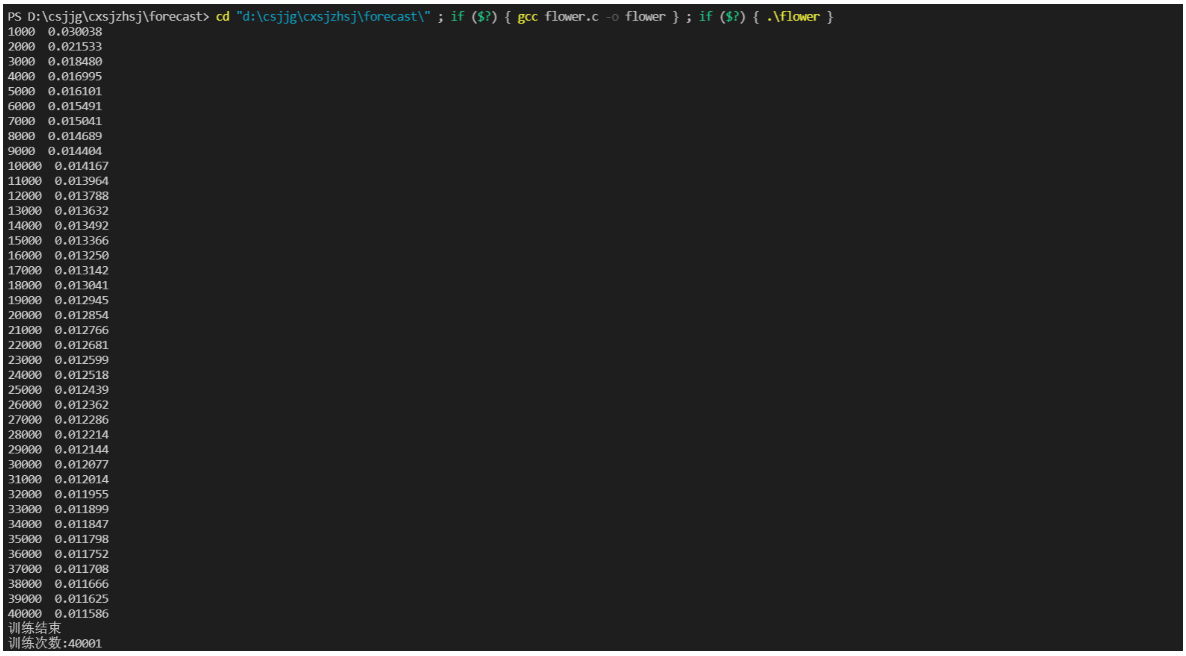
```
王一凡，李品成  花分类作业3层bp
#include<stdio.h>
#include<math.h>
#include<time.h>
#include<stdlib.h>
#include<string.h>
#define Data 113                //训练样本个数
#define TestData 37             //测试样本个数
#define In 4                    //输入层神经元个数（花萼长度，花萼宽度，花瓣长度，花瓣宽度）
#define Out 3                   //输出层神经元个数（样本为三种鸢尾花的可能性（0-1））
#define Neuron 5                //隐含层神经元个数
#define TrainC 40000            //最大训练次数
#define WAlta 0.1               //权值w的学习率
#define VAlta 0.1               //权值v的学习率
struct node {
    double in[In];
    double out[Out];
};
struct node iris[TestData + Data];//存储所有样本数据
double d_in[Data][In];          //训练样本输入
double d_out[Data][Out];        //训练样本输出
double t_in[TestData][In];      //测试样本输入
double t_out[TestData][Out];    //测试样本输出
double pre[TestData][Out];      //预测样本的实际输出
double v[Neuron][In];           //输入层到隐含层的权值
double y[Neuron];               //隐含层
double w[Out][Neuron];          //隐含层到输出层的权值
double Maxin[In], Minin[In];    //样本输入的最大值和最小值
double Maxout[Out],Minout[Out]; //样本输出的最大值和最小值
double OutputData[Out];         //神经网络的输出
double dw[Neuron][Out], dv[Neuron][In];//权值w和v的修正量
const char* Name[3] = { "Iris-setosa","Iris-versicolor","Iris-virginica" };//鸢尾花种类的名字
double mse;                     //均方误差
double rmse;
double ermse;                   //均方根误差
void ReadData() {               //读取文件中的数据
    srand(time(NULL));
    FILE* fp;
    char ch;
    char name[20];
