20220519-机器学习

1.学习内容

1.1机器学习

CNN类

C++ 复制代码

```
#pragma once
#ifndef CNN_H_
#define CNN_H_

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

class CNN
{
public:
    CNN();
    std::vector<std::vector<uint8_t>> GetFeature(std::string feature_file);

    std::vector<uint8_t> GetLabel(std::string label_file);

    std::vector<std::vector<uint8_t>> labelMatTran(std::vector<uint8_t>& labelMat);

    std::vector<std::vector<double>> filterInl(int num_f,int num_r, int num_c);

    std::vector<double> convBiasInl();

    std::vector<std::vector<double>> convLayer(
        std::vector<std::vector<uint8_t>> &FeatureMat,
        std::vector<std::vector<double>> &filter,
        std::vector<double> &biasMat,int picIndex);

    std::vector<std::vector<double>> convActivate(
        std::vector<std::vector<double>>& convMat);

    std::vector<std::vector<double>> poolingLayer(
        std::vector<std::vector<double>>& convMat_);

    std::vector<double> outputBiasInl();

    std::vector<std::vector<std::vector<double>>> outputWeightInl(int stride);

```

```
        std::vector<double> outputLayer(
            std::vector<std::vector<double>>& poolingMat,
            std::vector<std::vector<std::vector<double>>>& outputWeight,
            std::vector<double>& biasMat);

        std::vector<double> softmax(std::vector<double>& outputMat);

        void paramUpdate(
            int batchSize,
            std::vector<std::vector<uint8_t>>& featureMat,
            std::vector<uint8_t>& labelMat,
            std::vector<std::vector<uint8_t>>& labelMatZO,
            std::vector<std::vector<double>>& filterMat,
            std::vector<double>& convBias,
            std::vector<std::vector<std::vector<double>>>& outputWeight,
            std::vector<double>& outputBias);

        void train(double lr,int epoch,int batchSize,
            std::vector<std::vector<uint8_t>>& featureMat,
            std::vector<uint8_t>& labelMat,
            std::vector<std::vector<uint8_t>>& labelMatZO,
            std::vector<std::vector<double>>& filterMat,
            std::vector<double>& convBias,
            std::vector<std::vector<std::vector<double>>>& outputWeight,
            std::vector<double>& outputBias);

        void test(std::vector<std::vector<uint8_t>>& featureMat,
            std::vector<uint8_t>& labelMat,
            std::vector<std::vector<double>>& filterMat,
            std::vector<double>& convBias,
            std::vector<std::vector<std::vector<double>>>& outputWeight,
            std::vector<double>& outputBias);

    public:
        uint32_t convert_to_little_endian(const unsigned char* bytes);
        double MaxPool(std::vector<double> poolBlock);


    private:
        int numPic;
        int testnumPic;
        int numRowPixel;
        int numColPixel;
        int PicSize;
        int numFilter;
        int filterRow;
        int filterCol;
        int filterSize;
        int convEleNum;
        int numLabel;
        int poolStride;
        int convMatColNum;
        int poolMatColNum;
        int poolMatSize;
        double learnR;
        double softmaxMediator;
        std::vector<double> lossMat;
    };

    #endif
```

CNN.cpp C++ 复制代码

```cpp
#include "CNN.h"

CNN::CNN()
{
    numLabel = 10;
    numPic = 60000;
    testnumPic = 10000;
}

std::vector<std::vector<uint8_t>> CNN::GetFeature(std::string feature_file)
{
    std::ifstream fp(feature_file,std::ios::binary);
    while (!fp.is_open())
    {
        std::cout << "Can not open feature file!" << std::endl;
        exit(-1);
    }

    uint32_t header[4]={};
    unsigned char bytes[4];

    for (int i = 0; i < 4; i++)
    {
        if (fp.read((char*)bytes, sizeof(bytes)))
        {
            header[i] = convert_to_little_endian(bytes);
        }
    }
    //numPic = header[1];
    numRowPixel = header[2];
