20220519-机器学习

1.学习内容

1.1机器学习

CNN类

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)

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);

```

```
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,
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(std::vector<std::vector<uint8_t>>& featureMat,
69            std::vector<uint8_t>& labelMat,
70            std::vector<std::vector<double>>& filterMat,
71            std::vector<double>& convBias,
72            std::vector<std::vector<std::vector<double>>>& outputWeight,
73            std::vector<double>& outputBias);
74
75    public:
76        uint32_t convert_to_little_endian(const unsigned char* bytes);
77        double MaxPool(std::vector<double> poolBlock);
78
79
80    private:
81        int numPic;
82        int testnumPic;
83        int numRowPixel;
84        int numColPixel;
85        int PicSize;
86        int numFilter;
87        int filterRow;
88        int filterCol;
89        int filterSize;
```

```
90        int convEleNum;
91        int numLabel;
92        int poolStride;
93        int convMatColNum;
94        int poolMatColNum;
95        int poolMatSize;
96        double learnR;
97        double softmaxMediator;
98        std::vector<double> lossMat;
99    };
100
101   #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]);
            }
        }
```

```
45          featureMat.push_back(imageF);
46      }
47      //std::cout << static_cast<int>(featureMat[0][159]);
48      return featureMat;
49  }
50
51  std::vector<uint8_t> CNN::GetLabel(std::string label_file)
52  {
53      FILE* lp = fopen(label_file.c_str(), "r");
54      while (!lp)
55      {
56          std::cout << "Can not open label file" << std::endl;
57          exit(-1);
58      }
59      uint32_t lheader[2]={};
60      unsigned char lbytes[4];
61      for (int i = 0; i < 2; i++)
62      {
63          if (std::fread(lbytes, sizeof(lbytes), 1, lp))
64          {
65              lheader[i] = convert_to_little_endian(lbytes);
66          }
67      }
68      std::vector<uint8_t> labelData;
69      for (int j = 0; j < lheader[1]; j++)
70      {
71          uint8_t lelement[1];
72          if (std::fread(lelement, sizeof(lelement), 1, lp))
73          {
74              labelData.push_back(lelement[0]);
75          }
76      }
77      //std::cout << static_cast<int>(labelData[2]) << std::endl;
78      return labelData;
79  }
80
81  std::vector<std::vector<uint8_t>>
    CNN::labelMatTran(std::vector<uint8_t>& labelMat)
82  {
83      std::vector<std::vector<uint8_t>> labelMatZO;
84      for (int i = 0; i < numPic; i++)
85      {
86          std::vector<uint8_t> labelArrayZO;
87          for (int j = 0; j < numLabel; j++)
88          {
89              if (j == labelMat[i])
90              {
91                  labelArrayZO.push_back(1);
```

```
92              }
93              else
94              {
95                  labelArrayZO.push_back(0);
96              }
97          }
98          labelMatZO.push_back(labelArrayZO);
99      }
100     return labelMatZO;
101 }
102
103 std::vector<std::vector<double>> CNN::filterInl(int num_f, int num_r,
    int num_c)
104 {
105     numFilter = num_f;
106     filterRow = num_r;
107     filterCol = num_c;
108     filterSize = filterRow * filterCol;
109     std::vector<std::vector<double>> filter_matrix;
110     for (int i = 0; i < num_f; i++)
111     {
112         std::vector<double> filter_array;
113         for (int j = 0; j < filterSize; j++)
114         {
115             double randW = (-1) + 2 * rand() / double(RAND_MAX);
116             filter_array.push_back(randW);
117         }
118         filter_matrix.push_back(filter_array);
119     }
120     return filter_matrix;
121 }
122
123 std::vector<double> CNN::convBiasInl()
124 {
125     std::vector<double> biasMatrix;
126
127     for (int i = 0; i < numFilter; i++)
128     {
129         double randB = (-1) + 2 * rand() / double(RAND_MAX);
130
131         biasMatrix.push_back(randB);
132     }
133     return biasMatrix;
134 }
135
136 std::vector<std::vector<double>> CNN::convLayer(
137     std::vector<std::vector<uint8_t>> &FeatureMat,
138     std::vector<std::vector<double>> &filter,
```

