인공지능프로그래밍
Lab 01: Numpy Basic TEST

학 번 : 2021202058

성 명 : 송채영

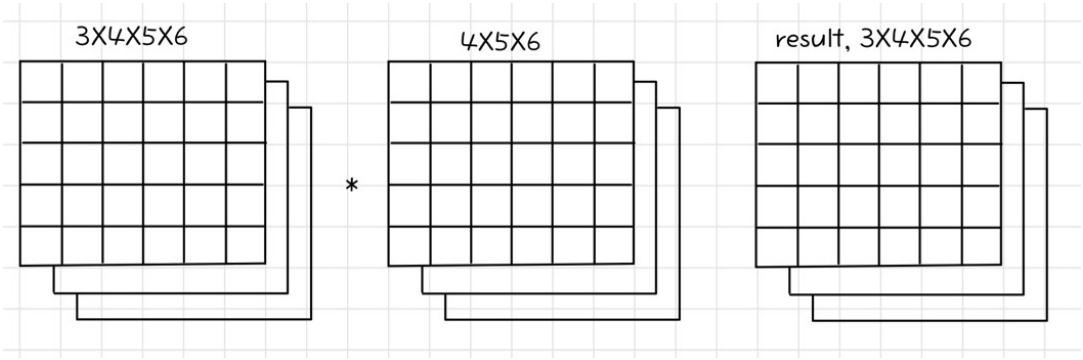
날 짜 : 2024.09.16

Lab Objective

이번 과제에서는 Numpy를 사용하여 배열을 다루고 연산하는 방법을 배운다. 배열 간 브로드캐스팅을 통해 과제로 주어진 크기가 다른 배열을 원소별로 연산하고 이를 다양한 함수와 Einsum을 활용하여 구현해본다. 마지막으로 각 방법을 통해 얻은 결과를 비교해보며 정확성을 검증해본다.

1.

A. Whole simulation program flow



3x4x5x6의 배열과 4x5x6의 배열을 * 연산을 진행했을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

4차원 배열과 3차원 배열을 브로드캐스팅을 사용해 곱셈을 진행하였다. Array3은 중첩 루프를 사용하여 직접 곱셈한 결과이고 array4는 np.multiply를 활용하여 진행한 결과이다. Array5는 np.einsum을 활용한 결과이다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(3,4,5,6)
array2 = np.random.rand(4,5,6)

result = array1 * array2                          # Broadcasting using the * operator

res_dim = result.shape
array3 = np.empty_like(result)
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        for k in range(res_dim[2]):
            for l in range(res_dim[3]):
                ### START CODE HERE ###
                # Calculate element-wise multiplication using nested loops
                array3[i,j,k,l] = array1[i,j,k,l] * array2[j,k,l]
                ###  END CODE HERE  ###

### START CODE HERE ###
array4 = np.multiply(array1, array2)                        # use multiply function
array5 = np.einsum('ijkl, jkl->ijkl', array1, array2)       # use einsum
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.array_equal(result, array3):
    print('Array3 fails the test.'); test_passed = False
if not np.array_equal(result, array4):
    print('Array4 fails the test.'); test_passed = False
if not np.array_equal(result, array5):
    print('Array5 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

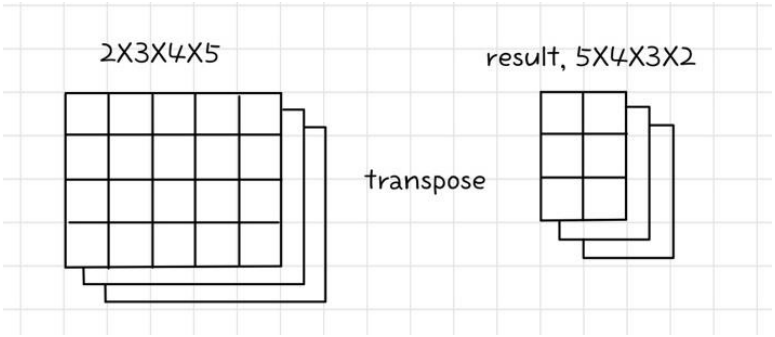
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3, array4, array5를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

2.

