인공지능프로그래밍

Lab1

NumPy_for_Arrays_Test_dist

2024/09/14

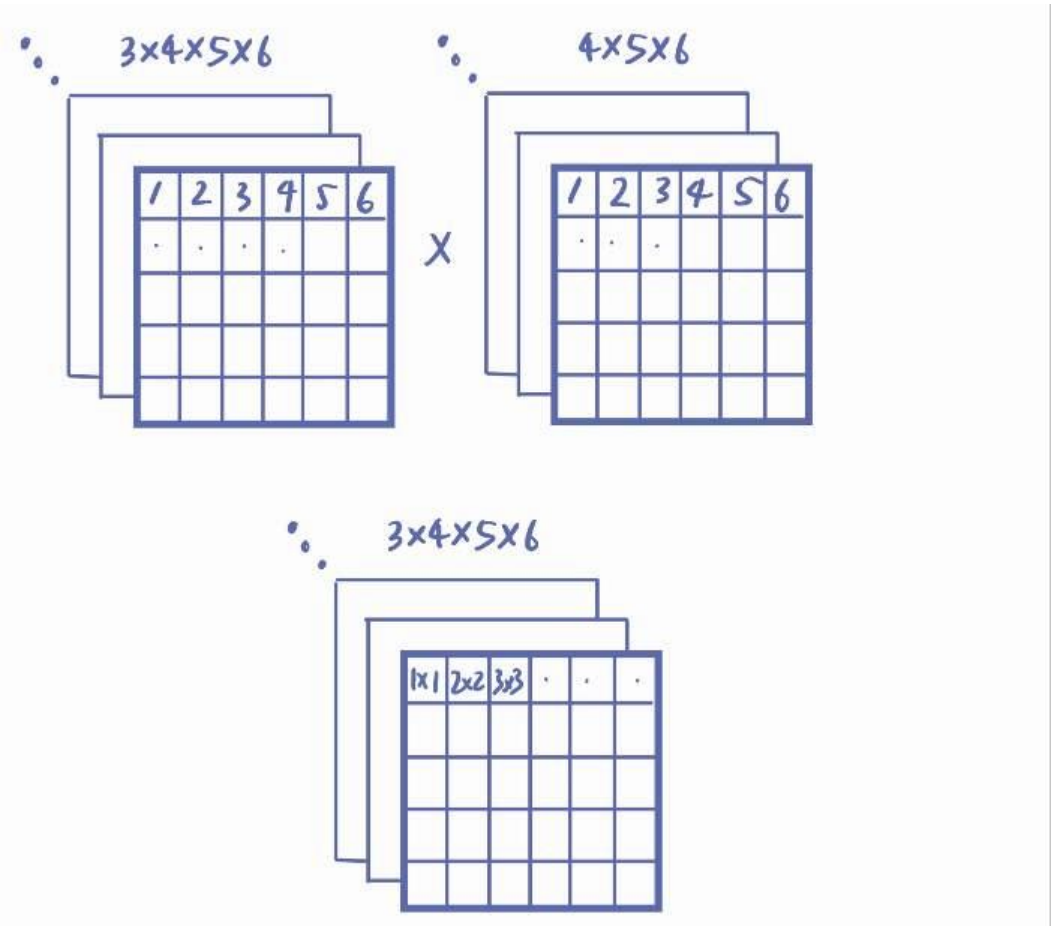
2019202050 이강현

Lab Objective

제공된 numpy 라이브러리 document를 읽고 제시된 여러 문제들을 해결해보면서 다양한 행렬 연산을 python에서 다루는 법을 익힌다. 실제 행렬연산이 요소별로 어떻게 이루어지는지, numpy에서 행렬을 어떻게 다루는지, 그러한 연산들을 진행하는데 있어서 사용가능한 함수들을 어떻게 사용하는지를 배우고 활용할 수 있는 능력을 기를 수 있다.

Whole simulation program flow and Result

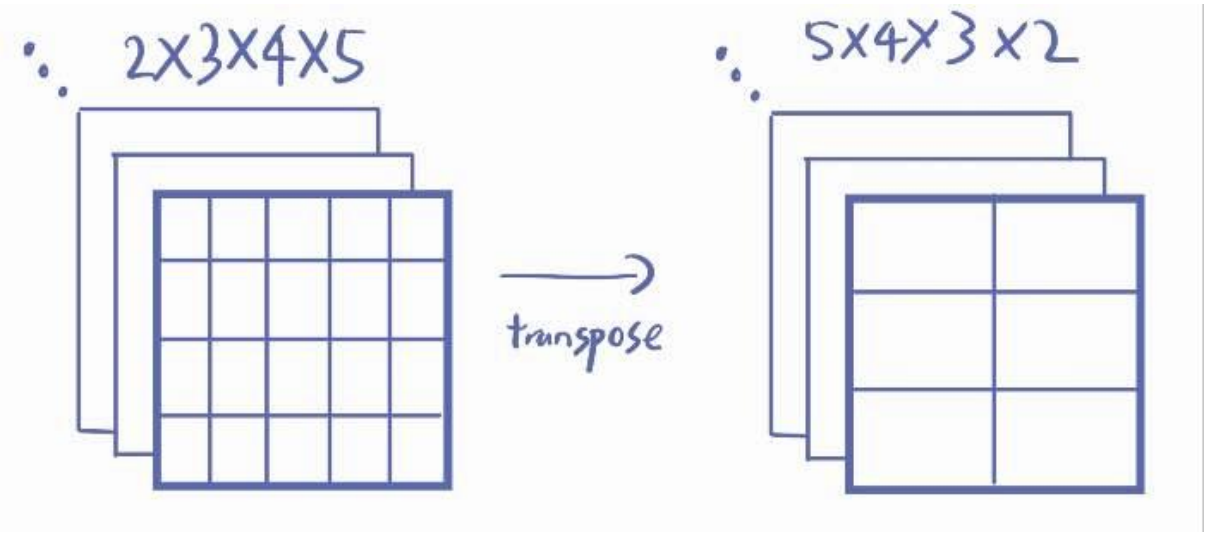
1.



