes

3.1 Experimental route

The number of neurons in establishing hidden layer neural network should follow the principle --- selecting compact structure under the condition that the precision requirement is satisfied, that is, as few hidden layer nodes as possible. In this experiment, the empirical formula is employed to determine the range of the number of hidden layer nodes, and the optimal value will be searched in this range. According to the empirical formula

$$N = (n+m)^{1/2} + a$$

(where N is the number of corresponding hidden layer nodes for minimum error, n is the dimension of input data, m is the dimension of output data , a is a constant number between 1 and 10), the optimal number of nodes is determined in the interval $[1,15]$ provided that input dimension and output dimension both change from 1 to 10. With the help of Matlab program, the number of hidden layer nodes is traversed from 1 to 15, then recording the error about different numbers of nodes and making comparison between these errors, determining and recording the corresponding numbers of hidden layer nodes for minimum error under different dimensions of input and output data, finally the corresponding formula is obtained according to the fitting between input and output dimensions and the number of hidden layer nodes with minimum error. The experimental processes are listed as follows:

(1) Obtaining data sets (sample sets and experimental sets);

(2) Inputting sample sets, and training the network;

(3) Inputting experimental sets to the network trained, and traversing different numbers of hidden layer nodes and then getting corresponding errors;

(4) Selecting the corresponding number of nodes for the minimum error under different input and output data with different dimensions, and then recording m (dimension of input layer), n (dimension of output layer), N (the

number of corresponding hidden layer nodes for minimum error);

(5) Fitting function relations according to the optimal number of codes and input and output data; [13]

3.2 Experimental data

3.2.1 Selection of experimental functions

In the experiment, the functions of generating experimental data are constructed by using the compound operations of elementary functions (including trigonometric function, exponential function, logarithmic function and so on), for example,

$$3x_1^4 + x_2^3 + 5x_3 + 1,$$

$$\log_{10}(x_1) + 2\log_{10}(x_2) + 3\log_{10}(x_3),$$

$$\tan(x_1) * \tan(x_2) + 3\log_{10}(x_3) - \cos(x_4),$$

$$\sin(x_1) + 3\cos(x_2) + 2\tan(x_3) * \cot(x_4)^4,$$

etc.

3.2.2 Normalization (pretreatment)

Normalization processing means that the input and output data of a network are limited to $[0,1]$ or $[-1,1]$ through transformation processing, and main reasons for normalization are:

(1) The input data of the network usually have different physical meanings or different dimensions, and the normalization makes all components change in $[0,1]$ or $[-1,1]$, so that the network training gives every input component equal position in the beginning.

(2) All neurons of BP neural network adopt transferring function of Sigmoid class, which can prevent the neuron output from being saturating because the absolute value of net input is too large, and then adjust the weight into the flat area of error surface.

(3) We know the output of Sigmoid transferring function changes in [0, 1] or [-1, 1], if the output data are not transformed, the absolute error of the output component owning big numerical value will be large, and the absolute error of the output component with small value will be small. The weight adjustment only for total error of output in network training leads to a problem that an output component with small proportion possessed in total error has large absolute error, while this problem can be settled after the normalization of output data.

Normalized formula is:

$$\bar{x}_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

where x_i is input or output data, $x_{\min}$ represents the minimum value of the data, $x_{\max}$ stands for the maximum value of the data. [14]

3.2.3 Normalized instance

Input x_1	Input x_2	Input x_3	Output y_1	Output y_2
Original value/ After normalization	Original value/ After normalization	Original value/ After normalization	Original value/ After normalization	Original value/After normalization
2771/-0.4465	191/-0.96219	5254/0.05042	9.44416/0.13394	19.166/0.23053
7132/0.426484	5717/0.143229	1322/-0.73629	10.7316/0.57939	20.731/0.485137

1649/-0.6711	6688/0.337467	6105/0.220688	10.828/0.61282	22.225/0.728122
3753/-0.24992	9646/0.929186	9090/0.817927	11.517/0.851245	23.419/0.922346
2611/-0.47853	375/-0.92539	9982/0.996399	9.990/0.322821	20.563/0.457689
8446/0.689521	5903/0.180436	1875/-0.62565	10.971/0.662133	21.288/0.575678
2224/-0.556	6960/0.391878	2508/-0.499	10.589/0.530081	21.230/0.56633
5984/0.196677	9894/0.978796	669-0.86695	10.598/0.533097	20.244/0.405874
113/-0.97858	8039/0.607722	9324/0.864746	9.928/0.301309	21.772/0.654496
8185/0.637274	8624/0.724745	7239/0.447579	11.708/0.917371	23.363/0.913351

3.3 Experimental parameters

In the process of training the network, the learning rate is set to be 0.001, and the training times are 300, and the training precision is 0.00005. The total error performance function is mean square error (MSE), and the weights and thresholds of the network are corrected successively. Then the total error decreases until the required error performance is reached. When there are lots of samples, the correction of weights and thresholds follows the input of all samples and the calculation of total error, so that the convergence speed can be accelerated. In general, the input-output relationships of the neurons in hidden layer are nonlinear functions, and the transferring function of hidden layer is defined as Tansig function, namely, the tangent Sigmoid transferring function, which limits the output value in the interval [0, 1] and reduces the amount of computation. The output layer acts as buffer memory, adding data source to the network. The input-output relationship of input layer is usually linear, and the Purelin function is selected, which is a pure linear transferring function, so that the result of output layer can be taken to any value without prejudice to the expression of the output result.

