Function approximation based on BP neural networks (Oct. 2019)

Ke. Liu,*3170105244,Automation1703*

***Abstract*—In the paper, BP artificial neural network (BP ANN) is used to approximate two functions. The BP ANN algorithm will be implemented with both matlab in the form of matrix and C language in sequence. Besides, the number of neurons in the hidden layer will be considered. With the increasing of training times, the error will be recorded and drawed in picture.**

***Index Terms*—Error Back Propagation, Function approximation, Sequencial and batch mode**

I. Introduction

BP artificial neural network is a basic algorithm in artificial neural networks. In order to be more proficient with it, in this paper, it will be used to approximate two functions: equation(1) with one input and one output and a more complex function,equation(2) with two input and one output.

$$f(x) = sinx, x \in [0, 2\pi] \tag{1}$$

$$f(x_1, x_2) = \frac{x_1}{sin(x_1)} \times \frac{x_2}{sin(x_2)}, x_1, x_2 \in [-10, +10] \tag{2}$$

To deal with the two functions, Matlab and C language are both used to try different approaches. Training data sets are input into neural network for training in two ways: sequence and group. Matlab is used to carry out the "group" way because of its well optimized matrix operation. And C language is used to handle the data sets input in sequence.

The main contributions of this paper are highlightes as follows.

1) Topology is firstly determined as a single layer which is easy but proper to deal with the question: funcion approximation. And the derivation of the equations is showed.
2) Experiment is carried out and the detail of code will be explained.
3) After experiment, the results are analyzed: the differences between squencial and batch mode, different number of neurons in hidden layer, different training times and different training sequence.

2019/10/3
Liuke
Student ID 3170105244
Tel 18888922995

II. Theory of BP neural network in this paper

Function approximation is a simple task for neural networks, so a neural network with single hidden layer is used in this paper. The symbols used in this paper is explained in tableI

Function3 sigmoid is used as activation function, the derivative of which is equation4

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

$$f'(x) = f(x)(1 - f(x)) \tag{4}$$

TABLE I
Explanation of symbols

Symbol	Meaning
n	Number of input variables
m	Number of test data
k	Number of hidden neurons
ω_{jp}	Connection power between hidden and input layer
l	Number of input variables
b_j	Hidden bias level
x	Input variable
ω_{oj}	Connection power between hidden and output layer
y	Output variable
b_o	Output bias level
t	Desired output
net_j	Output of the j^{th} hidden neuron
α, η	learning rates

So, for the i^{th} input data, the following can be derived.

A. Feed Forward

The input of j^{th} hidden neuron is:

$$net_j = \sum_{p=1}^{n} \omega_{jp} * x_p + b_j$$

So, its output is $f(net_j)$ The final output of bp model is:

$$y(i) = \sum_{j=1}^{k} \omega_{oj} * f(net_j) + b_o$$

and the error is defined as:

$$E(i) = \frac{1}{2}(t(i) - y(i))^2$$

B. Error Back Propagation

Modification gradient of the connection power between hidden and output layer can be calculated:

$$\frac{\partial E(i)}{\partial \omega_{oj}} = -[t(i)-y(i)]\frac{\partial y(i)}{\partial \omega_{oj}} = -[t(i)-y(i)]f(net_j)$$

Modification gradient of the bias level of output layer is:

$$\frac{\partial E(i)}{b_o} = -[t(i)-y(i)]$$

Modification gradient of the connection power between hidden and input layer can be calculated:

$$\begin{aligned}\frac{\partial E(i)}{\partial \omega_{jp}} &= -[t(i)-y(i)]\frac{\partial y(i)}{\partial \omega_{jp}} \\ &= -[t(i)-y(i)] * \omega_{oj} * f(net_j) * [1-f(net_j)]x_p\end{aligned}$$

Modification gradient of the bias level of hidden layer is:

$$\frac{\partial E(i)}{\partial b_j} = -[t(i)-y(i)] * \omega_{oj} * f(net_j) * [1-f(net_j)]$$

C. Renew weights and bias

$$\omega_{oj} = \omega_{oj} - \alpha\frac{\partial E(i)}{\partial \omega_{oj}}$$

$$b_o = b_o - \eta\frac{\partial E(i)}{b_o}$$

$$\omega_{jp} = \omega_{jp} - \alpha\frac{\partial E(i)}{\partial \omega_{jp}}$$

$$b_j = b_j - \eta\frac{\partial E(i)}{\partial b_j}$$

D. Conclusion

Generally, the main steps of BP ANN algorithm are

1) Feed forward, calculate the output and error.
2) Error back propagation, calculate the partial derivative of each weight and bias.
3) Renew the weights and bias.
4) Repeat the steps above until we get the desired output which means it has been trained for enough times or its error is small enough.

III. Approach

This part mainly includes the approachs to approximate two functions1 and 2

A. Sine function

To approximate sine function, both sequencial and batch mode are used in this paper. And both two ways worked effictively.

The topology is a single hidden layer ANN as figure1. And 9 pairs of equidistant points on the sine function are train data, and 361 pairs of equidistant points on the sine function are test data.

Fig. 1. ANN topology of approximating sine function

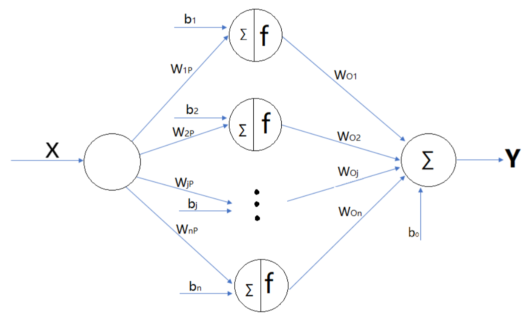
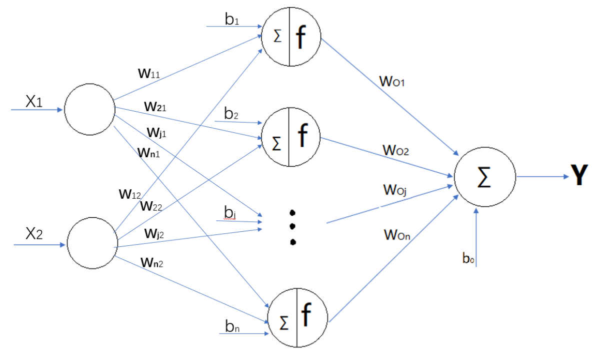


Fig. 2. ANN toppology of approximating bivariate function



1) batch mode: Batch mode is used because MATLAB is very efficient with matrix operations. Detail is showed in the matlab code in the appendix.

First of all, the equations derived in partII are realised. A slight difference from the equations is that train datas are input as a groud which means I input 9 pairs of train data and calculate total error each time.

And then, back propagate it to correct the weights and bias and record the changes of error.

Next, repeat the above steps until we get the desire output.

Finally, input the test data, get the actual output and calculate the error. The result is showed in the next part.

