20220521-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>>
convBN(std::vector<std::vector<double>>& convMat);

    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();

```

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

```
89        int numRowPixel;
90        int numColPixel;
91        int PicSize;
92        int numFilter;
93        int filterRow;
94        int filterCol;
95        int filterSize;
96        int convEleNum;
97        int numLabel;
98        int poolStride;
99        int convMatColNum;
100       int poolMatColNum;
101       int poolMatSize;
102       double learnR;
103       double softmaxMediator;
104       int actiType;
105       std::vector<double> lossMat;
106   };
107
108   #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     std::vector<std::vector<double>>
    CNN::convBN(std::vector<std::vector<double>>& convMat)
209     {
210         std::vector<std::vector<double>> BNMat;
211         BNMat = convMat;
212         for (int fil = 0; fil < numFilter; fil++)
213         {
214             double miu;
215             double sigma;
216             double sum = 0;
217             double dis = 0;
218             double const a = 0.000001;
219             for (int i = 0; i < convEleNum; i++)
220             {
221                 sum += convMat[fil][i];
222             }
223             miu = sum / numLabel;
224             for (int j = 0; j < convEleNum; j++)
225             {
226                 dis += std::pow((convMat[fil][j] - miu), 2);
```

```
227             }
228             sigma = dis / numLabel;
229             for (int k = 0; k < convEleNum; k++)
230             {
231                 BNMat[fil][k] = (BNMat[fil][k] - miu) / std::sqrt(sigma +
    a);
232             }
233         }
234         return BNMat;
235     }
236     /*卷积层激活函数应用*/
237     std::vector<std::vector<double>>
    CNN::convActivate(std::vector<std::vector<double>> convMat)
238     {
239         if (actiType == 0)
240         {
241             //sigmoid激活
242             for (auto i = convMat.begin(); i != convMat.end(); i++)
243             {
244                 for (auto j = (*i).begin(); j != (*i).end(); j++)
245                 {
246                     *j = 1 / (1 + std::exp(( - 1)*( * j)));
247                 }
248             }
249             return convMat;
250         }
251         else if (actiType == 1)
252         {
253             //relu激活
254             for (auto i = convMat.begin(); i != convMat.end(); i++)
255             {
256                 for (auto j = (*i).begin(); j != (*i).end(); j++)
257                 {
258                     if (*j <= 0)
259                     {
260                         *j = 0;
261                     }
262                     else
263                     {
264                         *j = *j;
265                     }
266                 }
267             }
268             return convMat;
269         }
270     }
271     /*池化层最大池化运算*/
272     std::vector<std::vector<double>> CNN::poolingLayer(
```

```
273         std::vector<std::vector<double>> &convMat_)
274     {
275         std::vector<std::vector<double>> poolingMat;
276 
277         for (int i = 0; i < numFilter; i++)
278         {
279             std::vector<double> poolingArray;
280             for (int j = 0; j < poolMatColNum; j++)
281             {
282                 for (int p = 0; p < poolMatColNum; p++)
283                 {
284                     std::vector<double> poolBlock;
285                     for (int q = 0; q < poolStride; q++)
286                     {
287                         for (int d = 0; d < poolStride; d++)
288                         {
289                             poolBlock.push_back(convMat_[i][j *
    convMatColNum * poolStride + p * poolStride + convMatColNum * q + d]);
290                         }
291                     }
292                     poolingArray.push_back(MaxV(poolBlock));
293                 }
294             }
295             poolingMat.push_back(poolingArray);
296         }
297         return poolingMat;
298     }
299     /*输出层运算*/
300     std::vector<double> CNN::outputLayer(
301         std::vector<std::vector<double>>& poolingMat,
302         std::vector<std::vector<std::vector<double>>>& outputWeight,
303         std::vector<double>& biasMat
304     )
305     {
306         std::vector<double> outputMat;
307         for (int i = 0; i < numLabel; i++)
308         {
309             double outputValue = 0;
310             for (int j = 0; j < numFilter; j++)
311             {
312                 for (int p = 0; p < poolMatSize; p++)
313                 {
314                     outputValue += outputWeight[i][j][p] * poolingMat[j][p];
315                 }
316             }
317             outputValue += biasMat[i];
318             outputMat.push_back(outputValue);
319         }
```