1번 문제는 3x4x5x6과 4x5x6 행렬끼리 곱셈을 진행하는 문제이다. 요소별 곱셈을 진행한다. 대응되는 numpy 함수로는 multiply함수가 있다.

```
Resulting shape:  (3, 4, 5, 6)
All tests passed.
```

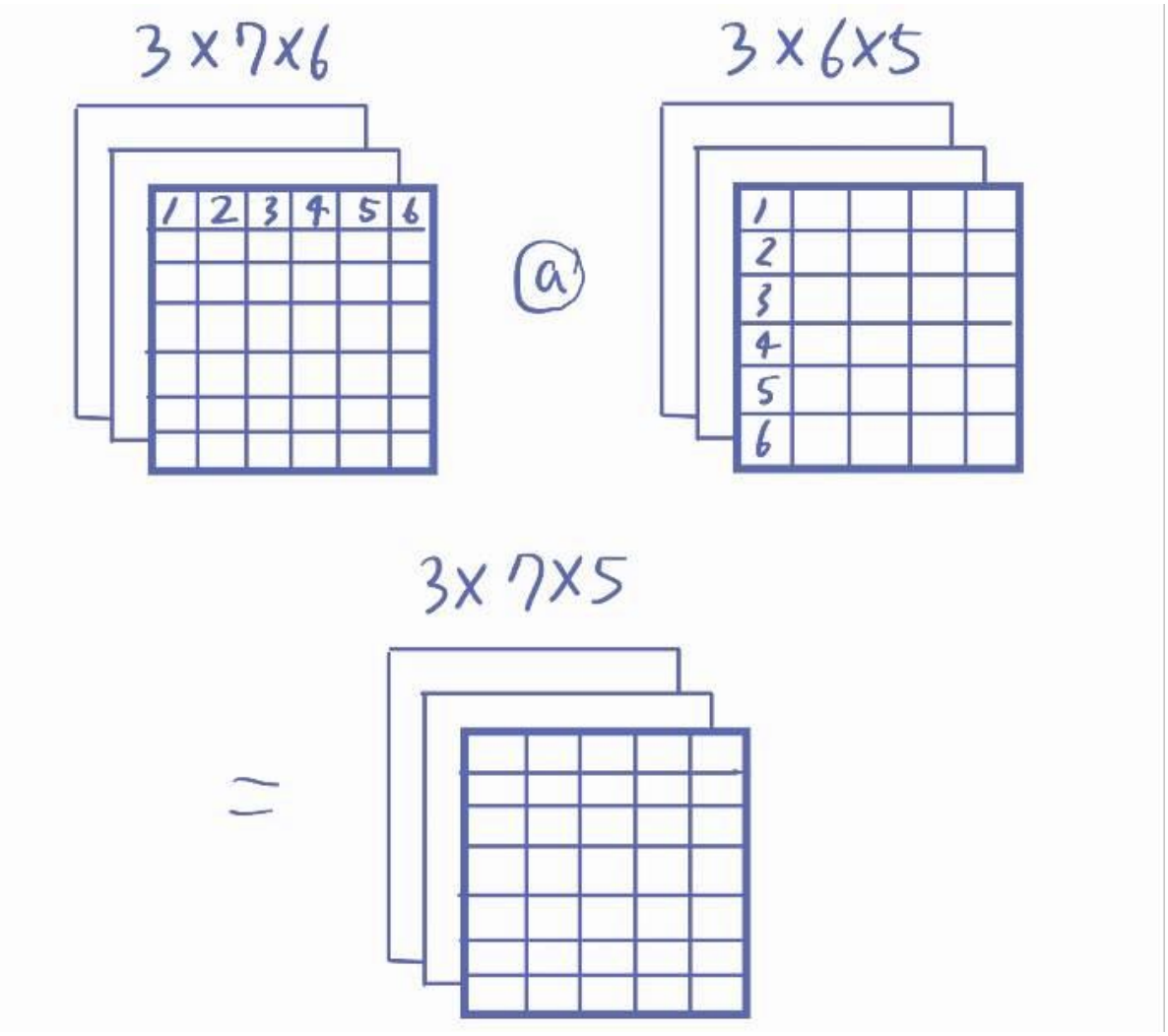
2.



2번 문제는 2x3x4x5 행렬을 transpose하는 문제이다. 기본적으로 transpose를 행 행렬 적용할 때 axis를 설정하지 않으면 shape이 뒤집어져서 나오는 것을 확인할 수 있다. 대응되는 numpy 함수에는 transpose 함수를 사용한다.

```
Resulting shape:  (5, 4, 3, 2)
All tests passed.
```