2) sequential mode: In sequential mode, test data is input one by one and the error is back propagated each time the train is input.

In order to complete sequential mode and make it faster, the program is realised in C language because there are more loops in this mode and C language is better at it.

The main program is to realise the equations in partII. And different number of hidden neuron is tried. We used 3, 6, 9 hidden neurons to test the changes of error, the final weights and bias.

B. Bivariate function

In this paper, bivariate function approximation is programmed in C language. The topology is shown as figure2

To complete the program, we just need to change the input number into two, and change the train data.

The topology is also a single hidden layer ANN. And 11×11 pairs of equidistant points on the sine function are train data, and 21×21 pairs of equidistant points on the sine function are test data. Test results result are plotted in figure, and the error of each test data is also plotted in figure to show the error visually.

In order to approximate it properly, ANNs with different number of hidden neurons are tryed. The training process, test results, train results are recorded in a document and are plotted in figures with matlab. For each group of test data, it is trained for 1.5e7 times.

IV. Experiment

The results are showed and analyzed in this part.

A. Sine function

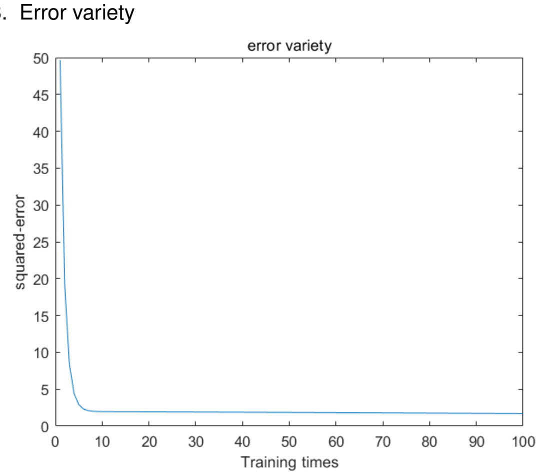
Sine function is approximated in two ways.

1) Batch mode: In this mode, I assign $\alpha = 0.01, \eta = 0.001$, $n = 1$, $k = 6$(Number of hidden neurons). And $\omega_p, \omega_o, b_o, b_j$ are random number in range from 0 to 1.

Finally, after training for 10^7 times, I get the parameters:
ω_p= [0.848777831080972 0.928310147037578 -1.27448783754993 0.586438641109643 0.558997384234099 1.33973057483642],
ω_o =[6.00186850394577 -2.19466051919320 3.14978676919043 -2.73889712218513 -3.86185657349878 6.10550031708145],
b_j =[-5.43767398781235 0.495640209234760 4.37093543610278 -1.02847722652615 1.62657955895883 -0.288697855554323],
b_o = -0.438974811087581
And the train error is 3.0724e-5;the test error is 1.4464e-5.

Using MATLAB to program, and plot the picture of error changes with the training times, as figure3 And input test data,

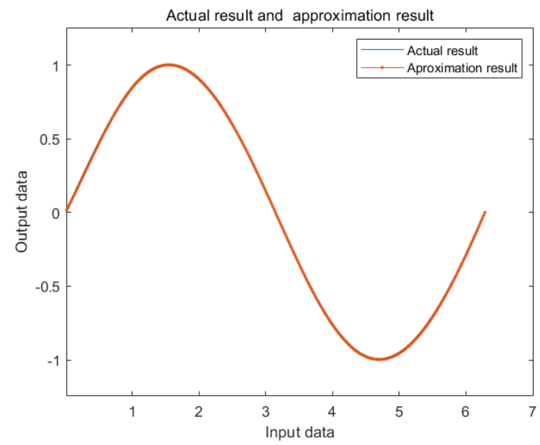
Fig. 3. Error variety



we get the result in figure4. Obviously, the actual output of ANN and the desired output is almost coincide.

In order to figure out the difference between different learning rates, I use the ANNs with 6 hidden neurons, but change the leaning rates.

Fig. 4. Test output and desired output



i $\alpha = 0.1$ $\eta = 0.01$. And the results are shown as figure5 and figure6 .And finally, the test error is 2.218e-4 and train error is 1.1099e-5.

Fig. 5. Test output and desired output

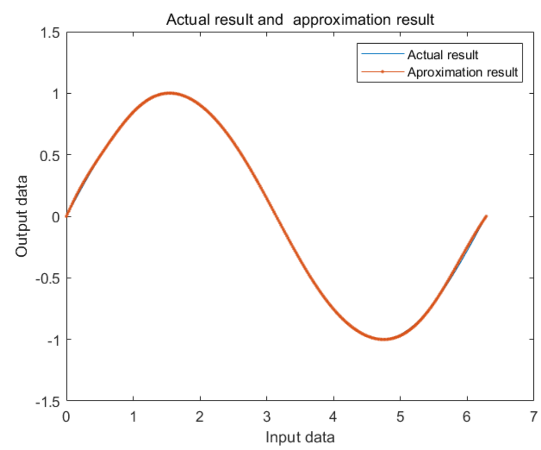
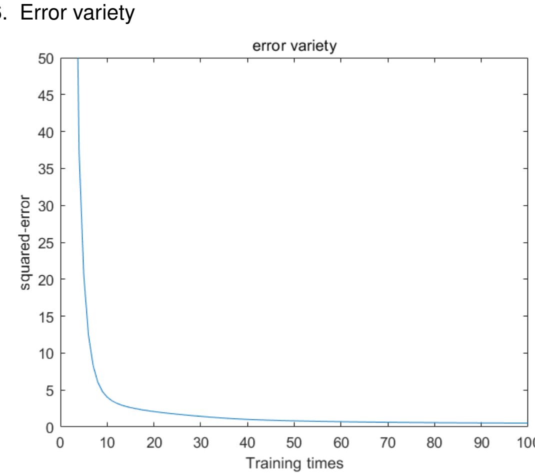


Fig. 6. Error variety



ii $\alpha = 0.05$ $\eta = 0.005$. And the results are shown as figure7 and figure8 .And finally, the test error is 1.4363e-6 and train error is 1.1733e-6.

