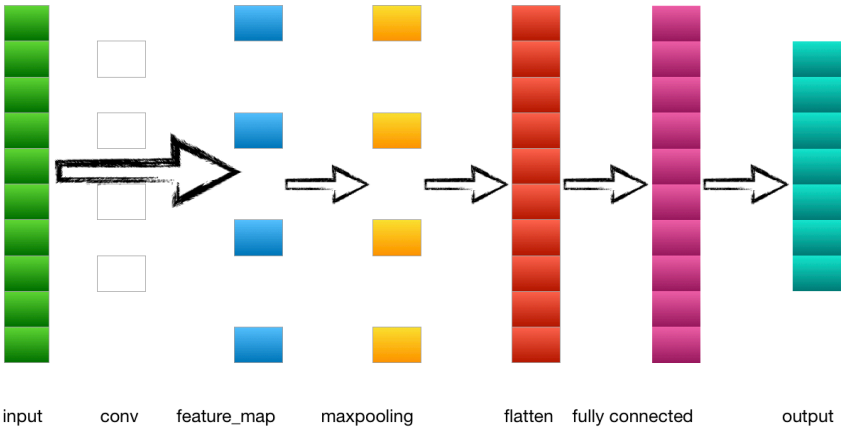
【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络

上篇文章中我们讲解了卷积神经网络的基本原理，包括几个基本层的定义、运算规则等。本文主要写卷积神经网络如何进行一次完整的训练，包括前向传播和反向传播，并自己手写一个卷积神经网络。如果不了解基本原理的，可以先看看上篇文章：【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理

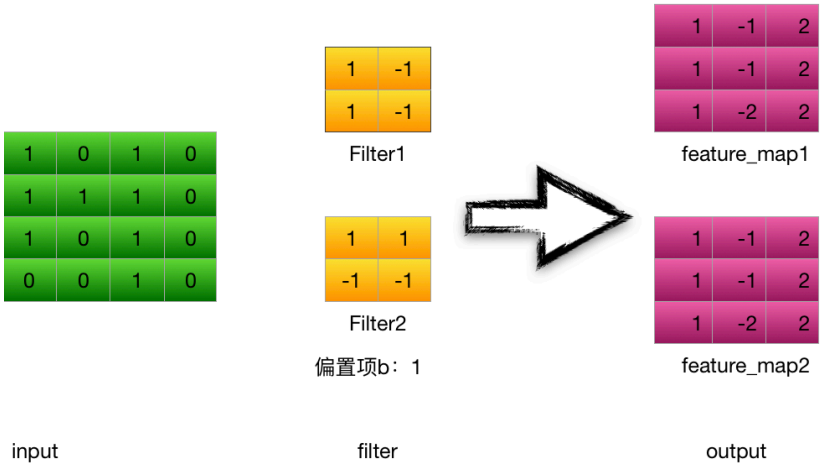
卷积神经网络的前向传播

首先我们来看一个最简单的卷积神经网络：



1.输入层----->卷积层

以上一节的例子为例，输入是一个4*4 的image，经过两个2*2的卷积核进行卷积运算后，变成两个3*3的feature_map



以卷积核filter1为例(stride = 1)：

计算第一个卷积层神经元o_{11}的输入:

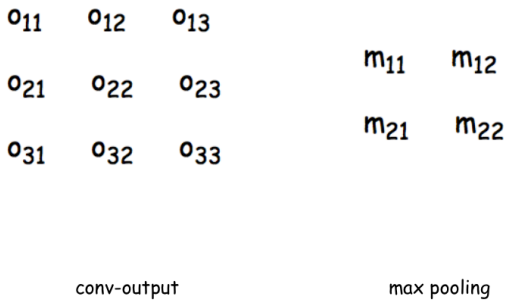
$$\begin{aligned} net_{o_{11}} &= conv(input, filter) \\ &= i_{11} \times h_{11} + i_{12} \times h_{12} + i_{21} \times h_{21} + i_{22} \times h_{22} \\ &= 1 \times 1 + 0 \times (-1) + 1 \times 1 + 1 \times (-1) = 1 \end{aligned} \tag{1}$$

神经元o_{11}的输出:(此处使用Relu激活函数)

$$\begin{aligned} out_{o_{11}} &= activators(net_{o_{11}}) \\ &= max(0, net_{o_{11}}) = 1 \end{aligned} \tag{2}$$

其他神经元计算方式相同

2.卷积层----->池化层



计算池化层m_{11} 的输入(取窗口为 2 * 2),池化层没有激活函数

我的评论
我的参与
最新评论
我的标签

最新随笔

1.谈谈坚持这件小事
2.我在北京这几年（全）
3.【原】深度学习的一些经验总结和建议 | To do v.s Not To Do
4.如何高效利用一场技术分享？
5.深度学习分布式训练及CTR预估模型应用
6.两个月刷完Leetcode前400题经验总结
7.【机器学习】如何解决数据不平衡问题
8.LeetCode刷题专栏第一篇--思维导图&时间安排
9.【资料总结】| Deep Reinforcement Learning 深度强化学习
10.2018年总结与2019年目标与计划

我的标签

深度学习(22)
机器学习(10)
数据挖据(5)
Spark(4)
学习心得(3)
数据挖掘(2)
推荐系统(2)
文本挖掘(2)
LeetCode(2)
年度总结(2)
更多

积分与排名

积分 - 177053
排名 - 3425

随笔分类 (56)

Spark(7)
机器学习笔记(12)
深度学习(23)
数据挖掘(9)
推荐系统(2)
文本挖掘(3)

随笔档案 (56)

2019年8月(2)
2019年7月(2)
2019年5月(2)
2019年3月(1)
2019年2月(1)
2019年1月(2)
2018年6月(1)
2018年5月(1)
2018年3月(1)
2018年2月(2)
2018年1月(4)
2017年12月(4)
2017年11月(4)
2017年10月(2)
2017年9月(1)
2016年12月(1)

$$net_{m_{11}} = max(o_{11}, o_{12}, o_{21}, o_{22}) = 1$$
$$out_{m_{11}} = net_{m_{11}} = 1 \quad (3)$$

3.池化层----->全连接层

池化层的输出到flatten层把所有元素"拍平"，然后到全连接层。

4.全连接层----->输出层

全连接层到输出层就是正常的神经元与神经元之间的邻接相连，通过softmax函数计算后输出到output，得到不同类别的概率值，输出概率值最大的即为该图片的类别。

卷积神经网络的反向传播

传统的神经网络是全连接形式的，如果进行反向传播，只需要由下一层对前一层不断的求偏导，即求链式偏导就可以求出每一层的误差敏感项，然后求出权重和偏置项的梯度，即可更新权重。而卷积神经网络有两个特殊的层：卷积层和池化层。池化层输出时不需要经过激活函数，是一个滑动窗口的最大值，一个常数，那么它的偏导是1。池化层相当于对上层图片做了一个压缩，这个反向求误差敏感项时与传统的反向传播方式不同。从卷积后的feature_map反向传播到前一层时，由于前向传播时是通过卷积核做卷积运算得到的feature_map，所以反向传播与传统的也不一样，需要更新卷积核的参数。下面我们介绍一下池化层和卷积层是如何做反向传播的。