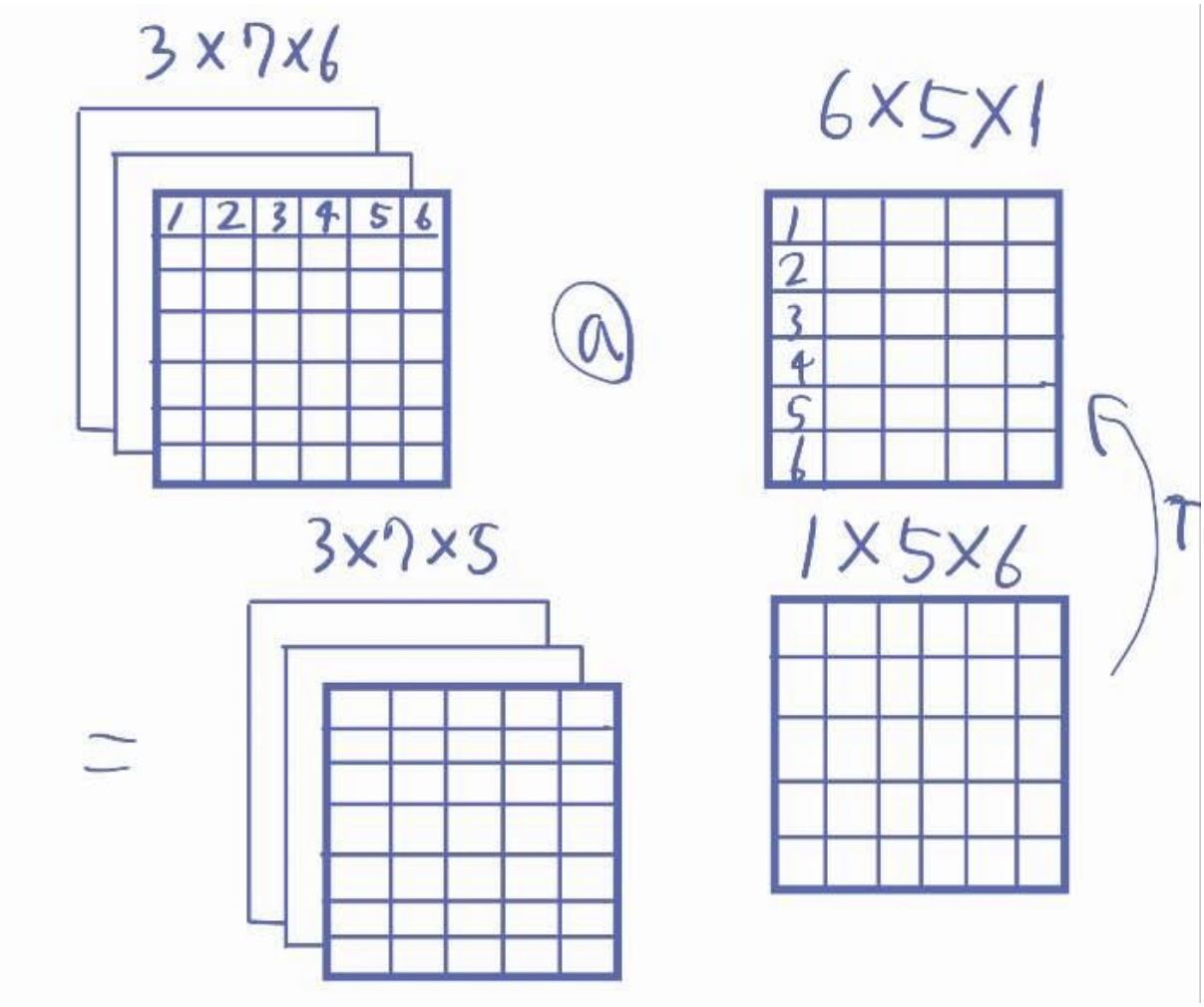
3.



3번 문제는 3x7x6 행렬과 3x6x5 행렬을 행렬곱하는 문제이다. 대응되는 함수로는 matmul함수가 있고 연산이 진행되기 위해서는 첫번째 행렬의 마지막 축과 두번째 행렬의 뒤에서 두번째 축의 크기가 서로 같아야한다. 또한 첫번째 축의 크기가 둘 다 3으로 동일하기에 앞의 연산이 짝지어 진행될 수 있다. 그림의 7 x 6 과 6 x 5가 서로 곱셈을 진행하여 7 x 5가 되고 이러한 결과가 3장 나오는 과정이다. 따라서 크기는 3 x 7 x 5가 된다.

```
Resulting shape:  (3, 7, 5)
All tests passed.
```