```
139         std::vector<double> &biasMat,
140         int picIndex)
141     {
142         std::vector<std::vector<double>> convMat;
143         convEleNum = (numRowPixel - filterRow + 1) * (numColPixel -
    filterCol + 1);
144         double conValue;
145         for (int i = 0; i < numFilter; i++)
146         {
147             std::vector<double> convArray;
148             for (int j = 0; j < (numRowPixel - filterRow + 1); j++)
149             {
150                 for (int k = 0; k < (numColPixel - filterCol + 1); k++)
151                 {
152                     conValue = 0;
153                     for (int p = 0; p < filterRow; p++)
154                     {
155                         for (int g = 0; g < filterCol; g++)
156                         {
157                             conValue += FeatureMat[picIndex][j * numRowPixel
    + k + p * numRowPixel + g] * filter[i][p * filterRow + g];
158                         }
159                     }
160                     conValue = conValue + biasMat[i];
161                     convArray.push_back(conValue);
162                 }
163             }
164             convMat.push_back(convArray);
165         }
166 
167         return convMat;
168     }
169 
170     std::vector<std::vector<double>>
    CNN::convActivate(std::vector<std::vector<double>> &convMat)
171     {
172         for (auto i =convMat.begin(); i != convMat.end(); i++)
173         {
174             for (auto j = (* i).begin(); j != (*i).end(); j++)
175             {
176                 *j = 1 / (1 + std::exp(*j));
177             }
178         }
179         return convMat;
180     }
181 
182     std::vector<std::vector<double>> CNN::poolingLayer(
183         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);
        }
```

```
230             outputWeightMat.push_back(outputWeightLabelMat);
231         }
232         return outputWeightMat;
233     }
234 
235     std::vector<double> CNN::outputBiasInl()
236     {
237         std::vector<double> biasMatrix;
238         for (int i = 0; i < numLabel; i++)
239         {
240             double randB = (-1) + 2 * rand() / double(RAND_MAX);
241             biasMatrix.push_back(randB);
242         }
243         return biasMatrix;
244     }
245 
246     std::vector<double> CNN::outputLayer(
247         std::vector<std::vector<double>>& poolingMat,
248         std::vector<std::vector<std::vector<double>>>& outputWeight,
249         std::vector<double>& biasMat
250     )
251     {
252         std::vector<double> outputMat;
253         for (int i = 0; i < numLabel; i++)
254         {
255             double outputValue = 0;
256             for (int j = 0; j < numFilter; j++)
257             {
258                 for (int p = 0; p < poolMatSize; p++)
259                 {
260                     outputValue += outputWeight[i][j][p] * poolingMat[j][p];
261                 }
262             }
263             outputValue += biasMat[i];
264             outputMat.push_back(outputValue);
265         }
266         return outputMat;
267     }
268 
269 
270     std::vector<double> CNN::softmax(std::vector<double> &outputMat)
271     {
272 
273         double sum = double(0);
274         for (int i = 0; i < numLabel; i++)
275         {
276             sum += std::exp(outputMat[i]);
277         }
```

```
278         softmaxMediator = sum;
279         for (int j = 0; j < numLabel; j++)
280         {
281             outputMat[j] = std::exp(outputMat[j]) / sum;
282         }
283         return outputMat;
284     }
285 
286 
287     uint32_t CNN::convert_to_little_endian(const unsigned char* bytes)
288     {
289         return(uint32_t)(
290             (bytes[0] << 24) |
291             (bytes[1] << 16) |
292             (bytes[2] << 8) |
293             (bytes[3])
294             );
295     }
296 
297     void CNN::train(double lr,int epoch,int batchSize,
298         std::vector<std::vector<uint8_t>>& featureMat,
299         std::vector<uint8_t>& labelMat,
300         std::vector<std::vector<uint8_t>>& labelMatZO,
301         std::vector<std::vector<double>>& filterMat,
302         std::vector<double>& convBias,
303         std::vector<std::vector<std::vector<double>>>& outputWeight,
304         std::vector<double>& outputBias)
305     {
306         learnR = lr;
307         for (int i = 0; i < epoch; i++)
308         {
309             paramUpdate(batchSize,featureMat,labelMat,
310                 labelMatZO,filterMat,convBias,outputWeight,outputBias);
311             std::cout << "epoch " << i << " train loss is: ";
312             for (auto it = lossMat.begin(); it != lossMat.end(); it++)
313             {
314                 std::cout << *it << " ";
315             }
316             std::cout << std::endl;
317         }
318     }
319 
320     double CNN::MaxPool(std::vector<double> poolBlock)
321     {
322         double max = double(-100);
323         for (auto it = poolBlock.begin(); it != poolBlock.end(); it++)
324         {
325             if (max < *it)
```

```
326                 max = *it;
327         }
328         return max;
329     }
330 
331     void CNN::test(
332         std::vector<std::vector<uint8_t>>& featureMat,
333         std::vector<uint8_t>& labelMat,
334         std::vector<std::vector<double>>& filterMat,
335         std::vector<double>& convBias,
336         std::vector<std::vector<std::vector<double>>>& outputWeight,
337         std::vector<double>& outputBias)
338     {
339         int rightNum=0;
340         int testused = testnumPic/100;
341         for (int i = 0; i < testused; i++)
342         {
343             std::vector<std::vector<double>> convMat = convLayer(featureMat,
    filterMat, convBias, i);
344             std::vector<std::vector<double>> activatedMat =
    convActivate(convMat);
345             std::vector<std::vector<double>> poolingMat =
    poolingLayer(activatedMat);
346             std::vector<double>              outputMat =
    outputLayer(poolingMat, outputWeight, outputBias);
347             std::vector<double>              softmaxed = softmax(outputMat);
348             double max = double(-100);
349             for (auto it = softmaxed.begin(); it != softmaxed.end(); it++)
350             {
351                 if (max < *it)
352                     max = *it;
353             }
354             for (int j = 0; j < softmaxed.size(); j++)
355             {
356                 if (softmaxed[j]==max)
357                 {
358                     if (j == labelMat[j])
359                     {
360                         rightNum++;
361                     }
362                 }
363             }
364         }
365         double accuracy = double(rightNum) / double(testused);
366         std::cout << "Test accuracy is " << accuracy << std::endl;
367     }
368 
369 
```