Fig. 7. Test output and desired output

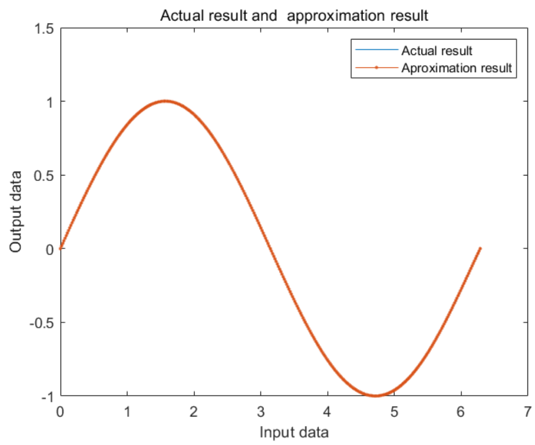


Fig. 8. Error variety

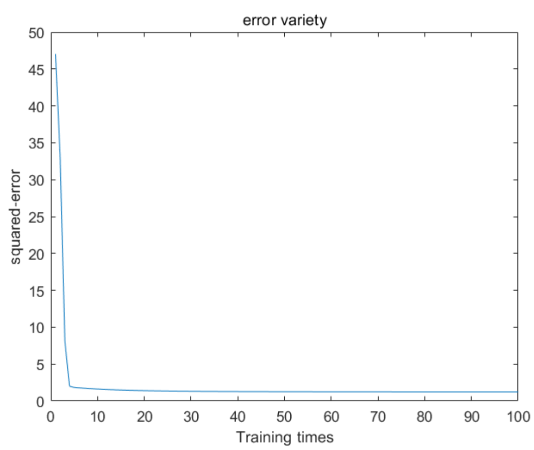


Fig. 9. Error variety(3 hidden neurons)

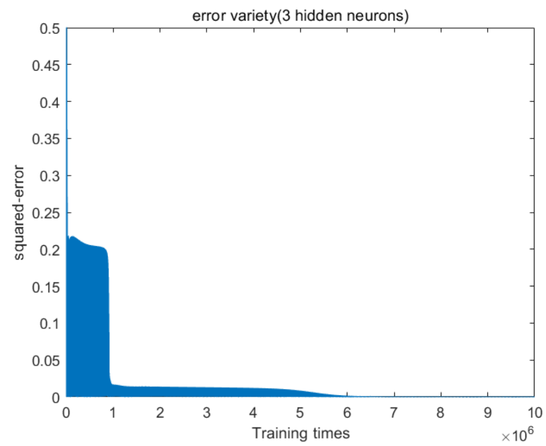
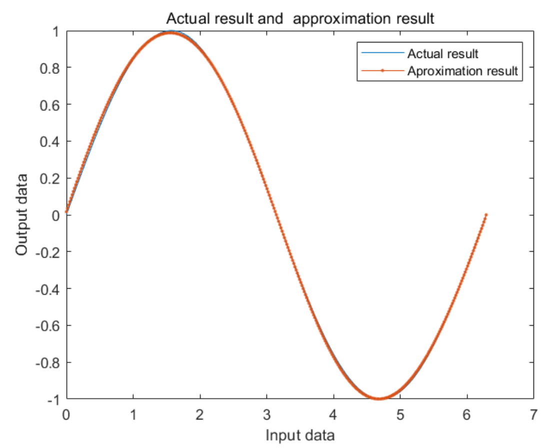


Fig. 10. Test output and desired output(3 hidden neurons)



2) Sequencial mode: In this mode, C language is used to program, and different number of hidden neurons are tried.

i 3 hidden neurons.
I assign $\alpha = 0.01$, $\eta = 0.001$, $n = 1$, $k = 3$. And $\omega_p, \omega_o, b_o, b_j$ are random number in range from 0 to 1. Finally, after training for 10^7 times, the Train Error = 2e-6, and Test Error = 3.5e-5.
Using MATLAB to program, and plot the picture of error changes with the training times, as figure9 And input test data, we get the result in figure10. Obviously, the actual output of ANN and the desired output is almost coincide.

ii 6 hidden neurons.
I assign $\alpha = 0.01$, $\eta = 0.001$, $n = 1$, $k = 6$. And $\omega_p, \omega_o, b_o, b_j$ are random number in range from 0 to 1. Finally, after training for 10^7 times, the Train Error = 10e-6, and Test Error = 2.3e-5.
Using MATLAB to program, and plot the picture of error changes with the training times, as figure11 And input test data, we get the result in figure12. Obviously, the actual output of ANN and the desired output is almost coincide.

iii 9 hidden neurons.
I assign $\alpha = 0.01$, $\eta = 0.001$, $n = 1$, $k = 9$. And $\omega_p, \omega_o, b_o, b_j$ are random number in range from 0 to 1. Finally, after training for 10^7 times, the Train Error is almost zero, and Test Error = 1.7e-5.
Using MATLAB to program, and plot the picture of error changes with the training times, as figure13 And input test data, we get the result in figure14. Obviously, the actual output of ANN and the desired output is almost coincide.

iv conclusion
As is shown above, all the three ANN with different number of hidden neurons can approximate the function perfectly. And it is more accurate with more hidden neurons(in this paper).

B. Bivariate function

To approximate bivariate, C language is used. And the results of 9, 12, 24 hidden neurons with 1.5e7 times training are shown below. The learning rates of the models above are $\alpha = 0.0001, \eta = 0.00001$. The desired output is shown as figure15. Besides, for this bivariate function, computer has a problem with calculated$\frac{sin(0)}{0} \times \frac{sin(0)}{0}$, which is $\frac{0}{0}$. This problem is solved in this way: when the input x_1 or x_2 is zero, it will be added a number, eps which is the precision of

Fig. 11. Error variety(6 hidden neurons)

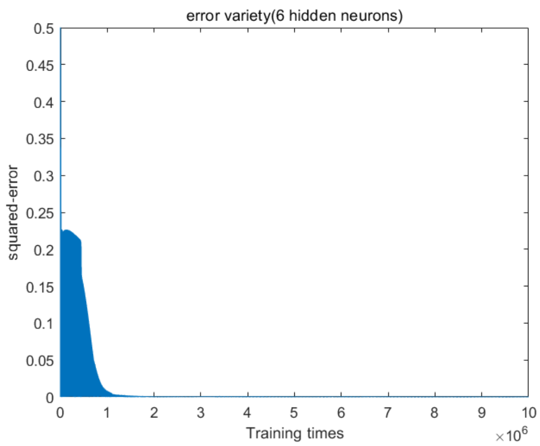
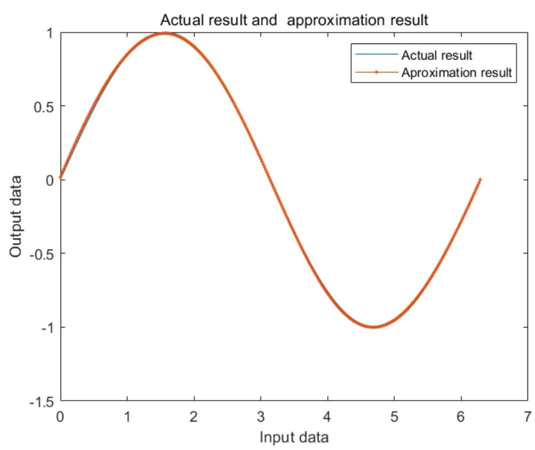


Fig. 12. Test output and desired output(6 hidden neurons)



double precision data. In this way computer can deal with all the input data.

i 9 hidden neurons
In this model, the train error is 3.244e-3, and test error is 3.962e-3. The results are shown as figure16, and the error between actual and desired output is shown as figure 17, the variate of error with the training times is shown as figure 18. Only the former 450 times are shown because the latter times vary very little.

ii 12 hidden neurons
In this model, the train error is 2.435e-3, and test error is 6.4e-3. The results are shown as figure19, and the error between actual and desired output is shown as figure 20, the variate of error with the training times is shown as figure 21.

iii 24 hidden neurons
In this model, the train error is 9.98e-4, and test error is 9.42e-4. The results are shown as figure22, and the error between actual and desired output is shown as figure 23, the variate of error with the training times is shown as figure 24.

iv In order to figure out the difference between different learning rates, I use the ANNs with 24 hidden neurons, but change the leaning rates.