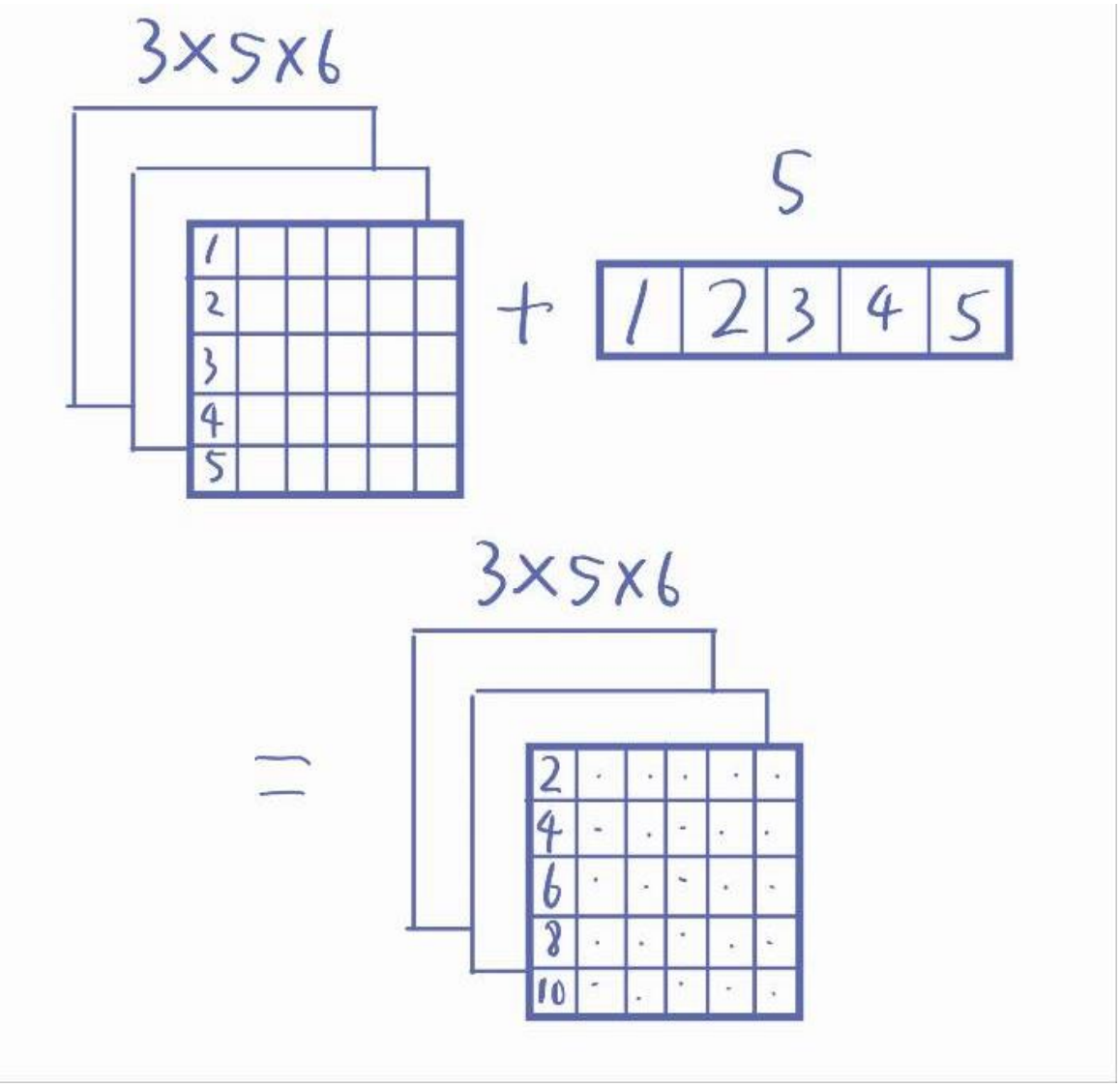
4.



4번 문제는 3 x 7 x 6 행렬과 1 x 5 x 6 행렬을 서로 행렬곱 하는 문제이다. 두 행렬을 연산하기 위해 1 x 5 x 6 행렬을 transpose하고 연산을 진행한다. 따라서 3 x 7 x 5의 shape이 나오게 된다. 대응되는 함수로는 matmul이 있는데 두번째 행렬의 마지막 축 크기가 1이므로 @ 연산자와 array indexing을 이용하여 그림과 같이 연산할 수 있으나 matmul함수를 이용할 경우 shape이 맞지않아 연산이 되지 않는다. 따라서 1 x 6 x 5로 크기를 맞추어서 transpose하고 연산을 진행하면 올바르게 작동한다.

```
Resulting shape:  (3, 7, 5)
All tests passed.
```

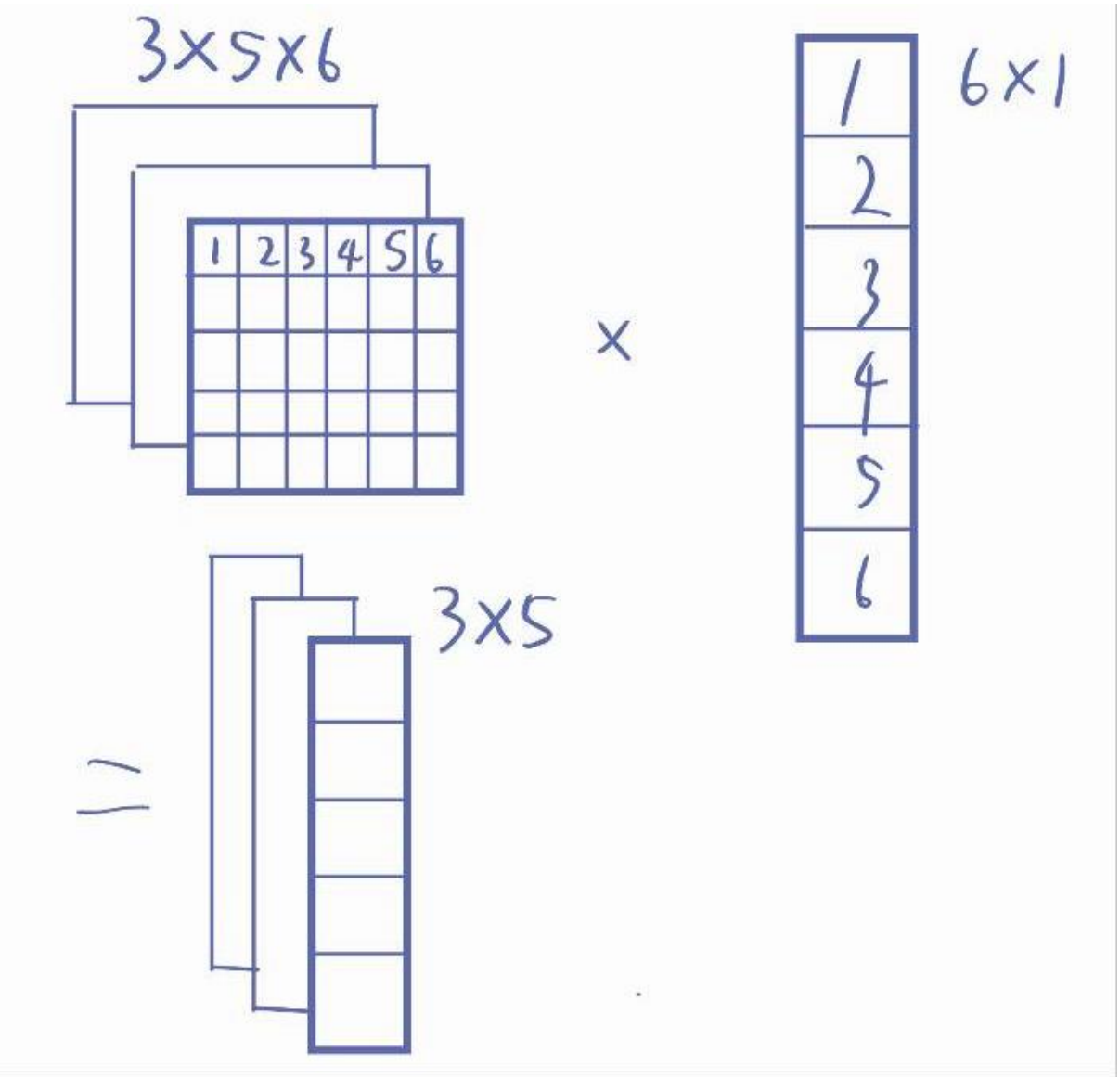
5.