    numColPixel = header[3];
    PicSize = numRowPixel * numColPixel;
    std::vector < std::vector<uint8_t>> featureMat;
    for (int j = 0; j < numPic; j++)
    {
        std::vector<uint8_t> imageF;
        for (int k = 0; k <PicSize; k++)
        {
            uint8_t element[1];
            if (fp.read((char*)&element, sizeof(element)))
            {
                imageF.push_back(element[0]);
            }
        }
        featureMat.push_back(imageF);
    }
    //std::cout << static_cast<int>(featureMat[0][159]);
    return featureMat;
}

std::vector<uint8_t> CNN::GetLabel(std::string label_file)
{
    FILE* lp = fopen(label_file.c_str(), "r");
    while (!lp)
    {
        std::cout << "Can not open label file" << std::endl;
        exit(-1);
    }
    uint32_t lheader[2]={};
    unsigned char lbytes[4];
    for (int i = 0; i < 2; i++)
    {
        if (std::fread(lbytes, sizeof(lbytes), 1, lp))
        {
            lheader[i] = convert_to_little_endian(lbytes);
        }
    }
    std::vector<uint8_t> labelData;
    for (int j = 0; j < lheader[1]; j++)
    {
        uint8_t lelement[1];
        if (std::fread(lelement, sizeof(lelement), 1, lp))
        {
            labelData.push_back(lelement[0]);
        }
    }
    //std::cout << static_cast<int>(labelData[2]) << std::endl;
    return labelData;
}

std::vector<std::vector<uint8_t>> CNN::labelMatTran(std::vector<uint8_t>& labelMat)
{
    std::vector<std::vector<uint8_t>> labelMatZO;
    for (int i = 0; i < numPic; i++)
    {
        std::vector<uint8_t> labelArrayZO;
        for (int j = 0; j < numLabel; j++)
        {
            if (j == labelMat[i])
            {
                labelArrayZO.push_back(1);
            }
            else
            {
                labelArrayZO.push_back(0);
            }
        }
        labelMatZO.push_back(labelArrayZO);
    }
    return labelMatZO;
}

std::vector<std::vector<double>> CNN::filterInl(int num_f, int num_r, int num_c)
{
    numFilter = num_f;
    filterRow = num_r;
    filterCol = num_c;
    filterSize = filterRow * filterCol;
    std::vector<std::vector<double>> filter_matrix;
    for (int i = 0; i < num_f; i++)
    {
        std::vector<double> filter_array;
        for (int j = 0; j < filterSize; j++)
        {
            double randW = (-1) + 2 * rand() / double(RAND_MAX);
            filter_array.push_back(randW);
        }
        filter_matrix.push_back(filter_array);
    }
    return filter_matrix;
}

std::vector<double> CNN::convBiasInl()
{
    std::vector<double> biasMatrix;

    for (int i = 0; i < numFilter; i++)
    {
        double randB = (-1) + 2 * rand() / double(RAND_MAX);

        biasMatrix.push_back(randB);
    }
    return biasMatrix;
}

std::vector<std::vector<double>> CNN::convLayer(
    std::vector<std::vector<uint8_t>> &FeatureMat,
    std::vector<std::vector<double>> &filter,
    std::vector<double> &biasMat,
    int picIndex)
{
    std::vector<std::vector<double>> convMat;
    convEleNum = (numRowPixel - filterRow + 1) * (numColPixel - filterCol + 1);
    double convValue;
    for (int i = 0; i < numFilter; i++)
    {
        std::vector<double> convArray;
        for (int j = 0; j < (numRowPixel - filterRow + 1); j++)
        {
            for (int k = 0; k < (numColPixel - filterCol + 1); k++)
            {
                conValue = 0;
                for (int p = 0; p < filterRow; p++)
                {
                    for (int g = 0; g < filterCol; g++)
                    {
                        conValue += FeatureMat[picIndex][j * numRowPixel + k + p * numRowPixel + g] * filter[i][p * filterRow + g];
                    }
                }
                conValue = conValue + biasMat[i];
                convArray.push_back(conValue);
            }
        }
        convMat.push_back(convArray);
    }

    return convMat;
