

SDM5013 Assignment 2

(一) Question

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

INPUT

1	0	-1
1	0	-1
1	0	-1

Kernel 1

1	0
0	1

Kernel 2

0	0
0	0

Reference

计算由INPUT经过Kernel 1与Kernel 2卷积层后得到的结果:

$$Result = INPUT * Kernel 1 * Kernel 2.$$

并且定义Loss为:

$$Loss = ||Result - Reference||_2^2.$$

通过反向传播求得过程中的梯度。

Answer:

1.1 The result after two kernels

(1) After the first kernel:

Size of result is $(5 - 3 + 1) \times (5 - 3 + 1) = 3 \times 3$

$$1 * 1 + 1 * 0 + 1 * (-1) = 0$$

$$1 * 1 + 1 * 0 = 1$$

$$1 * 1 = 1$$

...

$$1 * 1 + 1 * 0 + 1 * (-1) = 0$$

So,the first result1 is

0	1	1
-1	0	1
-1	-1	0

(2) After the second kernel:

Size of result is $(3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$

$$1 * 0 + 1 * 0 = 0$$

$$1 * 1 + 1 * 1 = 2$$

$$-1 * 1 + (-1 * 1) = -2$$

$$1 * 0 + 1 * 0 = 0$$

So,the second result2 is

0	2
-2	0

(3) Code

```
1. import torch
2. from torch import nn
3. from d2l import torch as d2l
4. # 定义卷积的函数
5. def corr2d(X, K):
6.     """Compute 2D cross-correlation"""
7.     h, w = K.shape
8.     Y = torch.zeros((X.shape[0] - h + 1, X.shape[1] - w + 1))
9.     for i in range(Y.shape[0]):
10.         for j in range(Y.shape[1]):
11.             Y[i, j] = (X[i:i+h, j:j+w] * K).sum()
12.     return Y
13.
14. # 定义输入矩阵和两个卷积核
15. input_matrix = torch.tensor([[1,0,0,0,0],
16.                               [0,1,0,0,0],
17.                               [0,0,1,0,0],
18.                               [0,0,0,1,0],
19.                               [0,0,0,0,1]], dtype=torch.float32, requires_grad=True)
20. kernel1 = torch.tensor([[1, 0, -1],
21.                          [1, 0, -1],
22.                          [1, 0, -1]], dtype=torch.float32, requires_grad=True)
23. kernel2 = torch.tensor([[1, 0],
24.                           [0, 1]], dtype=torch.float32, requires_grad=True)
25.
26. result1 = corr2d(input_matrix, kernel1)
27. result2 = corr2d(result1, kernel2)
28.
29. # 定义参考结果
30. reference = torch.tensor([[0, 0],
31.                             [0, 0]], dtype=torch.float32)
32.
33. # 计算 L2 范数损失
34. loss = (result2 - reference).pow(2).sum()
35.
36. # 反向传播求梯度
37. loss.backward()
38.
39. print("First result1:\n", result1)
40. print("Second result2:\n", result2)
41. print("loss:", loss.item())
```

```

D:\anaconda\Anaconda3\python.exe D:\pythonProject\main.py
First result1:
tensor([[ 0.,  1.,  1.],
        [-1.,  0.,  1.],
        [-1., -1.,  0.]]) grad_fn=<CopySlices>
Second result2:
tensor([[ 0.,  2.],
        [-2.,  0.]]) grad_fn=<CopySlices>
loss: 8.0

Process finished with exit code 0

```

图 1 Screenshot of the results

1.2 Calculation of gradients

(1) Formula

$$Result1 = INPUT * Kernel1$$

$$Result2 = Result1 * Kernel2$$

$$loss = ||Result2 - Reference||_2^2$$

$$\frac{\partial loss}{\partial Kernel} = Convolution(Input\ X, Loss\ gradient\ \frac{\partial Loss}{\partial O})$$

$$\frac{\partial loss}{\partial X} = Full\ Convolution(180^\circ\ rotated\ Filter\ Kernel, Loss\ gradient\ \frac{\partial Loss}{\partial O})$$

Note: CNN_Backprop_Recitation_5_F21.pdf & process.pdf for derivation and process

(2) Backpropagation

Loss for the gradient of Kernel2(2 × 2)

$$\frac{\partial loss}{\partial Kernel2} = Convolution(Result1, Loss\ gradient\ \frac{\partial Loss}{\partial O})$$

8	4
4	8

Loss for the gradient of Kernel1(3 × 3)