    if ((fp = fopen("data.txt", "rb")) == NULL) {
        printf("不能打开data.txt文件\n");
        exit(0);
    }
    int n = TestData + Data;
    for (int i = 0; i <n ; ++i) {   //转移数据到结构体数组
        for (int j = 0; j < In; ++j) {
            if (j != 0) {
                fscanf(fp, "%c", &ch);
            }
            fscanf(fp, "%lf", &iris[i].in[j]);
        }
```

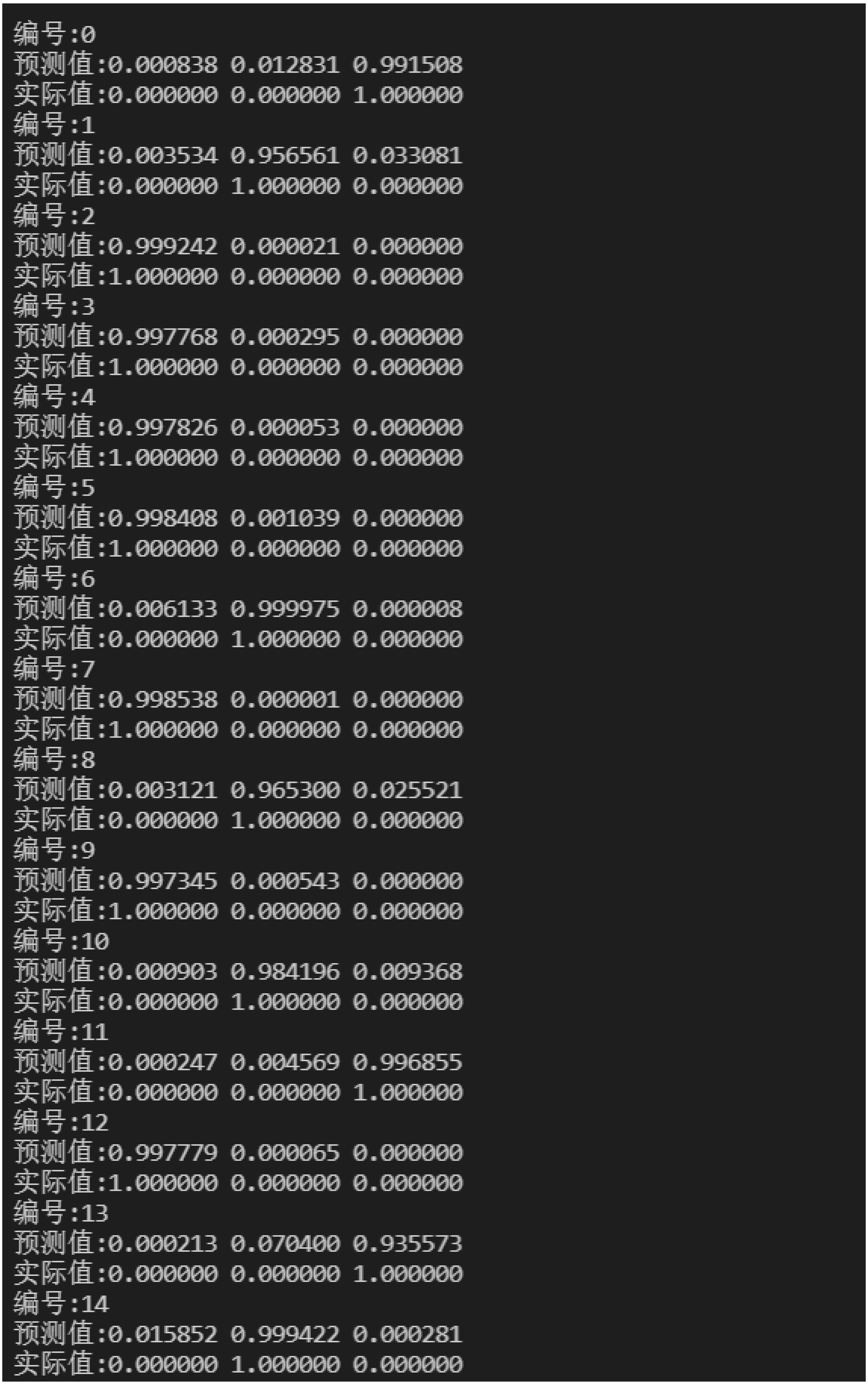
```c
        fscanf(fp, "%c", &ch);
        fscanf(fp, "%s", name);
        memset(iris[i].out, 0, sizeof(iris[i].out));
        for (int k = 0; k < 3; ++k) {   //种类维度的数据变换
            if (strcmp(Name[k],name)==0) {
                iris[i].out[k] =1;
                break;
            }
        }
    }
    struct node tmp;
    int t=0;
    for (int i = 0; i < n; ++i) {       //打乱数据
        t = rand() % n;
        tmp = iris[i];
        iris[i] = iris[t];
        iris[t] = tmp;
    }
    for (int i = 0; i < Data; ++i) {    //数据划分为训练集的输入和输出，测试集的输入和输出
        for (int j = 0; j < In; ++j) {
            d_in[i][j] = iris[i].in[j];
        }
        for (int j = 0; j < Out; ++j) {
            d_out[i][j] = iris[i].out[j];
        }
    }
    for (int i = Data; i < TestData+Data; ++i) {
        for (int j = 0; j < In; ++j) {
            t_in[i-Data][j] = iris[i].in[j];
        }
        for (int j = 0; j < Out; ++j) {
            t_out[i-Data][j] = iris[i].out[j];
        }
    }
    fclose(fp);
}
void InitBPNetwork() {
    srand(time(NULL));
