20220514-机器学习&C++

1.学习内容

1.1 CNN类

Matrix.h C++ 复制代码

```cpp
#pragma once
#ifndef DNN_H_
#define DHH_H_

#include <stdlib.h>
#include <iostream>
#include <iomanip>
class Matrix
{
public:
    Matrix();
    Matrix(int m_, int n_, int k_);
    Matrix(const Matrix& matIns);
    ~Matrix();
    float*** mat;
    friend std::ostream& operator<<(std::ostream &os,const Matrix& matIns);


    Matrix& operator=(const Matrix& matIns);
    Matrix operator+(const Matrix& matIns);
    Matrix operator-(const Matrix& matIns);
    Matrix operator*(const Matrix& matIns);
    Matrix operator/(const Matrix& matIns);

private:
    int m, n, k;
};

#endif
```

Matrix.cpp C++ 复制代码

```cpp
#include "Matrix.h"
Matrix::Matrix()
{

}
Matrix::Matrix(int m_,int n_,int k_):m(m_),n(n_),k(k_)
{
    mat = new float** [m];
    for (int i = 0; i < m; i++)
    {
        mat[i] = new float* [n];
        for (int j = 0; j < n; j++)
        {
            mat[i][j] = new float[k];
            for (int p = 0; p < k; p++)
            {
                mat[i][j][p] = rand()%100 / float(100);
            }
        }
    }
}
Matrix::Matrix(const Matrix& matIns):m(matIns.m),n(matIns.n),k(matIns.k)
{
    mat = new float** [m];
    for (int i = 0; i < m; i++)
    {
        mat[i] = new float* [n];
        for (int j = 0; j < n; j++)
        {
            mat[i][j] = new float[k];
            for (int p = 0; p < k; p++)
            {
                mat[i][j][p] = matIns.mat[i][j][p];
            }
        }
    }
}

Matrix::~Matrix()
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            delete[] mat[i][j];
        }
        delete[] mat[i];
    }
    delete[] mat;
}


std::ostream& operator<<(std::ostream &os,const Matrix& matIns)
{
    for (int i = 0; i < matIns.m; i++)
    {
        for (int j = 0; j < matIns.n; j++)
        {
            for (int p = 0; p < matIns.k; p++)
            {
                //os << std::fixed<<std::setprecision(2) <<
matIns.mat[i][j][p] << ' ';
                os << matIns.mat[i][j][p] << ' ';
            }
            os << std::endl;
        }
        os << std::endl;
    }
    return os;
}

Matrix& Matrix::operator=(const Matrix& matIns)
{
    if (this == &matIns)
    {
        return *this;
    }
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            delete[] mat[i][j];
        }
        delete[] mat[i];
    }
    delete[] mat;

    mat = new float** [m];
    for (int i = 0; i < m; i++)
    {
        mat[i] = new float* [n];
        for (int j = 0; j < n; j++)
        {
                mat[i][j] = new float[k];
                for (int p = 0; p < k; p++)
                {
                    mat[i][j][p] = matIns.mat[i][j][p];
                }
            }
        }
        return *this;
    }

    Matrix Matrix::operator+(const Matrix& matIns)
    {
        Matrix newMatIns(m,n,k);
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                for (int p = 0; p < k; p++)
                {
                    newMatIns.mat[i][j][p]= mat[i][j][p]+matIns.mat[i][j]
[p];
                }
            }
        }
        return newMatIns;
    }

    Matrix Matrix::operator-(const Matrix& matIns)
    {
        Matrix newMatIns(m, n, k);
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                for (int p = 0; p < k; p++)
                {
                    newMatIns.mat[i][j][p] = mat[i][j][p] - matIns.mat[i][j]
[p];
                }
            }
        }
        return newMatIns;
    }

    Matrix Matrix::operator*(const Matrix& matIns)
    {
        Matrix newMatIns(m, n, k);
        for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int p = 0; p < k; p++)
            {
                newMatIns.mat[i][j][p] = mat[i][j][p] * matIns.mat[i][j][p];
            }
        }
    }
    return newMatIns;
}

Matrix Matrix::operator/(const Matrix& matIns)
{
    Matrix newMatIns(m, n, k);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int p = 0; p < k; p++)
            {
                newMatIns.mat[i][j][p] = mat[i][j][p] / matIns.mat[i][j][p];
            }
        }
    }
    return newMatIns;
}
```