5번 문제는 3 x 7 x 6 행렬과 5 행렬을 서로 더하는 문제이다. 행렬의 shape이 달라 자동적으로 broadcasting 되지 않기 때문에 shape을 맞춰주어야 한다. 따라서 인덱스로 접근하여 요소별로 더해주거나 reshape을 통해 1 x 5 x 1로 두번째 행렬을 바꾸고 더해주어 broadcasting이 가능하게 하는 두 가지 방법 모두 가능하다.

```
Resulting shape:  (3, 5, 6)
All tests passed.
```

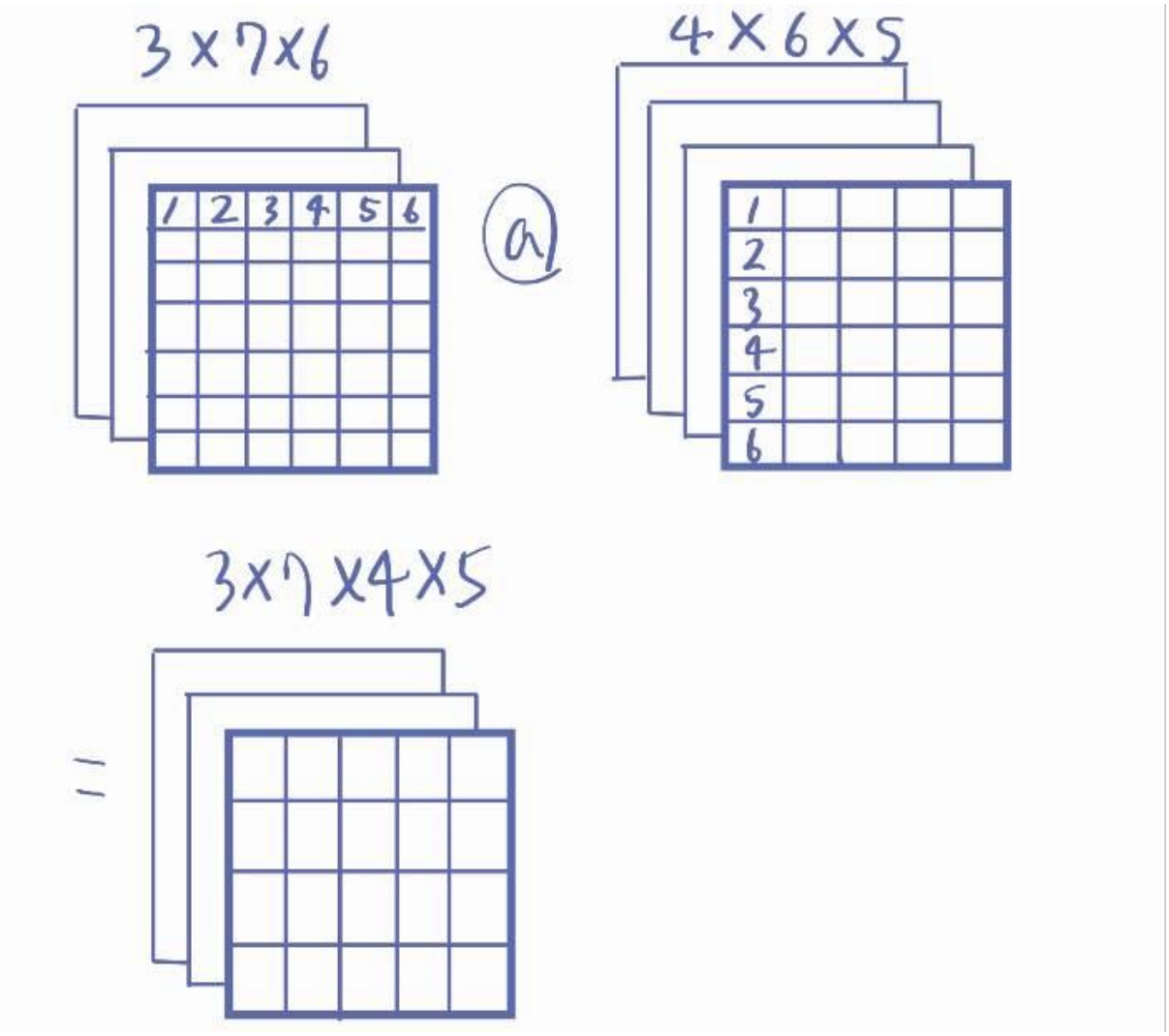
6.