A. Whole simulation program flow



2x3x4x5의 배열을 transpose 했을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

4차원 배열을 T 속성을 활용하여 전치한 후, 즉 5x4x3x2 크기의 배열로 변환하여 result에 저장되었고 array3은 중첩루프를 사용하여 직접 할당하였다. Array4는 np.transpose를 활용하여 배열의 차원을 전치하였다. 이는 T 속성과 동일한 역할을 한다. Array5는 np.einsum을 이용하여 진행한 결과이다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(2,3,4,5)

result = array1.T                                    # Transpose using T attribute

res_dim = result.shape
array3 = np.empty_like(result)
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        for k in range(res_dim[2]):
            for l in range(res_dim[3]):
                ### START CODE HERE ###
                # Transpose using nested loops
                array3[i,j,k,l] = array1[l,k,j,i]
                ### END CODE HERE ###

### START CODE HERE ###
array4 = np.transpose(array1)                                # use transpose function
array5 = np.einsum('ijkl->lkji', array1)                     # use einsum
### END CODE HERE ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.array_equal(result, array3):
    print('Array3 fails the test.'); test_passed = False
if not np.array_equal(result, array4):
    print('Array4 fails the test.'); test_passed = False
if not np.array_equal(result, array5):
    print('Array5 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

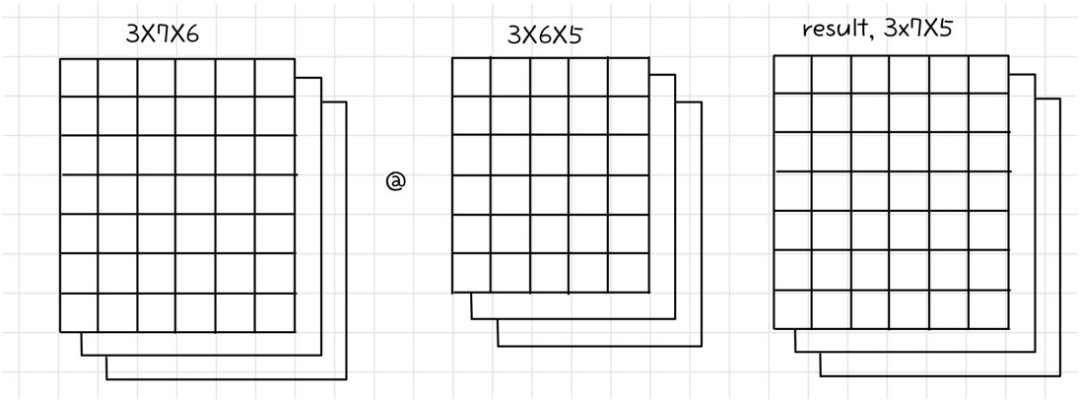
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3, array4, array5를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

3.

A. Whole simulation program flow



3x7x6의 배열과 3x6x5의 배열을 @(행렬 곱셈) 연산을 진행했을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

3차원 배열과 3차원 배열을 @(행렬 곱셈)을 수행한 결과를 result에 저장하였고, 중첩 루프를 활용하여 곱셈 결과를 array3에 저장하였다. Array4는 np.matmul을 사용하여 행렬 곱을 진행하였고, 이는 @ 연산자와 같은 기능을 수행한다. Array5는 np.einsum을 활용하였다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(3,7,6)
array2 = np.random.rand(3,6,5)

result = array1 @ array2                          # Matrix multiplication using @ operator

res_dim = result.shape
array3 = np.empty_like(result)
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        for k in range(res_dim[2]):
            ### START CODE HERE ###
            # matrix multiplication using nested loops
            array3[i,j,k] = array1[i,j,:] @ array2[i,:,k]
            ###  END CODE HERE  ###

### START CODE HERE ###
array4 = np.matmul(array1, array2)                # use matmul function
array5 = np.einsum('ijk, ikl->ijl', array1, array2) # use einsum
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.allclose(result, array3):
    print('Array3 fails the test.'); test_passed = False
if not np.array_equal(result, array4):
    print('Array4 fails the test.'); test_passed = False
if not np.allclose(result, array5):
    print('Array5 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