Training function is Trainrp, that is to say, the elastic reflection propagation algorithm is employed to train the network. The weight and threshold change according to the variation of the gradient for many training functions. However, if the function is very flat near the extreme value, the variation of the gradient is very small, and therefore the change of weight and threshold will be very small. Too small variation of the gradient not only drive the network training convergence speed being very slow, but also can the network not achieve the training performance requirements. Fortunately, Trainrp function has the ability to overcome this shortcoming, because Trainrp function uses the direction of error gradient to determine the variation of weight and threshold, and the size of the error gradient has no influence on the variation of weight and threshold. [14,15,16]

3.4 Experimental results

Dimension of input	Dimension of output	Number of hidden layer codes
1	1	3
2	1	4
3	1	3
4	1	9
5	1	5
6	1	6
7	1	3
8	1	8
9	1	9
10	1	3
1	2	4
2	2	4
3	2	13
4	2	12
5	2	8
6	2	8
7	2	13
8	2	10
9	2	13
10	2	6
1	3	5
2	3	14
3	3	8
4	3	7
5	3	12
6	3	14
7	3	8
8	3	6
9	3	14
10	3	10
1	4	5
2	4	13
3	4	11
4	4	5

5	4	9
6	4	14
7	4	14
8	4	11
9	4	11
10	4	9
1	5	4
2	5	9
3	5	5
4	5	10
5	5	12
6	5	4
7	5	8
8	5	13
9	5	14
10	5	3
1	6	6
2	6	12
3	6	14
4	6	8
5	6	10
6	6	13
7	6	9
8	6	14
9	6	12
10	6	8
1	7	12
2	7	14
3	7	3
4	7	7
5	7	10
6	7	5
7	7	5
8	7	3
9	7	4
10	7	13
1	8	4
2	8	11

3	8	5
4	8	9
5	8	14
6	8	7
7	8	12
8	8	13
9	8	7
10	8	9
1	9	6
2	9	11
3	9	8
4	9	8
5	9	14
6	9	7
7	9	14
8	9	10
9	9	7
10	9	9
1	10	6
2	10	8
3	10	8
4	10	6
5	10	12
6	10	12
7	10	11
8	10	11
9	10	5
10	10	6

Selecting some common continuous functions including sine function, cosine function, tangent function, logarithmic function, power function and quadratic function to combine as the source of experimental data, 1000 groups of function data are obtained by calculation provided that input dimension and output dimension are both restricted from 1 to 10, in which 100 groups are the sample sets for training network and the rest are experimental sets. The experimental parameters are set up according to section 3.3, and the data obtained are fitted to get the formula for figuring the number of hidden layer nodes (four decimal digits reserved):

$$N = 1.0366 + 1.6937n - 0.1201n^2 + 1.4814m - 0.1049m^2 - 0.0352nm.$$

IV. Simulation verification

4.1 Data

Select the function $f(x)=x_1*x_3+x_2*x_4$ as a test function for simulation experiments, and choose 10 sets of data as a test sample.

Function input (x_i) and expected output (Y)

x_1	x_2	x_3	x_4	Y
0. 565809	0. 276021	−0. 84075	−0. 96778	−0. 24107
−0. 74237	0. 253203	−0. 52786	−0. 26188	−0. 32922
−0. 18367	0. 101081	−0. 50645	−0. 84369	−0. 37273
−0. 3286	0. 428943	−0. 36021	−0. 37796	−0. 14799
−0. 73236	0. 058247	−0. 35541	0. 009507	−0. 25600
0. 061355	−0. 48179	0. 696509	0. 735415	0. 318415
0. 671905	−0. 95036	−0. 69831	−0. 13800	−0. 28288
−0. 98619	−0. 45757	0. 670101	−0. 14720	−0. 22783
−0. 05375	−0. 17014	−0. 67230	−0. 19824	−0. 27711
0. 472125	−0. 55705	−0. 47464	−0. 84389	−0. 36447

4.2 Experimental output

Input the function data to the network as experiment, obtaining the total error corresponding to the number of hidden nodes in [2,15].

