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。

```
torch.nn.Conv1d(in_channels, out_channels, kernel_size,
                stride=1, padding=0, dilation=1, groups=1, bias=True)

in_channels(int) - 输入信号的通道。在文本分类中，即为词向量的维度
out_channels(int) - 卷积产生的通道。卷积核的个数。

kernel_size(int or tuple) - 卷积核的宽度，长度由in_channels来决定的
                            卷积核的大小【in_channels，kernel_size】

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

bias(bool, optional) - 如果bias=True，添加偏置
```

5.2 实例

```
import torch
import torch.nn as nn

# inchannels = 4, out_channels = 2,kernel_size = 3
# 卷积核的大小 (inchannels * kernel_size)  (4,3)
# 输出的维度是2: 卷积核的个数 是 2.
m = nn.Conv1d(4, 2, 3, stride = 2)

# 第一个参数理解为batch的大小，输入是4 * 9格式
input = torch.randn(1, 4, 9)
output = m(input)

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。

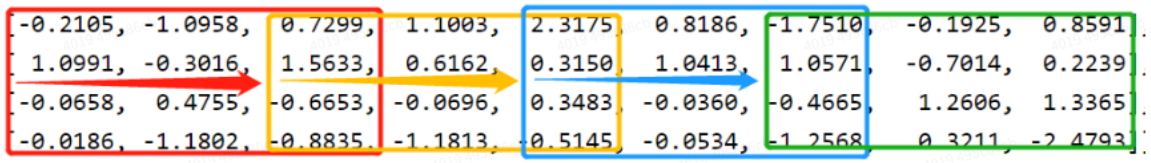
那么卷积核就是直接对【X,Y】矩阵进行**卷积**操作，得到一个结果。

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。

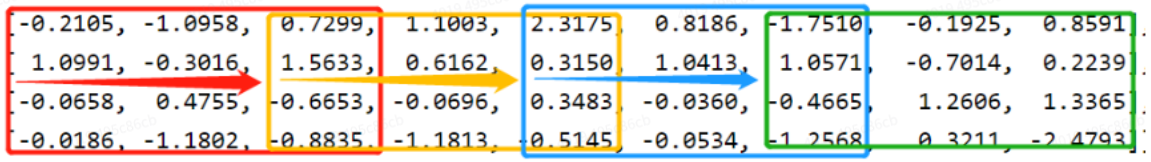
第一个卷积核进行如下操作：



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：



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

```python
class CNN(nn.Module):
    def __init__(self, B):
        super(CNN, self).__init__()
        self.B = B
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = nn.Sequential(
            nn.Conv1d(in_channels=15, out_channels=64, kernel_size=2),  # 24 - 2 + 1 = 23
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2, stride=1),  # 23 - 2 + 1 = 22
        )
        self.conv2 = nn.Sequential(
            nn.Conv1d(in_channels=64, out_channels=128, kernel_size=2),  # 22 - 2 + 1 = 21
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2, stride=1),  # 21 - 2 + 1 = 20
        )
        self.Linear1 = nn.Linear(self.B * 128 * 20, self.B * 50)
        self.Linear2 = nn.Linear(self.B * 50, self.B)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # print(x.size())  # 15 127 20
        x = x.view(-1)
        # print(x.size())
        x = self.Linear1(x)
        x = self.relu(x)
        x = self.Linear2(x)
        x = x.view(x.shape[0], -1)

        return x

```