0	-8	0
8	0	-8
0	8	0

(3) Code

```

1. import torch
2. from torch import nn

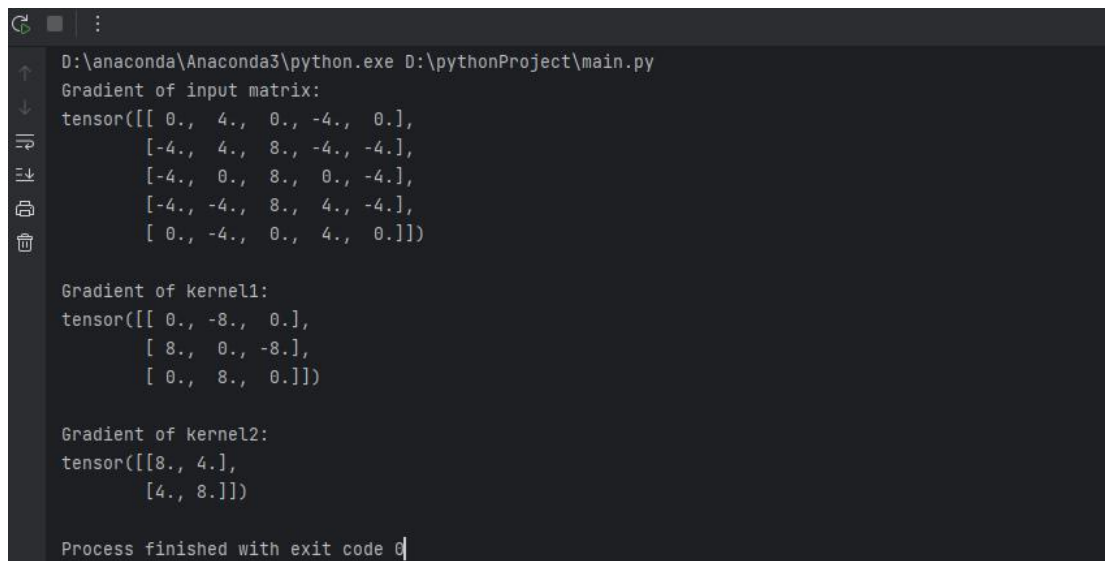
```

```

3. from d2l import torch as d2l
4.
5. # 定义卷积的函数
6. def corr2d(X, K):
7.     """Compute 2D cross-correlation"""
8.     h, w = K.shape
9.     Y = torch.zeros((X.shape[0] - h + 1, X.shape[1] - w + 1))
10.    for i in range(Y.shape[0]):
11.        for j in range(Y.shape[1]):
12.            Y[i, j] = (X[i:i+h, j:j+w] * K).sum()
13.    return Y
14.
15. # 定义输入矩阵和两个卷积核
16. input_matrix = torch.tensor([[1,0,0,0,0],
17.                               [0,1,0,0,0],
18.                               [0,0,1,0,0],
19.                               [0,0,0,1,0],
20.                               [0,0,0,0,1]], dtype=torch.float32, requires_grad=True)
21. kernel1 = torch.tensor([[1, 0, -1],
22.                          [1, 0, -1],
23.                          [1, 0, -1]], dtype=torch.float32, requires_grad=True)
24. kernel2 = torch.tensor([[1, 0],
25.                          [0, 1]], dtype=torch.float32, requires_grad=True)
26.
27. result1 = corr2d(input_matrix, kernel1)
28. result2 = corr2d(result1, kernel2)
29.
30. # 定义参考结果
31. reference = torch.tensor([[0, 0],
32.                            [0, 0]], dtype=torch.float32)
33.
34. # 计算 L2 范数损失
35. loss = (result2 - reference).pow(2).sum()
36.
37. # 反向传播求梯度
38. loss.backward()
39.
40. # 输出梯度
41. print("Gradient of input matrix:")
42. print(input_matrix.grad)
43. print("\nGradient of kernel1:")
44. print(kernel1.grad)

```

```
45. print("\nGradient of kernel2:")
46. print(kernel2.grad)
```



```
D:\anaconda\Anaconda3\python.exe D:\pythonProject\main.py
Gradient of input matrix:
tensor([[ 0.,  4.,  0., -4.,  0.],
        [-4.,  4.,  8., -4., -4.],
        [-4.,  0.,  8.,  0., -4.],
        [-4., -4.,  8.,  4., -4.],
        [ 0., -4.,  0.,  4.,  0.]])

Gradient of kernel1:
tensor([[ 0., -8.,  0.],
        [ 8.,  0., -8.],
        [ 0.,  8.,  0.]])

Gradient of kernel2:
tensor([[8., 4.],
        [4., 8.]])

Process finished with exit code 0
```

图 2 Screenshot of the results