Number of hidden layer nodes	Total error
2	0.014000
3	0.007991
4	0.009578
5	0.010067
6	0.008907

7	0.007160
8	0.006991
9	0.009317
10	0.008777
11	0.012671
12	0.016822
13	0.014904
14	0.022306
15	0.011066

4.3 Result analysis

It can be seen from the simulation results that in the case of selecting the function f (x) with four-dimension input and one-dimension output, the number of hidden neurons of the BP algorithm with the smallest total error is $N=8$.

Substituting $m=4, n=1$ into the formula:

$$N = 1.0366 + 1.6937n - 0.1201n^2 + 1.4814m - 0.1049m^2 - 0.0352nm$$

Easily, we know theoretically optimal number of hidden layer nodes: $N=7$. Based on the total error of test output, the total errors are almost identical when hidden layer nodes are 7 and 8. Furthermore, the total errors are all very small in the interval $[2,15]$. Considering the randomness of the training network, it is proved that the formula are able to improve the efficiency for determining the optimal number of hidden layer nodes within the allowable range of the error under some given input and output dimensions, and also improve the precision of the continuous function approximation by employing the BP algorithm network.

V. Conclusions

BP algorithm employing reverse propagation is the most widely used structure of artificial neural networks. In this paper, we took the basic multi-input and multi-output single hidden layer BP network as an example, and some typical continuous functions were trained under different input and output dimensions, and also the global error of each network output has been analyzed. The result is that when input dimension and output dimension are both less than or equal to 10, and the learning rate is 0.001, and the training times are 300, and the training precision is 0.00005, the formula for figuring the number of hidden layer nodes with the best learning convergence character and function approximation accuracy is obtained through approaching six kinds of functions. This investigation may provide inspiration for how to figure the number of hidden layer nodes in some BP networks with other input and output dimensions or the multi hidden layer structure.

There are still many deficiencies in this experiment. Firstly, interval range is derived from the most commonly used empirical formula which perhaps is not the interval where the optimal node locates. Secondly, most parameters are certain, such as learning rate and training parameters, while the experiment does not ensure that this formula is still the best when the learning rate and the number of training times change. Thirdly, the training sample set is not large enough to approximate the six kinds of functions, for each experimental data are only 1000 groups. In the future, we are about to deepen the study of BP algorithm, exploring the problem about the optimal number of hidden layer nodes in more complex cases. We plan to determine the interval range at first, and then make selection for the number of hidden layer nodes, so the amount of computation may be reduced remarkably. Moreover, the formula for determining the number of hidden layer nodes in this experiment can also be applied as reference and judging basis.

References:

[1] Nielson, R H., Theory of the Backpropagation Neural Network, *Neural Networks*, 1(Suppl.1): 593-594, 1989.

[2] Dayhoff, J. E. and DeLeo, J. M., Artificial neural networks: Opening the black box, *Cancer*, 91(s8): 1615–1635, 2001.

[3] Hamid, B. Mohamad R.M., A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks, *International Journal of Systems Science*, 40(1): 101-118, 2009.

[4] Han, L., *Introduction to Artificial Neural Network*, Beijing: Beijing University of Posts and Telecommunications Press, 2006.

[5] Chu, L., Chen, S., and Zhou, M., *Mathematical Basis of* Computing *Intelligence*, Beijing: Science Press, 2002.

[6] Xu, L., *Neural Network Control*, Beijing: Electronic Industry Press, 2009.

[7] Li, Y., *BP Neural Network Research and Improvement and Applications*, http://cdmd.cnki.com.cn/Article/CDMD-10361-1012421305.htm , 2012.

[8] Fan, J., Wang, Z. and Qian, F., Research progress of hidden layer structure design of BP artificial neural network, *Control Engineering of China*, 12(Suppl.1): 105-108, 2005.

[9] Gao, D., On Structures of supervised linear basics function feedforward three-layered neural networks, *Journal of Circuits and Systems*, 2 (03): 31-36. 1997.

[10] Wang, W. *Artificial Intelligence and Its Application*. Beijing: Higher Education Press, 2016.

[11] Yan, H. and Guan, Y., Method to determine the quantity of internal nodes of back propagation neural networks and its demonstration, *Control Engineering of China*, 16(S2): 100-102, 2009.

[12] Shi, Y., Han, L. and Lian, X., *Neural Network Design Method and Case Analysis*. Beijing: Beijing University of Posts and Telecommunications Press, 2009.

[13] Zhang, Y., and Cai, B., *Research Progress of Artificial* Neural *Network and Argument of Publication Process*, Beijing: Electronic Industry Press, 2010.

[14] Chen, M., Matlab Neural Network Principle and Examples *of Fine Solution*. Beijing: Tsinghua University Press, 2012.

[15] He, Z., *Matlab R2015b Neural Network Technology*,Beijing: Tsinghua University Press, 2016.

[16] Wang, X., Shi, F. and Yu, L., *43 Cases of Matlab Neural* Network *Analysis*, Beijing: Beihang University Press, 2013.