20220517-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)

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<float>> filterInl(int num_f,int num_r, int num_c);

    std::vector<float> convBiasInl();

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

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

    std::vector<std::vector<float>> poolingLayer(
        std::vector<std::vector<float>>& convMat_,
        int poolStride);

    std::vector<float> outputBiasInl();

    std::vector<std::vector<std::vector<float>>> outputWeightInl();

```

```
43        std::vector<float> outputLayer(
44            std::vector<std::vector<float>>& poolingMat,
45            std::vector<std::vector<std::vector<float>>>& outputWeight,
46            std::vector<float>& biasMat);
47
48        std::vector<float> softmax(std::vector<float>& outputMat);
49
50        std::vector<float> Train(
51            int batchSize,
52            std::vector<std::vector<uint8_t>>& featureMat,
53            std::vector<uint8_t>& labelMat,
54            std::vector<std::vector<uint8_t>>& labelMatZO,
55            std::vector<std::vector<float>>& filterMat,
56            std::vector<float>& convBias,
57            std::vector<std::vector<std::vector<float>>>& outputWeight,
58            std::vector<float>& outputBias);
59
60
61        void ParamUpdate();
62
63    public:
64        uint32_t convert_to_little_endian(const unsigned char* bytes);
65        float MaxPool(std::vector<float> poolBlock);
66
67
68        /*
69        void train();
70        float train_loss();
71        float test_loss();
72        void optimization();
73
74        void pooling_layer();
75        */
76    private:
77        int numPic;
78        int numRowPixel;
79        int numColPixel;
80        int PicSize;
81        int numFilter;
82        int filterRow;
83        int filterCol;
84        int numLabel;
85        int poolMatSize;
86
87        float softmaxMediator;
88    };
89
90    #endif
```

CNN.cpp C++ 复制代码

```cpp
#include "CNN.h"

CNN::CNN()
{
    numLabel = 10;
}

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);
    }
```

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

```
            {
                labelArrayZO.push_back(0);
            }
        }
        labelMatZO.push_back(labelArrayZO);
    }
    return labelMatZO;
}

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

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

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

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

std::vector<std::vector<float>> CNN::convLayer(
    std::vector<std::vector<uint8_t>> &FeatureMat,
    std::vector<std::vector<float>> &filter,
    std::vector<float> &biasMat,
    int picIndex)
{
```

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

```
184         int convMatColNum = numRowPixel - filterRow + 1;
185         int poolMatColNum = convMatColNum / poolStride;
186         poolMatSize = poolMatColNum * poolMatColNum;
187         for (int i = 0; i < numFilter; i++)
188         {
189             std::vector<float> poolingArray;
190             for (int j = 0; j < poolMatColNum; j++)
191             {
192                 for (int p = 0; p < poolMatColNum; p++)
193                 {
194                     std::vector<float> poolBlock;
195                     for (int q = 0; q < poolStride; q++)
196                     {
197                         for (int d = 0; d < poolStride; d++)
198                         {
199                             poolBlock.push_back(convMat_[i][j *
    convMatColNum * poolStride + p * poolStride + convMatColNum * q + d]);
200                         }
201                     }
202                     poolingArray.push_back(MaxPool(poolBlock));
203                 }
204             }
205             poolingMat.push_back(poolingArray);
206         }
207         return poolingMat;
208     }
209
210     std::vector<std::vector<std::vector<float>>> CNN::outputWeightInl()
211     {
212         std::vector<std::vector<std::vector<float>>> outputWeightMat;
213         for (int i = 0; i < numLabel; i++)
214         {
215             std::vector<std::vector<float>> outputWeightLabelMat;
216             for (int j = 0; j < numFilter; j++)
217             {
218                 std::vector<float> outputWeightArray;
219                 for (int p = 0; p < poolMatSize; p++)
220                 {
221                     float randW = (-1) + 2 * rand() / float(RAND_MAX);
222                     outputWeightArray.push_back(randW);
223                 }
224                 outputWeightLabelMat.push_back(outputWeightArray);
225             }
226             outputWeightMat.push_back(outputWeightLabelMat);
227         }
228         return outputWeightMat;
229     }
230
```

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