在介绍之前，首先回顾一下传统的反向传播方法：

1.通过前向传播计算每一层的输入值$net_{i,j}$ (如卷积后的feature_map的第一个神经元的输入：$net_{i_{11}}$)

2.反向传播计算每个神经元的误差项$\delta_{i,j}$，$\delta_{i,j} = \frac{\partial E}{\partial net_{i,j}}$，其中E为损失函数计算得到的总体误差，可以用平方差，交叉熵等表示。

3.计算每个神经元权重$w_{i,j}$的梯度，$\eta_{i,j} = \frac{\partial E}{\partial net_{i,j}} \cdot \frac{\partial net_{i,j}}{\partial w_{i,j}} = \delta_{i,j} \cdot out_{i,j}$

4.更新权重 $w_{i,j} = w_{i,j} - \lambda \cdot \eta_{i,j}$(其中$\lambda$为学习率)

卷积层的反向传播

由前向传播可得：

每一个神经元的值都是上一个神经元的输入作为这个神经元的输入，经过激活函数激活之后输出，作为下一个神经元的输入，在这里我用i_{11}表示前一层,o_{11}表示i_{11}的下一层。那么$net_{i_{11}}$就是i11这个神经元的输入，$out_{i_{11}}$就是i11这个神经元的输出，同理，$net_{o_{11}}$就是o11这个神经元的输入，$out_{o_{11}}$就是o_{11}这个神经元的输出,因为上一层神经元的输出 = 下一层神经元的输入，所以$out_{i_{11}} = net_{o_{11}}$，这里我为了简化，直接把$out_{i_{11}}$记为$i_{11}$

$$\begin{aligned} i_{11} &= out_{i_{11}} \\ &= activators(net_{i_{11}}) \\ net_{o_{11}} &= conv(input, filter) \\ &= i_{11} \times h_{11} + i_{12} \times h_{12} + i_{21} \times h_{21} + i_{22} \times h_{22} \\ out_{o_{11}} &= activators(net_{o_{11}}) \\ &= max(0, net_{o_{11}}) \end{aligned} \quad (4)$$

$net_{i_{11}}$表示上一层的输入，$out_{i_{11}}$表示上一层的输出

首先计算卷积的上一层的第一个元素i_{11}的误差项δ_{11}：

2016年7月(3)
2016年6月(3)
2016年5月(9)
2016年4月(6)
2016年3月(1)
2015年12月(3)

最新评论

1. Re:谈谈坚持这件小事
感谢，你已经帮助到我了。

--外方

2. Re:一文弄懂神经网络中的反向传播法——BackPropagation
隐藏层的误差是等于总误差，还是等于E_1w5+E_2w6啊

--Benys

3. Re:【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络
@风中de石头 谢谢，已经能够运行啦...

--3079779149

4. Re:我在北京这几年（全）
很幸运可以看到楼主的文章，关注楼主。大学的第一个暑假，因为疫情待在家好几个月，想在暑假学习深度学习相关知识却因为种种困难犹豫不前，效率低下，希望自己可以坚持下去，为了自己的目标，为了成为更好的自己。...

--ywqa

5. Re:我在北京这几年（全）
博主的文章很真实，比那些上来就鼓吹理想奋斗的文章的感触更深，能够更理性更客观的认识程序员这个职业，以及在北京一步一步奋斗所经历的点点滴滴。我也要冲向北京了，争取早日秋招上岸！！！

--MIIEo

6. Re:2018年总结与2019年目标与计划
这个复盘好，学习了

--6671

7. Re:一文弄懂神经网络中的反向传播法——BackPropagation
Charlotte you are so great Refer to there is back propagation.py that I modified will help you a lot ...

--fatalfeel

8. Re:我在北京这几年（全）
姐姐写的很好

--Details_K

9. Re:【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理
您好博主，为啥我感觉一开始最上面那个，参数个数是这样算的：784+(15784+151)+(1015+101)，您 784 1510，乘以10没有看懂是为什么， w^[l]的维数不是 (n^[l], n...

--douzujun

10. Re:【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络
我没看出来有偏置项了吖？

--小艾1

阅读排行榜

1. 一文弄懂神经网络中的反向传播法——BackPropagation(289091)

$$\delta_{11} = \frac{\partial E}{\partial net_{i_{11}}} = \frac{\partial E}{\partial out_{i_{11}}} \cdot \frac{\partial out_{i_{11}}}{\partial net_{i_{11}}} = \frac{\partial E}{\partial i_{11}} \cdot \frac{\partial i_{11}}{\partial net_{i_{11}}}$$

先计算$\frac{\partial E}{\partial i_{11}}$

此处我们并不清楚$\frac{\partial E}{\partial i_{11}}$怎么算，那可以先把input层通过卷积核做完卷积运算后的输出feature_map写出来:

$$\begin{aligned}
net_{o_{11}} &= i_{11} \times h_{11} + i_{12} \times h_{12} + i_{21} \times h_{21} + i_{22} \times h_{22} \\
net_{o_{12}} &= i_{12} \times h_{11} + i_{13} \times h_{12} + i_{22} \times h_{21} + i_{23} \times h_{22} \\
net_{o_{12}} &= i_{13} \times h_{11} + i_{14} \times h_{12} + i_{23} \times h_{21} + i_{24} \times h_{22} \\
net_{o_{21}} &= i_{21} \times h_{11} + i_{22} \times h_{12} + i_{31} \times h_{21} + i_{32} \times h_{22} \\
net_{o_{22}} &= i_{22} \times h_{11} + i_{23} \times h_{12} + i_{32} \times h_{21} + i_{33} \times h_{22} \\
net_{o_{23}} &= i_{23} \times h_{11} + i_{24} \times h_{12} + i_{33} \times h_{21} + i_{34} \times h_{22} \\
net_{o_{31}} &= i_{31} \times h_{11} + i_{32} \times h_{12} + i_{41} \times h_{21} + i_{42} \times h_{22} \\
net_{o_{32}} &= i_{32} \times h_{11} + i_{33} \times h_{12} + i_{42} \times h_{21} + i_{43} \times h_{22} \\
net_{o_{33}} &= i_{33} \times h_{11} + i_{34} \times h_{12} + i_{43} \times h_{21} + i_{44} \times h_{22}
\end{aligned} \tag{5}$$