    for (int i = 0; i < In; ++i) {              //寻找输入输出的最大值
        Maxin[i] = Minin[i] = d_in[0][i];
        for (int j = 0; j < Data; ++j) {
            Maxin[i] = Maxin[i] > d_in[j][i] ? Maxin[i] : d_in[j][i];
            Minin[i] = Minin[i] < d_in[j][i] ? Minin[i] : d_in[j][i];
        }
    }
    Maxout[0] = Maxout[1] = Maxout[2] = 1;      //最大可能为1，最小可能为0
    Minout[0] = Minout[1] = Minout[2] = 0;
    for (int i = 0; i < In; ++i) {              //根据最大最小值对数据进行归一化
        for (int j = 0; j < Data; ++j) {
            d_in[j][i] = (d_in[j][i] - Minin[i]) / (Maxin[i] - Minin[i]);
        }
    }
    for (int i = 0; i < Out; ++i) {             //归一化
        for (int j = 0; j < Data; ++j) {
            d_out[j][i] = (d_out[j][i] - Minout[i]) / (Maxout[i] - Minout[i]);
        }
    }
    for (int i = 0; i < Neuron; ++i) {          //使用随机值初始化权值
        for (int j = 0; j < In; ++j) {
            v[i][j] = rand() * 2.0 / RAND_MAX - 1;
            dv[i][j] = 0;
        }
    }
    for (int i = 0; i < Out; ++i) {             //使用随机值初始化权值
        for (int j = 0; j < Neuron; ++j) {
            w[i][j] = rand() * 2.0 / RAND_MAX - 1;
            dw[j][i] = 0;
        }
    }
}
void ComputO(int var) {                 //前向传播
    double sum;
    for (int i = 0; i < Neuron; ++i) {
        sum = 0;                        //存储累加和
        for (int j = 0; j < In; ++j) {  //计算隐含层每个神经元的输出
            sum += v[i][j] * d_in[var][j];
        }
        y[i] = 1 / (1 + exp(-1 * sum));