Fig. 13. Error variety(9 hidden neurons)

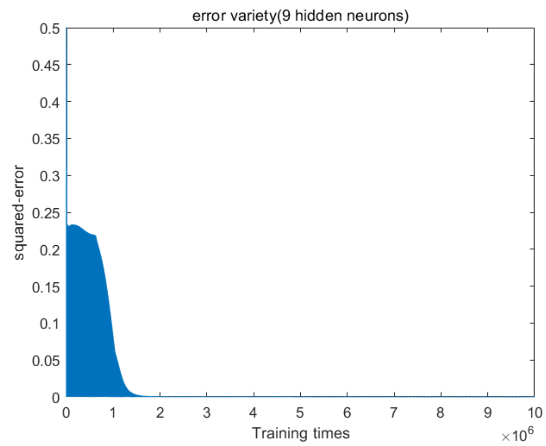
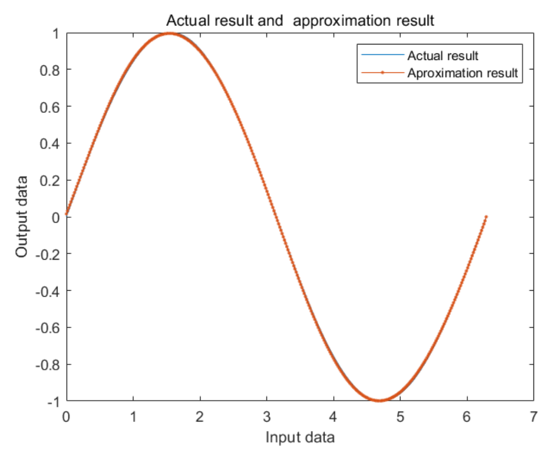


Fig. 14. Test output and desired output(9 hidden neurons)



a) $\alpha = 0.001$ $\eta = 0.0001$. And the results are shown as figure25, figure26 and figure27. And finally, the test error is 1.37e-4 and train error is 1.06e-4.

v Conclusion
As is shown above, the results of ANNs with more hidden neurons are more accurate. Although the learing rate gets bigger, the results get more accurate.

C. Analysis and discussion

i For Sine function, it is much easier to approximate, all the three ANNs with 3, 6, 9 hidden neurons can approximate it perfectly. But for the bivariate function, only the ANNs with 24 can approximate it properly in this experiment. So, we may come to a conclusion, the more complex the funtion is, the more hidden neurons we need to approximate it properly.

ii Compare the error of ANNs with different number of neurons, we come to a conclusion, the more hidden neurons the ANNs have, the more accurate the results are.

iii With a bigger learning rate, the error of ANNs get convergence more quickly, and in the bivariate function in this paper, the error is also smaller, but in the sine function the error is bigger. In my opinion, that

Fig. 15. Desired output

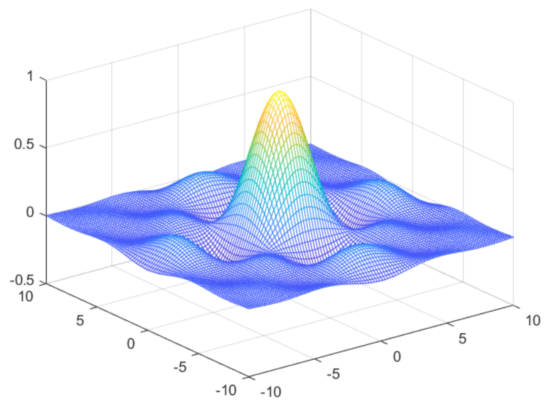
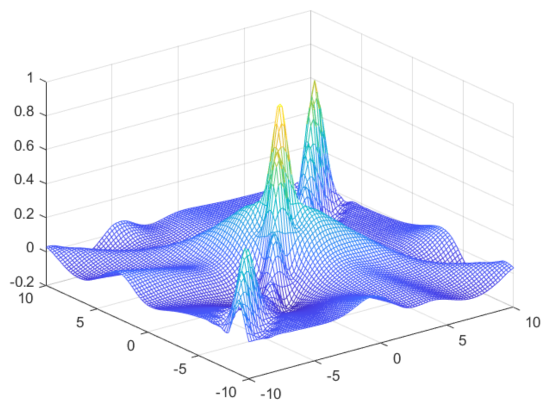


Fig. 16. actual output(9 hidden neurons)



is because for sine funcion, we can approximate it properly and easily, so the accurance depends on its learning rates, but for the bivariate function, changing the learning rates bigger may help it get out of the local optimum solution and get a better solution.

V. Conclusion

In this paper, the influence of input mode, learning rates, number of hidden neurons, and the training times is discussed. Through empirical research, this paper draws the following conclusions:

A. input mode

No matter it is sequencial mode or batch mode, both two ways can get the proper solution. if we are familiar with matrix operations, or the program language we use is better at matrix, we are willing to use batch mode.

B. Learning rates

As is shown in part IV-B, we are supposed to select a proper learning rate. On one hand, if it is too small, the error will decrease very slowly and the solution may be just local optimum. On the other hand, if it is too big, the final result

Fig. 17. Error between actual and desire output(9 hidden neurons)

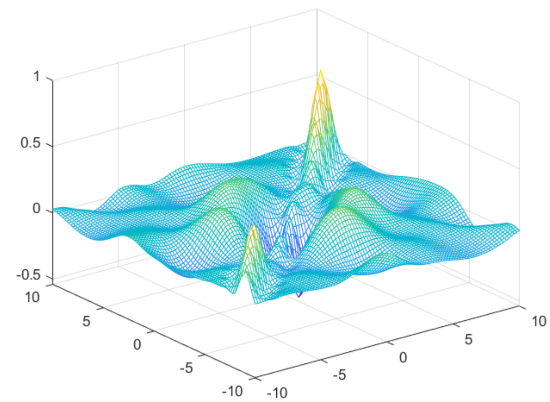
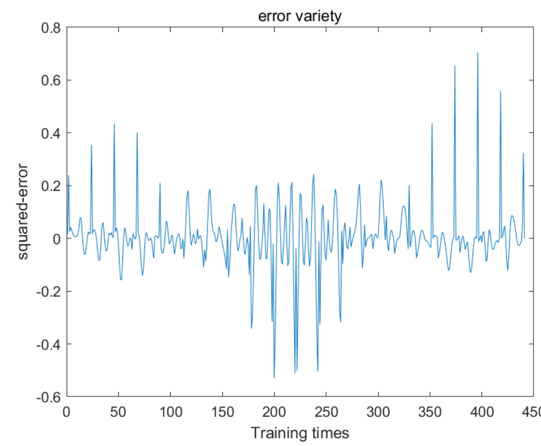


Fig. 18. Error variety(9 hidden neurons)



can not be very accurate. So it is also a great deal to select a proper learning rate. In this paper, I just use a hard way, and try different learning rates many times to observe the changes of train error. A great approach to select a initial learning rate is shown in Leslie N. Smith's paper, ”Cyclical learning rates for training Neural Networks”.