然后依次对输入元素$i_{i,j}$求偏导

i_{11}的偏导：

$$\begin{aligned}
\frac{\partial E}{\partial i_{11}} &= \frac{\partial E}{\partial net_{o_{11}}} \cdot \frac{\partial net_{o_{11}}}{\partial i_{11}} \\
&= \delta_{11} \cdot h_{11}
\end{aligned} \tag{6}$$

i_{12}的偏导：

$$\begin{aligned}
\frac{\partial E}{\partial i_{12}} &= \frac{\partial E}{\partial net_{o_{11}}} \cdot \frac{\partial net_{o_{11}}}{\partial i_{12}} + \frac{\partial E}{\partial net_{o_{12}}} \cdot \frac{\partial net_{o_{12}}}{\partial i_{12}} \\
&= \delta_{11} \cdot h_{12} + \delta_{12} \cdot h_{11}
\end{aligned} \tag{7}$$

i_{13}的偏导：

$$\begin{aligned}
\frac{\partial E}{\partial i_{13}} &= \frac{\partial E}{\partial net_{o_{12}}} \cdot \frac{\partial net_{o_{12}}}{\partial i_{13}} + \frac{\partial E}{\partial net_{o_{13}}} \cdot \frac{\partial net_{o_{13}}}{\partial i_{13}} \\
&= \delta_{12} \cdot h_{12} + \delta_{13} \cdot h_{11}
\end{aligned} \tag{8}$$

i_{21}的偏导：

$$\begin{aligned}
\frac{\partial E}{\partial i_{21}} &= \frac{\partial E}{\partial net_{o_{11}}} \cdot \frac{\partial net_{o_{11}}}{\partial i_{21}} + \frac{\partial E}{\partial net_{o_{21}}} \cdot \frac{\partial net_{o_{21}}}{\partial i_{21}} \\
&= \delta_{11} \cdot h_{21} + \delta_{21} \cdot h_{11}
\end{aligned} \tag{9}$$

i_{22}的偏导：

$$\begin{aligned}
\frac{\partial E}{\partial i_{22}} &= \frac{\partial E}{\partial net_{o_{11}}} \cdot \frac{\partial net_{o_{11}}}{\partial i_{22}} + \frac{\partial E}{\partial net_{o_{12}}} \cdot \frac{\partial net_{o_{12}}}{\partial i_{22}} \\
&+ \frac{\partial E}{\partial net_{o_{21}}} \cdot \frac{\partial net_{o_{21}}}{\partial i_{22}} + \frac{\partial E}{\partial net_{o_{22}}} \cdot \frac{\partial net_{o_{22}}}{\partial i_{22}} \\
&= \delta_{11} \cdot h_{22} + \delta_{12} \cdot h_{21} + \delta_{21} \cdot h_{12} + \delta_{22} \cdot h_{11}
\end{aligned} \tag{10}$$

观察一下上面几个式子的规律，归纳一下，可以得到如下表达式：

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & \delta_{13} & 0 \\ 0 & \delta_{21} & \delta_{22} & \delta_{23} & 0 \\ 0 & \delta_{31} & \delta_{32} & \delta_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} h_{22} & h_{21} \\ h_{12} & h_{11} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial i_{11}} & \frac{\partial E}{\partial i_{12}} & \frac{\partial E}{\partial i_{13}} & \frac{\partial E}{\partial i_{14}} \\ \frac{\partial E}{\partial i_{21}} & \frac{\partial E}{\partial i_{22}} & \frac{\partial E}{\partial i_{23}} & \frac{\partial E}{\partial i_{24}} \\ \frac{\partial E}{\partial i_{31}} & \frac{\partial E}{\partial i_{32}} & \frac{\partial E}{\partial i_{33}} & \frac{\partial E}{\partial i_{34}} \\ \frac{\partial E}{\partial i_{41}} & \frac{\partial E}{\partial i_{42}} & \frac{\partial E}{\partial i_{43}} & \frac{\partial E}{\partial i_{44}} \end{bmatrix} \tag{11}$$

2. 【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理(178877)
3. 三个月教你从零入门深度学习(59965)
4. 【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络(53628)
5. 机器学习基础与实践（一）----数据清洗(52755)
6. 如何用卷积神经网络CNN识别手写数字集？(33473)
7. 机器学习基础与实践（二）----数据转换(31279)
8. 用Tensorflow让神经网络自动创造音乐(27101)
9. 【深度学习Deep Learning】资料大全(25458)
10. 【原】数据分析/数据挖掘/机器学习---- 必读书目(23380)

评论排行榜

1. 三个月教你从零入门深度学习(219)
2. 我在北京这几年（全）(161)
3. 一文弄懂神经网络中的反向传播法——BackPropagation(154)
4. 2015年总结与2016年目标和计划(125)
5. 【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理(108)
6. 2018年总结与2019年目标与计划(88)
7. 2017年总结与2018年目标和计划(72)
8. 【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络(71)
9. 坑爹的2016年总结(57)
10. 两个月刷完Leetcode前400题经验总结(33)

推荐排行榜

1. 三个月教你从零入门深度学习(238)
2. 一文弄懂神经网络中的反向传播法——BackPropagation(160)
3. 我在北京这几年（全）(97)
4. 【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理(96)
5. 【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络(51)

图中的卷积核进行了180°翻转，与这一层的误差敏感项矩阵$delta_{i,j}$)周围补零后的矩阵做卷积运算后，就可以得到$\frac{\partial E}{\partial i_{11}}$，即

$$\frac{\partial E}{\partial i_{i,j}} = \sum_m \cdot \sum_n h_{m,n}\delta_{i+m,j+n}$$

第一项求完后，我们来求第二项$\frac{\partial i_{11}}{\partial net_{i_{11}}}$

$$\begin{aligned} \because i_{11} &= out_{i_{11}} \\ &= activators(net_{i_{11}}) \\ \therefore \frac{\partial i_{11}}{\partial net_{i_{11}}} &= f'(net_{i_{11}}) \\ \therefore \delta_{11} &= \frac{\partial E}{\partial net_{i_{11}}} \\ &= \frac{\partial E}{\partial i_{11}} \cdot \frac{\partial i_{11}}{\partial net_{i_{11}}} \\ &= \sum_m \cdot \sum_n h_{m,n}\delta_{i+m,j+n} \cdot f'(net_{i_{11}}) \end{aligned} \tag{12}$$