}

std::vector<std::vector<double>> CNN::convActivate(std::vector<std::vector<double>> &convMat)
{
    for (auto i =convMat.begin(); i != convMat.end(); i++)
    {
        for (auto j = (* i).begin(); j != (*i).end(); j++)
        {
            *j = 1 / (1 + std::exp(*j));
        }
    }
    return convMat;
}

std::vector<std::vector<double>> CNN::poolingLayer(
    std::vector<std::vector<double>> &convMat_)
```

```
{
    std::vector<std::vector<double>> poolingMat;

    for (int i = 0; i < numFilter; i++)
    {
        std::vector<double> poolingArray;
        for (int j = 0; j < poolMatColNum; j++)
        {
            for (int p = 0; p < poolMatColNum; p++)
            {
                std::vector<double> poolBlock;
                for (int q = 0; q < poolStride; q++)
                {
                    for (int d = 0; d < poolStride; d++)
                    {
                        poolBlock.push_back(convMat_[i][j * convMatColNum * poolStride + p * poolStride + convMatColNum * q + d]);
                    }
                }
                poolingArray.push_back(MaxPool(poolBlock));
            }
        }
        poolingMat.push_back(poolingArray);
    }
    return poolingMat;
}

std::vector<std::vector<std::vector<double>>> CNN::outputWeightInl(int stride)
{
    poolStride = stride;
    convMatColNum = numRowPixel - filterRow + 1;
    poolMatColNum = convMatColNum / poolStride;
    poolMatSize = poolMatColNum * poolMatColNum;
    std::vector<std::vector<std::vector<double>>> outputWeightMat;
    for (int i = 0; i < numLabel; i++)
    {
        std::vector<std::vector<double>> outputWeightLabelMat;
        for (int j = 0; j < numFilter; j++)
        {
            std::vector<double> outputWeightArray;
            for (int p = 0; p < poolMatSize; p++)
            {
                double randW = (-1) + 2 * rand() / double(RAND_MAX);
                outputWeightArray.push_back(randW);
            }
            outputWeightLabelMat.push_back(outputWeightArray);
        }
        outputWeightMat.push_back(outputWeightLabelMat);
    }
    return outputWeightMat;
}

std::vector<double> CNN::outputBiasInl()
{
    std::vector<double> biasMatrix;
    for (int i = 0; i < numLabel; i++)
    {
        double randB = (-1) + 2 * rand() / double(RAND_MAX);
        biasMatrix.push_back(randB);
    }
    return biasMatrix;
}

std::vector<double> CNN::outputLayer(
    std::vector<std::vector<double>>& poolingMat,
    std::vector<std::vector<std::vector<double>>>& outputWeight,
    std::vector<double>& biasMat
)
{
    std::vector<double> outputMat;
    for (int i = 0; i < numLabel; i++)
    {
        double outputValue = 0;
        for (int j = 0; j < numFilter; j++)
        {
            for (int p = 0; p < poolMatSize; p++)
            {
                outputValue += outputWeight[i][j][p] * poolingMat[j][p];
            }
        }
        outputValue += biasMat[i];
        outputMat.push_back(outputValue);
    }
    return outputMat;
}


std::vector<double> CNN::softmax(std::vector<double> &outputMat)
{

    double sum = double(0);
    for (int i = 0; i < numLabel; i++)
    {
        sum += std::exp(outputMat[i]);
    }
    softmaxMediator = sum;
    for (int j = 0; j < numLabel; j++)
    {
        outputMat[j] = std::exp(outputMat[j]) / sum;
    }
    return outputMat;
}


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])
        );
}

void CNN::train(double lr,int epoch,int batchSize,
    std::vector<std::vector<uint8_t>>& featureMat,
    std::vector<uint8_t>& labelMat,
    std::vector<std::vector<uint8_t>>& labelMatZO,
    std::vector<std::vector<double>>& filterMat,
    std::vector<double>& convBias,
    std::vector<std::vector<std::vector<double>>>& outputWeight,
    std::vector<double>& outputBias)
{
    learnR = lr;
    for (int i = 0; i < epoch; i++)