    }
    for (int i = 0; i < Out; ++i) {     //计算输出层每个神经元的输出
        sum = 0;
        for (int j = 0; j < Neuron; ++j) {
            sum += w[i][j] * y[j];
        }
        OutputData[i] = 1 / (1 + exp(-1 * sum));
    }
}
void BackUpdata(int var) {              //反向传播的权值修正
    double t;
    for (int i = 0; i < Neuron; ++i) {
        t = 0;
        for (int j = 0; j < Out; ++j) { //修正隐含层与输出层之间的各权值
            dw[j][i] = WAlta * (d_out[var][j] - OutputData[j]) * OutputData[j] * (1 - OutputData[j]) * y[i];
            t += (d_out[var][j] - OutputData[j]) * OutputData[j] * (1 - OutputData[j]) * w[j][i];
            w[j][i] += dw[j][i];
        }
        for (int j = 0; j < In; ++j) {  //修正隐含层与输出层之间的各权值
            dv[i][j] = VAlta * t * y[i] * (1 - y[i]) * d_in[var][j];
            v[i][j]+=dv[i][j];
        }
    }
}
void TrainNetwork() {                   //神经网络的训练
    int count = 1;                      //记录训练次数
    do {
        mse = 0;                        //均方误差设置为0
        for (int i = 0; i < Data; ++i) {//所有训练集的一轮训练
            ComputO(i);                 //前向传播
            BackUpdata(i);              //反向传播，调整权值
            for (int j = 0; j < Out; ++j) { //计算单个样本误差
```

```
                double tmp1 = OutputData[j] * (Maxout[j] - Minout[j]) + Minout[j];
                double tmp2 = d_out[i][j] * (Maxout[j] - Minout[j]) + Minout[j];
                mse += pow(tmp1 - tmp2, 2.0);
            }
        }