此时我们的误差敏感矩阵就求完了，得到误差敏感矩阵后，即可求权重的梯度。

由于上面已经写出了卷积层的输入$net_{o_{11}}$与权重$h_{i,j}$之间的表达式，所以可以直接求出：

$$\begin{aligned} \frac{\partial E}{\partial h_{11}} &= \frac{\partial E}{\partial net_{o_{11}}} \cdot \frac{\partial net_{o_{11}}}{\partial h_{11}} + \dots \\ &+ \frac{\partial E}{\partial net_{o_{33}}} \cdot \frac{\partial net_{o_{33}}}{\partial h_{11}} \\ &= \delta_{11} \cdot h_{11} + \dots + \delta_{33} \cdot h_{11} \end{aligned} \tag{13}$$

推论出**权重的梯度**：

$$\frac{\partial E}{\partial h_{i,j}} = \sum_m \sum_n \delta_{m,n} out_{o_{i+m,j+n}} \tag{14}$$

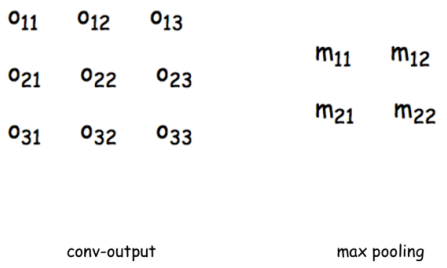
偏置项的梯度：

$$\begin{aligned} \frac{\partial E}{\partial b} &= \frac{\partial E}{\partial net_{o_{11}}} \frac{\partial net_{o_{11}}}{\partial w_b} + \frac{\partial E}{\partial net_{o_{12}}} \frac{\partial net_{o_{12}}}{\partial w_b} \\ &+ \frac{\partial E}{\partial net_{o_{21}}} \frac{\partial net_{o_{21}}}{\partial w_b} + \frac{\partial E}{\partial net_{o_{22}}} \frac{\partial net_{o_{22}}}{\partial w_b} \\ &= \delta_{11} + \delta_{12} + \delta_{21} + \delta_{22} \\ &= \sum_i \sum_j \delta_{i,j} \end{aligned} \tag{15}$$

可以看出，偏置项的偏导等于这一层所有误差敏感项之和。得到了权重和偏置项的梯度后，就可以根据梯度下降法更新权重和梯度了。

池化层的反向传播

池化层的反向传播就比较好求了，看着下面的图，左边是上一层的输出，也就是卷积层的输出feature_map，右边是池化层的输入，还是先根据前向传播，把式子都写出来，方便计算：



假设上一层这个滑动窗口的最大值是$out_{o_{11}}$

$$
\begin{aligned}
&\because net_{m_{11}} = max(out_{o_{11}}, out_{o_{12}}, out_{o_{21}}, out_{o_{22}}) \\
&\therefore \frac{\partial net_{m_{11}}}{\partial out_{o_{11}}} = 1 \\
&\frac{\partial net_{m_{11}}}{\partial out_{o_{12}}} = \frac{\partial net_{m_{11}}}{\partial out_{o_{21}}} = \frac{\partial net_{m_{11}}}{\partial out_{o_{22}}} = 0 \\
&\therefore \delta_{11}^{l-1} = \frac{\partial E}{\partial out_{o_{11}}} = \frac{\partial E}{\partial net_{m_{11}}} \cdot \frac{\partial net_{m_{11}}}{\partial out_{o_{11}}} = \delta_{11}^{l} \\
&\delta_{12}^{l-1} = \delta_{21}^{l-1} = \delta_{22}^{l-1} = 0
\end{aligned} \tag{16}
$$

这样就求出了池化层的误差敏感项矩阵。同理可以求出每个神经元的梯度并更新权重。

手写一个卷积神经网络

1.定义一个卷积层

首先我们通过ConvLayer来实现一个卷积层，定义卷积层的超参数

```
class ConvLayer(object):
    '''
    参数含义:
    input_width:输入图片尺寸—宽度
    input_height:输入图片尺寸—长度
    channel_number:通道数，彩色为3，灰色为1
    filter_width:卷积核的宽
    filter_height:卷积核的长
    filter_number:卷积核数量
    zero_padding：补零长度
    stride:步长
    activator:激活函数
    learning_rate:学习率
    '''
    def __init__(self, input_width, input_height,
                 channel_number, filter_width,
                 filter_height, filter_number,
                 zero_padding, stride, activator,
                 learning_rate):
        self.input_width = input_width
        self.input_height = input_height
        self.channel_number = channel_number
        self.filter_width = filter_width
        self.filter_height = filter_height
        self.filter_number = filter_number
        self.zero_padding = zero_padding
        self.stride = stride
        self.output_width = \
            ConvLayer.calculate_output_size(
            self.input_width, filter_width, zero_padding,
            stride)
        self.output_height = \
```

```
            ConvLayer.calculate_output_size(
            self.input_height, filter_height, zero_padding,
            stride)
        self.output_array = np.zeros((self.filter_number,
            self.output_height, self.output_width))
        self.filters = []
        for i in range(filter_number):
            self.filters.append(Filter(filter_width,
                filter_height, self.channel_number))
        self.activator = activator
        self.learning_rate = learning_rate
```

其中calculate_output_size用来计算通过卷积运算后输出的feature_map大小

```
@staticmethod
    def calculate_output_size(input_size,
            filter_size, zero_padding, stride):
        return (input_size - filter_size +
            2 * zero_padding) / stride + 1
```

2.构造一个激活函数

此处用的是RELU激活函数，因此我们在activators.py里定义，forward是前向计算，backforward是计算公式的导数：

```
class ReluActivator(object):
    def forward(self, weighted_input):
        #return weighted_input
        return max(0, weighted_input)

    def backward(self, output):
        return 1 if output > 0 else 0
```

其他常见的激活函数我们也可以放到activators里，如sigmoid函数，我们可以做如下定义：

```
class SigmoidActivator(object):
    def forward(self, weighted_input):
        return 1.0 / (1.0 + np.exp(-weighted_input))
    #the partial of sigmoid
    def backward(self, output):
        return output * (1 - output)
```

如果我们需要自动以其他的激活函数，都可以在activator.py定义一个类即可。

3.定义一个类，保存卷积层的参数和梯度

```
class Filter(object):
    def __init__(self, width, height, depth):
        #初始权重
        self.weights = np.random.uniform(-1e-4, 1e-4,
            (depth, height, width))
        #初始偏置
        self.bias = 0
        self.weights_grad = np.zeros(
            self.weights.shape)
        self.bias_grad = 0
```

```
11
12     def __repr__(self):
13         return 'filter weights:\n%s\nbias:\n%s' % (
14             repr(self.weights), repr(self.bias))
15
16     def get_weights(self):
17         return self.weights
18
19     def get_bias(self):
20         return self.bias
21
22     def update(self, learning_rate):
23         self.weights -= learning_rate * self.weights_grad
24         self.bias -= learning_rate * self.bias_grad
```