C. Number of hidden neurons

In this paper, the results of ANNs with more hidden neurons are more accurate. The empirical formulas of the hidden neuron number are:

$$m = \sqrt{n+l} + \alpha$$

$$m = log_2 N$$

$$m = \sqrt{nl}$$

where m is the number of hidden neurons, n is the number of input data, l is the number of output number and α is a constant between 0 and 10. So I try 3 numbers between the minimum and maximum number.

Fig. 19. actual output(12 hidden neurons)

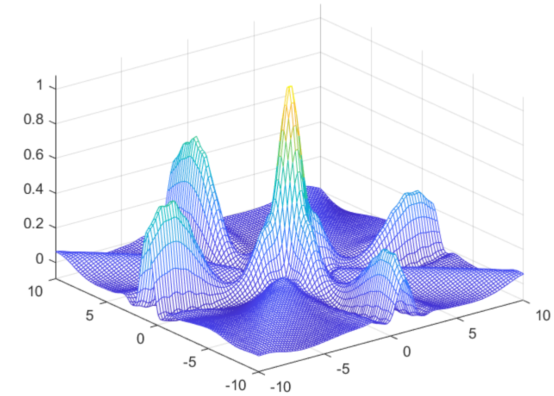
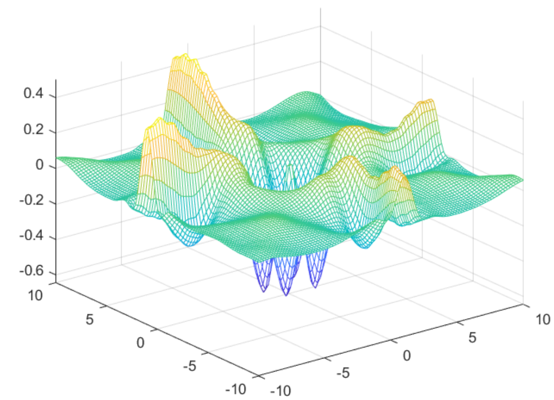


Fig. 20. Error between actual and desire output(12 hidden neurons)



D. the training times

The figures of error changes are shown above. Obviously, the error get smaller while the training times increase. But, the error decreases not equably, which means it decreases sharply in some period but in the latter times, it decreases just a little and spends a long time. So, in my opinion, we are supposed to set a bias. When the error is small enough, we will stop training it, which will save much time.

VI. Reference

1. G. Jiang, ”Research on Credit Rating Method Based on BP NN,” 2007 International Conference on Service Systems and Service Management, Chengdu, 2007, pp. 1-4. doi: 10.1109/ICSSSM.2007.4280185

2. Smith L N . Cyclical Learning Rates for Training Neural Networks[C]// 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2017.

Fig. 21. Error variety(12 hidden neurons)

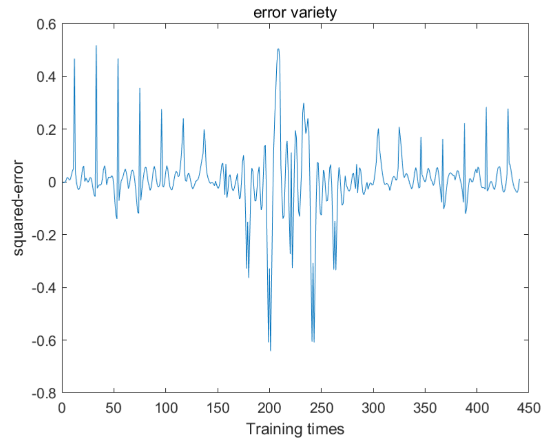


Fig. 22. actual output(24 hidden neurons)

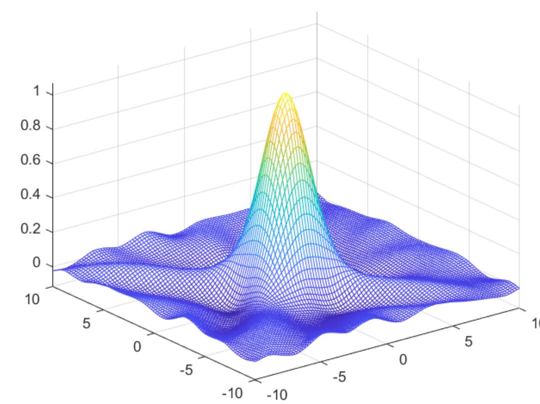


Fig. 23. Error between actual and desire output(24 hidden neurons)

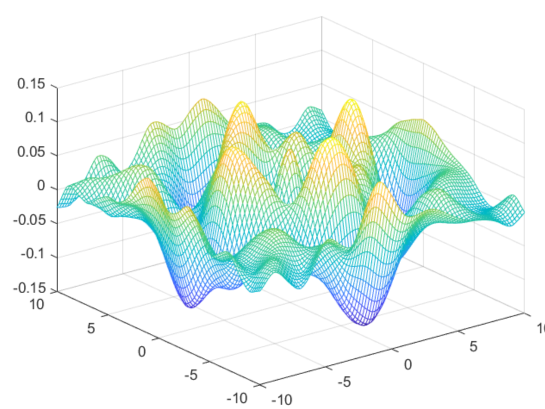


Fig. 24. Error variety(24 hidden neurons)

