****

**《程序设计课程实践》设计文档**

# 第6章 第1次作业

**学号： 21052115**

**姓名： 姜曜楠**

完成时间 2022 年 5 月

## 题目



## 程序设计思路

采用BP神经网络解决多分类问题，采用三层神经网络，结构为“4-5-3”

1. 全部数据读入到dataAll数组中，输出层采用one-hot编码
2. 数据预处理，包括打乱数据、划分训练集和测试集、Max-Min归一化
3. 初始化神经网络，包括权重初始化，范围为[-1,1]。偏置此处均不考虑
4. 训练神经网络，包括向前传播、向后传播、计算混淆矩阵进而计算准确率和召回率（此处两个指标只用来显示模型的训练是否有效）
5. 测试神经网络，包括向前传播、softmax处理输出层、输出每个数据元的预测结果和实际结果、计算混淆矩阵进而计算准确率和召回率

## 程序源码

main.c

1. #include <stdlib.h>
2. #include "irisBP.h"
3. int main(void)
4. {
5. ReadData();
6. InitBPNetwork();
7. TrainBPNetwork();
8. system("pause");
9. TestBPNetwork();
10. system("pause");
11. return 0;
12. }

irisBP.h

#ifndef IRISBP\_H

#define IRISBP\_H

#define DATAMAXN 150

#define DATATRAIN 113

#define DATATEST (DATAMAXN - DATATRAIN)

#define EPOCH\_MAX 10000

#define IN 4

#define OUT 3

#define NEURON 5

#define W01\_ALPHA 0.01

#define W12\_ALPHA 0.01

typedef char \*String;

void ReadData(void);

void InitBPNetwork(void);

void TrainBPNetwork(void);

void TestBPNetwork(void);

#endif

irisBP.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <math.h>

#include "irisBP.h"

String iris\_kinds[OUT] = {

    "Iris-setosa",

    "Iris-versicolor",

    "Iris-virginica",

};

double dataAll\_in[DATAMAXN][IN];

int dataAll\_out[DATAMAXN][OUT];

double maxIn[IN], minIn[IN];

double train\_in[DATATRAIN][IN];

double test\_in[DATATEST][IN];

double w01[IN][NEURON];

double neuron1[NEURON];

double w12[NEURON][OUT];

double pred\_out[OUT];

double softmax\_out[OUT];

int train\_out[DATATRAIN][OUT];

int test\_out[DATATEST][OUT];

int Matrix[OUT][OUT]; // Matrix[predition\_out][test\_out]

/\* 局部函数定义 \*/

void shuffle(void);

void divideData(void);

void normalizate(void);

/\* 接口函数实现 \*/

void ReadData(void)

{

    FILE \*fp = fopen("./data.txt", "rb");

    if (fp != NULL)

    {

        char ch;

        for (int i = 0; i < DATAMAXN; i++)

        {

            for (int j = 0; j < IN; j++)

            {

                if (j != 0)

                {

                    fscanf(fp, "%c", &ch);

                }

                fscanf(fp, "%lf", &dataAll\_in[i][j]);

            }

            char buffer[256];

            fscanf(fp, "%c%s", &ch, buffer);

            for (int j = 0; j < OUT; j++)

            {

                if (strcmp(buffer, iris\_kinds[j]) == 0)

                {

                    dataAll\_out[i][j] = 1;

                }

                else

                {

                    dataAll\_out[i][j] = 0;

                }

            }

        }

    }

    else

    {

        fputs("Open Error!", stderr);

    }

    fclose(fp);

}

void InitBPNetwork(void)

{

    shuffle();

    divideData();

    normalizate();

    srand((unsigned int)time(0));

    for (int i = 0; i < NEURON; i++)

    {

        for (int j = 0; j < IN; j++)

        {

            w01[j][i] = rand() % 2 - 1;

        }

        for (int j = 0; j < OUT; j++)

        {

            w12[i][j] = rand() % 2 - 1;

        }

    }

}

void TrainBPNetwork(void)

{

    int epoch = 0;

    double accuracy, recall;

    do

    {

        memset(Matrix, 0, sizeof(Matrix));

        for (int i = 0; i < DATATRAIN; i++)

        {

            /\* forward \*/

            for (int j = 0; j < NEURON; j++)

            {

                neuron1[j] = 0;

                for (int k = 0; k < IN; k++)

                {

                    neuron1[j] += train\_in[i][k] \* w01[k][j];

                }

                neuron1[j] = 1 / (1 + exp(-1 \* neuron1[j]));

            }

            for (int j = 0; j < OUT; j++)

            {

                pred\_out[j] = 0;

                for (int k = 0; k < NEURON; k++)

                {

                    pred\_out[j] += neuron1[k] \* w12[k][j];

                }

                pred\_out[j] = 1 / (1 + exp(-1 \* pred\_out[j]));

            }

            /\* backward \*/

            for (int j = 0; j < NEURON; j++)

            {

                double loss\_w01 = 0;

                for (int k = 0; k < OUT; k++)

                {

                    double loss = (pred\_out[k] - train\_out[i][k]) \* pred\_out[k] \* (1 - pred\_out[k]);

                    w12[j][k] -= W12\_ALPHA \* loss \* neuron1[j];

                    loss\_w01 += w12[j][k] \* loss;

                }

                for (int k = 0; k < IN; k++)

                {

                    double loss = loss\_w01 \* neuron1[j] \* (1 - neuron1[j]);

                    w01[k][j] -= W01\_ALPHA \* loss \* train\_in[i][k];

                }

            }

            /\* Matrix[prediction][truth] \*/

            int truth = 0, prediction = 0;

            for (int j = 1; j < OUT; j++)

            {

                prediction = pred\_out[j] > pred\_out[prediction] ? j : prediction;

                if (train\_out[i][j] == 1)