    {
        paramUpdate(batchSize,featureMat,labelMat,
            labelMatZO,filterMat,convBias,outputWeight,outputBias);
        std::cout << "epoch " << i << " train loss is: ";
        for (auto it = lossMat.begin(); it != lossMat.end(); it++)
        {
            std::cout << *it << " ";
        }
        std::cout << std::endl;
    }
}

double CNN::MaxPool(std::vector<double> poolBlock)
{
    double max = double(-100);
    for (auto it = poolBlock.begin(); it != poolBlock.end(); it++)
    {
        if (max < *it)
```

```
            max = *it;
    }
    return max;
}

void CNN::test(
    std::vector<std::vector<uint8_t>>& featureMat,
    std::vector<uint8_t>& labelMat,
    std::vector<std::vector<double>>& filterMat,
    std::vector<double>& convBias,
    std::vector<std::vector<std::vector<double>>>& outputWeight,
    std::vector<double>& outputBias)
{
    int rightNum=0;
    int testused = testnumPic/100;
    for (int i = 0; i < testused; i++)
    {
        std::vector<std::vector<double>> convMat = convLayer(featureMat, filterMat, convBias, i);
        std::vector<std::vector<double>> activatedMat = convActivate(convMat);
        std::vector<std::vector<double>> poolingMat = poolingLayer(activatedMat);
        std::vector<double>             outputMat = outputLayer(poolingMat, outputWeight, outputBias);
        std::vector<double>             softmaxed = softmax(outputMat);
        double max = double(-100);
        for (auto it = softmaxed.begin(); it != softmaxed.end(); it++)
        {
            if (max < *it)
                max = *it;
        }
        for (int j = 0; j < softmaxed.size(); j++)
        {
            if (softmaxed[j]==max)
            {
                if (j == labelMat[j])
                {
                    rightNum++;
                }
            }
        }
    }
    double accuracy = double(rightNum) / double(testused);
    std::cout << "Test accuracy is " << accuracy << std::endl;
}


```

```
void CNN::paramUpdate(
    int batchSize,
    std::vector<std::vector<uint8_t>>& featureMat,
    std::vector<uint8_t>& labelMat,
    std::vector<std::vector<uint8_t>>& labelMatZO,
    std::vector<std::vector<double>>& filterMat,
    std::vector<double>& convBias,
    std::vector<std::vector<std::vector<double>>>& outputWeight,
    std::vector<double>& outputBias)
{
    int numPerBatch = numPic / batchSize;
    double predLabel = 0;
    uint8_t trueLabel = 0;

    for (int i = 0; i < batchSize-11; i++)
    {
        lossMat.clear();
        double EntropyLoss = 0;
        std::vector<std::vector<std::vector<double>>> deltaWeightOutput;
        std::vector<double> deltaBiasOutput;
        std::vector<std::vector<double>> deltaWeightFilter;
        std::vector<double> deltaBiasFilter;
        //输出层参数更新值初始化
        for (int ii = 0; ii < numLabel; ii++)
        {
            std::vector<std::vector<double>> vec1;
            for (int jj = 0; jj < numFilter; jj++)
            {
                std::vector<double> vec2;
                for (int kk = 0; kk < poolMatSize; kk++)
                {
                    vec2.push_back(0);
                }
                vec1.push_back(vec2);
            }
            deltaWeightOutput.push_back(vec1);
            deltaBiasOutput.push_back(0);
        }
        //卷积层参数更新值初始化
        for (int qq = 0; qq < numFilter; qq++)
        {
            std::vector<double> vec3;
            for (int dd = 0; dd < filterSize; dd++)
            {
                vec3.push_back(0);
            }
            deltaWeightFilter.push_back(vec3);
            deltaBiasFilter.push_back(0);
        }
        //开始训练
        for (int j = 0; j < numPerBatch/100; j++)
        {
            //forward
            std::vector<std::vector<double>> convMat         = convLayer(featureMat, filterMat, convBias, i * numPerBatch + j);
