20220520-机器学习

1.学习内容

1.1 机器学习

CNN类

CNN.h C++ 复制代码

```cpp
#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)
#define PI 3.141592654
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>> convWeightInl(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);

```

```
42      std::vector<double> outputLayer(
43          std::vector<std::vector<double>>& poolingMat,
44          std::vector<std::vector<std::vector<double>>>& outputWeight,
45          std::vector<double>& biasMat);
46
47      std::vector<double> softmax(std::vector<double>& outputMat);
48
49      void paramUpdate(
50          int batchSize,
51          std::vector<std::vector<uint8_t>>& featureMat,
52          std::vector<uint8_t>& labelMat,
53          std::vector<std::vector<uint8_t>>& labelMatZO,
54          std::vector<std::vector<double>>& filterMat,
55          std::vector<double>& convBias,
56          std::vector<std::vector<std::vector<double>>>& outputWeight,
57          std::vector<double>& outputBias);
58
59      void train(double lr,int epoch,int batchSize,int actiTypein,
60          std::vector<std::vector<uint8_t>>& featureMat,
61          std::vector<uint8_t>& labelMat,
62          std::vector<std::vector<uint8_t>>& labelMatZO,
63          std::vector<std::vector<double>>& filterMat,
64          std::vector<double>& convBias,
65          std::vector<std::vector<std::vector<double>>>& outputWeight,
66          std::vector<double>& outputBias);
67
68      void test(
69          std::vector<std::vector<uint8_t>>& featureMat,
70          std::vector<uint8_t>& labelMat,
71          std::vector<std::vector<double>>& filterMat,
72          std::vector<double>& convBias,
73          std::vector<std::vector<std::vector<double>>>& outputWeight,
74          std::vector<double>& outputBias);
75
76  public:
77      uint32_t convert_to_little_endian(const unsigned char* bytes);
78      double MaxV(std::vector<double> poolBlock);
79      double GaussRand(double a,double b);
80
81  private:
82
83      int numPic;
84      int testnumPic;
85      int numRowPixel;
86      int numColPixel;
87      int PicSize;
88      int numFilter;
89      int filterRow;
```

```
90        int filterCol;
91        int filterSize;
92        int convEleNum;
93        int numLabel;
94        int poolStride;
95        int convMatColNum;
96        int poolMatColNum;
97        int poolMatSize;
98        double learnR;
99        double softmaxMediator;
100       int actiType;
101       std::vector<double> lossMat;
102   };
103
104   #endif
```

CNN.cpp C++ 复制代码

```cpp
#include "CNN.h"

CNN::CNN()
{
    numLabel = 10;
    numPic = 60000;
    testnumPic = 10000;
}
//----------------------------------<数据读取与存储>-----------------------------------
/*获取样本的特征数据*/
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;
    }
    /*获取样本的标签数据*/
    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;
    }
    /*将标签数据转化为0-1矩阵*/
    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;
}
//----------------------------------<参数初始化>--------------------------
---------------
/*卷积层（filter）权重初始化*/
std::vector<std::vector<double>> CNN::convWeightInl(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 = GaussRand(0,1/double(5));
            filter_array.push_back(randW);
        }
        filter_matrix.push_back(filter_array);
    }
    return filter_matrix;
}
/*卷积层bias初始化*/
std::vector<double> CNN::convBiasInl()
{
    std::vector<double> biasMatrix;

    for (int i = 0; i < numFilter; i++)
    {
        double randB = GaussRand(0, 1/double(5));

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

```
/*输出层（池化层到输出层）权重初始化*/
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 = GaussRand(0, 1/double(5));
                outputWeightArray.push_back(randW);
            }
            outputWeightLabelMat.push_back(outputWeightArray);
        }
        outputWeightMat.push_back(outputWeightLabelMat);
    }
    return outputWeightMat;
}
/*输出层bias初始化*/
std::vector<double> CNN::outputBiasInl()
{
    std::vector<double> biasMatrix;
    for (int i = 0; i < numLabel; i++)
    {
        double randB = GaussRand(0, 1/double(5));
        biasMatrix.push_back(randB);
    }
    return biasMatrix;
}
//----------------------<forward运算：卷积、池化、全连接输出>----------------------------
/*filter与原始数据进行卷积运算（互相关运算）*/
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;
```