                {

                    truth = j;

                }

            }

            Matrix[prediction][truth]++;

        }

        /\* Accuracy \*/

        accuracy = recall = 0;

        for (int tru = 0; tru < OUT; tru++)

        {

            accuracy += (Matrix[tru][tru] / (double)DATATRAIN);

            double sum\_tru = 0;

            for (int pred = 0; pred < OUT; pred++)

            {

                sum\_tru += (double)Matrix[pred][tru];

            }

            recall += (Matrix[tru][tru] / sum\_tru) / OUT;

        }

        if (epoch % 1000 == 0)

        {

            printf("\nEPOCH: %d\tAccurary: %.3lf\tRecall: %.3lf", epoch, accuracy, recall);

        }

        epoch++;

    } while (epoch <= EPOCH\_MAX);

    printf("\nEnd Train Success");

}

void TestBPNetwork(void)

{

    double accuracy, recall;

    memset(Matrix, 0, sizeof(Matrix));

    for (int i = 0; i < DATATEST; i++)

    {

        /\* forward \*/

        for (int j = 0; j < NEURON; j++)

        {

            neuron1[j] = 0;

            for (int k = 0; k < IN; k++)

            {

                neuron1[j] += test\_in[i][k] \* w01[k][j];

            }

            neuron1[j] = 1 / (1 + exp(-1 \* neuron1[j]));

        }

        double lnC = 0;

        for (int j = 0; j < OUT; j++)

        {

            pred\_out[j] = 0;

            for (int k = 0; k < NEURON; k++)

            {

                pred\_out[j] += neuron1[k] \* w12[k][j];

            }

            pred\_out[j] = 1 / (1 + exp(-1 \* pred\_out[j]));

            lnC = pred\_out[j] > lnC ? pred\_out[j] : lnC;

        }

        double sum\_ex = 0;

        for (int j = 0; j < OUT; j++)

        {

            sum\_ex += exp(pred\_out[j] - lnC);

        }

        for (int j = 0; j < OUT; j++)

        {

            softmax\_out[j] = exp(pred\_out[j] - lnC) / sum\_ex;

        }

        /\* Test Data \*/

        printf("\nPrediction:");

        for (int j = 0; j < OUT; j++)

        {

            printf("%6.3lf", softmax\_out[j]);

        }

        printf("\nTruth Data:");

        for (int j = 0; j < OUT; j++)

        {

            printf("%6d", test\_out[i][j]);

        }

        /\* Matrix[prediction][truth] \*/

        int truth = 0, prediction = 0;

        for (int j = 1; j < OUT; j++)

        {

            prediction = softmax\_out[j] > softmax\_out[prediction] ? j : prediction;

            if (test\_out[i][j] == 1)

            {

                truth = j;

            }

        }

        Matrix[prediction][truth]++;

    }

    /\* Accuracy \*/

    accuracy = recall = 0;

    for (int tru = 0; tru < OUT; tru++)

    {

        accuracy += (Matrix[tru][tru] / (double)DATATEST);

        double sum\_tru = 0;

        for (int pred = 0; pred < OUT; pred++)

        {

            sum\_tru += (double)Matrix[pred][tru];

        }

        recall += (Matrix[tru][tru] / sum\_tru) / OUT;

    }

    printf("\nAccurary: %.3lf\tRecall: %.3lf", accuracy, recall);

}

/\* 局部函数实现 \*/

void shuffle(void)

{

    srand((unsigned int)time(0));

    for (int i = 0; i < DATAMAXN; i++)

    {

        int index1 = rand() % DATAMAXN;

        int index2 = rand() % DATAMAXN;

        while (index2 == index1)

        {

            index2 = rand() % DATAMAXN;

        }

        for (int j = 0; j < IN; j++)

        {

            double temp;

            temp = dataAll\_in[index1][j];

            dataAll\_in[index1][j] = dataAll\_in[index2][j];

            dataAll\_in[index2][j] = temp;

        }

        for (int j = 0; j < OUT; j++)

        {

            int temp;

            temp = dataAll\_out[index1][j];

            dataAll\_out[index1][j] = dataAll\_out[index2][j];

            dataAll\_out[index2][j] = temp;

        }

    }

}

void divideData(void)

{

    for (int i = 0; i < DATAMAXN; i++)

    {

        if (i < DATATRAIN)

        {

            for (int j = 0; j < IN; j++)

            {

                train\_in[i][j] = dataAll\_in[i][j];

            }

            for (int j = 0; j < OUT; j++)

            {

                train\_out[i][j] = dataAll\_out[i][j];

            }

        }

        else

        {

            for (int j = 0; j < IN; j++)

            {

                test\_in[i - DATATRAIN][j] = dataAll\_in[i][j];

            }

            for (int j = 0; j < OUT; j++)

            {

                test\_out[i - DATATRAIN][j] = dataAll\_out[i][j];

            }

        }

    }

}

void normalizate(void) // min-max normalizate

{

    for (int i = 0; i < IN; i++)

    {

        maxIn[i] = minIn[i] = train\_in[0][i];

        for (int j = 1; j < DATATRAIN; j++)

        {

            maxIn[i] = train\_in[j][i] > maxIn[i] ? train\_in[j][i] : maxIn[i];

            minIn[i] = train\_in[j][i] < minIn[i] ? train\_in[j][i] : minIn[i];

        }

        for (int j = 0; j < DATATRAIN; j++)

        {

            train\_in[j][i] = (train\_in[j][i] - minIn[i]) / (maxIn[i] - minIn[i]);

        }

        for (int j = 0; j < DATATEST; j++)

        {

            test\_in[j][i] = (test\_in[j][i] - minIn[i]) / (maxIn[i] - minIn[i]);

        }

    }

}

## 运行截图



