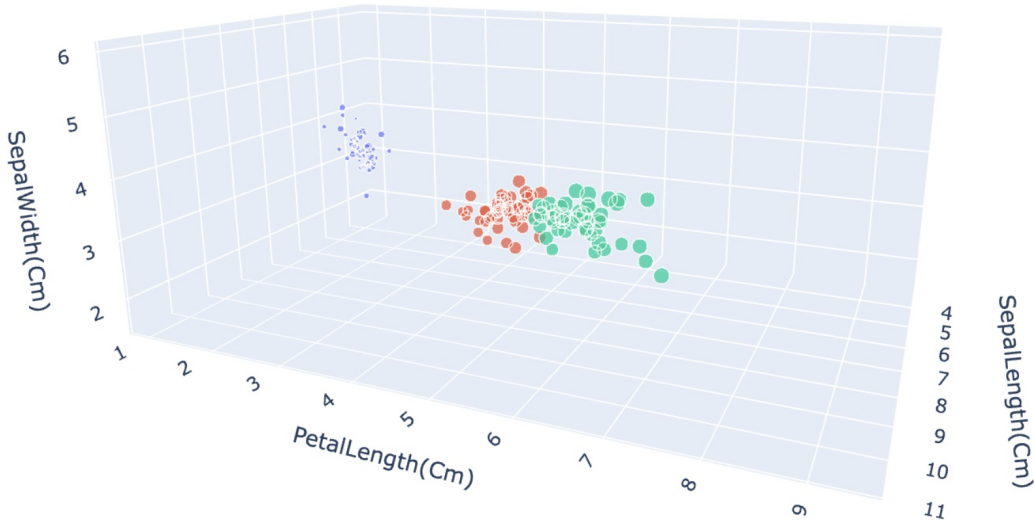


# C++程序设计大作业：误差反向传播算法

## 一、实验介绍

- 本次实验提供3类共150个鸢尾花样本（iris-setosa, iris-versicolor, iris-virginica），每个鸢尾花样本包括总共4个属性以及归属某一类的样本标签，保存在iris.data文件内。



- 通过误差反向传播算法（以下简称BP），从网络的输入层输入训练数据（鸢尾花的4个特征值），在输出层获得结果，将该结果与样本归属标签进行对比获得误差。然后根据误差对网络权重和偏置进行更新。通过大量的训练数据、大批量的迭代次数，使得网络参数收敛。
- 经过以上过程，使得当我们输入新的数据时，能够准确预测准确的分类结果，让该网络对于一类问题形成智能分类。

## 二、算法分析

### Step1.整体网络结构

$$a^0 = p$$

- 网络初始输入 $p$ 将会是一个4x1的行向量，每个分量对应鸢尾花样本的特征值

$$a^{m+1} = f^{m+1} (W^{m+1} a^m + b^{m+1}), m = 0, 1, \dots, M-1$$

- 每一层的输入会和所有神经元进行全连接，因此需要权值矩阵 $W^{m+1}$ 与输入向量 $a^m$ 相乘，同时加上bias矩阵 $b^{m+1}$ ，通过 $f^{m+1}(x)$ 映射输出，将数据归一化到一定区间

$$a = a^M$$

- 网络的输出 $a$ 将会是一个3x1的向量，在某个分量取到的值最大，意味着它从属于某个样本集合

### Step2.敏感度回传

$$s^M = -2\dot{F}^M (n^M) (t - a)$$

- 定义 $s$ 为回传敏感度， $s^M$ 为敏感度回传的起点，整个公式实际上是最均方误差函数进行矩阵求导后的结果， $\dot{F}^M$ 为每层神经网络输出对净输出 $n^M$ 的偏导

$$s^m = \dot{F}^m (n^m) (W^{m+1})^T s^{m+1}, m = M-1, \dots, 2, 1$$

- 由最终层计算出的 $s^M$ 将不断回传至开始

### Step3.最速下降更新参数

$$\begin{aligned} W^m(k+1) &= W^m(k) - l_r s^m (a^{m-1})^T \\ b^m(k+1) &= b^m(k) - l_r s^m \end{aligned}$$

- 在得到敏感度后，我们将其乘以一个学习率 $l_r$ （取值范围[0,1]）对于权值矩阵和偏置矩阵进行修正

## 三、程序设计

### 1. Matrix类

BP网络运行需要大量使用矩阵相乘、加减和求导，为了便于将数学公式直接转换为代码，我们引入Matrix类，仿照Matlab的代码格式，我们重载了大量运算符以及函数调用。

此外，为了代码的复用性，不局限于此题，Matrix类设计成模板类，由于本题中鸢尾花数据都是以小数形式出现，权值、偏置矩阵也将在运算过程中出现小数。因此，本次实验在不加说明的情况下均为double型Matrix类，有效精度在15-16位。