6번 문제는 3 x 5 x 6 행렬과 6 x 1 행렬을 서로 곱하는 문제이다. Matrix와 vector의 곱으로 수학적으로는 3 x 5 가 나와야하나 일반적으로 numpy의 matmul을 적용하면 축이 제거되지 않아 3 x 5 x 1의 shape을 가지게 된다. 이때 요소별로 인덱스로 접근하여 곱한 후 제거할 축을 기점으로 덧셈을 진행하거나 squeeze를 이용하여 크기가 1인 축을 제거하고 matmul함수를 통해 연산을 진행하면 3 x 5의 행렬을 얻을 수 있다.

```
Resulting shape:  (3, 5)
All tests passed.
```

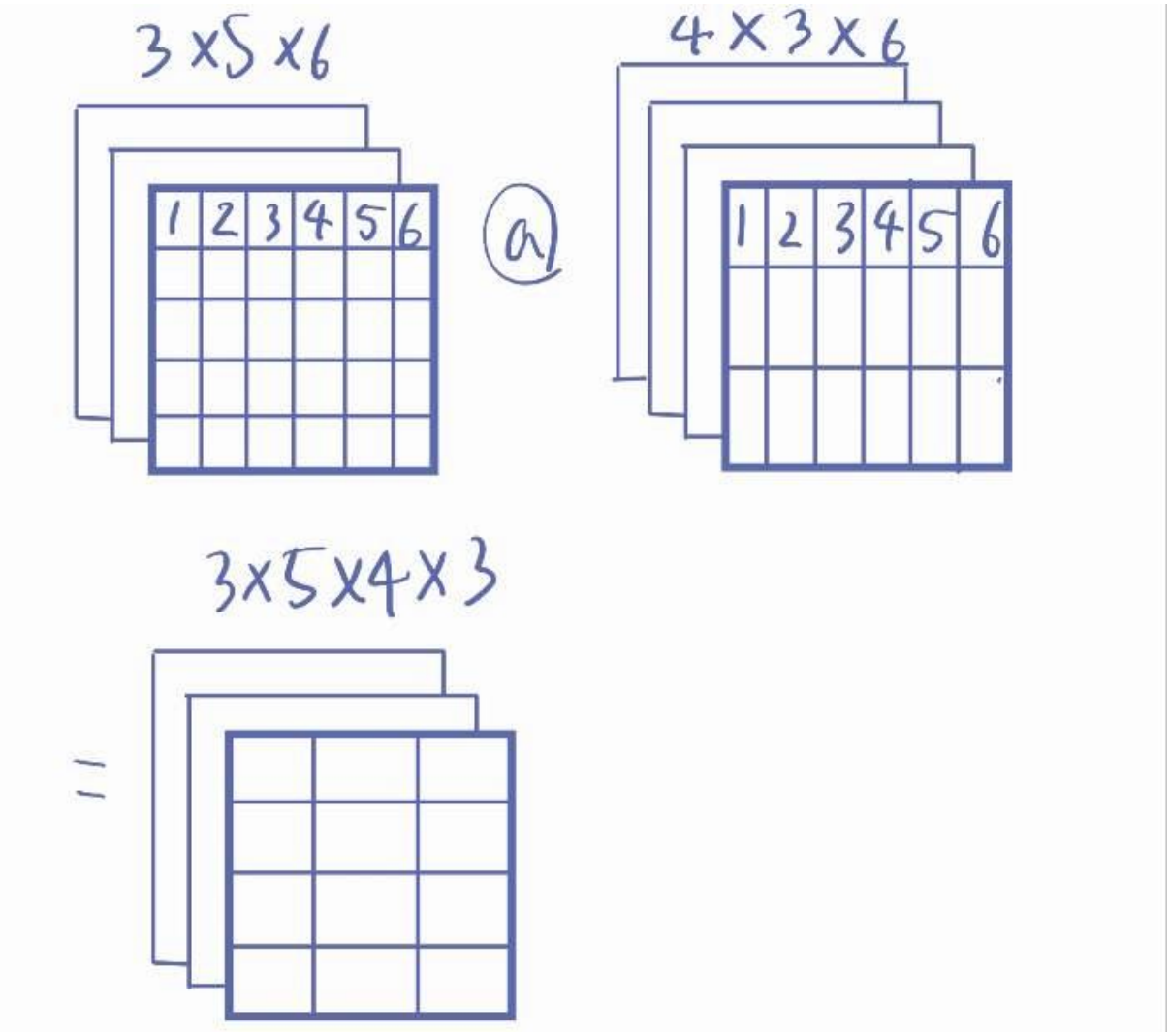
7.



7번 문제는 3 x 7 x 6 행렬과 4 x 6 x 5 행렬을 행렬곱 하는 문제이다. 이때 첫번째 행렬의 마지막 축의 크기과 두번째 행렬의 뒤에서 두번째 축의 크기가 같기에 곱셈이 가능하고 행렬곱을 하게되면 스칼라 값을 얻는데 첫번째 행렬은 6이 3 x 7개 있고 두번째 행렬은 4 x 5 개 만큼 존재하기에 최종 shape은 3 x 7 x 4 x 5가 된다. 대응되는 함수는 dot함수가 있다.

```
Resulting shape:  (3, 7, 4, 5)
All tests passed.
```

8.