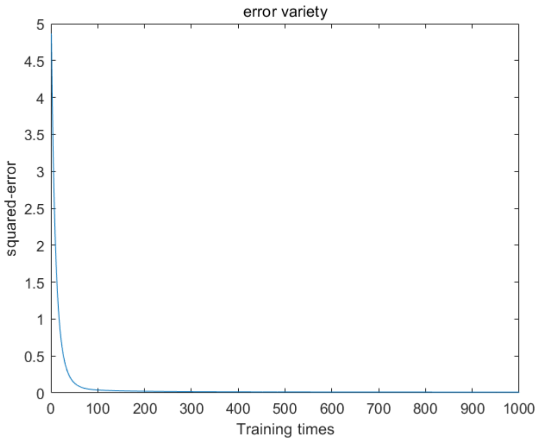


Fig. 25. Actual output

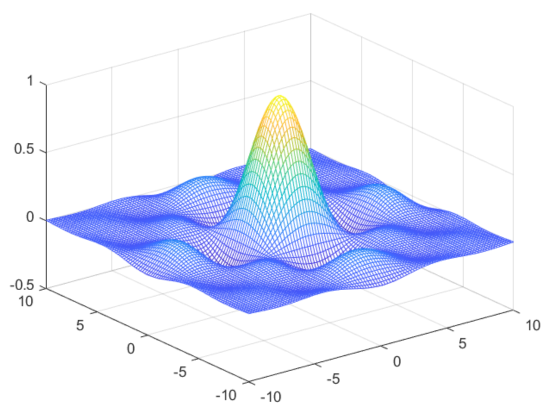


Fig. 26. Error between actual and desired output

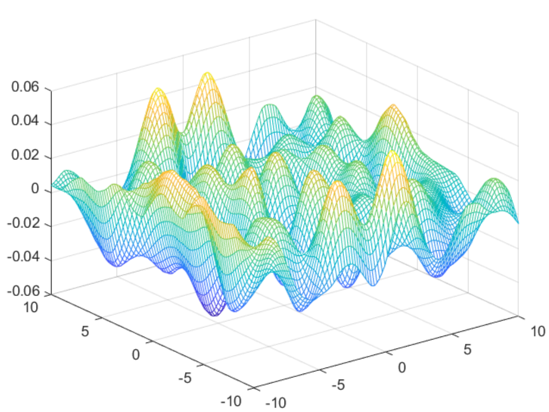
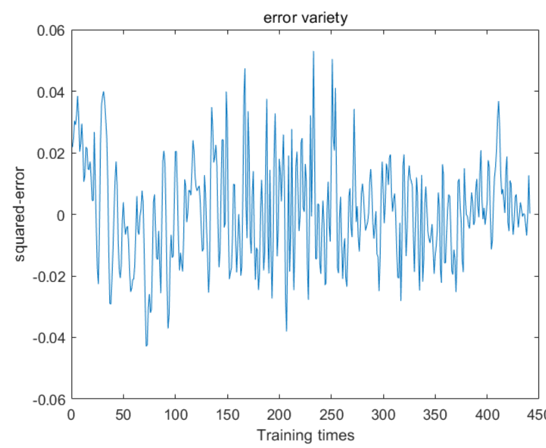


Fig. 27. Variety of error



VII. Appendix

The code used in this paper is shown as below

A. Batch mode :MLP1_sin.m

```
clear
clc

%%
%Train data
Traindata_I = linspace(0,2*pi,9);
Traindata_T = sin(Traindata_I);
%test data
Testdata_I = linspace(0,2*pi,361);
Testdata_T = sin(Testdata_I);

%%
%initial
HiddenNeuron = 6; %Number of hidden
    neurons
MaxTimes = 10000000; %Maximum number of
    training
MinErr =0.00000001; %Margin of error
W1 = rand(HiddenNeuron,1); %Initial the
    weight of output of input neurons
W2 = rand(1,HiddenNeuron); %Initial the
    weight of output of hidden neurons
Theta1 = rand(HiddenNeuron,1);
Theta2 = rand(1);
%Fix the random data, so we can compare
    the other elements
%W1 =
    [0.959492426392903;0.655740699156587;0.0357116785741896;0.849129305868777;0.933993247757551

%W2 =
    [0.757740130578333,0.743132468124916,0.392227019534168,0.655477890177557,0.171186687811562,

%Theta1 =
    [0.0318328463774207;0.276922984960890;0.0461713906311539;0.0971317812358475;0.823457828327293

%Theta2 = 0.317099480060861;

Alpha =0.05;% %0.01;  0.1;          %
    learning rate
Eta = 0.005; %0.001;   0.01;
   %floating item
ErrRecord = zeros(1,MaxTimes);
%Train
for i = 1:MaxTimes
HiddenOutput = logsig(W1 * Traindata_I +
    Theta1); %Each column is the
%output of Hidden layer
ActualOutput = W2 * HiddenOutput + Theta2
    ; %A row vector
Err = Traindata_T - ActualOutput;
ErrorSum = sumsqr(Err)/2;
ErrRecord(i) = ErrorSum;
if ErrorSum <MinErr          %Accurate
    enough
break;
end
%Error back propagation
delta2 = HiddenOutput * Err'; %Column
    Vector,Output layer
delta_Theta2 = sum(Err);
%delta1 = sum(ActualOutput .* (1 -
    ActualOutput) .*Err )* W2; %Row vetor,
temp1 = ones(1,9);
delta1 =  HiddenOutput.*(1 - HiddenOutput
    ).*(W2'*temp1)*(Err .* Traindata_I)';
delta_Theta1 = HiddenOutput.*(1 -
    HiddenOutput).*(W2'*temp1)*Err';
%Change the value of W1,W2 and Theta;
W2 = W2 + Alpha * delta2';
W1 = W1 + Alpha * delta1 ;
Theta1 = Theta1 + Eta * delta_Theta1;
Theta2 = Theta2 + Eta * delta_Theta2;
end

%%
%calculate the results
temp2 = ones(1,361);
ActualOutputOfNN = W2 * logsig(W1 *
    Testdata_I + Theta1) + Theta2;
TestErr = sum((ActualOutputOfNN -
    Testdata_T).^2)*0.5/361;
%plot pictures
figure
plot(Testdata_I,Testdata_T);
hold on
plot(Testdata_I, ActualOutputOfNN,'.-');
title('Actual result and  approximation
    result')
xlabel('Input data')
ylabel('Output data')
legend('Actual result','Aproximation
    result');
%Test the accuracy
figure
plot(1:MaxTimes, ErrRecord);
title('error variety')
xlabel('Training times')
ylabel('squared-error')
axis([0,100 , 0,50]);
```

B. sequencial mode: sequencial.c

```
/*BP neural networks trained in sequence
    */

#include<stdio.h>
#include<math.h>
#include<stdlib.h>