#### • Matrix中基本数据类型

为了保证Matrix类的矩阵元素不被随意修改，数据成员均被设计为`private`，但我们也提供了许多API供使用者调用

```
private:
    int Row, Col;
    vector<vector<T>>>
```

#### • Matrix初始化以及赋值拷贝支持三种形式

根据其他contributors的需求：要求在非初始化阶段也能矩阵赋值，我们重载operator()

```
Matrix(int Row, int Col);           //矩阵将被赋值为 Row行Col列的零矩阵

Matrix(const Matrix<T>&);           //赋值拷贝
Matrix(vector<vector<T>>&);         //将二维数组整体复制给Matrix
```

```
Matrix<int> m, m2;
m(2,2)                               //2x2初值为0
m2(m)                                //与m2相同
m(vector<vector<int>> a)             //同上
```

#### • Matrix运算符重载

```
Matrix<int> m, m2, c;
c = m * m2;
m2 = 1.0 * m;                        //重载* 支持点乘以及矩阵乘法
m2 = m - c;
m2 = m + c;                          //重载± 矩阵加减
m[i];                                //重载[] 直接获取某一行，返回值将是一个1xN的Matrix
!m;                                  //重载! 对矩阵进行转置，不修改本身
```

- **Matrix打印矩阵**

```
m.display(); //在控制台打印矩阵
```

- **Matrix获取单个元素**

```
m.get_element(2,2); //返回单个元素 超出范围返回矩阵第一个值  
m.find_max();  
m.find_min(); //返回矩阵元素最大值
```

- **Matrix获取行列信息**

```
m.getRowandCol(); //返回一个1*2的Matrix 第一个元素为行，第二个元素为列
```

- **Matrix对整体元素操作**

```
int add(int a) //function temp  
{  
    return a+1;  
}  
  
Matrix<int> m;  
int (*ptr)(int)=add; //传入函数句柄对所有矩阵元素进行操作  
m.self_function(ptr);
```

- **Matrix元素归一化**

```
m.Normalize(); //按列元素归一化
```

- **Matrix行重排**

```
m.shuffle(); //若要对列乱序，务必转置
```

## 2. data\_loader类

实验提供数据无法直接被BP网络调用，因此我们通过data\_loader类实现文件数据到内存数据的转化，它是网络训练和数据提供之间必不可缺的桥梁。

- **data\_loader数据成员**

```
public:  
    Matrix<double> xtrain,ytrain;
```

注：本次实验中我们将提供的数据划分为4:1，比率可调

```
public:  
    Matrix<int> train_index; //训练集索引  
    Matrix<int> test_index; //测试集索引 大小为1xN
```

- data\_loader获取特定文件数据

```
void read_file("../data/iris.data");    //相对位置
```

- data\_loader创建数据标签

标签向量是3x1行向量，对应分量为1则对应某一类，其余分量为0

```
vector<double> label_to_num(const string);
```

### 3. BP\_net类

BP\_net类是整个程序的精华部分，以Matrix类为底层数据架构，以data\_loader类为数据来源，完整封装了网络训练和网络测试两大基本功能。同时，在类内部自带了Sigmoid、ReLU、Randn等函数，误差评估函数以及反向传播算法的实现，但这些对外都是封闭的，仅仅为用户提供使用接口。

注：在本次实验中，整个神经网络结构为4xNx3，其中N可调

- BP\_net私有数据成员

```
private:
    int iteration_num;        //迭代次数
    double lr_w;              //w学习率
    double lr_b;              //b学习率

    int input_layer;          //输入层维数 3
    int hidden_layer;         //隐藏层维数 n
    int output_layer;         //输出层维数 2

    Matrix<double> x;          //输入矩阵 4*1
    Matrix<double> y;          //标签矩阵 3*1

    Matrix<double> w1;         //一二层间权值 n*4
    Matrix<double> b1;         //一二层间偏置 n*1
    Matrix<double> w2;         //二三层间权值 3*n
    Matrix<double> b2;         //二三层间偏置 3*1

    Matrix<double> a1;         //输入矩阵 同x 4*1
    Matrix<double> a2;         //第二层结果 5*1
    Matrix<double> a3;         //第三层结果 即网络输出 3*1

    Matrix<double> delta1;     //前一层敏感度 5*1
    Matrix<double> delta2;     //结果和y的偏差导出的敏感度 3*1
```

- BP\_net公有数据成员

```
public:
    double* loss;              //动态数组，length=迭代数
    double* acc;
```

