20220516-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<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();

    std::vector<float> outputLayer(
        std::vector<std::vector<float>>& poolingMat,
        std::vector<std::vector<std::vector<float>>>& outputWeight,
```

```
44          std::vector<float>& biasMat);
45
46      std::vector<float> softmax(std::vector<float>& outputMat);
47
48      float Loss(int batchSize, std::vector<uint8_t>&
    labelMat,std::vector<float>& softmaxed);
49  public:
50      uint32_t convert_to_little_endian(const unsigned char* bytes);
51      float MaxPool(std::vector<float> poolBlock);
52
53  private:
54      int numPic;
55      int numRowPixel;
56      int numColPixel;
57      int PicSize;
58      int numFilter;
59      int filterRow;
60      int filterCol;
61      int numLabel;
62      int poolMatSize;
63
64  };
65
66  #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]-55000;
    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<float>> CNN::filterInl(int num_f, int num_r, int
    num_c)
80    {
81        numFilter = num_f;
82        filterRow = num_r;
83        filterCol = num_c;
84        std::vector<std::vector<float>> filter_matrix;
85        for (int i = 0; i < num_f; i++)
86        {
87            std::vector<float> filter_array;
88            for (int j = 0; j < num_r * num_c; j++)
89            {
90                float randW = (-1) + 2 * rand() / float(RAND_MAX);
91                filter_array.push_back(randW);
```

```
92          }
93          filter_matrix.push_back(filter_array);
94      }
95      return filter_matrix;
96  }
97
98  std::vector<float> CNN::convBiasInl()
99  {
100     std::vector<float> biasMatrix;
101
102     for (int i = 0; i < numFilter; i++)
103     {
104         float randB = (-1) + 2 * rand() / float(RAND_MAX);
105
106         biasMatrix.push_back(randB);
107     }
108     return biasMatrix;
109 }
110
111 std::vector<std::vector<float>> CNN::convLayer(
112     std::vector<std::vector<uint8_t>> &FeatureMat,
113     std::vector<std::vector<float>> &filter,
114     std::vector<float> &biasMat,
115     int picIndex)
116 {
117     std::vector<std::vector<float>> convMat;
118     int convEleNum = (numRowPixel - filterRow + 1) * (numColPixel -
    filterCol + 1);
119     float conValue;
120     for (int i = 0; i < numFilter; i++)
121     {
122         std::vector<float> convArray;
123         for (int j = 0; j < (numRowPixel - filterRow + 1); j++)
124         {
125             for (int k = 0; k < (numColPixel - filterCol + 1); k++)
126             {
127                 conValue = 0;
128                 for (int p = 0; p < filterRow; p++)
129                 {
130                     for (int g = 0; g < filterCol; g++)
131                     {
132                         conValue += FeatureMat[picIndex][j * numRowPixel
    + k + p * numRowPixel + g] * filter[i][p * filterRow + g];
133                     }
134                 }
135                 conValue = conValue + biasMat[i];
136                 convArray.push_back(conValue);
137             }
```

```
138             }
139             convMat.push_back(convArray);
140         }
141 
142         return convMat;
143     }
144 
145     std::vector<std::vector<float>>
        CNN::convActivate(std::vector<std::vector<float>> &convMat)
146     {
147         for (auto i =convMat.begin(); i != convMat.end(); i++)
148         {
149             for (auto j = (* i).begin(); j != (*i).end(); j++)
150             {
151                 *j = 1 / (1 + std::exp(*j));
152             }
153         }
154         return convMat;
155     }
156 
157     std::vector<std::vector<float>> CNN::poolingLayer(
158         std::vector<std::vector<float>> &convMat_,
159         int poolStride)
160     {
161         std::vector<std::vector<float>> poolingMat;
162         int convMatColNum = numRowPixel - filterRow + 1;
163         int poolMatColNum = convMatColNum / poolStride;
164         poolMatSize = poolMatColNum * poolMatColNum;
165         for (int i = 0; i < numFilter; i++)
166         {
167             std::vector<float> poolingArray;
168             for (int j = 0; j < poolMatColNum; j++)
169             {
170                 for (int p = 0; p < poolMatColNum; p++)
171                 {
172                     std::vector<float> poolBlock;
173                     for (int q = 0; q < poolStride; q++)
174                     {
175                         for (int d = 0; d < poolStride; d++)
176                         {
177                             poolBlock.push_back(convMat_[i][j *
        convMatColNum * poolStride + p * poolStride + convMatColNum * q + d]);
178                         }
179                     }
180                     poolingArray.push_back(MaxPool(poolBlock));
181                 }
182             }
183             poolingMat.push_back(poolingArray);
```