```

```
double sigmod(double x);
double test(double* weight1, double* weight2, double* theta1, double theta2);

constexpr auto PI = 3.1415926;
constexpr auto Hidden =9;
constexpr auto MaxTime = 10000000;

void main( )
{
/*initial the train data*/
double TrainInput[9] = { 0 , 0.25 * PI ,0.5 * PI, 0.75 * PI,  PI, 1.25 * PI, 1.5 * PI, 1.75 * PI, 2 * PI };
double TrainOutput[9] = { 0, 0.707106781186548,1,0.707106781186548, 0 , -0.707106781186548, -1, -0.707106781186548, 0 };
/**********************/
/***The train process****/
//initial the weight and bias(Fix the random data, so we can compare the other elements)
double W1[Hidden] = { 0.959492426392903, 0.655740699156587, 0.0357116785741896 , 0.849129305868777, 0.933993247757551, 0.678735154857774 , 0.655740699156587, 0.0357116785741896 , 0.849129305868777 };
double Theta1[Hidden] = { 0.757740130578333, 0.743132468124916, 0.392227019534168 ,0.655477890177557,0.171186687811562,0.706046088019609 , 0.743132468124916, 0.392227019534168,0.655477890177557 };
double W2[Hidden] = { 0.0318328463774207, 0.276922984960890, 0.0461713906311539 , 0.0971317812358475, 0.823457828327293, 0.694828622975817 , 0.276922984960890, 0.0461713906311539 , 0.0971317812358475 };
double Theta2 = 0.317099480060861;
double Alpha = 0.01;
double Eta = 0.001;
double ErrRecord[MaxTime];
//define the process variable
double HiddenOutput[Hidden];
int RandNum;

for (int count1 = 0; count1 < MaxTime; count1++){
//RandNum = rand() % 9; //9 reprents the total number of input
RandNum = count1 % 9;
for (int count2 = 0; count2 < Hidden; count2++) {
HiddenOutput[count2] = sigmod(W1[count2] * TrainInput[RandNum] + Theta1[count2]);
}//calculate the output of Hidden layer

double ActualOutput = 0;
for (int count3 = 0; count3 < Hidden; count3++) {
ActualOutput = ActualOutput + W2[count3] * HiddenOutput[count3];
}//calculate the Actualoutput
ActualOutput += Theta2;
double Error = TrainOutput[RandNum] - ActualOutput;
ErrRecord[count1] = 0.5 * pow(TrainOutput[RandNum] - ActualOutput, 2);

//Update W1,W2, Theta1, Theta2
Theta2 += (Eta * Error);
for (int count4 = 0; count4 < Hidden; count4++) {

W1[count4] = W1[count4] + Alpha * Error * HiddenOutput[count4] * (1 - HiddenOutput[count4]) * W2[count4] * TrainInput[RandNum];
Theta1[count4] = Theta1[count4] + Eta * Error * HiddenOutput[count4] * (1 - HiddenOutput[count4]) * W2[count4];
//The above sentences execute first in case W2 changes.
W2[count4] = W2[count4] + Alpha * Error * HiddenOutput[count4];
}
if(count1%100000==0)
printf("Train Error = %lf\n", ErrRecord[count1]);

}

double TestErr = test(W1, W2, Theta1, Theta2);
printf("Test Error = %lf", TestErr);
/*save data to txt files */
FILE* fp;
fp = fopen("D:\\err.txt", "w");
for (int i = 0; i < MaxTime; i++) {
fprintf(fp, "%lf\n", ErrRecord[i]);
}
fclose(fp);

FILE* fp3;
fp3 = fopen("D:\\weight.txt", "w");
for (int i = 0; i < Hidden; i++) {
fprintf(fp3, "%lf\n", W1[i]);
}
for (int i = 0; i < Hidden; i++) {
fprintf(fp3, "%lf\n", W2[i]);
```

```
}
for (int i = 0; i < Hidden; i++) {
fprintf(fp3, "%lf\n", Theta1[i]);
}
fprintf(fp3, "%lf\n", Theta2);
fclose(fp3);

}

/*
    ****************************************
    */
/*This function is used to calculate
    sigmod*/
double sigmod(double x) {
return 1.0 / (1.0 + pow(2.71828183, -x));
}
/*
    ****************************************
    */
/*This function is used to test and
    return the correction rate*/
double test(double* weight1, double*
    weight2, double* theta1, double theta2
    )
{
double temp[361] = {0};
double Error = 0;
int i = 0;
for (; i < 361; i++)
{
double input = 2.0 * PI *i / 361;
double HiddenOutput[Hidden];
for (int count2 = 0; count2 < Hidden;
    count2++) {
HiddenOutput[count2] = sigmod(weight1[
    count2] * input + theta1[count2]);
}
double ActualOutput = 0;
for (int count3 = 0; count3 < Hidden;
    count3++) {
ActualOutput += weight2[count3] *
    HiddenOutput[count3];
}
ActualOutput += theta2;
temp[i - 1] = ActualOutput;
Error += (0.5 * pow((ActualOutput - sin(
    input)), 2));
}

FILE* fp2;
fp2 = fopen("D:\\approximation.txt","w");
for (int i = 0; i < 361; i++) {
fprintf(fp2, "%lf\n", temp[i]);
}
fclose(fp2);

double AveErr = Error * 1.0 / i;
return AveErr;
}
```