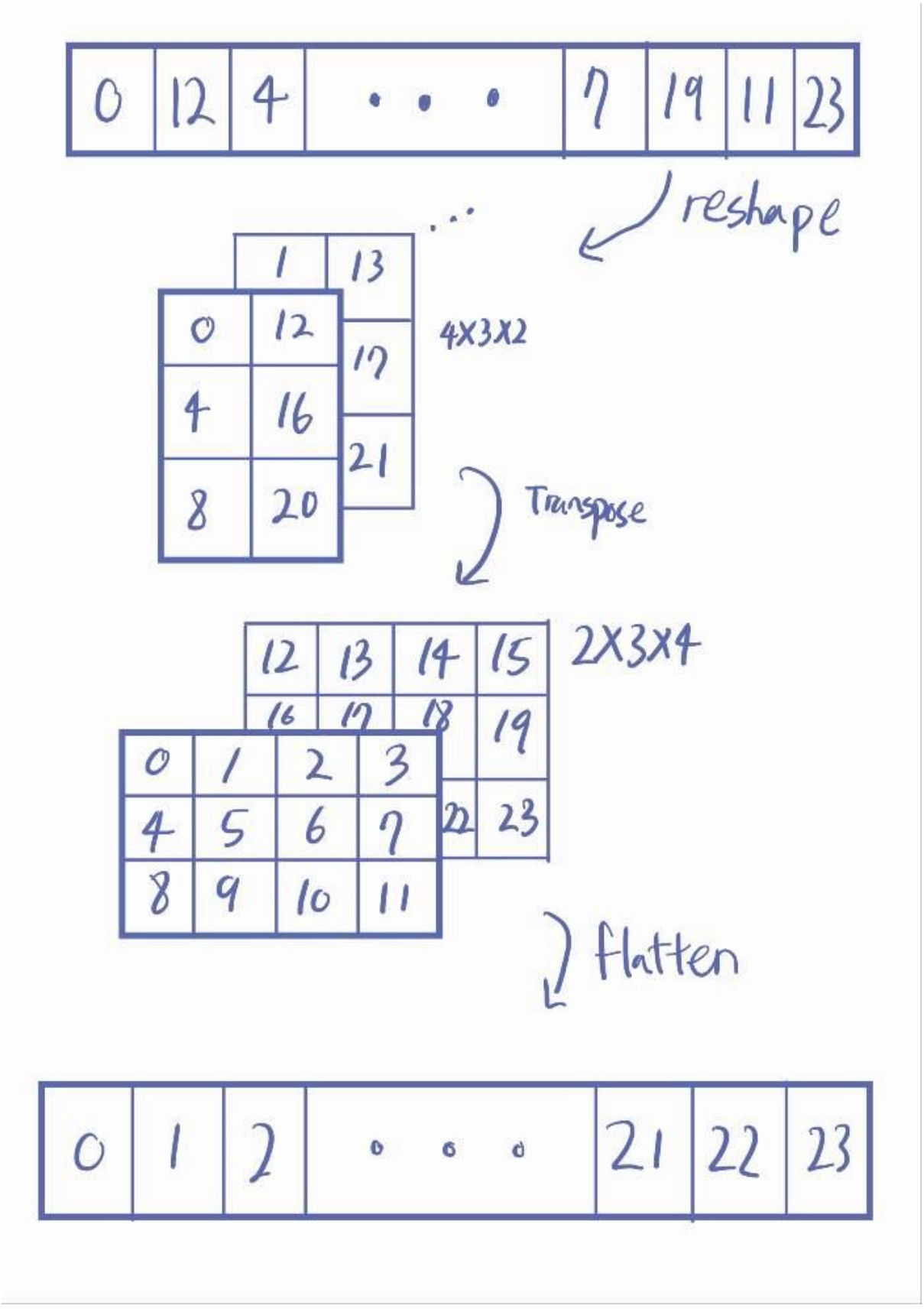


8번 문제는 3 x 5 x 6 행렬과 4 x 3 x 6 행렬끼리 내적을 진행하는 문제이다.

Inner 함수를 사용하는데 dot함수와 1차원 벡터 간 연산에서는 동일한 결과를 얻지만 다차원 행렬간 연산에서는 다른 결과를 얻는다. Inner 함수는 마지막 축을 기준으로 요소별 곱셈을 진행한 후 합산하고 dot함수는 첫번째 행렬의 마지막 축과 두번째 행렬의 뒤에서 두번째 축과 요소별 곱셈후 합산을 진행한다.

```
Resulting shape:  (3, 5, 4, 3)
All tests passed.
```