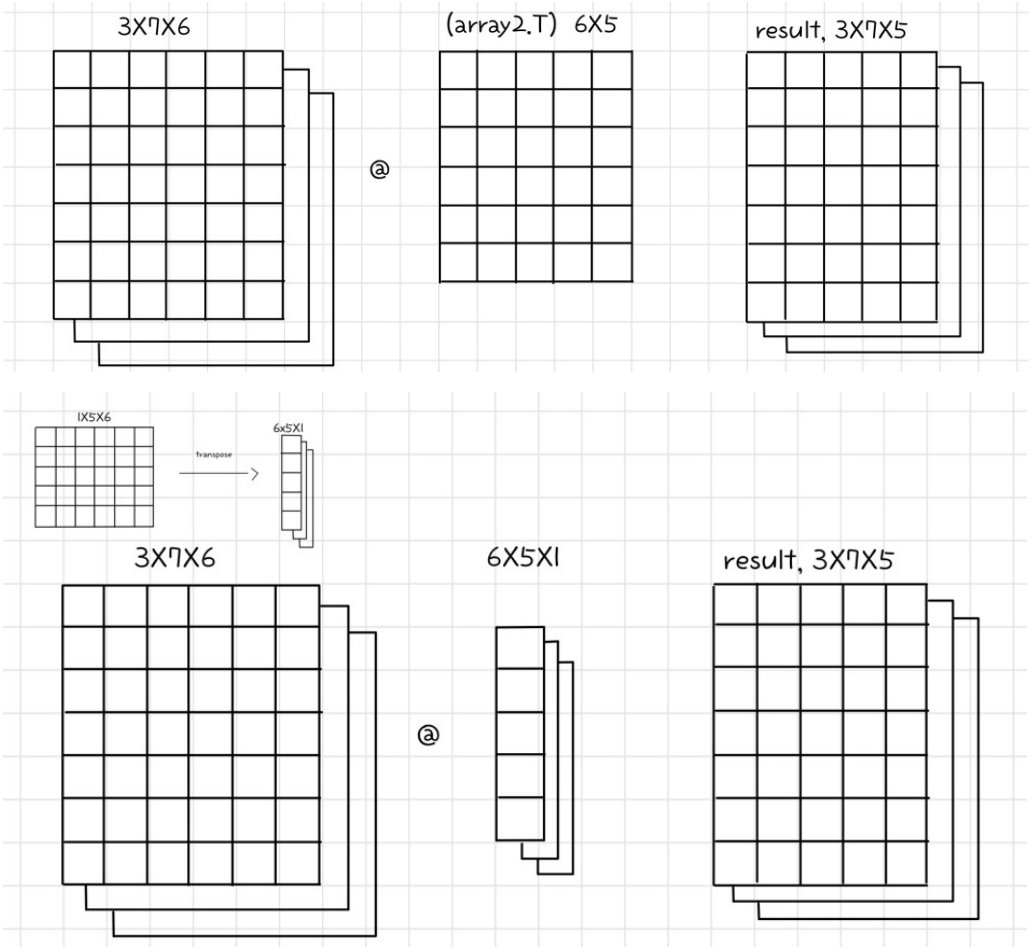
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3, array4, array5를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

4.

A. Whole simulation program flow



3x7x6의 배열과 5x6의 배열을 행렬 곱셈 연산을 진행했을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다. 먼저 첫 번째 그림은 result에 해당하는 그림이며 두 번째 그림은 array2를 reshape 한 후 진행한 결과에 해당한다.

B. Brief comments on each code block

우선 array2를 T 속성을 활용하여 크기가 (6,5)가 되었고 이 결과를 행렬 곱셈을 진행한 후 result에 저장하였다. 이어서 array2의 차원을 (1,5,6)으로 reshape 한 후 아래의 코드를 진행하였다. array3은 중첩루프를 활용해 수동으로 계산한 결과이다. Array4는 np.matmul과 np.transpose를 활용하여 계산하였다. Array5는 np.einsum을 활용하였다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(3,7,6)
array2 = np.random.rand(5,6)

result = array1 @ array2.T                          # Matrix multiplication using @ operator and T attribute

array2 = array2.reshape(1,5,6)                      # same array but one more higher dimension

res_dim = result.shape
array3 = np.empty_like(result)
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        for k in range(res_dim[2]):
            ### START CODE HERE ###
            # matrix multiplication using nested loops and adjusted dimensions
            array3[i,j,k] = array1[i,j,:] @ array2.T[:,k,0]
            ###  END CODE HERE  ###

