

# 20220518-机器学习

---

## 1.学习内容

### 1.1 机器学习

CNN类

## 2.结果描述

## 1.学习内容

### 1.1 机器学习

CNN类

```
1  #pragma once
2  #ifndef CNN_H_
3  #define CNN_H_
4
5  #include "Matrix.h"
6  #include <string>
7  #include <fstream>
8  #include <iostream>
9  #include <vector>
10 #include <math.h>
11 #pragma warning(disable:4996)
12
13 class CNN
14 {
15 public:
16     CNN();
17     std::vector<std::vector<uint8_t>> GetFeature(std::string
feature_file);
18
19     std::vector<uint8_t> GetLabel(std::string label_file);
20
21     std::vector<std::vector<uint8_t>> labelMatTran(std::vector<uint8_t>&
labelMat);
22
23     std::vector<std::vector<float>> filterInl(int num_f,int num_r, int
num_c);
24
25     std::vector<float> convBiasInl();
26
27     std::vector<std::vector<float>> convLayer(
28         std::vector<std::vector<uint8_t>> &FeatureMat,
29         std::vector<std::vector<float>> &filter,
30         std::vector<float> &biasMat,int picIndex);
31
32     std::vector<std::vector<float>> convActivate(
33         std::vector<std::vector<float>>& convMat);
34
35     std::vector<std::vector<float>> poolingLayer(
36         std::vector<std::vector<float>>& convMat_);
37
38     std::vector<float> outputBiasInl();
39
40     std::vector<std::vector<std::vector<float>>> outputWeightInl(int
stride);
41
```

```

42     std::vector<float> outputLayer(
43         std::vector<std::vector<float>>& poolingMat,
44         std::vector<std::vector<std::vector<float>>>& outputWeight,
45         std::vector<float>& biasMat);
46
47     std::vector<float> softmax(std::vector<float>& outputMat);
48
49     std::vector<float> Train(
50         int batchSize,
51         std::vector<std::vector<uint8_t>>& featureMat,
52         std::vector<uint8_t>& labelMat,
53         std::vector<std::vector<uint8_t>>& labelMatZ0,
54         std::vector<std::vector<float>>& filterMat,
55         std::vector<float>& convBias,
56         std::vector<std::vector<std::vector<float>>>& outputWeight,
57         std::vector<float>& outputBias);
58
59
60     void ParamUpdate();
61
62     public:
63         uint32_t convert_to_little_endian(const unsigned char* bytes);
64         float MaxPool(std::vector<float> poolBlock);
65
66
67     private:
68         int numPic;
69         int numRowsPixel;
70         int numColPixel;
71         int PicSize;
72         int numFilter;
73         int filterRow;
74         int filterCol;
75         int filterSize;
76         int convEleNum;
77         int numLabel;
78         int poolStride;
79         int convMatColNum;
80         int poolMatColNum;
81         int poolMatSize;
82         float softmaxMediator;
83     };
84
85     #endif

```

```
1  #include "CNN.h"
2
3  CNN::CNN()
4  {
5      numLabel = 10;
6  }
7
8  std::vector<std::vector<uint8_t>> CNN::GetFeature(std::string
feature_file)
9  {
10     std::ifstream fp(feature_file, std::ios::binary);
11     while (!fp.is_open())
12     {
13         std::cout << "Can not open feature file!" << std::endl;
14         exit(-1);
15     }
16
17     uint32_t header[4]={};
18     unsigned char bytes[4];
19
20     for (int i = 0; i < 4; i++)
21     {
22         if (fp.read((char*)bytes, sizeof(bytes)))
23         {
24             header[i] = convert_to_little_endian(bytes);
25         }
26     }
27     numPic = header[1];
28     numRowsPixel = header[2];
29     numColPixel = header[3];
30     PicSize = numRowsPixel * numColPixel;
31     std::vector < std::vector<uint8_t>> featureMat;
32     for (int j = 0; j < numPic; j++)
33     {
34         std::vector<uint8_t> imageF;
35         for (int k = 0; k < PicSize; k++)
36         {
37             uint8_t element[1];
38             if (fp.read((char*)&element, sizeof(element)))
39             {
40                 imageF.push_back(element[0]);
41             }
42         }
43         featureMat.push_back(imageF);
44     }
```

```

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>>
80     CNN::labelMatTran(std::vector<uint8_t>& labelMat)
81     {
82         std::vector<std::vector<uint8_t>> labelMatZ0;
83         for (int i = 0; i < numPic; i++)
84         {
85             std::vector<uint8_t> labelArrayZ0;
86             for (int j = 0; j < numLabel; j++)
87             {
88                 if (j == labelMat[j])
89                 {
90                     labelArrayZ0.push_back(1);
91                 }
92                 else