4.卷积层的前向传播

1).获取卷积区域

```
1 # 获取卷积区域
2 def get_patch(input_array, i, j, filter_width,
3                filter_height, stride):
4     '''
5     从输入数组中获取本次卷积的区域,
6     自动适配输入为2D和3D的情况
7     '''
8     start_i = i * stride
9     start_j = j * stride
10    if input_array.ndim == 2:
11        input_array_conv = input_array[
12            start_i : start_i + filter_height,
13            start_j : start_j + filter_width]
14        print "input_array_conv:",input_array_conv
15        return input_array_conv
16
17    elif input_array.ndim == 3:
18        input_array_conv = input_array[:,
19            start_i : start_i + filter_height,
20            start_j : start_j + filter_width]
21        print "input_array_conv:",input_array_conv
22        return input_array_conv
```

2).进行卷积运算

```
1 def conv(input_array,
2          kernel_array,
3          output_array,
4          stride, bias):
5     '''
6     计算卷积，自动适配输入为2D和3D的情况
7     '''
8     channel_number = input_array.ndim
9     output_width = output_array.shape[1]
10    output_height = output_array.shape[0]
11    kernel_width = kernel_array.shape[-1]
12    kernel_height = kernel_array.shape[-2]
13    for i in range(output_height):
14        for j in range(output_width):
15            output_array[i][j] = (
```

```
                get_patch(input_array, i, j, kernel_width,
                    kernel_height, stride) * kernel_array
                ).sum() + bias
```

3).增加zero_padding

```
#增加Zero padding
def padding(input_array, zp):
    '''
    为数组增加Zero padding，自动适配输入为2D和3D的情况
    '''
    if zp == 0:
        return input_array
    else:
        if input_array.ndim == 3:
            input_width = input_array.shape[2]
            input_height = input_array.shape[1]
            input_depth = input_array.shape[0]
            padded_array = np.zeros((
                input_depth,
                input_height + 2 * zp,
                input_width + 2 * zp))
            padded_array[:,
                zp : zp + input_height,
                zp : zp + input_width] = input_array
            return padded_array
        elif input_array.ndim == 2:
            input_width = input_array.shape[1]
            input_height = input_array.shape[0]
            padded_array = np.zeros((
                input_height + 2 * zp,
                input_width + 2 * zp))
            padded_array[zp : zp + input_height,
                zp : zp + input_width] = input_array
            return padded_array
```

4).进行前向传播

```
def forward(self, input_array):
        '''
        计算卷积层的输出
        输出结果保存在self.output_array
        '''
        self.input_array = input_array
        self.padded_input_array = padding(input_array,
            self.zero_padding)
        for f in range(self.filter_number):
            filter = self.filters[f]
            conv(self.padded_input_array,
                filter.get_weights(), self.output_array[f],
                self.stride, filter.get_bias())
        element_wise_op(self.output_array,
                        self.activator.forward)
```

其中element_wise_op函数是将每个组的元素对应相乘

```
# 对numpy数组进行element wise操作，将矩阵中的每个元素对应相乘
```

```
def element_wise_op(array, op):
    for i in np.nditer(array,
                       op_flags=['readwrite']):
        i[...] = op(i)
```

5.卷积层的反向传播

1).将误差传递到上一层

```
def bp_sensitivity_map(self, sensitivity_array,
                       activator):
        '''
        计算传递到上一层的sensitivity map
        sensitivity_array: 本层的sensitivity map
        activator: 上一层的激活函数
        '''
        # 处理卷积步长，对原始sensitivity map进行扩展
        expanded_array = self.expand_sensitivity_map(
            sensitivity_array)
        # full卷积，对sensitivitiy map进行zero padding
        # 虽然原始输入的zero padding单元也会获得残差
        # 但这个残差不需要继续向上传递，因此就不计算了
        expanded_width = expanded_array.shape[2]
        zp = (self.input_width +
              self.filter_width - 1 - expanded_width) / 2
        padded_array = padding(expanded_array, zp)
        # 初始化delta_array，用于保存传递到上一层的
        # sensitivity map
        self.delta_array = self.create_delta_array()
        # 对于具有多个filter的卷积层来说，最终传递到上一层的
        # sensitivity map相当于所有的filter的
        # sensitivity map之和
        for f in range(self.filter_number):
            filter = self.filters[f]
            # 将filter权重翻转180度
            flipped_weights = np.array(map(
                lambda i: np.rot90(i, 2),
                filter.get_weights()))
            # 计算与一个filter对应的delta_array
            delta_array = self.create_delta_array()
            for d in range(delta_array.shape[0]):
                conv(padded_array[f], flipped_weights[d],
                     delta_array[d], 1, 0)
            self.delta_array += delta_array
        # 将计算结果与激活函数的偏导数做element-wise乘法操作
        derivative_array = np.array(self.input_array)
        element_wise_op(derivative_array,
                        activator.backward)
        self.delta_array *= derivative_array
```

2).保存传递到上一层的sensitivity map的数组

```
def create_delta_array(self):
        return np.zeros((self.channel_number,
            self.input_height, self.input_width))
```

3).计算代码梯度

```
def bp_gradient(self, sensitivity_array):
```

```
        # 处理卷积步长，对原始sensitivity map进行扩展
        expanded_array = self.expand_sensitivity_map(
            sensitivity_array)
        for f in range(self.filter_number):
            # 计算每个权重的梯度
            filter = self.filters[f]
            for d in range(filter.weights.shape[0]):
                conv(self.padded_input_array[d],
                     expanded_array[f],
                     filter.weights_grad[d], 1, 0)
            # 计算偏置项的梯度
            filter.bias_grad = expanded_array[f].sum()
```

4).按照梯度下降法更新参数

```
def update(self):
        '''
        按照梯度下降，更新权重
        '''
        for filter in self.filters:
            filter.update(self.learning_rate)
```

6.MaxPooling层的训练

1).定义MaxPooling类

```
class MaxPoolingLayer(object):
    def __init__(self, input_width, input_height,
                 channel_number, filter_width,
                 filter_height, stride):
        self.input_width = input_width
        self.input_height = input_height
        self.channel_number = channel_number
        self.filter_width = filter_width
        self.filter_height = filter_height
        self.stride = stride
        self.output_width = (input_width -
            filter_width) / self.stride + 1
        self.output_height = (input_height -
            filter_height) / self.stride + 1
        self.output_array = np.zeros((self.channel_number,
            self.output_height, self.output_width))
```

2).前向传播计算

```
# 前向传播
    def forward(self, input_array):
        for d in range(self.channel_number):
            for i in range(self.output_height):
                for j in range(self.output_width):
                    self.output_array[d,i,j] = (
                        get_patch(input_array[d], i, j,
                            self.filter_width,
                            self.filter_height,
                            self.stride).max())
```

