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

▼ CNN.h C++ □ 复制代码

```
1
     #pragma once
 2
     #ifndef CNN H
     #define CNN H
 4
 5 ▼ #include "Matrix.h"
     #include <string>
     #include <fstream>
7
     #include <iostream>
     #include <vector>
     #include <math.h>
10
11
     #pragma warning(disable:4996)
12
13
     class CNN
14 ▼ {
15
     public:
16
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<float>>> outputWeightInl(int
     stride):
41
```

```
42
         std::vector<float> outputLayer(
             std::vector<std::vector<float>>& poolingMat,
43
             std::vector<std::vector<float>>>& outputWeight,
44
             std::vector<float>& biasMat);
45
46
47
         std::vector<float> softmax(std::vector<float>& outputMat);
48
49
         std::vector<float> Train(
50
             int batchSize,
             std::vector<std::vector<uint8 t>>& featureMat,
51
             std::vector<uint8 t>& labelMat,
52
53
             std::vector<std::vector<uint8_t>>& labelMatZO,
             std::vector<std::vector<float>>& filterMat,
54
             std::vector<float>& convBias,
55
56
             std::vector<std::vector<float>>>& outputWeight,
57
             std::vector<float>& outputBias);
58
59
         void ParamUpdate();
60
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 numRowPixel;
         int numColPixel;
70
         int PicSize;
71
72
         int numFilter;
         int filterRow;
73
         int filterCol;
74
75
         int filterSize;
76
         int convEleNum;
77
         int numLabel;
78
         int poolStride;
79
         int convMatColNum;
80
         int poolMatColNum;
         int poolMatSize;
81
82
         float softmaxMediator;
83
     };
84
85
     #endif
```

```
CNN.cpp
```

```
1 ▼ #include "CNN.h"
3
     CNN::CNN()
4 ▼ {
 5
          numLabel = 10;
6
     }
7
     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 -
          {
              std::cout << "Can not open feature file!" << std::endl;</pre>
13
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
          numRowPixel = header[2];
29
          numColPixel = header[3];
30
          PicSize = numRowPixel * 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++)</pre>
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
     std::vector<uint8_t> CNN::GetLabel(std::string label_file)
49
50 ▼
          FILE* lp = fopen(label file.c str(), "r");
51
         while (!lp)
52
53 ▼
          {
              std::cout << "Can not open label file" << std::endl;</pre>
54
55
              exit(-1);
56
          }
57
          uint32 t lheader[2]={};
58
          unsigned char lbytes[4];
          for (int i = 0; i < 2; i++)
59
60 -
          {
              if (std::fread(lbytes, sizeof(lbytes), 1, lp))
61
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];
              if (std::fread(lelement, sizeof(lelement), 1, lp))
70
71 -
              {
72
                  labelData.push_back(lelement[0]);
              }
73
74
          }
75
         //std::cout << static_cast<int>(labelData[2]) << std::endl;</pre>
76
          return labelData;
77
     }
78
79
     std::vector<std::vector<uint8_t>>
     CNN::labelMatTran(std::vector<uint8_t>& labelMat)
80 ▼ {
          std::vector<std::vector<uint8_t>> labelMatZO;
81
          for (int i = 0; i < numPic; i++)
82
83 🔻
          {
84
              std::vector<uint8_t> labelArrayZ0;
              for (int j = 0; j < numLabel; j++)
85
86 -
              {
                  if (j == labelMat[j])
87
88 ▼
                  {
                      labelArrayZ0.push_back(1);
89
90
91
                  else
```

```
92 -
                   {
 93
                       labelArrayZ0.push_back(0);
                   }
 94
               }
 95
               labelMatZO.push_back(labelArrayZO);
 96
 97
           }
 98
           return labelMatZ0;
      }
99
100
      std::vector<std::vector<float>>> CNN::filterInl(int num_f, int num_r, int
101
      num_c)
102 ▼ {
103
           numFilter = num f;
104
           filterRow = num r;
           filterCol = num c;
105
106
           filterSize = filterRow * filterCol;
           std::vector<std::vector<float>> filter_matrix;
107
          for (int i = 0; i < num f; i++)
108
          {
109 -
110
               std::vector<float> filter_array;
111
               for (int j = 0; j < filterSize; j++)</pre>
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;
           convEleNum = (numRowPixel - filterRow + 1) * (numColPixel -
141
       filterCol + 1);
142
           float conValue;
           for (int i = 0; i < numFilter; i++)
143
144 ▼
           {
145
               std::vector<float> convArray;
146
               for (int j = 0; j < (numRowPixel - filterRow + 1); j++)</pre>