        mse /= (double)Data * Out;      //计算均方误差
        if (count % 1000 == 0) {        //每1000此训练，显示一次训练误差，以便观察
            printf("%d  %lf\n", count, mse);
        }
        count++;
    } while (count <= TrainC && mse >= 0.01);//训练次数或达到要求，训练结束
    printf("训练结束\n");
    printf("训练次数:%d\n", count);
}
void TestNetwork() {            //神经网络模型评估
    for (int i = 0; i < In; ++i) {  //预测数据归一化
        for (int j = 0; j < TestData; ++j) {
            t_in[j][i] = (t_in[j][i] - Minin[i]) / (Maxin[i] - Minin[i]);
        }
    }
    double sum;
    int m = 0;
    for (int k = 0; k < TestData; ++k) {//计算每一个样本的预测结果
        for (int i = 0; i < Neuron; ++i) {//计算隐含层输出
            sum = 0;
            for (int j = 0; j < In; ++j) {
                sum += v[i][j] * t_in[k][j];
            }
            y[i] = 1 / (1 + exp(-1 * sum));
        }
        for (int i = 0; i < Out; ++i) {//计算输出层的预测结果
            sum = 0;
            for (int j = 0; j < Neuron; ++j) {
                sum += w[i][j] * y[j];
            }
            pre[k][i] = 1 / (1 + exp(-1 * sum)) * (Maxout[i] - Minout[i]) + Minout[i];
        }
        double max = 0;
        int t=0;
        printf("\n编号:%d\n",k);      //进行对比
        printf("预测值:");
        for (int i = 0; i < Out; ++i) {
            printf("%lf ", pre[k][i]);
            if (pre[k][i] > max) {
                max = pre[k][i];
                t = i;
            }
        }
        if (t_out[k][t] == 1) {
            m++;
        }
