卷积层梯度反向传播

卷积层的梯度反向传播可以使用大矩阵乘法来实现。

梯度反向传播的计算过程和卷积层的正向计算过程相反。

利用矩阵乘法实现卷积层的正向传播过程：

（1）将输入特征图变换为矩阵matric_data

（2）进行矩阵相乘和非线性激活（等价于全连接层）后得到filter_data

（3）将filter_data变换为输出特征图

卷积层梯度反向传播是已知输出特征图的梯度，求输入特征图的梯度及卷积核梯度。

利用矩阵乘法实现卷积层的反向传播过程：

（1）把特征图的梯度dout_data变换为矩阵形式（正向传播第（3）步的逆过程）

（2）将全连接层和激活曾的梯度进行反向传播

（3）把第（2）步得到的矩阵梯度变换为特征图形状的梯度，即得到输入特征图的梯度。

梯度反向传播时，需要正向计算过程得到的matric_data和filter_data.

例：卷积层梯度反向传播代码实现

```
import numpy as np

# 定义超参数
filter_size = 3
filter_size2 = filter_size * filter_size
stride = 1
padding = (filter_size - 1) // 2

batch = 8
 (in_height, in_width, in_depth) = (32, 48, 16)
 (out_height, out_width, out_depth) = (32, 48, 32)
 out_size = out_width * out_height
 # 随机生成一些可以在正向传播种获得的数据
 dout_data = np.random.randn(batch, out_height, out_width, out_depth) # 上一次梯度反向传播得到，是输入梯度
 matric_data = np.random.randn(out_size * batch, filter_size2 * in_depth)
 filter_data = np.random.randn(out_size * batch, out_depth)
 weights = 0.01 * np.random.randn(filter_size2 * in_depth, out_depth)

 # 将dout_data变换成矩阵形式dfilter_data,dout_data的每个深度列就是dfilter_data的一行
 dfilter_data = np.zeros_like(filter_data)
```

```
21 for i_batch in range(batch):
22  i_batch_size = i_batch * out_size
23  for i_height in range(out_height):
24  i_height_size = i_batch_size + i_height * out_width
25  for i_width in range(out_width):
26  dfilter_data[i_height_size + i_width, :] = dout_data[i_batch, i_height,
i_width, :]
27
28 # 进行激活层和全连接层反向传播，得到权重梯度dweights、dbias、矩阵形式的梯度dma
tric_data:
29 dfilter_data[filter_data <= 0] = 0
30
31 dweights = np.dot(matric_data.T, dfilter_data)
32 dbias = np.sum(dfilter_data, axis=0, keepdims=True)
33 dmatric_data = np.dot(dfilter_data, weights.T)
34
35 # 将dmatric_data变换为特征图形状的梯度，得到输入特征图的梯度
36 padding_height = in_height + 2*padding
37 padding_width = in_width + 2*padding
38 dpadding_data = np.zeros((batch, padding_height, padding_width,
in_depth))
39
40 height_ef = padding_height - filter_size + 1
41 width_ef = padding_width - filter_size + 1
42 for i_batch in range(batch):
43  i_batch_size = i_batch*out_size
44  for i_h, i_height in zip(range(out_height), range(0, height_ef,
stride)):
45  i_height_size = i_batch_size + i_h*out_width
46  for i_w, i_width in zip(range(out_width), range(0, width_ef, stride)):
47  dpadding_data[i_batch, i_height:i_height + filter_size,
i_width:i_width+filter_size, :] += dmatric_data[i_height_size + i_w,:].resh
ape(filter_size, filter_size, -1)
48  # dmatric_data的每一行数据是dpadding_data的局部窗口的一个数据，行向量需要res
hape成3D矩阵
49  # dpadding_data数据有多条路径（局部窗口有重叠）得到梯度，所以需要累加
50 if padding:
51  din_data = dpadding_data[:, padding:-padding, padding:-padding, :]
52 else:
53  din_data = dpadding_data
```

有了din_data就可以向前一层进行梯度反向传播。