### START CODE HERE ###
array4 = np.matmul(array1, np.transpose(array2,(0,2,1)))  # use matmul and transpose; keep the dimension
array5 = np.einsum('ijk, mk->ijm', array1, array2[0]) # use einsum
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.allclose(result, array3):
    print('Array3 fails the test.'); test_passed = False
if not np.array_equal(result, array4):
    print('Array4 fails the test.'); test_passed = False
if not np.allclose(result, array5):
    print('Array5 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

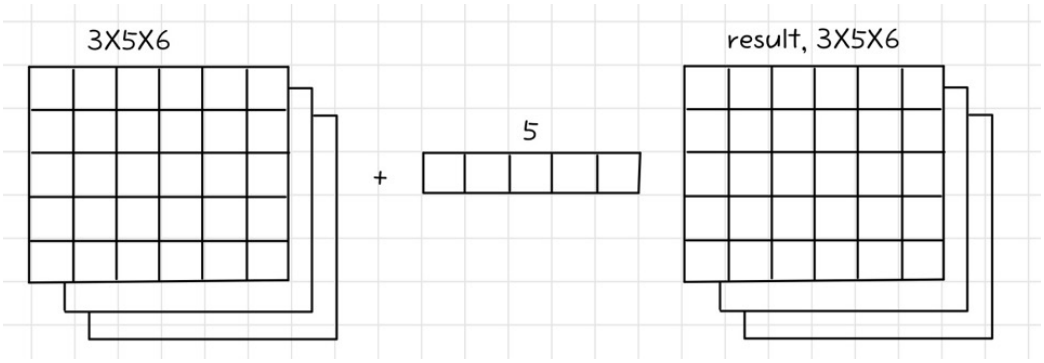
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3, array4, array5를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

5.

A. Whole simulation program flow



3x5x6의 배열과 크기가 5인 배열을 브로드캐스팅을 했을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

Result는 array1과 array2를 중첩 루프를 통해 덧셈을 진행하였을 때의 결과이고 이와 동일한 결과가 나오도록 array3을 구성하였다. Array2를 (1,5,1)로 reshape 한 후 array1과 브로드캐스팅을 수행하였다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(3,5,6)
array2 = np.random.rand(5)

# to make the following addition work
#   result = array1 + array2
# array2 must be broadcasted to shape of (3,5,6)

res_dim = array1.shape
result = np.empty(res_dim)
# Compute the result using nested loops
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        for k in range(res_dim[2]):
            result[i,j,k] = array1[i,j,k] + array2[j]

### START CODE HERE ###
# Broadcast array2 to shape (1, 5, 1) and add to array1
array3 = array1 + np.reshape(array2, (1,-1,1))              # make + operation possible with column broadcasting
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.array_equal(result, array3):
    print('Array3 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

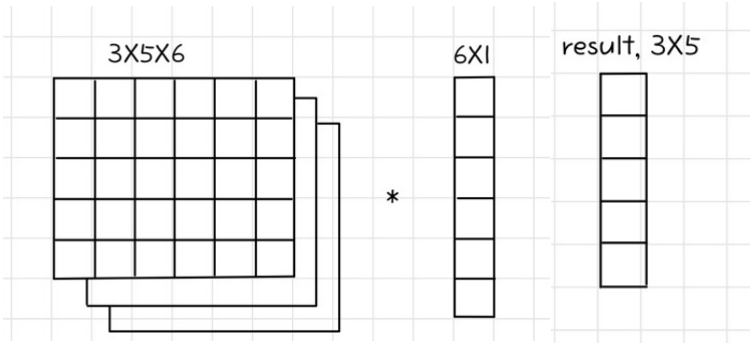
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3을 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

6.

A. Whole simulation program flow



3x5x6의 배열과 크기가 6x1인 배열을 곱했을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

result에는 array1과 array2의 열 벡터를 곱한 후, 곱셈 결과를 더해서 저장하였다. array3은 수학적으로는 다차원 행렬에 열 벡터를 곱하면 마지막 차원이 제거되지만, numpy에서는 그런식으로 동작되지 않기 때문에 squeeze를 활용하여 축을 제거한 후 진행하여야 한다. 따라서 np.squeeze(array2)를 함으로써 불필요한 차원을 제거하여 (6,)의 형태인 1D 배열로 만들어주었고 np.matmul을 활용해 행렬 곱셈을 수행하였다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(3,5,6)
array2 = np.random.rand(6,1)