        printf("\n实际值:");
        for (int i = 0; i < Out; ++i) {
            printf("%lf ", t_out[k][i]);
        }
    }
    double summse=0.0;
    for (int i = 0; i < Out; ++i) { //计算均方根误差，预测准确率
        double t = 0.0;
        for (int k = 0; k < TestData; ++k) {
            t +=pow(pre[k][i] - t_out[k][i], 2.0);
        }
        summse += sqrt(t / TestData);
    }
    rmse = summse / Out;
    printf("\nrmse: %.4lf,准确率:%lf\n", rmse,(double)m/TestData);
}
int main() {//主函数
    ReadData();
    InitBPNetwork();
    TrainNetwork();
    TestNetwork();
    return 0;
}
```

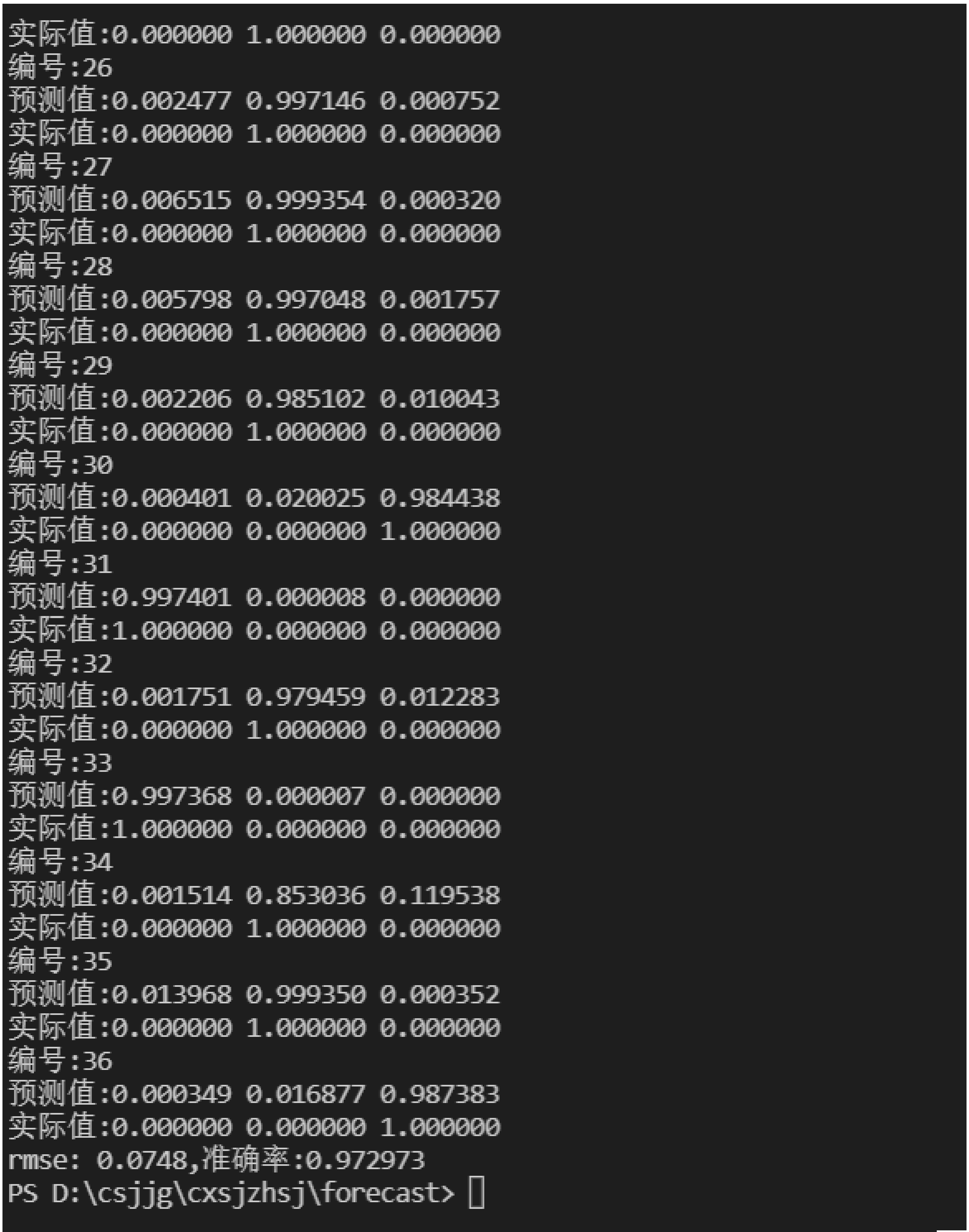
王



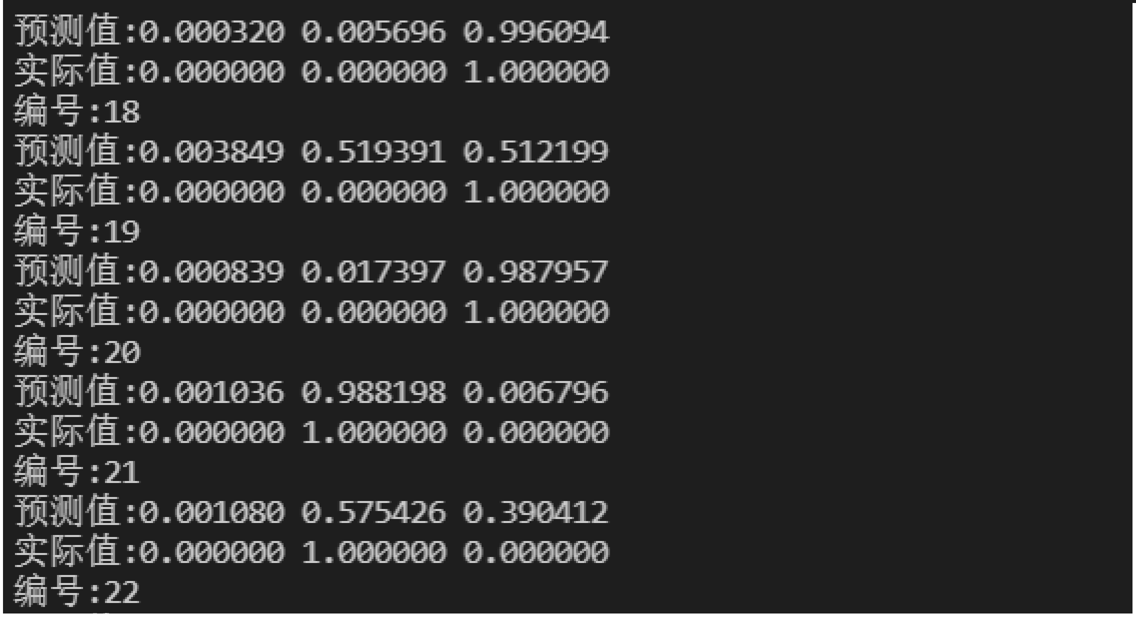
```
PS D:\csjjg\cxsjzhsj\forecast> cd "d:\csjjg\cxsjzhsj\forecast\" ; if ($?) { gcc flower.c -o flower } ; if ($?) { .\flower }
1000  0.030038
2000  0.021533
3000  0.018480
4000  0.016995
5000  0.016101
6000  0.015491
7000  0.015041
8000  0.014689
9000  0.014404
10000  0.014167
11000  0.013964
12000  0.013788
13000  0.013632
14000  0.013492
15000  0.013366
16000  0.013250
17000  0.013142
18000  0.013041
19000  0.012945
20000  0.012854
21000  0.012766
22000  0.012681
23000  0.012599
24000  0.012518
25000  0.012439
26000  0.012362
27000  0.012286
28000  0.012214
29000  0.012144
30000  0.012077
31000  0.012014
32000  0.011955
33000  0.011899
34000  0.011847
35000  0.011798
36000  0.011752
37000  0.011708
38000  0.011666
39000  0.011625
40000  0.011586
训练结束
训练次数:40001
```



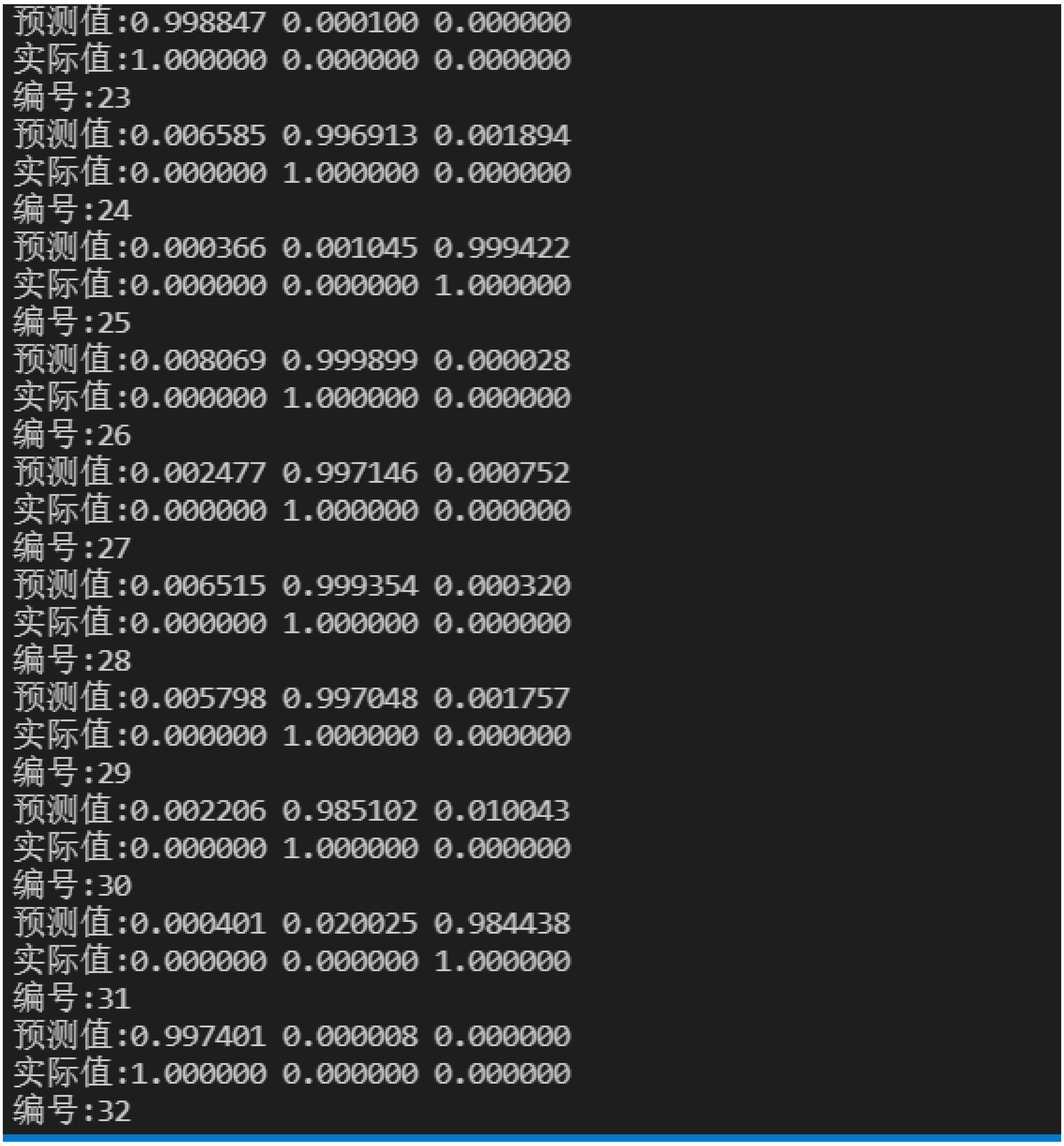