```

```

92     {
93         labelArrayZ0.push_back(0);
94     }
95 }
96 labelMatZ0.push_back(labelArrayZ0);
97 }
98 return labelMatZ0;
99 }
100
101 std::vector<std::vector<float>> CNN::filterInl(int num_f, int num_r, int
num_c)
102 {
103     numFilter = num_f;
104     filterRow = num_r;
105     filterCol = num_c;
106     filterSize = filterRow * filterCol;
107     std::vector<std::vector<float>> filter_matrix;
108     for (int i = 0; i < num_f; i++)
109     {
110         std::vector<float> filter_array;
111         for (int j = 0; j < filterSize; j++)
112         {
113             float randW = (-1) + 2 * rand() / float(RAND_MAX);
114             filter_array.push_back(randW);
115         }
116         filter_matrix.push_back(filter_array);
117     }
118     return filter_matrix;
119 }
120
121 std::vector<float> CNN::convBiasInl()
122 {
123     std::vector<float> biasMatrix;
124
125     for (int i = 0; i < numFilter; i++)
126     {
127         float randB = (-1) + 2 * rand() / float(RAND_MAX);
128
129         biasMatrix.push_back(randB);
130     }
131     return biasMatrix;
132 }
133
134 std::vector<std::vector<float>> CNN::convLayer(
135     std::vector<std::vector<uint8_t>> &FeatureMat,
136     std::vector<std::vector<float>> &filter,
137     std::vector<float> &biasMat,
138     int picIndex)

```

```

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

```

```

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

```



```

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

```

```

277     for (int j = 0; j < numLabel; j++)
278     {
279         outputMat[j] = std::exp(outputMat[j]) / sum;
280     }
281     return outputMat;
282 }
283
284
285 uint32_t CNN::convert_to_little_endian(const unsigned char* bytes)
286 {
287     return(uint32_t)(
288         (bytes[0] << 24) |
289         (bytes[1] << 16) |
290         (bytes[2] << 8) |
291         (bytes[3])
292     );
293 }
294
295 float CNN::MaxPool(std::vector<float> poolBlock)
296 {
297     float max = float(-100);
298     for (auto it = poolBlock.begin(); it != poolBlock.end(); it++)
299     {
300         if (max < *it)
301             max = *it;
302     }
303     return max;
304 }
305
306 std::vector<float> CNN::Train(
307     int batchSize,
308     std::vector<std::vector<uint8_t>>& featureMat,
309     std::vector<uint8_t>& labelMat,
310     std::vector<std::vector<uint8_t>>& labelMatZ0,
311     std::vector<std::vector<float>>& filterMat,
312     std::vector<float>& convBias,
313     std::vector<std::vector<std::vector<float>>>& outputWeight,
314     std::vector<float>& outputBias)
315 {
316     int numPerBatch = numPic / batchSize;
317     float predLabel = 0;
318     uint8_t trueLabel = 0;
319     std::vector<float> lossMat;
320
321     for (int i = 0; i < batchSize; i++)
322     {
323         float EntropyLoss = 0;
324         std::vector<std::vector<std::vector<float>>> deltaWeightOutput;