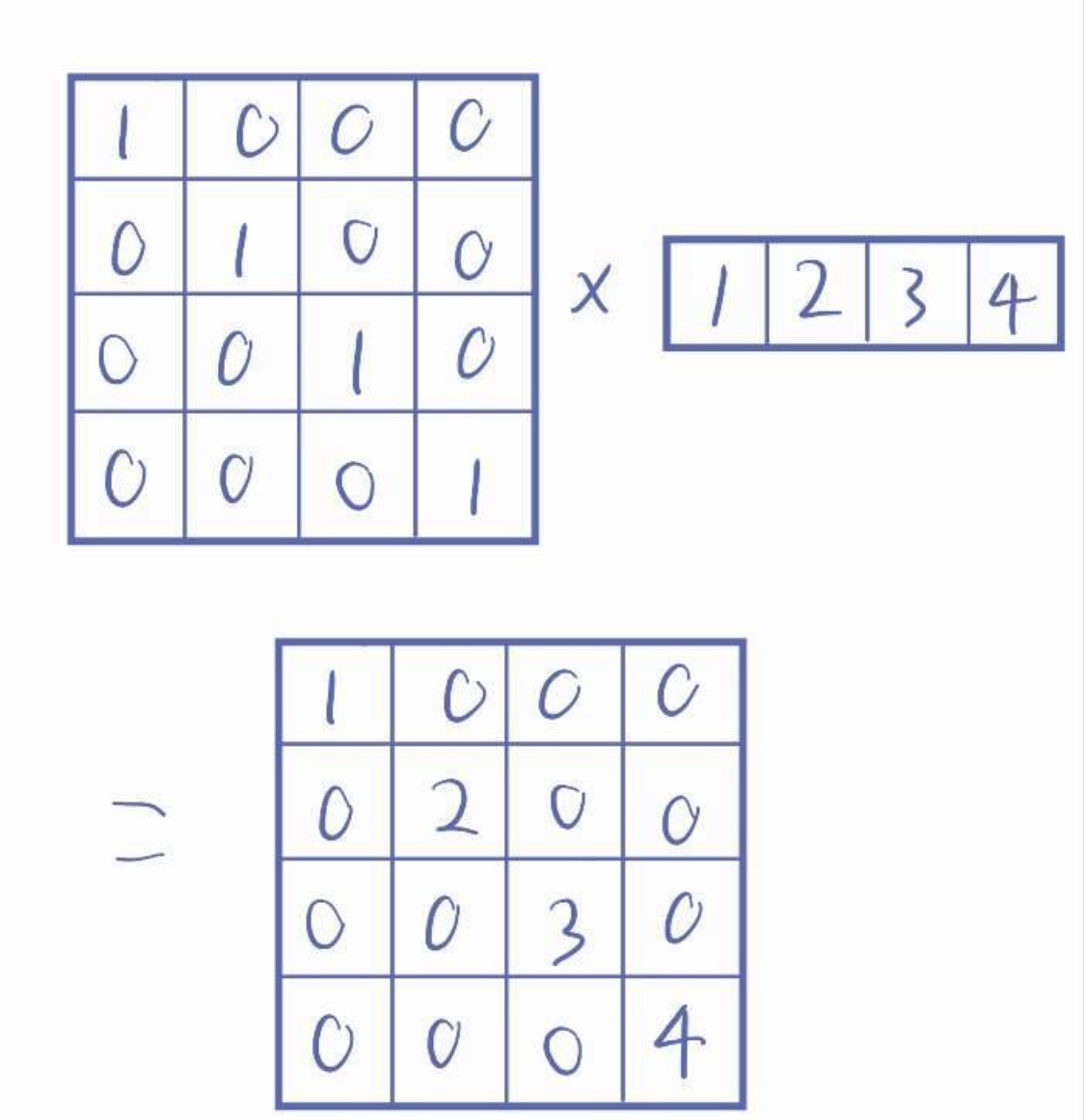
9.


0 12 4 . . . 7 19 11 23
reshape
1 13
0 12 17
4 16 21
8 20
4X3X2
Transpose
12 13 14 15
16 17 18 19
0 1 2 3
4 5 6 7 22 23
8 9 10 11
2X3X4
flatten
0 1 2 . . . 21 22 23

9번 문제는 행렬을 reshape과 transpose를 이용하여 섞여 있는 순열들을 올바르게 돌려놓는 문제이다. 먼저 0이 나오고 그 후 7번째 요소에서 1이 나왔고 0과 1 사이에 4, 8이 두 칸 간격으로 나왔기에 4 x 3 x 2로 reshape을 진행하였고 이후 0,1,2,3이 들어있는 첫번째 축을 마지막 축으로 이동하고 마지막 축의 12가 첫번째 3 x 4 행렬 이후에 나올 수 있도록 첫번째 축으로 이동해주고 평탄화하여 0부터 23까지 정렬된 행렬을 얻었다.

```
Resulting shape:  (24,)
All tests passed.
```

10.



10번 문제는 1,2,3,4를 요소로 가진 행렬과 곱하여 대각 성분이 1,2,3,4이고 그외 모든 성분이 0인 4x4행렬이 되게하는 행렬을 구하는 문제이다. eye라는 함수로 대각성분이 모두 1인 대각행렬을 생성하고 이를 [1,2,3,4]와 곱하여 result와 동일한 행렬을 구할 수 있었다. 행렬의 크기가 다르지만 자동으로 broadcasting되어 연산이 진행된 것을 확인한다.

```
Resulting shape:  (4, 4)
All tests passed.
```

Discussion

Numpy내 라이브러리 함수들을 이용하여 다양한 행렬연산을 진행해보았다. 반복문을 중첩시켜서 연산의 결과를 얻는 것과 함수를 사용하는 것의 결과의 차이를 확인하고 함수를 이용하는 방식의 편리함을 알 수 있고 직접 접근하는 방식을 통해 어떻게 동작하는지에 대한 이해가 쉬웠다. 또 연산자와 함수의 상관관계를 확인할 수 있었고 입력에 따라 연산의 결과가 같아질 수도 있고 달라질 수도 있기에 입력에 대해 꼼꼼히 확인해봐야 함을 알 수 있었다. 추가적으로 행렬 연산 문제에서 모두 einsum이라는 함수를 사용했는데 축을 I,j,k와 같이 나타내고 이를 이용하여 하나의 행렬을 transpose시키거나 두가지 행렬끼리 연산할 때 어떤 축을 건드느냐에 따라 덧셈 곱셈을 결정할 수 있는 것이 유용하게 사용될 수 있을 것 같았다.