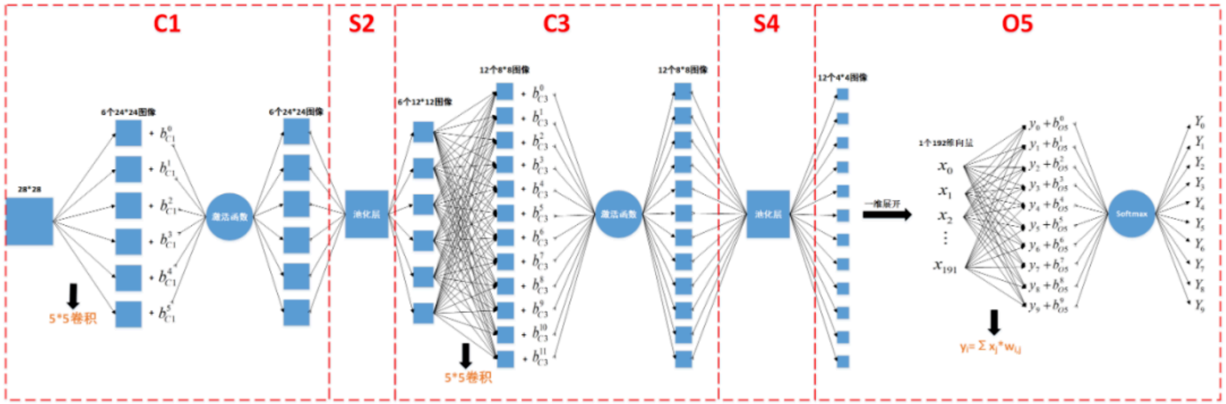
卷積神經網路原理及其C++/Opencv實作(7)—誤反向傳播程式碼實現

原創 sdff20201029 萌萌噠程序猴 2021-04-02 21:39

首先列出本系列部落格的連結：

1. 卷積神經網路原理及其C++/Opencv實作(1)
2. 卷積神經網路原理及其C++/Opencv實作(2)
3. 卷積神經網路原理及其C++/Opencv實作(3)
4. 卷積神經網路原理及其C++/Opencv實作(4)—誤反向傳播法
5. 卷積神經網路原理及其C++/Opencv實作(5)—參數更新
6. 卷積神經網路原理及其C++/Opencv實作(6)—前向傳播程式碼實現

上篇文章中我們講了5層網路的前向傳播的程式碼實現，有前向就有反向，本文就讓我們同樣使用C++和Opencv來實現反向傳播的程式碼吧~



如上圖所示，誤差訊息的反向傳播過程可分為以下5步：

1. Softmax-->Affine
2. Affine-->S4
3. S4-->C3
4. C3-->S2

5. S2-->C1

公式推導我們前文已經詳細講過，核心思想是複合函數的鍊式求導法則，下面我們分別闡述以上5個步驟的程式碼實作。

1. Softmax-->Affine

根據前文的推導，此步驟的反向傳播公式為，其中y為Affine層的輸出，Y為Softmax函數的輸出，t為標籤，0≤i<10。

$$d_{O5}^{i} = Y_{O5}^{i} - t_i$$

程式碼實現如下：

```
void softmax_bp(Mat outputData, Mat &e, OutLayer &O)
{
  for (int i = 0; i < O.outputNum; i++)
    e.ptr<float>(0)[i] = O.y.ptr<float>(0)[i] - outputData.ptr<float>(0)[i];

  //将Y-t保存到O5层的局部梯度中
  for (int i = 0; i < O.outputNum; i++)
    O.d.ptr<float>(0)[i] = e.ptr<float>(0)[i];// *sigma_derivation(O.y.ptr<flo
}
```

2. Affine-->S4

本步驟的反向傳播公式如下，其中x為Affine的輸入，w為Affine層的權重，0≤j<192。

$$\frac{\partial E}{\partial x_j} = \sum_{i=0}^{9} \frac{\partial E}{\partial y_{O5}^{i}} \cdot \frac{\partial y_{O5}^{i}}{\partial x_j} = \sum_{i=0}^{9} d_{O5}^{i} . w_{O5}^{ij}$$