```
184         }
185         return poolingMat;
186     }
187
188     std::vector<std::vector<std::vector<float>>> CNN::outputWeightInl()
189     {
190         std::vector<std::vector<std::vector<float>>> outputWeightMat;
191         for (int i = 0; i < numLabel; i++)
192         {
193             std::vector<std::vector<float>> outputWeightLabelMat;
194             for (int j = 0; j < numFilter; j++)
195             {
196                 std::vector<float> outputWeightArray;
197                 for (int p = 0; p < poolMatSize; p++)
198                 {
199                     float randW = (-1) + 2 * rand() / float(RAND_MAX);
200                     outputWeightArray.push_back(randW);
201                 }
202                 outputWeightLabelMat.push_back(outputWeightArray);
203             }
204             outputWeightMat.push_back(outputWeightLabelMat);
205         }
206         return outputWeightMat;
207     }
208
209     std::vector<float> CNN::outputBiasInl()
210     {
211         std::vector<float> biasMatrix;
212         for (int i = 0; i < numLabel; i++)
213         {
214             float randB = (-1) + 2 * rand() / float(RAND_MAX);
215             biasMatrix.push_back(randB);
216         }
217         return biasMatrix;
218     }
219
220     std::vector<float> CNN::outputLayer(
221         std::vector<std::vector<float>>& poolingMat,
222         std::vector<std::vector<std::vector<float>>>& outputWeight,
223         std::vector<float>& biasMat
224     )
225     {
226         std::vector<float> outputMat;
227         for (int i = 0; i < numLabel; i++)
228         {
229             float outputValue = 0;
230             for (int j = 0; j < numFilter; j++)
231             {
```

```
232                 for (int p = 0; p < poolMatSize; p++)
233                 {
234                     //outputValue += outputWeight[i][j][p] * poolingMat[j]
    [p];
235                 }
236             }
237             outputValue += biasMat[i];
238             outputMat.push_back(outputValue);
239         }
240         return outputMat;
241     }
242
243
244     std::vector<float> CNN::softmax(std::vector<float> &outputMat)
245     {
246
247         float sum = float(0);
248         for (int i = 0; i < numLabel; i++)
249         {
250             sum += std::exp(outputMat[i]);
251         }
252         for (int j = 0; j < numLabel; j++)
253         {
254             outputMat[j] = std::exp(outputMat[j]) / sum;
255         }
256         return outputMat;
257     }
258
259
260     uint32_t CNN::convert_to_little_endian(const unsigned char* bytes)
261     {
262         return(uint32_t)(
263             (bytes[0] << 24) |
264             (bytes[1] << 16) |
265             (bytes[2] << 8) |
266             (bytes[3])
267             );
268     }
269
270     float CNN::MaxPool(std::vector<float> poolBlock)
271     {
272         float max = float(-100);
273         for (auto it = poolBlock.begin(); it != poolBlock.end(); it++)
274         {
275             if (max < *it)
276                 max = *it;
277         }
278         return max;
```

```
279    }
280
281    float CNN::Loss(int batchSize, std::vector<uint8_t>& labelMat,
       std::vector<float>& softmaxed)
282    {
283
284    }
```

2.结果描述

今天成功解决了昨天的bug，并完成了卷积、池化、全连接输出等内容，目前还剩余损失函数以及梯度下降算法两部分未完成，这两块也是整个CNN类的重难点。明天争取啃下这两块硬骨头。