3).反向传播计算

```
#反向传播
    def backward(self, input_array, sensitivity_array):
        self.delta_array = np.zeros(input_array.shape)
        for d in range(self.channel_number):
            for i in range(self.output_height):
                for j in range(self.output_width):
                    patch_array = get_patch(
                        input_array[d], i, j,
                        self.filter_width,
                        self.filter_height,
                        self.stride)
                    k, l = get_max_index(patch_array)
                    self.delta_array[d,
                        i * self.stride + k,
                        j * self.stride + l] = \
                        sensitivity_array[d,i,j]
```

完整代码请见：cnn.py (https://github.com/huxiaoman7/PaddlePaddle_code/blob/master/1.mnist/cnn.py)

View Code

最后，我们用之前的4 * 4的image数据检验一下通过一次卷积神经网络进行前向传播和反向传播后的输出结果：

```
def init_test():
    a = np.array(
        [[[0,1,1,0,2],
          [2,2,2,2,1],
          [1,0,0,2,0],
          [0,1,1,0,0],
          [1,2,0,0,2]],
         [[1,0,2,2,0],
          [0,0,0,2,0],
          [1,2,1,2,1],
          [1,0,0,0,0],
          [1,2,1,1,1]],
         [[2,1,2,0,0],
          [1,0,0,1,0],
          [0,2,1,0,1],
          [0,1,2,2,2],
          [2,1,0,0,1]]])
    b = np.array(
        [[[0,1,1],
          [2,2,2],
          [1,0,0]],
         [[1,0,2],
          [0,0,0],
          [1,2,1]]])
    cl = ConvLayer(5,5,3,3,3,2,1,2,IdentityActivator(),0.001)
    cl.filters[0].weights = np.array(
        [[[-1,1,0],
```

```
            [0,1,0],
            [0,1,1]],
           [[-1,-1,0],
            [0,0,0],
            [0,-1,0]],
           [[0,0,-1],
            [0,1,0],
            [1,-1,-1]]], dtype=np.float64)
    cl.filters[0].bias=1
    cl.filters[1].weights = np.array(
        [[[1,1,-1],
          [-1,-1,1],
          [0,-1,1]],
         [[0,1,0],
          [-1,0,-1],
          [-1,1,0]],
         [[-1,0,0],
          [-1,0,1],
          [-1,0,0]]], dtype=np.float64)
    return a, b, cl
```

运行一下：

```
def test():
    a, b, cl = init_test()
    cl.forward(a)
    print "前向传播结果:", cl.output_array
    cl.backward(a, b, IdentityActivator())
    cl.update()
    print "反向传播后更新得到的filter1:",cl.filters[0]
    print "反向传播后更新得到的filter2:",cl.filters[1]

if __name__ == "__main__":
        test()
```

运行结果：

```
前向传播结果: [[[ 6.  7.  5.]
  [ 3. -1. -1.]
  [ 2. -1.  4.]]

 [[ 2. -5. -8.]
  [ 1. -4. -4.]
  [ 0. -5. -5.]]]
反向传播后更新得到的filter1: filter weights:
array([[[-1.008,  0.99 , -0.009],
        [-0.005,  0.994, -0.006],
        [-0.006,  0.995,  0.996]],

       [[-1.004, -1.001, -0.004],
        [-0.01 , -0.009, -0.012],
        [-0.002, -1.002, -0.002]],

       [[-0.002, -0.002, -1.003],
        [-0.005,  0.992, -0.005],
        [ 0.993, -1.008, -1.007]]])
bias:
0.99099999999999999
反向传播后更新得到的filter2: filter weights:
```

```
array([[[  9.98000000e-01,   9.98000000e-01,  -1.00100000e+00],
        [ -1.00400000e+00,  -1.00700000e+00,   9.97000000e-01],
        [ -4.00000000e-03,  -1.00400000e+00,   9.98000000e-01]],

       [[  0.00000000e+00,   9.99000000e-01,   0.00000000e+00],
        [ -1.00900000e+00,  -5.00000000e-03,  -1.00400000e+00],
        [ -1.00400000e+00,   1.00000000e+00,   0.00000000e+00]],

       [[ -1.00400000e+00,  -6.00000000e-03,  -5.00000000e-03],
        [ -1.00200000e+00,  -5.00000000e-03,   9.98000000e-01],
        [ -1.00200000e+00,  -1.00000000e-03,   0.00000000e+00]]])
bias:
-0.0070000000000000001
```

PaddlePaddle卷积神经网络源码解析

卷积层

在上篇文章中，我们对paddlepaddle实现卷积神经网络的的函数简单介绍了一下。在手写数字识别中，我们设计CNN的网络结构时，调用了一个函数simple_img_conv_pool(上篇文章的链接已失效，因为已经把framework--->fluid，更新速度太快了 = =)使用方式如下：

```
conv_pool_1 = paddle.networks.simple_img_conv_pool(
        input=img,
        filter_size=5,
        num_filters=20,
        num_channel=1,
        pool_size=2,
        pool_stride=2,
        act=paddle.activation.Relu())
```

这个函数把卷积层和池化层两个部分封装在一起，只用调用一个函数就可以搞定，非常方便。如果只需要单独使用卷积层，可以调用这个函数img_conv_layer,使用方式如下：

```
conv = img_conv_layer(input=data, filter_size=1, filter_size_y=1,
                              num_channels=8,
                              num_filters=16, stride=1,
                              bias_attr=False,
                              act=ReluActivation())
```