```
278         }
279         return outputMat;
280     }
281
282
283     uint32_t CNN::convert_to_little_endian(const unsigned char* bytes)
284     {
285         return(uint32_t)(
286             (bytes[0] << 24) |
287             (bytes[1] << 16) |
288             (bytes[2] << 8) |
289             (bytes[3])
290             );
291     }
292
293     float CNN::MaxPool(std::vector<float> poolBlock)
294     {
295         float max = float(-100);
296         for (auto it = poolBlock.begin(); it != poolBlock.end(); it++)
297         {
298             if (max < *it)
299                 max = *it;
300         }
301         return max;
302     }
303
304     std::vector<float> CNN::Train(
305         int batchSize,
306         std::vector<std::vector<uint8_t>>& featureMat,
307         std::vector<uint8_t>& labelMat,
308         std::vector<std::vector<uint8_t>>& labelMatZO,
309         std::vector<std::vector<float>>& filterMat,
310         std::vector<float>& convBias,
311         std::vector<std::vector<std::vector<float>>>& outputWeight,
312         std::vector<float>& outputBias)
313     {
314         int numPerBatch = numPic / batchSize;
315         float predLabel = 0;
316         uint8_t trueLabel = 0;
317         std::vector<float> lossMat;
318
319         for (int i = 0; i < batchSize-11; i++)
320         {
321             float EntropyLoss = 0;
322             std::vector<std::vector<std::vector<float>>> deltaWeightOutput;
323             std::vector<float> deltaBiasOutput;
324             std::vector<std::vector<float>> deltaWeightFilter;
325             std::vector<float> deltaBiasFilter;
```

```
        //参数变化矩阵的初始化
        for (int ii = 0; ii < numLabel; ii++)
        {
            std::vector<std::vector<float>> vec1;
            for (int jj = 0; jj < numFilter; jj++)
            {
                std::vector<float> vec2;
                for (int kk = 0; kk < poolMatSize; kk++)
                {
                    vec2.push_back(0);
                }
                vec1.push_back(vec2);
            }
            deltaWeightOutput.push_back(vec1);
        }
        for (int pp = 0; pp < numLabel; pp++)
        {
            deltaBiasOutput.push_back(0);
        }

        for (int j = 0; j < numPerBatch/1000; j++)
        {
            std::vector<std::vector<float>> convMat      = convLayer(featureMat, filterMat, convBias, i * numPerBatch + j);
            std::vector<std::vector<float>> activatedMat = convActivate(convMat);
            std::vector<std::vector<float>> poolingMat   = poolingLayer(activatedMat, 2);
            std::vector<float>              outputMat    = outputLayer(poolingMat,outputWeight, outputBias);
            std::vector<float>              softmaxed    = softmax(outputMat);

            float Mediator = 1 / (softmaxMediator * softmaxMediator) * (-1) * 1 / float(numPerBatch);
            for (int k = 0; k < numLabel; k++)
            {
                float output = outputMat[k];
                float softmaxValue = softmaxed[k];
                float backBar = 0;
                for (int d = 0; d < numLabel; d++)
                {
                    if (d = k)
                    {
                        backBar += labelMatZO[i * numPerBatch + j][d] / 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++)
                    {
                        float delW= Mediator*poolingMat[p][q] * std::exp(output)*backBar;
                        deltaWeightOutput[k][p][q] += delW;
                    }
                }
                float delB= Mediator * std::exp(output) * backBar;
                deltaBiasOutput[k] += delB;
            }
            trueLabel = labelMat[i * numPerBatch + j];
            EntropyLoss += (-1) * trueLabel * std::log(softmaxed[trueLabel]);
        }
        EntropyLoss = 1 / float(numPerBatch) * EntropyLoss;
        lossMat.push_back(EntropyLoss);
    }
    return lossMat;
}

```

2.结果描述

今天依旧围绕CNN类开展编程，过程中梯度下降公式的推导及代码实现花费了比较多的功夫。目前已经基本完成输出层的权重以及bias的更新代码的编写，但由于嵌套了大量循环，试验中还没能在短时间内跑出结果。明天继续。