- **BP\_net初始化**

注：迭代次数\_iters默认1000次，但实际并不需要这么多，50次左右即可

权值矩阵、偏置矩阵 $lr$ 学习率默认0.01

隐藏层神经元cell\_num可调，默认为5

```
BPnet(int _iters = 1000, double _lr_w = 0.01, double _lr_b = 0.01, int  
cell_num=5);
```

- **BP\_net公有函数 (API)**

其他私有函数声明及实现见附件BP\_net.h&BP\_net.cpp

```
void train(data_loader&); //训练,调用数据集reader()  
void checkparameter(); //检查参数函数 调试用
```

## 4. 主函数

主函数内定义了网络类&数据类的全局变量，还实现了对训练结果可视化 (VisualDL)，依靠的是C++头文件Python.h,将C++代码转化为Python语言，众所周知Python具有快速优美的matplotlib绘图模块。

- **全局变量**

```
data_loader data; //reader()  
BPnet net1(100, 0.01, 0.01, 4); //BP网络
```

- **VisualDL**

注：第一个参数n为训练迭代次数

```
void draw(int n, double* loss1, double* loss2, double* loss3, double* acc1,  
double* acc2, double* acc3)
```

## 四、实验结果

### 1. 训练结果

- 以隐藏层神经元为5时为例，50次迭代最高可到**96%**，由于初始权值矩阵是随机生成的，所以每次得到的acc、loss不尽相同。但是小幅度的变动不影响训练模型在测试集优异表现**acc=1**，我们将最终模型参数保存在了**train\_model.pdparams**文件中
- 事实上，由于鸢尾花数据集的空间分布良好，在尝试多次后发现隐藏层只需要一层即可
- Release版本对代码编译执行进行了许多优化，整体网络运算速度远胜于Debug版本

```
Iteration: 33 acc:0.941667
Iteration: 34 acc:0.95
Iteration: 35 acc:0.958333
Iteration: 36 acc:0.958333
Iteration: 37 acc:0.958333
Iteration: 38 acc:0.958333
Iteration: 39 acc:0.958333
Iteration: 40 acc:0.958333
Iteration: 41 acc:0.958333
Iteration: 42 acc:0.958333
Iteration: 43 acc:0.958333
Iteration: 44 acc:0.958333
Iteration: 45 acc:0.958333
Iteration: 46 acc:0.95
Iteration: 47 acc:0.941667
Iteration: 48 acc:0.941667
Iteration: 49 acc:0.941667
acc on test: 1
-----END-----
请按任意键继续...
```

```
//keep the
}
cout << "Iterat

cout << "acc on tes
model_save("./data/

return;

}

void BPnet::checkparam
cout << "----check p
cout << "iteration_n
cout << iteration_n
cout << "dimensions
```

```
BPnet net(50, 0.01, 0.01);
net.train(test);
cout << "----END-----" << endl;
test.shuffle_index();
/*
int id = test.test_index.get_element(0, 12);
test.xtrain[id].display();
test.ytrain[id].display();
cout << net.forecast(test.xtrain[id]) << endl;
*/
return 0;
```