```
182         convEleNum = (numRowPixel - filterRow + 1) * (numColPixel -
    filterCol + 1);
183         double conValue;
184         for (int i = 0; i < numFilter; i++)
185         {
186             std::vector<double> convArray;
187             for (int j = 0; j < (numRowPixel - filterRow + 1); j++)
188             {
189                 for (int k = 0; k < (numColPixel - filterCol + 1); k++)
190                 {
191                     conValue = 0;
192                     for (int p = 0; p < filterRow; p++)
193                     {
194                         for (int g = 0; g < filterCol; g++)
195                         {
196                             conValue += FeatureMat[picIndex][j * numRowPixel
    + k + p * numRowPixel + g] * filter[i][p * filterRow + g];
197                         }
198                     }
199                     conValue = conValue + biasMat[i];
200                     convArray.push_back(conValue);
201                 }
202             }
203             convMat.push_back(convArray);
204         }
205 
206         return convMat;
207     }
208     /*卷积层激活函数应用*/
209     std::vector<std::vector<double>>
    CNN::convActivate(std::vector<std::vector<double>> convMat)
210     {
211         if (actiType == 0)
212         {
213             //sigmoid激活
214             for (auto i = convMat.begin(); i != convMat.end(); i++)
215             {
216                 for (auto j = (*i).begin(); j != (*i).end(); j++)
217                 {
218                     *j = 1 / (1 + std::exp(( - 1)*( * j)));
219                 }
220             }
221             return convMat;
222         }
223         else if (actiType == 1)
224         {
225             //relu激活
226             for (auto i = convMat.begin(); i != convMat.end(); i++)
```

```
        {
            for (auto j = (*i).begin(); j != (*i).end(); j++)
            {
                if (*j <= 0)
                {
                    *j = 0;
                }
                else
                {
                    *j = *j;
                }
            }
        }
        return convMat;
    }
}
/*池化层最大池化运算*/
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(MaxV(poolBlock));
            }
        }
        poolingMat.push_back(poolingArray);
    }
    return poolingMat;
}
/*输出层运算*/
std::vector<double> CNN::outputLayer(
    std::vector<std::vector<double>>& poolingMat,
```

```
274         std::vector<std::vector<std::vector<double>>>& outputWeight,
275         std::vector<double>& biasMat
276     )
277     {
278         std::vector<double> outputMat;
279         for (int i = 0; i < numLabel; i++)
280         {
281             double outputValue = 0;
282             for (int j = 0; j < numFilter; j++)
283             {
284                 for (int p = 0; p < poolMatSize; p++)
285                 {
286                     outputValue += outputWeight[i][j][p] * poolingMat[j][p];
287                 }
288             }
289             outputValue += biasMat[i];
290             outputMat.push_back(outputValue);
291         }
292         return outputMat;
293     }
294     /*softmax函数应用*/
295     std::vector<double> CNN::softmax(std::vector<double> &outputMat)
296     {
297
298         double sum = double(0);
299         for (int i = 0; i < numLabel; i++)
300         {
301             sum += std::exp(outputMat[i]);
302         }
303         softmaxMediator = sum;
304         for (int j = 0; j < numLabel; j++)
305         {
306             outputMat[j] = std::exp(outputMat[j]) / sum;
307         }
308         return outputMat;
309     }
310     /*参数更新*/
311     void CNN::paramUpdate(
312         int batchSize,
313         std::vector<std::vector<uint8_t>>& featureMat,
314         std::vector<uint8_t>& labelMat,
315         std::vector<std::vector<uint8_t>>& labelMatZO,
316         std::vector<std::vector<double>>& filterMat,
317         std::vector<double>& convBias,
318         std::vector<std::vector<std::vector<double>>>& outputWeight,
319         std::vector<double>& outputBias)
320     {
321         int numPerBatch = numPic / batchSize;
```

```
322         double predLabel = 0;
323         uint8_t trueLabel = 0;
324 
325         for (int i = 0; i < batchSize - 11; i++)
326         {
327             lossMat.clear();
328             double EntropyLoss = 0;
329             std::vector<std::vector<std::vector<double>>> deltaWeightOutput;
330             std::vector<double> deltaBiasOutput;
331             std::vector<std::vector<double>> deltaWeightFilter;
332             std::vector<double> deltaBiasFilter;
333             //输出层参数更新值初始化
334             for (int ii = 0; ii < numLabel; ii++)
335             {
336                 std::vector<std::vector<double>> vec1;
337                 for (int jj = 0; jj < numFilter; jj++)
338                 {
339                     std::vector<double> vec2;
340                     for (int kk = 0; kk < poolMatSize; kk++)
341                     {
342                         vec2.push_back(0);
343                     }
344                     vec1.push_back(vec2);
345                 }
346                 deltaWeightOutput.push_back(vec1);
347                 deltaBiasOutput.push_back(0);
348             }
349             //卷积层参数更新值初始化
350             for (int qq = 0; qq < numFilter; qq++)
351             {
352                 std::vector<double> vec3;
353                 for (int dd = 0; dd < filterSize; dd++)
354                 {
355                     vec3.push_back(0);
356                 }
357                 deltaWeightFilter.push_back(vec3);
358                 deltaBiasFilter.push_back(0);
359             }
360             //开始训练
361             for (int j = 0; j < numPerBatch / 100; j++)
362             {
363                 //forward
364                 std::vector<std::vector<double>> convMat =
    convLayer(featureMat, filterMat, convBias, i * numPerBatch + j);
365                 std::vector<std::vector<double>> activatedMat =
    convActivate(convMat);
366                 std::vector<std::vector<double>> poolingMat =
    poolingLayer(activatedMat);
```