            std::vector<std::vector<double>> activatedMat = convActivate(convMat);
            std::vector<std::vector<double>> poolingMat  = poolingLayer(activatedMat);
            std::vector<double>             outputMat    = outputLayer(poolingMat,outputWeight, outputBias);
            std::vector<double>             softmaxed    = softmax(outputMat);

            //输出层参数更新
            double Mediator = 1 / (softmaxMediator * softmaxMediator) * (-1) * 1 / double(numPerBatch);
            for (int k = 0; k < numLabel; k++)
            {
                double output = outputMat[k];
                double softmaxValue = softmaxed[k];
                double backBar = 0;
                for (int d = 0; d < numLabel; d++)
                {
                    if (d == k)
                    {
                        backBar += labelMatZO[i * numPerBatch + j][d]*(softmaxMediator-std::exp(output)) / softmaxed[d];
                    }
                    else
                    {
                        backBar += (-1) * labelMatZO[i * numPerBatch + j][d] * std::exp(outputMat[d]) / softmaxed[d];
                    }
                }
                for (int p = 0; p < numFilter; p++)
                {
                    for (int q = 0; q < poolMatSize; q++)
                    {
                        double delW= Mediator*poolingMat[p][q] * std::exp(output)*backBar;
                        deltaWeightOutput[k][p][q] += delW;
                    }
                }
                double delB= Mediator * std::exp(output) * backBar;
                deltaBiasOutput[k] += delB;
            }
            //filter权重及bias更新
            std::vector<double> filterBackBarVec;
            double filterBackBarMed = softmaxMediator * softmaxMediator * (-1) * 1 / double(numPerBatch);
            for (int a1 = 0; a1 < numLabel; a1++)
            {
                double filterBackBar = 0;
                for (int a2 = 0; a2 < numLabel; a2++)
                {
                    if (a2 == a1)
                    {
                        filterBackBar += labelMatZO[i * numPerBatch + j][a2] * (softmaxMediator - std::exp(outputMat[a1]))*std::exp(outputMat[a1]) / softmaxed[a2];
                    }
                    else
                    {
                        filterBackBar += (-1) * labelMatZO[i * numPerBatch + j][a2] * std::exp(outputMat[a2]) *std::exp(outputMat[a1]) / softmaxed[a2];
                    }
                }
                filterBackBar = (1 / filterBackBarMed) * filterBackBar;
                filterBackBarVec.push_back(filterBackBar);
            }

            //卷积层的元素对filter的权重参数的求导
            std::vector<std::vector<double>> filToPixMat;
            for (int fw = 0; fw < filterSize; fw++)
            {
                std::vector<double> filToPixArray;
                for (int ele = 0; ele < convEleNum; ele++)
                {
                    int index = (fw / filterRow) * numRowPixel + (filterRow - 1) * (ele / convMatColNum) + ele + (fw + filterRow) % filterRow;
                    filToPixArray.push_back(featureMat[i * numPerBatch + j][index]);
                }
                filToPixMat.push_back(filToPixArray);
            }

            for (int a3 = 0; a3 < numFilter; a3++)
            {
                //权重更新
                for (int a4 = 0; a4 < filterSize; a4++)
                {
                    //激活函数对filter权重的求导
                    std::vector<double> actTowMat;
                    for (int actw = 0; actw < convEleNum; actw++)
                    {
                        double actwV = activatedMat[a3][actw] * (1 - activatedMat[a3][actw]) * filToPixMat[a4][actw];
                        actTowMat.push_back(actw);
                    }
                    //池化矩阵对filter权重的求导
                    std::vector<double> poolTow;