C. Bivariate function approxiamation:xsinxxsinx.c

```
/*BP neural networks trained in
    sequence*/

#include<stdio.h>
#include<math.h>
#include<stdlib.h>

double sigmod(double x);
double test(double* weight1,
    double* weight2, double*
    theta1, double theta2);
double DesiredOutput(double x1,
    double x2);

constexpr auto Hidden = 24;//
    6;//9;//12;
constexpr auto InputNum = 2;
constexpr auto MaxTime =
    15000000;
constexpr auto eps =
    0.0000000000000001;
void main()
{
/*initial the train data*/
double X1[22] = { -10, -8, - 6, -
     4, - 2, 0, 2, 4,       6, 8,
    10,-10, -8, -6, -4, - 2, 0, 2,
     4, 6, 8, 10 };
//output data can be calculated
    we it is used;
/***********************/
/****The train process****/
//initial the weight and bias(Fix
     the random data, so we can
    compare the other elements)
double W1[Hidden * InputNum];
double Theta1[Hidden];
double W2[Hidden];
double Theta2 = ((rand() % 10) -
    5) / 5.0;
for (int i = 0; i < Hidden; i++)
    {
for (int j = 0; j < InputNum; j
    ++) {
W1[j*Hidden + i] = ((rand() % 10)
     - 5) / 5.0;
}
W2[i] = ((rand() % 10) - 5) /
    5.0;
Theta1[i] = ((rand() % 10) - 5) /
     5.0;
}
```

```
        //double W1[Hidden * InputNum] = { 0.959492426392903, 0.655740699156587, 0.0357116785741896 , 0.849129305868777, 0.933993247757551, 0.678735154857774 ,0.959492426392903 , 0.655740699156587, 0.0357116785741896 , 0.849129305868777, 0.933993247757551, 0.678735154857774 };//,0.959492426392903, 0.655740699156587, 0.0357116785741896 //, 0.849129305868777, 0.933993247757551, 0.678735154857774 };//,0.959492426392903 , 0.655740699156587, 0.0357116785741896 ,// 0.849129305868777, 0.933993247757551, 0.678735154857774 };
        //double Theta1[Hidden] = { 0.757740130578333, 0.743132468124916, 0.392227019534168,0.655477890177557,0.171186687811562,0.706046088019609 };//,0.757740130578333, 0.743132468124916 , 0.392227019534168 };//,0.655477890177557,0.171186687811562,0.706046088019609 };
        //double W2[Hidden] = { 0.0318328463774207, 0.276922984960890, 0.0461713906311539 , 0.0971317812358475, 0.823457828327293, 0.694828622975817 };//,0.0318328463774207, 0.276922984960890, 0.0461713906311539 };//, 0.0971317812358475, 0.823457828327293, 0.694828622975817 };
        //double Theta2 = 0.317099480060861;
        double Alpha = 0.001;//0.0001;
        double Eta = 0.0001;//0.00001;
        double ErrRecord[MaxTime];
        //define the process variable
        double HiddenOutput[Hidden];
        double HiddenInput[Hidden];
        double DesiredOut;
        int RandNum1;
        int RandNum2;

            for (int count1 = 0; count1 < MaxTime; count1++) {
            double ErrSum = 0;
            for (RandNum1 = 0; RandNum1 < 11; RandNum1++) {
            for (RandNum2 = 0; RandNum2 < 11; RandNum2++) {
            double Input[2] = { X1[RandNum1], X1[RandNum2] };

            DesiredOut = DesiredOutput(Input[0], Input[1]);

            for (int count2 = 0; count2 < Hidden; count2++) {
            HiddenInput[count2] = Input[0] * W1[count2] + Input[1] * W1[count2 + Hidden] + Theta1[count2];
            }//calculate the input of Hidden layer

            for (int count3 = 0; count3 < Hidden; count3++) {
            HiddenOutput[count3] = sigmod(HiddenInput[count3]);
            }//calculate the output of Hidden layer

            double ActualOutput = 0;
            for (int count4 = 0; count4 < Hidden; count4++) {
            ActualOutput = ActualOutput + W2[count4] * HiddenOutput[count4];
            }//calculate the Actualoutput
            ActualOutput += Theta2;

            double Error = DesiredOut - ActualOutput;

            ErrSum = ErrSum + 0.5 * pow(Error, 2);
            //Update W1,W2, Theta1, Theta2
            Theta2 += (Eta * Error);
            for (int count5 = 0; count5 < Hidden * InputNum; count5++) {
            W1[count5] = W1[count5] + Alpha * Error * HiddenOutput[count5 % Hidden] * (1 - HiddenOutput[count5 % Hidden]) * W2[count5 % Hidden] * Input[count5 / Hidden];
            //The above sentences execute first in case W2 changes.
            }
```

```
for (int count6 = 0; count6 < Hidden; count6++) {
Theta1[count6] = Theta1[count6] + Eta * Error * HiddenOutput[count6] * (1 - HiddenOutput[count6]) * W2[count6];
W2[count6] = W2[count6] + Alpha * Error * HiddenOutput[count6];
}
}
}
ErrRecord[count1] = ErrSum *1.0/121;
if(count1%1000000 == 0)
printf("%lf \n", ErrRecord[count1]);
}
printf("TestError = %lf\n", test(W1, W2, Theta1, Theta2));
FILE* fp1;
fp1 = fopen("D:\\err.txt", "w");
for (int i = 0; i < MaxTime; i++) {
fprintf(fp1, "%lf \n", ErrRecord[i]);
}
fclose(fp1);
}

/* ****************************************** */
/*This function is used to calculate sigmod*/
double sigmod(double x) {
return 1.0 / (1.0 + pow(2.71828183, -x));
}
/* ****************************************** */
/*This function is used to test and return the correction rate */
double test(double* weight1, double* weight2, double* theta1, double theta2)
{
double Error = 0;
double d;
double HiddenInput[Hidden];
double HiddenOutput[Hidden];
/*initial the test data sets*/
double testI[21] = { -10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
//double testI[11] = { -10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10 };

FILE* fp2;
fp2 = fopen("D:\\approximation.txt", "w");
for (int i = 0; i < 21; i++) {
for (int j = 0; j < 21; j++) {
d = DesiredOutput(testI[i], testI[j]);
for (int count2 = 0; count2 < Hidden; count2++) {
HiddenInput[count2] = testI[i] * weight1[count2] + testI[j] * weight1[count2 + Hidden] + theta1[count2];
}//calculate the input of Hidden layer
for (int count3 = 0; count3 < Hidden; count3++) {
HiddenOutput[count3] = sigmod(HiddenInput[count3]);
}//calculate the output of Hidden layer
double ActualOutput = 0;
for (int count4 = 0; count4 < Hidden; count4++) {
ActualOutput = ActualOutput + weight2[count4] * HiddenOutput[count4];
}//calculate the Actualoutput
ActualOutput = ActualOutput + theta2;
//printf("%lf\n ", (ActualOutput - d));
//printf("%lf, %lf, %lf, %lf ,%lf\n", testI[i], testI[j], ActualOutput, d, (ActualOutput - d));
Error += 0.5 * pow((ActualOutput - d), 2);
fprintf(fp2, "%lf, %lf, %lf, %lf ,%lf\n", testI[i], testI[j], ActualOutput, d, (ActualOutput - d));
}
fprintf(fp2, "\n");
}
fclose(fp2);
double AveErr = Error * 1.0 / 441;
return AveErr;
}

/* ****************************************** */
```

```
        /*This function is used to
            calculate the desired output
            of x1,x2*/
        double DesiredOutput(double x1,
            double x2)
        {
        if (x1 == 0 || x2 == 0) {
        x1 += eps;
        x2 += eps;
        }
        return (sin(x1) * sin(x2))* 1.0
            / (x1 * x2);
        }
```

D. Functions

i PlotPicture.m

```
        clear
        clc

        approximation = load('
            approximation.txt');
        err = load('err.txt');
        weight = load('weight.txt');

        figure
        plot(linspace(0,2*pi,361),sin(
            linspace(0,2*pi,361)));
        hold on
        plot(linspace(0,2*pi,361),
            approximation,'.-');
        title('Actual␣result␣and␣␣
            approximation␣result')
        xlabel('Input␣data')
        ylabel('Output␣data')
        legend('Actual␣result','
            Aproximation␣result');

        %Test the accuracy
        figure
        plot(1:length(err), err);
        title('error␣variety(9␣hidden␣
            neurons)')
        xlabel('Training␣times')
        ylabel('squared-error')
        axis([0,length(err) , 0,0.5]);
```

ii PlotPic.m

```
        clear
        clc

        table = load('approximation.txt')
            ;
        err = load('err.txt');
        x = table(1:end,1);
        y = table(1:end, 2);
        z = table(1:end, 3);
        d = table(1:end, 4);
        err = table(1:end,5);

        [X,Y,Z] = griddata(x,y,z,linspace
            (min(x),max(x))',linspace(min(
            y),max(y)),'v4');
        figure
        mesh(X,Y,Z)
        saveas(gcf , "ActualOutput,png");

        [U,V,W] = griddata(x,y,d,linspace
            (min(x),max(x))',linspace(min(
            y),max(y)),'v4');
        figure
        mesh(U,V,W)
        saveas(gcf , "DesiredOutput,png")
            ;

        [B,N,M] =griddata(x,y,err,
            linspace(min(x),max(x))',
            linspace(min(y),max(y)),'v4');
        figure
        mesh(B,N,M)
        saveas(gcf , "Error,png");

        figure
        plot(1:length(err), err);
        title('error␣variety')
        xlabel('Training␣times')
        ylabel('squared-error')
        saveas(gcf, "Err.png")
```