我们来看一下这个函数具体有哪些参数(注释写明了参数的含义和怎么使用)

```
def img_conv_layer(input,
                   filter_size,
                   num_filters,
                   name=None,
                   num_channels=None,
                   act=None,
                   groups=1,
                   stride=1,
                   padding=0,
                   dilation=1,
                   bias_attr=None,
```

```
                    param_attr=None,
                    shared_biases=True,
                    layer_attr=None,
                    filter_size_y=None,
                    stride_y=None,
                    padding_y=None,
                    dilation_y=None,
                    trans=False,
                    layer_type=None):
    """
    适合图像的卷积层。Paddle可以支持正方形和长方形两种图片尺寸的输入

    也可适用于图像的反卷积(Convolutional Transpose，即deconv)。
    同样可支持正方形和长方形两种尺寸输入。

    num_channel:输入图片的通道数。可以是1或者3，或者是上一层的通道数(卷积核数目 * 组的数量)
    每一个组都会处理图片的一些通道。举个例子，如果一个输入如偏的num_channel是256，设置4个group，
    32个卷积核，那么会创建32*4 = 128个卷积核来处理输入图片。通道会被分成四块，32个卷积核会先
    处理64(256/4=64)个通道。剩下的卷积核组会处理剩下的通道。

    name:层的名字。可选，自定义。
    type:basestring

    input:这个层的输入
    type:LayerOutPut

    filter_size:卷积核的x维，可以理解为width。
                如果是正方形，可以直接输入一个元祖组表示图片的尺寸
    type:int/ tuple/ list

    filter_size_y:卷积核的y维，可以理解为height。
                PaddlePaddle支持长方形的图片尺寸，所以卷积核的尺寸为(filter_size,filter_size_y)

    type:int/ None

    act: 激活函数类型。默认选Relu
    type:BaseActivation

    groups:卷积核的组数量
    type:int


    stride: 水平方向的滑动步长。或者世界输入一个元祖，代表水平数值滑动步长相同。
    type:int/ tuple/ list

    stride_y:垂直滑动步长。
    type:int

    padding: 补零的水平维度，也可以直接输入一个元祖，水平和垂直方向上补零的维度相同。
    type:int/ tuple/ list

    padding_y:垂直方向补零的维度
    type:int

    dilation:水平方向的扩展维度。同样可以输入一个元祖表示水平和初值上扩展维度相同
    :type:int/ tuple/ list

    dilation_y:垂直方向的扩展维度
    type:int

    bias_attr:偏置属性
              False：不定义bias    True：bias初始化为0
    type: ParameterAttribute/ None/ bool/ Any

    num_channel：输入图片的通道channel。如果设置为None，自动生成为上层输出的通道数
```

```
    type: int

    param_attr:卷积参数属性。设置为None表示默认属性
    param_attr:ParameterAttribute

    shared_bias:设置偏置项是否会在卷积核中共享
    type:bool

    layer_attr: Layer的 Extra Attribute
    type:ExtraLayerAttribute

    param trans:如果是convTransLayer，设置为True，如果是convlayer设置为conv
    type:bool

    layer_type:明确layer_type，默认为None。
                如果trans= True，必须是exconvt或者cudnn_convt，否则的话要么是exconv，要么是cudnn_conv
                ps:如果是默认的话，paddle会自动选择适合cpu的ExpandConvLayer和适合GPU的CudnnConvLayer
                当然，我们自己也可以明确选择哪种类型
    type:string
    return:LayerOutput object
    rtype:LayerOutput

    """


def img_conv_layer(input,
                   filter_size,
                   num_filters,
                   name=None,
                   num_channels=None,
                   act=None,
                   groups=1,
                   stride=1,
                   padding=0,
                   dilation=1,
                   bias_attr=None,
                   param_attr=None,
                   shared_biases=True,
                   layer_attr=None,
                   filter_size_y=None,
                   stride_y=None,
                   padding_y=None,
                   dilation_y=None,
                   trans=False,
                   layer_type=None):

    if num_channels is None:
        assert input.num_filters is not None
        num_channels = input.num_filters

    if filter_size_y is None:
        if isinstance(filter_size, collections.Sequence):
            assert len(filter_size) == 2
            filter_size, filter_size_y = filter_size
        else:
            filter_size_y = filter_size

    if stride_y is None:
        if isinstance(stride, collections.Sequence):
            assert len(stride) == 2
            stride, stride_y = stride
        else:
            stride_y = stride

    if padding_y is None:
        if isinstance(padding, collections.Sequence):
            assert len(padding) == 2
```

```
144             padding, padding_y = padding
145         else:
146             padding_y = padding
147 
148     if dilation_y is None:
149         if isinstance(dilation, collections.Sequence):
150             assert len(dilation) == 2
151             dilation, dilation_y = dilation
152         else:
153             dilation_y = dilation
154 
155     if param_attr.attr.get('initial_smart'):
156         # special initial for conv layers.
157         init_w = (2.0 / (filter_size**2 * num_channels))**0.5
158         param_attr.attr["initial_mean"] = 0.0
159         param_attr.attr["initial_std"] = init_w
160         param_attr.attr["initial_strategy"] = 0
161         param_attr.attr["initial_smart"] = False
162 
163     if layer_type:
164         if dilation > 1 or dilation_y > 1:
165             assert layer_type in [
166                 "cudnn_conv", "cudnn_convt", "exconv", "exconvt"
167             ]
168         if trans:
169             assert layer_type in ["exconvt", "cudnn_convt"]
170         else:
171             assert layer_type in ["exconv", "cudnn_conv"]
172         lt = layer_type
173     else:
174         lt = LayerType.CONVTRANS_LAYER if trans else LayerType.CONV_LAYER
175 
176     l = Layer(
177         name=name,
178         inputs=Input(
179             input.name,
180             conv=Conv(
181                 filter_size=filter_size,
182                 padding=padding,
183                 dilation=dilation,
184                 stride=stride,
185                 channels=num_channels,
186                 groups=groups,
187                 filter_size_y=filter_size_y,
188                 padding_y=padding_y,
189                 dilation_y=dilation_y,
190                 stride_y=stride_y),
191             **param_attr.attr),
192         active_type=act.name,
193         num_filters=num_filters,
194         bias=ParamAttr.to_bias(bias_attr),
195         shared_biases=shared_biases,
196         type=lt,
197         **ExtraLayerAttribute.to_kwargs(layer_attr))
198     return LayerOutput(
199         name,
200         lt,
201         parents=[input],
202         activation=act,
203         num_filters=num_filters,
204         size=l.config.size)
```

我们了解这些参数的含义后，对比我们之前自己手写的CNN，可以看出paddlepaddle有几个优点：

- 支持长方形和正方形的图片尺寸

- 支持滑动步长stride、补零zero_padding、扩展dilation在水平和垂直方向上设置不同的值
- 支持偏置项卷积核中能够共享
- 自动适配cpu和gpu的卷积网络

在我们自己写的CNN中，只支持正方形的图片长度，如果是长方形会报错。滑动步长，补零的维度等也只支持水平和垂直方向上的维度相同。了解卷积层的参数含义后，我们来看一下底层的源码是如何实现的：ConvBaseLayer.py 有兴趣的同学可以在这个链接下看看底层是如何用C++写的ConvLayer

池化层同理，可以按照之前的思路分析，有兴趣的可以一直顺延看到底层的实现，下次有机会再详细分析。(占坑明天补一下tensorflow的源码实现)

总结

本文主要讲解了卷积神经网络中反向传播的一些技巧，包括卷积层和池化层的反向传播与传统的反向传播的区别，并实现了一个完整的CNN，后续大家可以自己修改一些代码，譬如当水平滑动长度与垂直滑动长度不同时需要怎么调整等等，最后研究了一下paddlepaddle中CNN中的卷积层的实现过程，对比自己写的CNN，总结了4个优点，底层是C++实现的，有兴趣的可以自己再去深入研究。写的比较粗糙，如果有问题欢迎留言：)