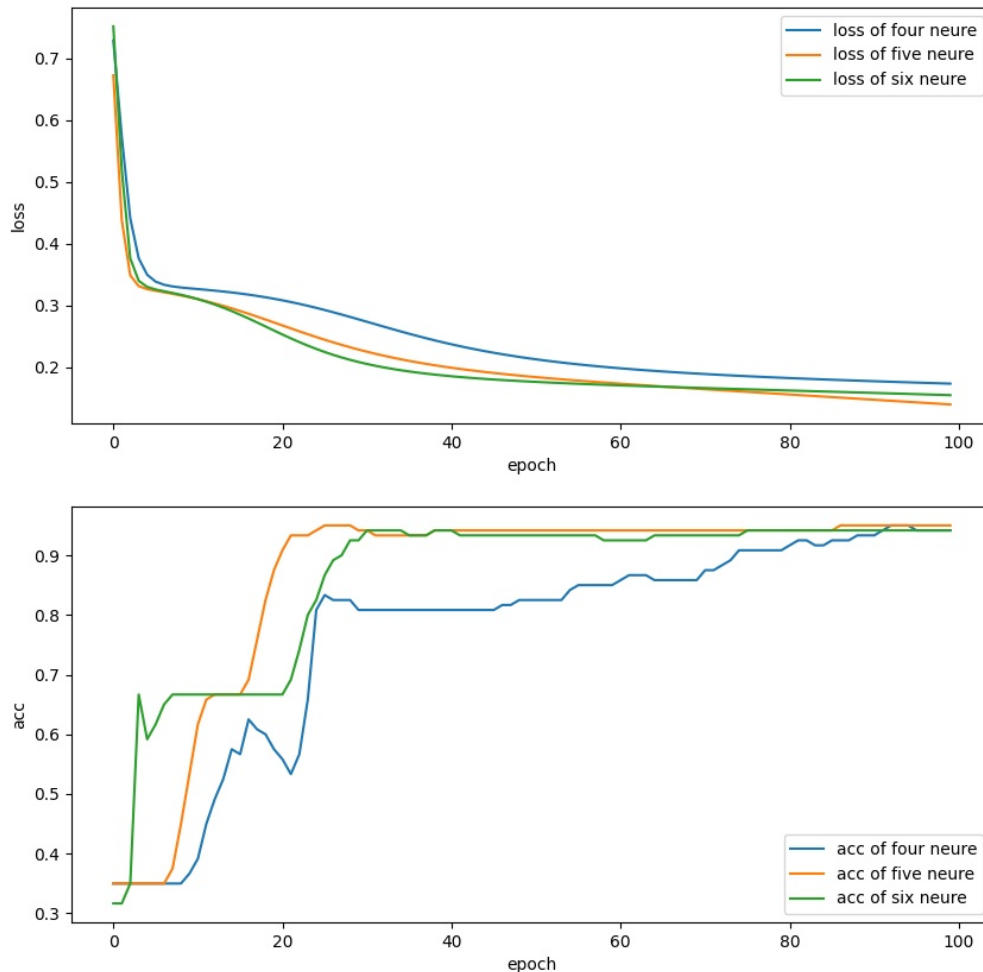
```
220 savefile << "b2" << endl;
221 b2.display();
222 cout.rdbuf(strmout_buf);
223 savefile.close();
224 }
225
226 double sigmoid(double x) {
227     return 1 / (1 + exp(-x));
228 }
229 double _1subx(double x) {
230     return 1 - x;
231 }
232 double myrand(double x) {
233     return (double)(rand() % 1
234 }
```

```
Iteration: 34 acc:0.941667
Iteration: 35 acc:0.941667
Iteration: 36 acc:0.941667
Iteration: 37 acc:0.941667
Iteration: 38 acc:0.941667
Iteration: 39 acc:0.941667
Iteration: 40 acc:0.941667
Iteration: 41 acc:0.941667
Iteration: 42 acc:0.941667
Iteration: 43 acc:0.941667
Iteration: 44 acc:0.933333
Iteration: 45 acc:0.933333
Iteration: 46 acc:0.933333
Iteration: 47 acc:0.933333
Iteration: 48 acc:0.933333
Iteration: 49 acc:0.933333
acc on test: 1
-----END-----
```

## 2. 参数优化

- 针对隐藏层的神经元个数，我们给出其在N=4、5、6下的acc、loss图像
- 由图像可知，当隐藏层神经元N=5时，acc、loss收敛速度、最终成绩均优于N=4、6时的情况，而N=7、8...时BP网络可能将出现过拟合，训练结果大大下降

loss and accuracy of the BPnet







## 五、实验总结

本次实验综合运用了本学期所学：数据保护与共享、运算符重载、模板类和STL库，将神经网络知识与C++面向对象相结合。我们充分利用C++各种特性，直接对数据内存化操作，从底层数据操作到API封装都有涉及，可以说是一次良好的学科实践。

- 在实验中，我们也遇到了大量困难，例如VS2019对模板类声明定义强制要求在一个文件中，C++链接器中子系统需设置为Console，类运算符“=”可以系统默认将内存复制给对象，但如果已经定义复制拷贝函数，“=”自动调用复制拷贝函数等等。
- 不断克服实践中的困难，锻炼了C++面向对象编程的能力，能够通过阅读开发文档，求助社区问答来自行解决产生的问题。课堂上老师教授的许多关于C++语言的语法规则，听起来十分枯燥无味，也不容易记住，只有通过实践-失败-成功的方式，才能真正掌握C++。
- 那些自认为万无一失的程序，实际上机运行时可能不断出现麻烦。如编译程序检测出一大堆错误。有时程序本身不存在语法错误，也能够顺利运行，但是运行结果显然是错误的。这时候就需要设置断点，单步调试或者依靠自己平时的经验教训做出修改。

## 六、关于作者

## Contributors:

-  把徐进: 编写BP\_net类以及PPT制作
-  朱业帆: 编写Matrix、data\_loader类以及报告撰写
-  杨佳晖: 编写绘图函数, 主函数, 优化BP\_net类参数设置
-  陈佳鹏: 优化数据结构及算法, 代码版本管理

## Orgnization:

团队: (<https://github.com/c-Lris-xmu>)

代码仓库: (<https://github.com/c-Lris-xmu/bp>)