```
367             std::vector<double>             outputMat = outputLayer(poolingMat, outputWeight, outputBias);
368             std::vector<double>             softmaxed = softmax(outputMat);
369             /*
370             std::cout << "convMat" << std::endl;
371             for (auto it = convMat.begin(); it != convMat.end(); it++)
372             {
373                 for (auto i = (*it).begin(); i != (*it).end(); i++)
374                 {
375                     std::cout << *i << " ";
376                 }
377                 std::cout << std::endl;
378             }
379             std::cout << "activatedMat" << std::endl;
380             for (auto it = activatedMat.begin(); it != activatedMat.end(); it++)
381             {
382                 for (auto i = (*it).begin(); i != (*it).end(); i++)
383                 {
384                     std::cout << *i << " ";
385                 }
386                 std::cout << std::endl;
387             }
388             exit(-1);
389             */
390
391             //输出层参数更新
392             double Mediator = 1 / (softmaxMediator * softmaxMediator) * (-1) * 1 / double(numPerBatch);
393             for (int k = 0; k < numLabel; k++)
394             {
395                 double output = outputMat[k];
396                 double softmaxValue = softmaxed[k];
397                 double backBar = 0;
398                 for (int d = 0; d < numLabel; d++)
399                 {
400
401                     if (d == k)
402                     {
403                         backBar += labelMatZO[i * numPerBatch + j][d] * (softmaxMediator - std::exp(output)) / softmaxed[d];
404                     }
405                     else
406                     {
407                         backBar += (-1) * labelMatZO[i * numPerBatch + j][d] * std::exp(outputMat[d]) / softmaxed[d];
408                     }
```

```
409                 }
410                 for (int p = 0; p < numFilter; p++)
411                 {
412                     for (int q = 0; q < poolMatSize; q++)
413                     {
414                         double delW = Mediator * poolingMat[p][q] *
    std::exp(output) * backBar;
415                         deltaWeightOutput[k][p][q] += delW;
416                     }
417                 }
418                 double delB = Mediator * std::exp(output) * backBar;
419                 deltaBiasOutput[k] += delB;
420             }
421             //filter权重及bias更新
422             std::vector<double> filterBackBarVec;
423             double filterBackBarMed = softmaxMediator * softmaxMediator
    * (-1) * 1 / double(numPerBatch);
424             for (int a1 = 0; a1 < numLabel; a1++)
425             {
426                 double filterBackBar = 0;
427                 for (int a2 = 0; a2 < numLabel; a2++)
428                 {
429                     if (a2 == a1)
430                     {
431                         filterBackBar += labelMatZO[i * numPerBatch + j]
    [a2] * (softmaxMediator - std::exp(outputMat[a1])) *
    std::exp(outputMat[a1]) / softmaxed[a2];
432                     }
433                     else
434                     {
435                         filterBackBar += (-1) * labelMatZO[i *
    numPerBatch + j][a2] * std::exp(outputMat[a2]) * std::exp(outputMat[a1])
    / softmaxed[a2];
436                     }
437                 }
438                 filterBackBar = (1 / filterBackBarMed) * filterBackBar;
439                 filterBackBarVec.push_back(filterBackBar);
440             }
441
442             //卷积层的元素对filter的权重参数的求导
443             std::vector<std::vector<double>> filToPixMat;
444             for (int fw = 0; fw < filterSize; fw++)
445             {
446                 std::vector<double> filToPixArray;
447                 for (int ele = 0; ele < convEleNum; ele++)
448                 {
449                     int index = (fw / filterRow) * numRowPixel +
    (filterRow - 1) * (ele / convMatColNum) + ele + (fw + filterRow) %
```