147 ▼
                   for (int k = 0; k < (numColPixel - filterCol + 1); k++)</pre>
148
                   {
149 ▼
150
                       conValue = 0;
151
                       for (int p = 0; p < filterRow; p++)</pre>
152 ▼
153
                            for (int g = 0; g < filterCol; g++)</pre>
154 ▼
                            {
                                conValue += FeatureMat[picIndex][j * numRowPixel
155
      + 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
      std::vector<std::vector<float>>
168
      CNN::convActivate(std::vector<std::vector<float>> &convMat)
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
           }
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
           for (int i = 0; i < numFilter; i++)
185
186 ▼
               std::vector<float> poolingArray;
187
               for (int j = 0; j < poolMatColNum; j++)</pre>
188
189 ▼
190
                   for (int p = 0; p < poolMatColNum; p++)</pre>
191 ▼
                   {
192
                       std::vector<float> poolBlock;
193
                       for (int q = 0; q < poolStride; q++)</pre>
194 ▼
                       {
                           for (int d = 0; d < poolStride; d++)</pre>
195
196 ▼
197
                               poolBlock.push back(convMat [i][j *
       convMatColNum * poolStride + p * poolStride + convMatColNum * q + d]);
198
199
                       }
                       poolingArray.push back(MaxPool(poolBlock));
200
                   }
201
               }
202
               poolingMat.push_back(poolingArray);
203
204
           }
           return poolingMat;
205
206
      }
207
208
      std::vector<std::vector<float>>> CNN::outputWeightInl(int
      stride)
209 ▼ {
210
           poolStride = stride;
211
           convMatColNum = numRowPixel - filterRow + 1;
           poolMatColNum = convMatColNum / poolStride;
212
213
           poolMatSize = poolMatColNum * poolMatColNum;
214
           std::vector<std::vector<float>>> outputWeightMat;
215
           for (int i = 0; i < numLabel; i++)</pre>
216 -
           {
217
               std::vector<std::vector<float>> outputWeightLabelMat;
218
               for (int j = 0; j < numFilter; j++)
               {
219 -
                   std::vector<float> outputWeightArray;
220
221
                   for (int p = 0; p < poolMatSize; p++)</pre>
222 -
                   {
223
                       float randW = (-1) + 2 * rand() / float(RAND_MAX);
                       outputWeightArray.push_back(randW);
224
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:
          for (int i = 0; i < numLabel; i++)
236
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<float>>>& outputWeight,
          std::vector<float>& biasMat
247
248
      )
249 ▼ {
250
          std::vector<float> outputMat;
          for (int i = 0; i < numLabel; i++)</pre>
251
252 ▼
              float outputValue = 0;
253
              for (int j = 0; j < numFilter; j++)</pre>
254
255 ▼
              {
                  for (int p = 0; p < poolMatSize; p++)</pre>
256
                  {
257 ▼
258
                       //outputValue += outputWeight[i][j][p] * poolingMat[j]
      [p];
                  }
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
          float sum = float(0);
271
          for (int i = 0; i < numLabel; i++)
272
273 ▼
          {
274
              sum += std::exp(outputMat[i]);
275
276
          softmaxMediator = sum;
```

```
for (int j = 0; j < numLabel; j++)</pre>
277
278 -
          {
              outputMat[j] = std::exp(outputMat[j]) / sum;
279
280
          }
          return outputMat;
281
      }
282
283
284
      uint32_t CNN::convert_to_little_endian(const unsigned char* bytes)
285
286 ▼ {
287
          return(uint32 t)(
288
              (bytes[0] << 24) |
              (bytes[1] << 16) |
289
              (bytes[2] << 8) |
290
              (bytes[3])
291
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 -
          {
              if (max < *it)</pre>
300
                  max = *it;
301
302
          }
303
          return max;
      }
304
305
      std::vector<float> CNN::Train(
306
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,
          std::vector<std::vector<float>>>& outputWeight,
313
          std::vector<float>& outputBias)
314
315 ▼ {
316
          int numPerBatch = numPic / batchSize;
317
          float predLabel = 0;
          uint8_t trueLabel = 0;
318
          std::vector<float> lossMat;
319
320
321
          for (int i = 0; i < batchSize; i++)</pre>
322 ▼
          {
323
              float EntropyLoss = 0;
324
              std::vector<std::vector<float>>> deltaWeightOutput;
```