# in mathamtics, multiplying a multi-dimensional matrix and a column vector
# removes the last dimension of the matrix.
# but numpy does not work that way. make the correction.

res_dim = array1.shape[0:2]
result = np.empty(res_dim)
# Compute the result using nested loops and summation
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        result[i,j] = np.sum(array1[i,j,:] * array2[:,0])

### START CODE HERE ###
# Matrix multiplication with squeezed array2
array3 = np.matmul(array1, np.squeeze(array2))
# use matmul and squeeze
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.allclose(result, array3):
    print('Array3 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

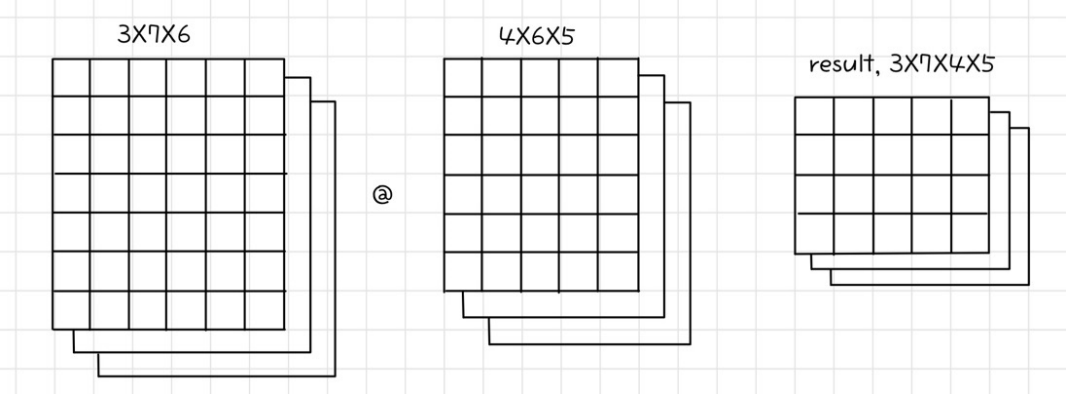
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3을 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

7.

A. Whole simulation program flow



3x7x6의 배열과 4x6x5 배열을 행렬 곱셈을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

Np.dot을 활용하여 두 배열의 행렬곱셈의 결과가 result에 저장되었고 array3에는 중첩루프를 통해 행렬 곱셈을 진행하였고, array4는 np.matmul을 사용하였다. Array5는 np.einsum을 활용하였다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(3,7,6)
array2 = np.random.rand(4,6,5)

result = np.dot(array1, array2)                    # Matrix multiplication using dot function

res_dim = result.shape
array3 = np.empty_like(result)
# Compute the result using nested loops and matrix multiplication
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        for k in range(res_dim[2]):
            for l in range(res_dim[3]):
                ### START CODE HERE ###
                # Matrix product of slices
                array3[i,j,k,l] = array1[i,j,:] @ array2[k,:,l]
                ###  END CODE HERE  ###

array4 = np.empty_like(result)
for i in range(res_dim[2]):
    ### START CODE HERE ###
    array4[:,:,i,:] = array1 @ array2[i,:,:]                                    # use @ or matmul
    ###  END CODE HERE  ###

### START CODE HERE ###
array5 = np.einsum('ijm, kml -> ijkl', array1, array2)                           # use einsum
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.allclose(result, array3):
    print('Array3 fails the test.'); test_passed = False
if not np.allclose(result, array4):
    print('Array4 fails the test.'); test_passed = False
if not np.allclose(result, array5):
    print('Array5 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

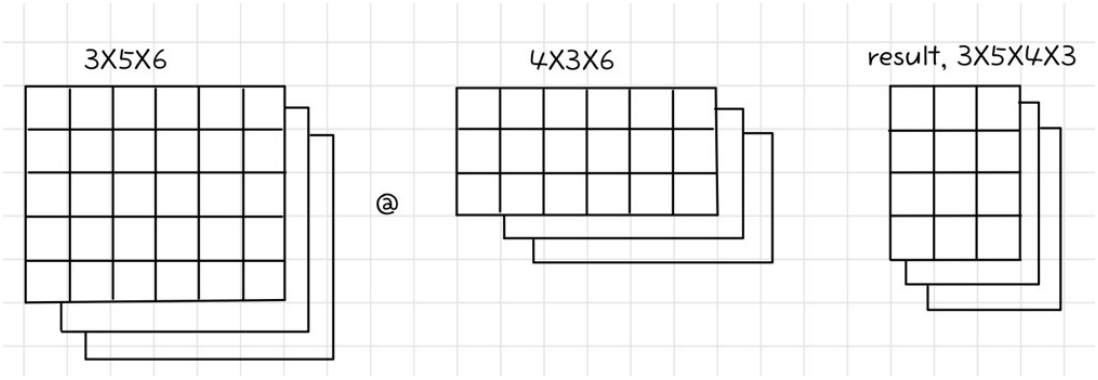
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3, array4, array5를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