```
320         return outputMat;
321     }
322     //全连接层批量初始化
323     std::vector<double> CNN::fclBN(std::vector<double>& outputMat)
324     {
325         std::vector<double> BNMat;
326         BNMat = outputMat;
327         double miu;
328         double sigma;
329         double sum=0;
330         double dis = 0;
331         double const a = 0.000001;
332         for (int i = 0; i < numLabel; i++)
333         {
334             sum += outputMat[i];
335         }
336         miu = sum / numLabel;
337         for (int j = 0; j < numLabel; j++)
338         {
339             dis += std::pow((outputMat[j] - miu), 2);
340         }
341         sigma = dis / numLabel;
342         for (int k = 0; k < numLabel; k++)
343         {
344             BNMat[k] = (BNMat[k] - miu) / std::sqrt(sigma + a);
345         }
346         return BNMat;
347     }
348     /*softmax函数应用*/
349     std::vector<double> CNN::softmax(std::vector<double> &outputMat)
350     {
351
352         double sum = double(0);
353         for (int i = 0; i < numLabel; i++)
354         {
355             sum += std::exp(outputMat[i]);
356         }
357         softmaxMediator = sum;
358         for (int j = 0; j < numLabel; j++)
359         {
360             outputMat[j] = std::exp(outputMat[j]) / sum;
361         }
362         return outputMat;
363     }
364     /*参数更新*/
365     void CNN::paramUpdate(
366         int batchSize,
367         std::vector<std::vector<uint8_t>>& featureMat,
```

```
368         std::vector<uint8_t>& labelMat,
369         std::vector<std::vector<uint8_t>>& labelMatZO,
370         std::vector<std::vector<double>>& filterMat,
371         std::vector<double>& convBias,
372         std::vector<std::vector<std::vector<double>>>& outputWeight,
373         std::vector<double>& outputBias)
374     {
375         int numPerBatch = numPic / batchSize;
376         double predLabel = 0;
377         uint8_t trueLabel = 0;
378 
379         for (int i = 0; i < batchSize - 11; i++)
380         {
381             lossMat.clear();
382             double EntropyLoss = 0;
383             std::vector<std::vector<std::vector<double>>> deltaWeightOutput;
384             std::vector<double> deltaBiasOutput;
385             std::vector<std::vector<double>> deltaWeightFilter;
386             std::vector<double> deltaBiasFilter;
387             //输出层参数更新值初始化
388             for (int ii = 0; ii < numLabel; ii++)
389             {
390                 std::vector<std::vector<double>> vec1;
391                 for (int jj = 0; jj < numFilter; jj++)
392                 {
393                     std::vector<double> vec2;
394                     for (int kk = 0; kk < poolMatSize; kk++)
395                     {
396                         vec2.push_back(0);
397                     }
398                     vec1.push_back(vec2);
399                 }
400                 deltaWeightOutput.push_back(vec1);
401                 deltaBiasOutput.push_back(0);
402             }
403             //卷积层参数更新值初始化
404             for (int qq = 0; qq < numFilter; qq++)
405             {
406                 std::vector<double> vec3;
407                 for (int dd = 0; dd < filterSize; dd++)
408                 {
409                     vec3.push_back(0);
410                 }
411                 deltaWeightFilter.push_back(vec3);
412                 deltaBiasFilter.push_back(0);
413             }
414             //开始训练
415             for (int j = 0; j < numPerBatch/100 ; j++)
```

```
416         {
417             //forward
418             std::vector<std::vector<double>> convMatF =
    convLayer(featureMat, filterMat, convBias, i * numPerBatch + j);
419             std::vector<std::vector<double>> convMat = convBN(convMatF);
420             std::vector<std::vector<double>> activatedMat =
    convActivate(convMat);
421             std::vector<std::vector<double>> poolingMat =
    poolingLayer(activatedMat);
422             std::vector<double>             outputMatF =
    outputLayer(poolingMat, outputWeight, outputBias);
423             std::vector<double>             outputMat
    =fclBN(outputMatF);
424             std::vector<double>             softmaxed =
    softmax(outputMat);
425             /*
426             std::cout << "convMat" << std::endl;
427             for (auto it = convMat.begin(); it != convMat.end(); it++)
428             {
429                 for (auto i = (*it).begin(); i != (*it).end(); i++)
430                 {
431                     std::cout << *i << " ";
432                 }
433                 std::cout << std::endl;
434             }
435             std::cout << "activatedMat" << std::endl;
436             for (auto it = activatedMat.begin(); it !=
    activatedMat.end(); it++)
437             {
438                 for (auto i = (*it).begin(); i != (*it).end(); i++)
439                 {
440                     std::cout << *i << " ";
441                 }
442                 std::cout << std::endl;
443             }
444             exit(-1);
445             */
446
447             //输出层参数更新
448             double Mediator = 1 / (softmaxMediator * softmaxMediator) *
    (-1) * 1 / double(numPerBatch);
449             for (int k = 0; k < numLabel; k++)
450             {
451                 double output = outputMat[k];
452                 double softmaxValue = softmaxed[k];
453                 double backBar = 0;
454                 for (int d = 0; d < numLabel; d++)
455                 {
```