Affine層的輸入有192個x，也就是說有192個E關於x的偏導數，把這192個偏導數依序重組成12個4*4的二維矩陣，作為S4層的局部梯度，其中有12個d，每個d都是4*4矩陣：

$$d_{S4}^{i}, 0 \le i < 12$$

程式碼實現如下：

```
void full2pool_bp(OutLayer O, PoolLayer &S)
{
  int outSize_r = S.inputHeight / S.mapSize;
  int outSize_c = S.inputWidth / S.mapSize;
  for (int i = 0; i < S.outChannels; i++)  //输出12张4*4图像
  {
    for (int r = 0; r < outSize_r; r++)
    {
      for (int c = 0; c < outSize_c; c++)
      {
        int wInt = i*outSize_c*outSize_r + r*outSize_c + c;  //i*outSize.c*ou
        for (int j = 0; j < O.outputNum; j++)   //O5输出层的输出个数
        {
          //把192个偏导数重组成12个4*4的二维矩阵，作为S4层的局部梯度
          S.d[i].ptr<float>(r)[c] = S.d[i].ptr<float>(r)[c] + O.d.ptr<float>(
        }
      }
    }
  }
}
```

3. S4-->C3

本步驟的反向傳播公式如下，其中upsample為我們前文講過的池化層向上採樣操作，DerivativeRelu為Relu函數的導數，我們前文也講過。本層的局部梯度是12個8*8的矩陣（0≤i<12）：

$$d_{C3}^{i}=upsample(d_{S4}^{i}).Derivative\,\mathrm{Re}\,lu(y_{C3}^{i})$$

上述公式的計算程式碼如下：

```
/*
矩阵上采样，upc及upr是池化窗口的列、行
如果是最大值池化模式，则把局域梯度放到池化前最大值的位置，比如池化窗口2*2，池化前最大值
5 9         5 0 0 9
     -->    0 0 0 0
3 6         0 0 0 0
                 3 0 0 6
如果是均值池化模式，则把局域梯度除以池化窗口的尺寸2*2=4:
5 9         1.25 1.25 2.25 2.25
     -->    1.25 1.25 2.25 2.25
3 6         0.75 0.75 1.5  1.5
            0.75 0.75 1.5  1.5
*/
Mat UpSample(Mat mat, int upc, int upr)   //均值池化层的向上采样
{
  //int i, j, m, n;
  int c = mat.cols;
  int r = mat.rows;

  Mat res(r*upr, c*upc, CV_32FC1);

  float pooling_size = 1.0 / (upc*upr);

```

```
  for (int j = 0; j < r*upr; j += upr)
  {
    for (int i = 0; i < c*upc; i += upc)  // 宽的扩充
    {
      for (int m = 0; m < upc; m++)
      {

        //res[j][i + m] = mat[j / upr][i / upc] * pooling_size;
        res.ptr<float>(j)[i + m] = mat.ptr<float>(j/upr)[i/upc] * pooling_siz
      }
    }

    for (int n = 1; n < upr; n++)      //   高的扩充
    {
      for (int i = 0; i < c*upc; i++)
      {
        //res[j + n][i] = res[j][i];
        res.ptr<float>(j+n)[i] = res.ptr<float>(j)[i];

      }
    }
  }
  return res;
}


//最大值池化层的向上采样
Mat maxUpSample(Mat mat, Mat max_position, int upc, int upr)
{
  int c = mat.cols;
```

```
  int r = mat.rows;

  int outsize_r = r*upr;
  int outsize_c = c*upc;

  Mat res = Mat::zeros(outsize_r, outsize_c, CV_32FC1);

  for (int j = 0; j < r; j++)
  {
    for (int i = 0; i < c; i++)
    {
      int index_r = max_position.ptr<int>(j)[i] / outsize_c;   //计算最大值的索
      int index_c = max_position.ptr<int>(j)[i] % outsize_c;
      res.ptr<float>(index_r)[index_c] = mat.ptr<float>(j)[i];
    }
  }
  return res;
}


void pool2cov_bp(PoolLayer S, CovLayer &C)
{
  for (int i = 0; i < C.outChannels; i++)   //12通道
  {
    Mat C3e;
    if (S.poolType == AvePool)  //均值
      C3e = UpSample(S.d[i], S.mapSize, S.mapSize);    //向上采样，把S4层的局域
    else if (S.poolType == MaxPool)  //最大值
      C3e = maxUpSample(S.d[i], S.max_position[i], S.mapSize, S.mapSize);

