5.一维卷积

5.1函数

- 一维卷积不代表卷积核只有一维,也不代表被卷积的feature是一维的。
- 一维的意思是,卷积的方向是一维的。

```
1 torch.nn.Conv1d(in_channels, out_channels, kernel_size,
 2
                 stride=1, padding=0, dilation=1, groups=1, bias=True)
 3
  in_channels(int) – 输入信号的通道。在文本分类中,即为词向量的维度
 5 out_channels(int) - 卷积产生的通道。卷积核的个数。
6
  kernel_size(int or tuple) - 卷积核的宽度,长度由in_channels来决定的
7
                           卷积核的大小【in_channels, kernel_size】
9
  stride(int or tuple, optional) - 卷积步长
  padding (int or tuple, optional) - 输入的每一条边补充0的层数
11
12
13 bias(bool, optional) - 如果bias=True,添加偏置
14
```

5.2 实例

```
1 import torch
2 import torch.nn as nn
3
4 # inchannels = 4, out_channels = 2,kernel_size = 3
5 # 卷积核的大小 (inchannels * kernel_size) (4,3)
6 # 输出的维度是2: 卷积核的个数 是 2.
7 m = nn.Convld(4, 2, 3, stride = 2)
8
9 # 第一个参数理解为batch的大小,输入是4 * 9格式
10 input = torch.randn(1, 4, 9)
11 output = m(input)
12
13 print(output.size()) # (1,2,4)
```

理解输入:输入是一个三通道的矩阵【N,X,Y】。

N: 样本的数量, 那么每一个样本的特征就是一个二维的矩阵。

X,Y: 就是这个样本的特征矩阵。

那么卷积核就是直接对【X,Y】矩阵进行**卷积**操作,得到一个结果。

二维卷积的滑动窗口,向右滑动,向下滑动。**一维卷积的滑动窗口就是一个方向滑动,那就是向** 右/ **向下**。

原始的输入的矩阵大小为: 【4,9】

那么卷积核的大小是: 【4,9】,每隔两个步骤卷积一次,【4*3】卷4次即可。

第一个卷积核进行如下操作:

	-0.2105, -1.0958,	0.7299,	ୀ.1003,	2.3175	o 0.8186,	-1.7510	ে -0.1925,	0.8591	્
	1.0991, -0.3016,	1.5633,	0.6162,	0.3150	1.0413,	1.0571	-0.7014,	0.2239	ı
	-0.0658, 0.4755,	-0.6653,	-0.0696,	0.3483	-0.0360,	-0.4665	1.2606,	1.3365	i
sci	-0.0186, -1.1802,	-0.8835.	-1.1813.	0.5145	-0.0534,	-1 2568	A 3211	-2 4793	

得到输出1*4的输出:

[-0.8012, 0.0589, 0.1576, -0.8222]

第二个卷积核进行类似操作:

```
-0.2105, -1.0958, 0.7299, 1.1003,
                                    2.3175 0.8186, -1.7510,
                                                               -0.1925,
 1.0991, -0.3016,
                                             1.0413,
                  1.5633, 0.6162,
                                    0.3150
                                                      1.0571
                                                              -0.7014, 0.2239
-0.0658, 0.4755, -0.6653, -0.0696,
                                    0.3483
                                            -0.0360,
                                                     -0.4665
                                                               1.2606, 1.3365
-0.0186, -1.1802, -0.8835, -1.1813
                                    0.5145
                                            -0.0534,
```

得到输出1*4的输出:

[-0.8231, -0.4233, 0.7178, -0.6621]

合并得到最后的2*4的结果:

```
tensor([[[-0.8012, 0.0589, 0.1576, -0.8222], [-0.8231, -0.4233, 0.7178, -0.6621]]], grad_fn=<SqueezeBackward1>)
```

输入的input为 4 * 9 , 输出为 2 * 4

5.3完整分类实例

```
1 class CNN(nn.Module):
 2 def __init__(self, B):
           super(CNN, self).__init__()
           self.B = B
 4
           self.relu = nn.ReLU(inplace=True)
           self.conv1 = nn.Sequential(
 6
               nn.Conv1d(in_channels=15, out_channels=64, kernel_size=2), # 24 - 2
 7
 4019 + 1 = 23
 8
               nn.ReLU(),
               nn.MaxPool1d(kernel_size=2, stride=1), # 23 - 2 + 1 = 22
 9
10
           self.conv2 = nn.Sequential(
11
               nn.Conv1d(in_channels=64, out_channels=128, kernel_size=2), # 22 - 2
12
   + 1 = 21
               nn.ReLU(),
13
14
            nn.MaxPoolld(kernel_size=\frac{2}{2}, stride=\frac{1}{2}), # 21 - 2 + 1 = 20
15
           self.Linear1 = nn.Linear(self.B * 128 * 20, self.B * 50)
16
           self.Linear2 = nn.Linear(self.B * 50, self.B)
17
18
19
       def forward(self, x):
          x = self.conv1(x)
20
           x = self.conv2(x)
21
22
           # print(x.size()) # 15 127 20
           x = x.view(-1)
23
           # print(x.size())
24
           x = self.Linear1(x)
25
           x = self.relu(x)
26
27
           x = self.Linear2(x)
28
           x = x.view(x.shape[0], -1)
29
30
           return x
31
```