```
456
457                     if (d == k)
458                     {
459                         backBar += labelMatZO[i * numPerBatch + j][d] *
    (softmaxMediator - std::exp(output)) / softmaxed[d];
460                     }
461                     else
462                     {
463                         backBar += (-1) * labelMatZO[i * numPerBatch +
    j][d] * std::exp(outputMat[d]) / softmaxed[d];
464                     }
465                 }
466                 for (int p = 0; p < numFilter; p++)
467                 {
468                     for (int q = 0; q < poolMatSize; q++)
469                     {
470                         double delW = Mediator * poolingMat[p][q] *
    std::exp(output) * backBar;
471                         deltaWeightOutput[k][p][q] += delW;
472                     }
473                 }
474                 double delB = Mediator * std::exp(output) * backBar;
475                 deltaBiasOutput[k] += delB;
476             }
477             //filter权重及bias更新
478             std::vector<double> filterBackBarVec;
479             double filterBackBarMed = softmaxMediator * softmaxMediator
    * (-1) * 1 / double(numPerBatch);
480             for (int a1 = 0; a1 < numLabel; a1++)
481             {
482                 double filterBackBar = 0;
483                 for (int a2 = 0; a2 < numLabel; a2++)
484                 {
485                     if (a2 == a1)
486                     {
487                         filterBackBar += labelMatZO[i * numPerBatch + j]
    [a2] * (softmaxMediator - std::exp(outputMat[a1])) *
    std::exp(outputMat[a1]) / softmaxed[a2];
488                     }
489                     else
490                     {
491                         filterBackBar += (-1) * labelMatZO[i *
    numPerBatch + j][a2] * std::exp(outputMat[a2]) * std::exp(outputMat[a1])
    / softmaxed[a2];
492                     }
493                 }
494                 filterBackBar = (1 / filterBackBarMed) * filterBackBar;
495                 filterBackBarVec.push_back(filterBackBar);
```

```
496                 }
497
498                 //卷积层的元素对filter的权重参数的求导
499                 std::vector<std::vector<double>> filToPixMat;
500                 for (int fw = 0; fw < filterSize; fw++)
501                 {
502                     std::vector<double> filToPixArray;
503                     for (int ele = 0; ele < convEleNum; ele++)
504                     {
505                         int index = (fw / filterRow) * numRowPixel +
    (filterRow - 1) * (ele / convMatColNum) + ele + (fw + filterRow) %
    filterRow;
506                         filToPixArray.push_back(featureMat[i * numPerBatch +
    j][index]);
507                     }
508                     filToPixMat.push_back(filToPixArray);
509                 }
510
511                 for (int a3 = 0; a3 < numFilter; a3++)
512                 {
513                     //权重更新
514                     for (int a4 = 0; a4 < filterSize; a4++)
515                     {
516                         //激活函数对filter权重的求导
517                         std::vector<double> actTowMat;
518                         for (int actw = 0; actw < convEleNum; actw++)
519                         {
520                             double actwV = 0;
521                             if (actiType == 0)
522                             {
523                                 actwV = activatedMat[a3][actw] * (1 -
    activatedMat[a3][actw]) * filToPixMat[a4][actw];
524                             }
525                             else if (actiType == 1)
526                             {
527                                 if (activatedMat[a3][actw] != 0)
528                                 {
529                                     actwV = 1 * filToPixMat[a4][actw];
530                                 }
531                                 else
532                                 {
533                                     actwV = 0;
534                                 }
535
536                             }
537                             actTowMat.push_back(actwV);
538                         }
539                         //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];
                    }

```

```
578                     deltaWeightFilter[a3][a4] += filterWeightUpdate;
579
580                 }
581                 //bias更新
582                 std::vector<double> actTobMat;
583                 for (int actb = 0; actb < convEleNum; actb++)
584                 {
585                     double actbV=0;
586                     if (actiType == 0)
```

2.结果描述

今天上午稍微尝试了下在linux环境下编译并运行C++程序，感觉还挺方便的。下午给CNN类加入了批初始化的代码，但依旧会出现NAN的状况。此外真正的BN应该有待学习的参数，感觉还挺麻烦的，后续看情况加进去。此外，目前的类并不能很好地支持网络结构的拓展，尤其在反向传播方面，没能把核心逻辑解耦出来。后续还得继续优化。