```

```

325     std::vector<float> deltaBiasOutput;
326     std::vector<std::vector<float>> deltaWeightFilter;
327     std::vector<float> deltaBiasFilter;
328     //输出层参数初始化
329     for (int ii = 0; ii < numLabel; ii++)
330     {
331         std::vector<std::vector<float>> vec1;
332         for (int jj = 0; jj < numFilter; jj++)
333         {
334             std::vector<float> vec2;
335             for (int kk = 0; kk < poolMatSize; kk++)
336             {
337                 vec2.push_back(0);
338             }
339             vec1.push_back(vec2);
340         }
341         deltaWeightOutput.push_back(vec1);
342         deltaBiasOutput.push_back(0);
343     }
344     //卷积层参数初始化
345     for (int qq = 0; qq < numFilter; qq++)
346     {
347         std::vector<float> vec3;
348         for (int dd = 0; dd < filterSize; dd++)
349         {
350             vec3.push_back(0);
351         }
352         deltaWeightFilter.push_back(vec3);
353         deltaBiasFilter.push_back(0);
354     }
355     //开始训练
356     for (int j = 0; j < numPerBatch; j++)
357     {
358         std::vector<std::vector<float>> convMat =
convLayer(featureMat, filterMat, convBias, i * numPerBatch + j);
359         std::vector<std::vector<float>> activatedMat =
convActivate(convMat);
360         std::vector<std::vector<float>> poolingMat =
poolingLayer(activatedMat);
361         std::vector<float> outputMat =
outputLayer(poolingMat, outputWeight, outputBias);
362         std::vector<float> softmaxed =
softmax(outputMat);
363         //输出层权重及bias更新
364         float Mediator = 1 / (softmaxMediator * softmaxMediator) *
(-1) * 1 / float(numPerBatch);
365         for (int k = 0; k < numLabel; k++)
366     {

```

```

367         float output = outputMat[k];
368         float softmaxValue = softmaxed[k];
369         float backBar = 0;
370         for (int d = 0; d < numLabel; d++)
371         {
372             if (d == k)
373             {
374                 backBar += labelMatZ0[i * numPerBatch + j][d] *
375 (softmaxMediator - std::exp(output)) / softmaxed[d];
376             }
377             else
378             {
379                 backBar += (-1) * labelMatZ0[i * numPerBatch +
380 j][d] * std::exp(outputMat[d]) / softmaxed[d];
381             }
382         }
383         for (int p = 0; p < numFilter; p++)
384         {
385             for (int q = 0; q < poolMatSize; q++)
386             {
387                 float delW = Mediator * poolingMat[p][q] *
388 std::exp(output) * backBar;
389                 deltaWeightOutput[k][p][q] += delW;
390             }
391         }
392         float delB = Mediator * std::exp(output) * backBar;
393         deltaBiasOutput[k] += delB;
394     }
395     //filter权重及bias更新
396     std::vector<float> filterBackBarVec;
397     float filterBackBarMed = softmaxMediator * softmaxMediator;
398     for (int a1 = 0; a1 < numLabel; a1++)
399     {
400         float filterBackBar = 0;
401         for (int a2 = 0; a2 < numLabel; a2++)
402         {
403             if (a2 == a1)
404             {
405                 filterBackBar += labelMatZ0[i * numPerBatch + j]
406 [a2] * (softmaxMediator -
407 std::exp(outputMat[a1])) * std::exp(outputMat[a1]) / softmaxed[a2];
408             }
409             else
410             {
411                 filterBackBar += (-1) * labelMatZ0[i *
412 numPerBatch + j][a2] * std::exp(outputMat[a2]) * std::exp(outputMat[a1])
413 / softmaxed[a2];
414             }
415         }
416     }

```

```

408     }
409     filterBackBar = (1 / filterBackBarMed) * filterBackBar;
410     filterBackBarVec.push_back(filterBackBar);
411 }
412
413 //卷积层的元素对filter的权重参数的求导
414 std::vector<std::vector<float>> filToPixMat;
415 for (int fw = 0; fw < filterSize; fw++)
416 {
417     std::vector<float> filToPixArray;
418     for (int ele = 0; ele < convEleNum; ele++)
419     {
420         int index = (fw / filterRow) * numRowsPixel +
421 (filterRow - 1) * (ele / convMatColNum) + ele + (fw + filterRow) %
422 filterRow;
423         filToPixArray.push_back(featureMat[i * numPerBatch +
424 j][index]);
425     }
426     filToPixMat.push_back(filToPixArray);
427 }
428
429 for (int a3 = 0; a3 < numFilter; a3++)
430 {
431     //权重更新
432     for (int a4 = 0; a4 < filterSize; a4++)
433     {
434         //激活函数对filter权重的求导
435         std::vector<float> actTowMat;
436         for (int actw = 0; actw < convEleNum; actw++)
437         {
438             int actwV = activatedMat[a3][actw] * (1 -
439 activatedMat[a3][actw]) * filToPixMat[a4][actw];
440             actTowMat.push_back(actw);
441         }
442         //池化矩阵对filter权重的求导
443         std::vector<float> poolTow;
444         for (int poolindex = 0; poolindex < poolMatSize;
445 poolindex++)
446         {
447             float poolTowV = 0;
448             std::vector<float> poolCmp;
449             std::vector<int> poolCmpIndex;
450             for (int poolstr = 0; poolstr < 2 * poolStride;
451 poolstr++)
452             {
453                 int cuteIndex = (poolindex / poolMatColNum)
454 * (2 * poolMatColNum) + (poolStride * poolindex) +