CNN.h C++ 复制代码

```cpp
#pragma once
#ifndef CNN_H_
#define CNN_H_
/*
1)读取训练数据并将其存储进一个Matrix对象中

2)



存储模型参数


*/


#include "Matrix.h"
#include <string>
#include <fstream>
#include <iostream>
#pragma warning(disable:4996)

class CNN
{
public:
    CNN();
    void loadData(std::string feature_file,std::string label_file);
    uint32_t convert_to_little_endian(const unsigned char* bytes);
    /*
    void train();
    float train_loss();
    float test_loss();
    void optimization();
    void conv_layer();
    void pooling_layer();
    */
private:



};

#endif
```

CNN.cpp   C++   复制代码

```cpp
#include "CNN.h"

CNN::CNN()
{

}

void CNN::loadData(std::string feature_file,std::string label_file)
{
    FILE* fp = fopen(feature_file.c_str(), "r");
    while (!fp)
    {
        std::cout << "Can not open feature file!" << std::endl;
        exit(-1);
    }
    FILE* lp = fopen(label_file.c_str(), "r");
    while(!lp)
    {
        std::cout << "Can not open label file" << std::endl;
        exit(-1);
    }
    uint32_t header[4];
    uint32_t test[4];
    unsigned char bytes[4];

    for (int i = 0; i < 4; i++)
    {
        if (std::fread(bytes, sizeof(bytes), 1, fp))
        {
            header[i] = convert_to_little_endian(bytes);
            std::cout << header[i] << std::endl;
        }
    }
    Matrix featureMat(header[1], header[2], header[3]);
    for (int j = 0; j < header[1]; j++)
    {
        for (int k = 0; k < header[2]; k++)
        {
            for (int p = 0; p < header[3]; p++)
            {
                uint8_t element[1];
                if (std::fread(element, sizeof(element),1, fp))
                {

                }
                }
            }
        }
    }

    uint32_t CNN::convert_to_little_endian(const unsigned char* bytes)
    {
        return(uint32_t)(
            (bytes[0]<<24)|
            (bytes[1]<<16)|
            (bytes[2]<<8)|
            (bytes[3])
            );
    }
```

main.cpp   C++ | 复制代码

```cpp
#include "Matrix.h"
#include "CNN.h"
int main()
{
    srand((unsigned)time(NULL));
    Matrix myMatrix1(1, 3, 4);
    std::cout << myMatrix1;
    /*复制构造函数与赋值运算符重载
    Matrix myMatrix1(1, 3, 4);
    Matrix myMatrix2(1, 3, 4);
    Matrix myMatrix= myMatrix1 + myMatrix2;//这里调用了copy constructor来创建一个新的对象
    std::cout << myMatrix;
    Matrix myMatrix_a(1, 3, 4);
    myMatrix_a = myMatrix;//这里调用了重载的赋值运算符，对已存在的对象进行赋值
    Matrix myMatrix_b(1, 3, 4);
    myMatrix_b = myMatrix1 + myMatrix2;//这里由于加法运算符以值的方式返回对象，在返回时会调用copy constructor创建一个临时对象tmp作为返回值，返回后再利用赋值运算符将临时对象tmp赋值给已存在的对象
    */

    CNN myConvNet;
    myConvNet.loadData("train-images.idx3-ubyte", "train-labels.idx1-ubyte");
}
```

1.2 C++知识补充

复制构造函数与赋值运算符重载

https://blog.csdn.net/huangjw_806/article/details/79134330

- 二者的区别

复制构造函数是在创建对象的时候调用的，产生一个新的对象，只是这个对象成员的值与传入的对象相等；
重载的赋值运算符则是对已存在的对象进行赋值，使其对象成员的值与另一个对象相等

- 在构造函数中使用new——复制构造函数

复制构造函数在利用已有对象来初始化一个新的对象时调用。当构造函数中使用new初始化对象的指针成员时，需要定义一个复制构造函数，分配足够的空间来存储复制的数据，并复制数据

- 包含指针成员——重载赋值运算符

当类包含指针成员时，需要重载赋值运算符。如果使用原始的赋值运算符，则只是将左边的对象的指针成员指向右边的对象的指针成员所指向的内存。这样会导致，当右边的对象销毁时，则左边的对象的指针成员将指向空值，引发错误。这也是俗称的“浅复制”。
使用重载赋值符，首先会delete掉原有通过new分配的内存，之后利用new申请新的内存并进行赋值，从而实现“深复制”。

2.结果描述

今天没能完成CNN类的程序编写，设想是挺简单，但等真正动起手来发现需要考虑的东西还是挺多的，尤其是数据的转换以及描述。昨晚编写的矩阵类并不能很好地适应Mnist数据集，本应用模板编写的，但由于这一块不是很熟练所以还没实施。争取明天完成整体框架的编写。