参考文章：

1.https://www.cnblogs.com/pinard/p/6494810.html

2.https://www.zybuluo.com/hanbingtao/note/476663

分类: 深度学习

标签: 深度学习

« 上一篇： 【深度学习系列】数据预处理
» 下一篇： 【深度学习系列】用Tensorflow进行图像分类

posted @ 2017-11-22 17:20 Charlotte77 阅读(53629) 评论(71) 编辑 收藏

< Prev 1 2

评论

#51楼 2018-11-15 14:42 | shellcom

如果用cnn来识别0-9数字，反向传播来修正卷积核，那cnn在全连接层，怎么知道输出是第一个数字是0呢，也就是∂E怎么算。

支持(0) 反对(0)

#52楼 2018-11-22 10:56 | rxguo

想测试一下你的代码，结果出现：
TypeError: 'float' object cannot be interpreted as an integer
还有：
Traceback (most recent call last):
File "C:/Users/Dell/PycharmProjects/HelloWorld/NetworkPractice.py", line 502, in <module>
test()
File "C:/Users/Dell/PycharmProjects/HelloWorld/NetworkPractice.py", line 408, in test
a, b, cl = init_test()
File "C:/Users/Dell/PycharmProjects/HelloWorld/NetworkPractice.py", line 382, in init_test
cl = ConvLayer(5, 5, 3, 3, 3, 2, 1, 2, IdentityActivator(), 0.001)
File "C:/Users/Dell/PycharmProjects/HelloWorld/NetworkPractice.py", line 184, in __init__
self.output_height, self.output_width))

是什么原因呢？另，感觉您提供的DEMO都不能复现呀？

支持(0) 反对(0)

#53楼 [楼主] 2018-11-22 17:57 | Charlotte77

@ rxguo
我的代码都是测试过的才放上来的，跑不通不代表都是我的问题，先自己改一下吧。
我的代码都不能复现？？具体提供一下吧

支持(0) 反对(0)

#54楼 2018-11-22 18:13 | 会长

@ Charlotte77
也许是python版本的问题

支持(0) 反对(0)

#55楼 2018-12-10 13:27 | clemente

@ Charlotte77
请问 你的公式是怎么写的？？？这么整齐

支持(0) 反对(0)

#56楼 [楼主] 2018-12-11 09:16 | Charlotte77

@ clemente
用latex，搜一下对齐怎么写

支持(0) 反对(0)

#57楼 2018-12-15 16:13 | 小男孩的Code

大神，收藏了有时间自己跑跑代码

支持(0) 反对(0)

#58楼 2018-12-16 20:33 | wz0919

楼主你好，有一个地方不太理解：
反向传播计算sensitivity_map时，filter旋转180度到底是按行和列旋转还是层和行旋转呢？
我看一般例子里的filter都是二维的，这时确实是行和列旋转。
但我看你的代码bp_sensitivity_map这个函数里面，旋转filter的操作是

```
flipped_weights = np.array(map(
lambda i: np.rot90(i, 2),
filter.get_weights()))
```

对于三维filter，np.rot90(i, 2)就按层和行旋转了，好像不是很直观？希望能解释下。谢谢！

（直观感觉：对于三维的filter，filter的第k层和input_array的第k层一一对应，所以对input_array的第k层求偏导时，只需要考虑filter的第k层，这时就相当于二维了，直观上就是按行和列旋转。）

支持(2) 反对(0)

#59楼 [楼主] 2019-01-03 18:36 | Charlotte77

@ huxiaoman777
你这个id看得我吓了一跳，还以为是我自己呢 = =!
op是一种运算符，这里定义为内积

支持(1) 反对(0)

#60楼 2019-02-26 13:10 | wupeng1131

博主，我有几个小问题啊。1.卷积核翻转和不翻转，都是卷积操作，那什么时候需要翻转，什么时候不需要？
2.我看到博主在写神经元输出公式（4）的时候，是不是把bias漏了？
3.对博主的公式（13）我有疑问，∂net/∂h怎么会得到和h相关的乘积呢，应该是上一层的输出吧？
4.我对于卷积的反向传播有一个较简单的思路总结：（设L为层数）（1）delta^{L} 的求解就是 delta^{L+1} (后一层的 delta) 和权重 W 的卷积
（2）dW 的求解就是 delta^{L+1} 和前一层的输入 x^{L} 的卷积
，这个思路和博主的公式，本质上是一样的，不过，加上这个总结，会更加利于理解，博主以为若何？

支持(0) 反对(0)

#61楼 2019-04-04 14:16 | weifengm

@ rxguo
同样的问题，你解决了吗？不知道哪里默认是float格式所以报错 'float' object cannot be interpreted as an integer

支持(0) 反对(0)

#62楼 2019-05-08 10:42 | 没事想想达尔文

如果做数据回归的话，是不是每卷积一次就要跟着一个标准化的层啊？

支持(0) 反对(0)

#63楼 2019-07-25 14:45 | 风中de石头

@ weifengm
是因为python 2.x和python 3.x有些地方不兼容导致的，除了在报错的地方将float强制转换成int外（或者将里面的除号/改成双斜杠//,这是根原因），还要更改里面的一个map()函数将其强制转换成list,因为在python 3.x map()返回的是一个迭代器

支持(1) 反对(0)

#64楼 2019-08-07 09:10 | CG大魔王

文章里面反向传播推算过程中说Outi11=Neto11这里我觉得是有问题的，Outi11是上一层的输出，没错，Neto11是下一层的输入也没错，但是Outi11到Neto11是经过一次卷积的，可能是我的理解有问题，请大佬解惑，话说后面的m和n表示啥？

支持(0) 反对(0)

#65楼 2019-08-07 14:22 | CG大魔王

@ Andre_Ma
这个貌似是根据导数的正负来的，所以加减都可以

支持(0) 反对(0)

#66楼 2019-08-07 15:08 | CG大魔王

@ Charlotte77
可是最后还是h11啊

支持(0) 反对(0)

#67楼 2019-08-07 15:09 | CG大魔王

@ zongcm
可以有多个卷积层

支持(0) 反对(0)

#68楼 2020-01-01 16:12 | 原味辣鸡

请问可以引用一下吗？我会标注原作者的。

支持(0) 反对(0)

#69楼 2020-04-16 14:38 | gii_walter

全是公式看得好懵

支持(0) 反对(0)

#70楼 2020-05-28 21:23 | 小艾1

我没看出来有偏置项了吖？

支持(0) 反对(0)

#71楼 2020-07-29 18:14 | 3079779149

@风中de石头
谢谢，已经能够运行啦

支持(0) 反对(0)

< Prev 1 2

刷新评论 刷新页面 返回顶部