```

```

448             (2 * poolMatColNum) * (poolstr /
poolStride) + (poolstr + poolStride) % poolStride;
449             poolCmpIndex.push_back(cuteIndex);
450             poolCmp.push_back(activatedMat[a3]
[cuteIndex]));
451         }
452         for (int poolcmpI = 0; poolcmpI < 2 *
poolStride; poolcmpI++)
453     {
454         if (poolCmp[poolcmpI] == poolingMat[a3]
[poolindex])
455     {
456         poolTowV = 1 *
actTowMat[poolCmpIndex[poolcmpI]];
457         poolTow.push_back(poolTowV);
458     }
459     else
460     {
461         poolTow.push_back(0);
462     }
463     }
464     }
465     float filterWeightUpdate = 0;
466     for (int a5 = 0; a5 < numLabel; a5++)
467     {
468         float filterForeBarW = 0;
469         for (int a6 = 0; a6 < poolMatSize; a6++)
470         {
471             filterForeBarW += outputWeight[a5][a3][a6]
*poolTow[a6];
472         }
473
474         filterWeightUpdate+= filterForeBarW *
filterBackBarVec[a5];
475     }
476
477     deltaWeightFilter[a3][a4] +=filterWeightUpdate ;
478
479     }
480     //bias更新
481     std::vector<float> actTobMat;
482     for (int actb = 0; actb < convEleNum; actb++)
483     {
484         int actbV = activatedMat[a3][actb] * (1 -
activatedMat[a3][actb]);
485         actTobMat.push_back(actbV);
486     }
487     std::vector<float> poolTob;

```

```

488         for (int poolindex = 0; poolindex < poolMatSize;
poolindex++)
489     {
490         float poolTobV = 0;
491         std::vector<float> poolCmp;
492         std::vector<int> poolCmpIndex;
493         for (int poolstr = 0; poolstr < 2 * poolStride;
poolstr++)
494     {
495         int cuteIndex = (poolindex / poolMatColNum) * (2
* poolMatColNum) + (poolStride * poolindex) +
496         (2 * poolMatColNum) * (poolstr / poolStride)
+ (poolstr + poolStride) % poolStride;
497         poolCmpIndex.push_back(cuteIndex);
498         poolCmp.push_back(activatedMat[a3][cuteIndex]);
499     }
500         for (int poolcmpI = 0; poolcmpI < 2 * poolStride;
poolcmpI++)
501     {
502         if (poolCmp[poolcmpI] == poolingMat[a3]
[poolindex])
503     {
504         poolTobV = 1 *
actTobMat[poolCmpIndex[poolcmpI]];
505         poolTob.push_back(poolTobV);
506     }
507         else
508     {
509         poolTob.push_back(0);
510     }
511     }
512     }
513     float filterBiasUpdate = 0;
514     for (int a5 = 0; a5 < numLabel; a5++)
515     {
516         float filterForeBarB = 0;
517         for (int a6 = 0; a6 < poolMatSize; a6++)
518     {
519         filterForeBarB += outputWeight[a5][a3][a6] *
poolTob[a6];
520     }
521
522         filterBiasUpdate += filterForeBarB *
filterBackBarVec[a5];
523     }
524     deltaBiasFilter[a3] += filterBiasUpdate ;
525 }
526 trueLabel = labelMat[i * numPerBatch + j];

```

```
527         EntropyLoss += (-1) * trueLabel *  
std::log(softmaxed[trueLabel]);  
528     }  
529     EntropyLoss = 1 / float(numPerBatch) * EntropyLoss;  
530     lossMat.push_back(EntropyLoss);  
531 }  
532 return lossMat;  
533 }
```

## 2.结果描述

今天完成了卷积层权重及bias更新的代码编写，其中有几处需要公式推导，费了不少脑细胞，不过最后总算还是想明白了。目前代码中由于存在大量循环，还没能在短时间内完成一次梯度更新。明天重点优化一下代码。