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

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

```
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++)</pre>
416 -
417
                       std::vector<float> filToPixArray;
418
                       for (int ele = 0; ele < convEleNum; ele++)</pre>
419 ▼
                           int index = (fw / filterRow) * numRowPixel +
420
      (filterRow - 1) * (ele / convMatColNum) + ele + (fw + filterRow) %
      filterRow;
421
                           filToPixArray.push back(featureMat[i * numPerBatch +
      i][index]);
422
423
                       filToPixMat.push_back(filToPixArray);
424
                   }
425
426
                   for (int a3 = 0; a3 < numFilter; a3++)
427 ▼
428
                       //权重更新
429
                       for (int a4 = 0; a4 < filterSize; a4++)
430 ▼
                           //激活函数对filter权重的求导
431
432
                           std::vector<float> actTowMat;
433
                           for (int actw = 0; actw < convEleNum; actw++)</pre>
434 ▼
                               int actwV = activatedMat[a3][actw] * (1 -
435
      activatedMat[a3][actw]) * filToPixMat[a4][actw];
436
                               actTowMat.push back(actw);
437
                           }
                           //池化矩阵对filter权重的求导
438
439
                           std::vector<float> poolTow;
                           for (int poolindex = 0; poolindex < poolMatSize;</pre>
440
      poolindex++)
441 -
                           {
442
                               float poolTowV = 0;
443
                               std::vector<float> poolCmp;
                               std::vector<int> poolCmpIndex;
444
                               for (int poolstr = 0; poolstr < 2 * poolStride;</pre>
445
      poolstr++)
446 ▼
                               {
                                   int cuteIndex = (poolindex / poolMatColNum)
447
      * (2 * poolMatColNum) + (poolStride * poolindex) +
```

```
448
                                         (2 * poolMatColNum) * (poolstr /
       poolStride) + (poolstr + poolStride) % poolStride;
449
                                    poolCmpIndex.push back(cuteIndex);
                                    poolCmp.push back(activatedMat[a3]
450
       [cuteIndex]);
                                }
451
                                for (int poolcmpI = 0; poolcmpI < 2 *</pre>
452
       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;
                            for (int a5 = 0; a5 < numLabel; a5++)</pre>
466
                            {
467 ▼
468
                                float filterForeBarW = 0;
469
                                for (int a6 = 0; a6 < poolMatSize; a6++)</pre>
                                {
470 -
                                    filterForeBarW += outputWeight[a5][a3][a6]
471
      *poolTow[a6];
                                }
472
473
474
                                filterWeightUpdate+= filterForeBarW *
       filterBackBarVec[a5];
                            }
475
476
477
                            deltaWeightFilter[a3][a4] +=filterWeightUpdate;
478
                        }
479
                        //bias更新
480
481
                        std::vector<float> actTobMat;
482
                        for (int actb = 0; actb < convEleNum; actb++)</pre>
483 ▼
                            int actbV = activatedMat[a3][actb] * (1 -
484
       activatedMat[a3][actb]);
485
                            actTobMat.push_back(actbV);
486
487
                        std::vector<float> poolTob;
```

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

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。