```
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);
```

```
418             }
419             //开始训练
420             for (int j = 0; j < numPerBatch/100; j++)
421             {
422                 //forward
423                 std::vector<std::vector<double>> convMat         =
    convLayer(featureMat, filterMat, convBias, i * numPerBatch + j);
424                 std::vector<std::vector<double>> activatedMat =
    convActivate(convMat);
425                 std::vector<std::vector<double>> poolingMat  =
    poolingLayer(activatedMat);
426                 std::vector<double>              outputMat    =
    outputLayer(poolingMat,outputWeight, outputBias);
427                 std::vector<double>              softmaxed    =
    softmax(outputMat);
428 
429                 //输出层参数更新
430                 double Mediator = 1 / (softmaxMediator * softmaxMediator) *
    (-1) * 1 / double(numPerBatch);
431                 for (int k = 0; k < numLabel; k++)
432                 {
433                     double output = outputMat[k];
434                     double softmaxValue = softmaxed[k];
435                     double backBar = 0;
436                     for (int d = 0; d < numLabel; d++)
437                     {
438                         if (d == k)
439                         {
440                             backBar += labelMatZO[i * numPerBatch + j][d]*
    (softmaxMediator-std::exp(output)) / softmaxed[d];
441                         }
442                         else
443                         {
444                             backBar += (-1) * labelMatZO[i * numPerBatch +
    j][d] * std::exp(outputMat[d]) / softmaxed[d];
445                         }
446                     }
447                     for (int p = 0; p < numFilter; p++)
448                     {
449                         for (int q = 0; q < poolMatSize; q++)
450                         {
451                             double delW= Mediator*poolingMat[p][q] *
    std::exp(output)*backBar;
452                             deltaWeightOutput[k][p][q] += delW;
453                         }
454                     }
455                     double delB= Mediator * std::exp(output) * backBar;
456                     deltaBiasOutput[k] += delB;
```

```
457                 }
458                 //filter权重及bias更新
459                 std::vector<double> filterBackBarVec;
460                 double filterBackBarMed = softmaxMediator * softmaxMediator
    * (-1) * 1 / double(numPerBatch);
461                 for (int a1 = 0; a1 < numLabel; a1++)
462                 {
463                     double filterBackBar = 0;
464                     for (int a2 = 0; a2 < numLabel; a2++)
465                     {
466                         if (a2 == a1)
467                         {
468                             filterBackBar += labelMatZO[i * numPerBatch + j]
    [a2] * (softmaxMediator -
    std::exp(outputMat[a1]))*std::exp(outputMat[a1]) / softmaxed[a2];
469                         }
470                         else
471                         {
472                             filterBackBar += (-1) * labelMatZO[i *
    numPerBatch + j][a2] * std::exp(outputMat[a2]) *std::exp(outputMat[a1])
    / softmaxed[a2];
473                         }
474                     }
475                     filterBackBar = (1 / filterBackBarMed) * filterBackBar;
476                     filterBackBarVec.push_back(filterBackBar);
477                 }
478
479                 //卷积层的元素对filter的权重参数的求导
480                 std::vector<std::vector<double>> filToPixMat;
481                 for (int fw = 0; fw < filterSize; fw++)
482                 {
483                     std::vector<double> filToPixArray;
484                     for (int ele = 0; ele < convEleNum; ele++)
485                     {
486                         int index = (fw / filterRow) * numRowPixel +
    (filterRow - 1) * (ele / convMatColNum) + ele + (fw + filterRow) %
    filterRow;
487                         filToPixArray.push_back(featureMat[i * numPerBatch +
    j][index]);
488                     }
489                     filToPixMat.push_back(filToPixArray);
490                 }
491
492                 for (int a3 = 0; a3 < numFilter; a3++)
493                 {
494                     //权重更新
495                     for (int a4 = 0; a4 < filterSize; a4++)
496                     {
```

```
                    //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
```

```
574                         {
575                             poolTob.push_back(0);
576                         }
577                     }
578                 }
579                 double filterBiasUpdate = 0;
580                 for (int a5 = 0; a5 < numLabel; a5++)
581                 {
582                     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。