8.

A. Whole simulation program flo



3x5x6의 배열과 4x3x6 배열을 내적 했을 때의 결과이다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

Np.inner을 활용하여 두 배열 간의 내적을 수행한 후 result에 저장하였다. Array3에는 중첩루프를 통해 두 배열의 내적을 계산하고 np.sum을 통해 결과를 저장해주었다. Array4는 np.transpose를 사용하여 array2의 차원을 (4,6,3)으로 전치하였고 array1과 array2를 행렬 곱셈을 수행하였다. Array5는 np.einsum을 활용하였다.

C. Codes

```
np.random.seed(0)
array1 = np.random.rand(3,5,6)
array2 = np.random.rand(4,3,6)

result = np.inner(array1, array2)                    # Inner product using inner function

res_dim = (3,5,4,3)
array3 = np.empty(res_dim)
# Compute the result using nested loops and inner product
for i in range(res_dim[0]):
    for j in range(res_dim[1]):
        for k in range(res_dim[2]):
            for l in range(res_dim[3]):
                ### START CODE HERE ###
                # Sum of element-wise product
                array3[i,j,k,l] = np.sum(array1[i,j,:] @ array2[k,l,:])
                ### END CODE HERE ###

### START CODE HERE ###
array4 = np.dot(array1, np.transpose(array2, (0,2,1)))                    # use dot
array5 = np.einsum('ijm, klm->ijkl', array1, array2)                      # use einsum
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.array_equal(result, array3):
    print('Array3 fails the test.'); test_passed = False
if not np.array_equal(result, array4):
    print('Array4 fails the test.'); test_passed = False
if not np.allclose(result, array5):
    print('Array5 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

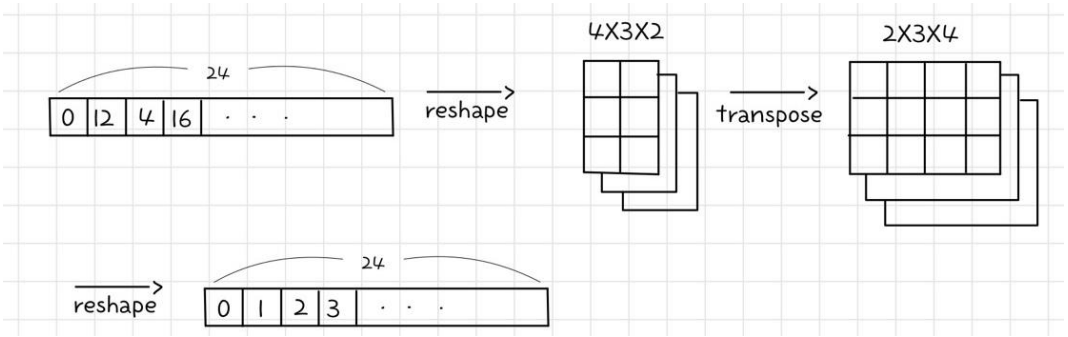
D. Simulation results

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

위 코드를 실행한 결과로 result와 array3, array4, array5를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

9.

A. Whole simulation program flow



1D 배열을 4x3x2로 reshape 한 후 이를 다시 2x3x4로 transpose 한 후 1D로 reshape 하여 result의 결과를 얻어내는 그림이다. 먼저 4x3x2로 reshape 한 이유는 result가 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,

21, 22, 23의 순서로 나와야 하기 때문에 여러 경우의 수를 그려서 확인해본 결과 위 그림과 같이 reshape 및 transpose를 진행하였다. 그림은 차원축소를 활용하여 2차원으로 나타내었다.

B. Brief comments on each code block

1D의 배열을 np.reshape을 활용하여 (4,3,2)로 변환하고 이를 다시 np.transpose를 활용하여 (2,3,4)로 변경하였다. 이를 다시 1D 배열로 변환하기 위해 reshpae을 해주어 길이가 24인 1D로 만들어주었다.

C. Codes