```
filterRow;
450                     filToPixArray.push_back(featureMat[i * numPerBatch + j][index]);
451                 }
452                 filToPixMat.push_back(filToPixArray);
453             }
454 
455             for (int a3 = 0; a3 < numFilter; a3++)
456             {
457                 //权重更新
458                 for (int a4 = 0; a4 < filterSize; a4++)
459                 {
460                     //激活函数对filter权重的求导
461                     std::vector<double> actTowMat;
462                     for (int actw = 0; actw < convEleNum; actw++)
463                     {
464                         double actwV = 0;
465                         if (actiType == 0)
466                         {
467                             actwV = activatedMat[a3][actw] * (1 - activatedMat[a3][actw]) * filToPixMat[a4][actw];
468                         }
469                         else if (actiType == 1)
470                         {
471                             if (activatedMat[a3][actw] != 0)
472                             {
473                                 actwV = 1 * filToPixMat[a4][actw];
474                             }
475                             else
476                             {
477                                 actwV = 0;
478                             }
479 
480                         }
481                         actTowMat.push_back(actwV);
482                     }
483                     //池化矩阵对filter权重的求导
484                     std::vector<double> poolTow;
485                     for (int poolindex = 0; poolindex < poolMatSize; poolindex++)
486                     {
487                         double poolTowV = 0;
488                         std::vector<double> poolCmp;
489                         std::vector<int> poolCmpIndex;
490                         for (int poolstr = 0; poolstr < 2 * poolStride; poolstr++)
491                         {
```

```
                                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=0;
                        if (actiType == 0)
                        {
```

```
                        actbV = activatedMat[a3][actb] * (1 - activatedMat[a3][actb]);
                    }
                    else if (actiType == 1)
                    {
                        if (activatedMat[a3][actb] != 0)
                        {
                            actbV = 1;
                        }
                        else
                        {
                            actbV = 0;
                        }

                    }
                    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);
                        }
```

```
572                     }
573                 }
574                 double filterBiasUpdate = 0;
575                 for (int a5 = 0; a5 < numLabel; a5++)
576                 {
577                     double filterForeBarB = 0;
578                     for (int a6 = 0; a6 < poolMatSize; a6++)
579                     {
580                         filterForeBarB += outputWeight[a5][a3][a6] *
```

2.结果描述

今天主要还是优化了CNN的代码，发现了几个比较不显眼的问题。但还是很容易出现NAN的问题。查了下大概有几种情况，其中前三种主要跟梯度爆炸有关：

1.参数初始化问题

2.网络结构设计

3.激活函数的选择

4.出现分母为零的情况

针对参数初始化的问题，目前用的比较多的Xavier初始化，为此参考了一些生成服从正态分布随机数的代码，并加入到CNN类中。虽然感觉通过调节正态分布的方差可以取得一定效果，但实测下来还是经常出现NAN的问题。网络结构设计这块感觉应该不是主要问题，毕竟这只是一个最基本的卷积神经网络，比起那些好几百层的网络还差得远。激活函数的不同所带来的影响目前还不是很清楚，虽然Relu用的比sigmoid多，但在加入代码后并没有带来什么改善，反而出现了loss inf的情况，也没收敛过哪怕一次。至于分母为零的情况，在实测中确实出现过，比如最后softmax的输出。但从softmax的公式出发，按理说不应出现为零的情况，具体是为什么目前也没搞懂。

此外，还有几个比较让人摸不着头脑的现象。在某些学习率下，几轮学习的loss会一直保持一样，说明参数没有更新；测试数据经常出现只识别某个特定数字的情况，除了该数字以外的其它数字都没能成功识别，也挺tricky的。明天再想办法优化一轮，包括加入其它激活函数、进行批初始化等。之后学习一下一个youtuber写的C++ deep learning的代码。

此外，在写参数更新部分代码的时候感觉选择了一种最蠢的方式，即把链式求导的过程一个个展开。虽然思路没错，但总感觉没有应用到之前读的日本作者写的深度学习的书里提到的两个关键的求导中间量。后续也得好好再复习一下。

在完成基本的CNN类的编写之后，之后有三块想主要学习的：

- 计算机网络（看完书之后已经很久没复习了，这可是以后的饭碗
- C++软件开发（主要是带界面的。之前学习过一点点QT，但感觉挺枯燥的。还是得学起来
- C++机器学习（想把主要得机器学习算法都实现一遍，但感觉不一定有足够的时间。毕竟七八月份就要入职当社畜了，在那之前还想pick一些其它的技能，包括吉他 德语等等。机器学习是一个大大的坑，真要学到一点皮毛确实没那么简单，不应该有太多不切实际的幻想，比如吃透每种算法的内在数学逻辑。这次简单的CNN实现就已经花了快一周的时间，其它的估计也大差不差。如果只是简

单的使用，用python的很多工具包应该足够应付（这也需要对算法有足够的熟悉度），但这确实不够cool。

- 数据结构与算法（这部分之前学的毫无章法，也没能坚持把书跟视频看完，基础还十分薄弱
- C++高级编程（在编写CNN类的过程，明显感觉到自己的C++水平停留在一个很低的程度，写来写去都是vector跟for循环，像指针、map、模板、多态等基本没有使用，这其实会让自己的程序编写水平一直被锁死在一个low level的水准。虽然目前而言好像还勉强够用，但已经面临到代码低水平重复的情况。理应用一种更合理、更简洁的方式进行表示