                    for (int poolindex = 0; poolindex < poolMatSize; poolindex++)
                    {
                        double poolTowV = 0;
                        std::vector<double> poolCmp;
                        std::vector<int> poolCmpIndex;
                        for (int poolstr = 0; poolstr < 2 * poolStride; poolstr++)
                        {
                            int cuteIndex = (poolindex / poolMatColNum) * (2 * poolMatColNum) + (poolStride * poolindex) +
                                (2 * poolMatColNum) * (poolstr / poolStride) + (poolstr + poolStride) % poolStride;
                            poolCmpIndex.push_back(cuteIndex);
                            poolCmp.push_back(activatedMat[a3][cuteIndex]);
                        }
                        for (int poolcmpI = 0; poolcmpI < 2 * poolStride; poolcmpI++)
                        {
                            if (poolCmp[poolcmpI] == poolingMat[a3][poolindex])
                            {
                                poolTowV = 1 * actTowMat[poolCmpIndex[poolcmpI]];
                                poolTow.push_back(poolTowV);
                            }
                            else
                            {
                                poolTow.push_back(0);
                            }
                        }
                    }
                    double filterWeightUpdate = 0;
                    for (int a5 = 0; a5 < numLabel; a5++)
                    {
                        double filterForeBarW = 0;
                        for (int a6 = 0; a6 < poolMatSize; a6++)
                        {
                            filterForeBarW += outputWeight[a5][a3][a6] *poolTow[a6];
                        }

                        filterWeightUpdate+= filterForeBarW * filterBackBarVec[a5];
                    }

                    deltaWeightFilter[a3][a4] +=filterWeightUpdate ;

                }
                //bias更新
                std::vector<double> actTobMat;
                for (int actb = 0; actb < convEleNum; actb++)
                {
                    double actbV = activatedMat[a3][actb] * (1 - activatedMat[a3][actb]);
                    actTobMat.push_back(actbV);
                }
                std::vector<double> poolTob;
                for (int poolindex = 0; poolindex < poolMatSize; poolindex++)
                {
                    double poolTobV = 0;
                    std::vector<double> poolCmp;
                    std::vector<int> poolCmpIndex;
                    for (int poolstr = 0; poolstr < 2 * poolStride; poolstr++)
                    {
                        int cuteIndex = (poolindex / poolMatColNum) * (2 * poolMatColNum) + (poolStride * poolindex) +
                            (2 * poolMatColNum) * (poolstr / poolStride) + (poolstr + poolStride) % poolStride;
                        poolCmpIndex.push_back(cuteIndex);
                        poolCmp.push_back(activatedMat[a3][cuteIndex]);
                    }
                    for (int poolcmpI = 0; poolcmpI < 2 * poolStride; poolcmpI++)
                    {
                        if (poolCmp[poolcmpI] == poolingMat[a3][poolindex])
                        {
                            poolTobV = 1 * actTobMat[poolCmpIndex[poolcmpI]];
                            poolTob.push_back(poolTobV);
                        }
                        else
                        {
                            poolTob.push_back(0);
                        }
                    }
                }
                double filterBiasUpdate = 0;
                for (int a5 = 0; a5 < numLabel; a5++)
                {
                    double filterForeBarB = 0;
```

2.结果描述

今天完成了CNN类的基本构建，但目前大多数时候会出现NAN的情况，只有少数几次试验出现比较明显的损失下降。之所以出现NAN，个人猜测是损失值爆炸性增长，导致超出了C++的数据类型的范围。通过调整学习率有一定机会可以使损失收敛。明天继续优化代码。