```
编号:0
预测值:0.000838 0.012831 0.991508
实际值:0.000000 0.000000 1.000000
编号:1
预测值:0.003534 0.956561 0.033081
实际值:0.000000 1.000000 0.000000
编号:2
预测值:0.999242 0.000021 0.000000
实际值:1.000000 0.000000 0.000000
编号:3
预测值:0.997768 0.000295 0.000000
实际值:1.000000 0.000000 0.000000
编号:4
预测值:0.997826 0.000053 0.000000
实际值:1.000000 0.000000 0.000000
编号:5
预测值:0.998408 0.001039 0.000000
实际值:1.000000 0.000000 0.000000
编号:6
预测值:0.006133 0.999975 0.000008
实际值:0.000000 1.000000 0.000000
编号:7
预测值:0.998538 0.000001 0.000000
实际值:1.000000 0.000000 0.000000
编号:8
预测值:0.003121 0.965300 0.025521
实际值:0.000000 1.000000 0.000000
编号:9
预测值:0.997345 0.000543 0.000000
实际值:1.000000 0.000000 0.000000
编号:10
预测值:0.000903 0.984196 0.009368
实际值:0.000000 1.000000 0.000000
编号:11
预测值:0.000247 0.004569 0.996855
实际值:0.000000 0.000000 1.000000
编号:12
预测值:0.997779 0.000065 0.000000
实际值:1.000000 0.000000 0.000000
编号:13
预测值:0.000213 0.070400 0.935573
实际值:0.000000 0.000000 1.000000
编号:14
预测值:0.015852 0.999422 0.000281
实际值:0.000000 1.000000 0.000000
```



```
实际值:0.000000 1.000000 0.000000
编号:26
预测值:0.002477 0.997146 0.000752
实际值:0.000000 1.000000 0.000000
编号:27
预测值:0.006515 0.999354 0.000320
实际值:0.000000 1.000000 0.000000
编号:28
预测值:0.005798 0.997048 0.001757
实际值:0.000000 1.000000 0.000000
编号:29
预测值:0.002206 0.985102 0.010043
实际值:0.000000 1.000000 0.000000
编号:30
预测值:0.000401 0.020025 0.984438
实际值:0.000000 0.000000 1.000000
编号:31
预测值:0.997401 0.000008 0.000000