```
array1 = np.array([ 0, 12,  4, 16,  8, 20,  1, 13,  5, 17,  9, 21,  2, 14,  6, 18, 10, 22,  3, 15,  7, 19, 11, 23])

result = np.array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])

### START CODE HERE ###
# Reshape to (4,3,2), transpose to (2,3,4), reshape to 1D
array2 = np.transpose(np.reshape(array1, (4,3,2)), (2,1,0)).reshape(24)  # use rehape and transpose
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.array_equal(result, array2):
    print('Array2 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

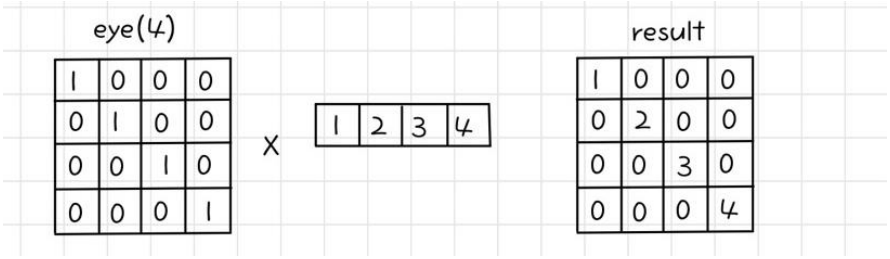
D. Simulation results

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

위 코드를 실행한 결과로 result와 array2를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

10.

A. Whole simulation program flow



4x4 단위 행렬과 길이가 4인 1D 배열의 곱셈을 진행한 결과이다.

B. Brief comments on each code block

Array1은 길이가 4인 1D 배열이며 result는 각 대각선 요소가 array1의 값으로 설정된 4x4 행렬이다. Np.eye(4)는 4x4의 단위행렬을 생성하며 이는 대각선 요소는 모두 1이고 나머지는 0인 행렬을 의미한다. Np.eye(4)와 Array1과의 곱셈을

진행한 결과를 array2에 저장해주었다.

C. Codes

```
array1 = np.array([1., 2., 3., 4.])

result = np.array([[1., 0., 0., 0.], [0., 2., 0., 0.], [0., 0., 3., 0.], [0., 0., 0., 4.]])

### START CODE HERE ###
# Create identity matrix and scale by array1
array2 =  np.eye(4) * array1                              # use eye
###  END CODE HERE  ###

print('Resulting shape: ', result.shape); test_passed = True
if not np.array_equal(result, array2):
    print('Array2 fails the test.'); test_passed = False
if test_passed: print('All tests passed.')
```

D. Simulation results

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

위 코드를 실행한 결과로 result와 array2를 비교했을 때 모두 동일한 결과를 나타내는 것을 확인할 수 있었다.

Discussion

이번 과제를 진행하며 Numpy의 다양한 차원에서의 행렬의 여러 연산들을 해볼 수 있었다. 처음에는 뭘 의미하고 뭘 구현해야 하는지 쉽게 와닿지 않았는데 여러 번 반복해서 시도해보고 또 그림을 그려가며 이해를 더해가니 어렵지 않고 재미있었다. 특히 여러 방법들 중 einsum이 가장 쉽고 간결했던 것 같다. 배열이 복잡해져도 배열의 인덱스 규칙을 사용한다는 점에서 원하는 대로 구현할 수 있어 편리했다. 여러 문제들 중 9번 문제에 가장 많은 시간을 썼는데, 문제 조건에 reshpae과 transpose를 활용하여 result와 같은 결과를 내라고 하다 보니 24가 될 수 있는 모든 경우의 수를 다 그려가며 같은 결과가 나오는 크기를 찾아보았다. 이 과정을 거치다 보니 시간이 오래 걸렸지만, 이렇게 하나하나 해보지 않고 빠르게 크기를 찾을 수 있다면 좋을 것 같다. 수업 때 개념으로만 보았던 부분들이라 익숙하지 않았는데 이번 과제를 통해 많은 도움이 됐던 것 같다.