```

```
    for (int r = 0; r < S.inputHeight; r++)    //8*8
    {
      for (int c = 0; c < S.inputWidth; c++)
      {
        C.d[i].ptr<float>(r)[c] = C3e.ptr<float>(r)[c] * sigma_derivation(C.y
      }
    }
  }
}
```

4. C3-->S2

本步驟的反向傳播公式如下，其中rotate180為我們前文講過的矩陣順時針旋轉180度操作，本層的局部梯度為6個(8+5-1)* (8+5-1) =12 *12的矩陣（0≤j<6）：

$$d_{S2}^{j} = \sum_{i=0}^{11} d_{C3}^{i} * rotate180(k_{C3}^{ij})$$

程式碼實現如下：

```

Mat cov(Mat map, Mat inputData, int type)
{
  Mat flipmap;
  flip(map, flipmap, -1);   //卷积核先顺时针旋转180度
  Mat res = correlation(flipmap, inputData, type);   //然后再进行卷积

  return res;
```

```
}

void cov2pool_bp(CovLayer C, int cov_type, PoolLayer &S)
{

  for (int i = 0; i < S.outChannels; i++)   //S2有6通道
  {
    for (int j = 0; j < S.inChannels; j++)  //C3有12通道
    {
      //得到12*12矩阵：full模式下为(inSize+mapSize-1)*(inSize+mapSize-1)
      Mat corr = cov(C.mapData[i][j], C.d[j], cov_type);
      S.d[i] = S.d[i] + corr;   //矩阵累加：cnn->S2->d[i] = cnn->S2->d[i] + cc
    }
  }
}
```

5. S2-->C1

本步驟的反向傳播公式如下，其中upsample為池化層向上採樣操作，DerivativeRelu為Relu函數的導數。本層的局部梯度是6個24*24的矩陣（0≤j<6）：

$$d_{C1}^{j} = upsample(d_{S2}^{j}).Derivative\,\mathrm{Re}\,lu(y_{C1}^{j})$$

由於本步驟的操作與上述第3步一樣，只是輸入、輸出參數不一樣，所以也可以呼叫第3步實現的pool2cov_bp函數來實現本步驟的反向傳播。

最後把上述5個步驟合起來，反向傳播的代號為：

```
//outputData为标签
```

```
void cnnbp(CNN &cnn, Mat outputData)
{
  softmax_bp(outputData, cnn.e, cnn.O5);
  full2pool_bp(cnn.O5, cnn.S4);
  pool2cov_bp(cnn.S4, cnn.C3);
  cov2pool_bp(cnn.C3, full, cnn.S2);
  pool2cov_bp(cnn.S2, cnn.C1);
}
```

歡迎掃碼追蹤以下微信公眾號，接下來會不定時更新更加精彩的內容噢～

人工智慧 27 深度學習 26 機器學習 33 C++ 70 Opencv 50

人工智慧 目錄

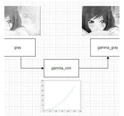
上一篇
卷積神經網路原理及其C++/Opencv實作(6)—前向傳播程式碼實現

下一篇
卷積神經網路原理及其C++/Opencv實作(8)—手寫數位影像辨識

閱讀原文

喜歡此內容的人還喜歡

數位影像處理之gamma矯正
FPGA開源工作室



混凝土模板荷載與壓力計算
忒修斯破船

NJ系列電子凸輪應用分享
Karl工控