实际值:1.000000 0.000000 0.000000
编号:32
预测值:0.001751 0.979459 0.012283
实际值:0.000000 1.000000 0.000000
编号:33
预测值:0.997368 0.000007 0.000000
实际值:1.000000 0.000000 0.000000
编号:34
预测值:0.001514 0.853036 0.119538
实际值:0.000000 1.000000 0.000000
编号:35
预测值:0.013968 0.999350 0.000352
实际值:0.000000 1.000000 0.000000
编号:36
预测值:0.000349 0.016877 0.987383
实际值:0.000000 0.000000 1.000000
rmse: 0.0748,准确率:0.972973
PS D:\csjjg\cxsjzhsj\forecast> 
```



```
预测值:0.000320 0.005696 0.996094
实际值:0.000000 0.000000 1.000000
编号:18
预测值:0.003849 0.519391 0.512199
实际值:0.000000 0.000000 1.000000
编号:19
预测值:0.000839 0.017397 0.987957
实际值:0.000000 0.000000 1.000000
编号:20
预测值:0.001036 0.988198 0.006796
实际值:0.000000 1.000000 0.000000
编号:21
预测值:0.001080 0.575426 0.390412
实际值:0.000000 1.000000 0.000000
编号:22
```



```
预测值:0.998847 0.000100 0.000000
实际值:1.000000 0.000000 0.000000
编号:23
预测值:0.006585 0.996913 0.001894
实际值:0.000000 1.000000 0.000000
编号:24
预测值:0.000366 0.001045 0.999422
实际值:0.000000 0.000000 1.000000
编号:25
预测值:0.008069 0.999899 0.000028
实际值:0.000000 1.000000 0.000000
编号:26
预测值:0.002477 0.997146 0.000752
实际值:0.000000 1.000000 0.000000
编号:27
预测值:0.006515 0.999354 0.000320
实际值:0.000000 1.000000 0.000000
编号:28
预测值:0.005798 0.997048 0.001757
实际值:0.000000 1.000000 0.000000
编号:29
预测值:0.002206 0.985102 0.010043
实际值:0.000000 1.000000 0.000000
编号:30
预测值:0.000401 0.020025 0.984438
实际值:0.000000 0.000000 1.000000
编号:31
预测值:0.997401 0.000008 0.000000
实际值:1.000000 0.000000 0.000000
编号:32
```