MAT6012 - Programming for Data Analysis

NumPy - Broadcasting

- The term **broadcasting** refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations.
- Arithmetic operations on arrays are usually done on corresponding elements.
- If two arrays are exactly the same shape, these operations are smoothly performed.

```
a = np.array([1, 2, 3])
b = np.array([2, 2, 2])
a * b
array([2,  4,  6])
```

- If the dimensions of two arrays are dissimilar, element-to-element operations are not possible.
- However, operations on arrays of non-similar shapes are still possible in NumPy, because of the broadcasting capability.
- The smaller array is **broadcast** to the size of the larger array so that they have compatible shapes.

```
a = np.array([1, 2, 3])
b = 2
a * b
array([2,  4,  6])
```

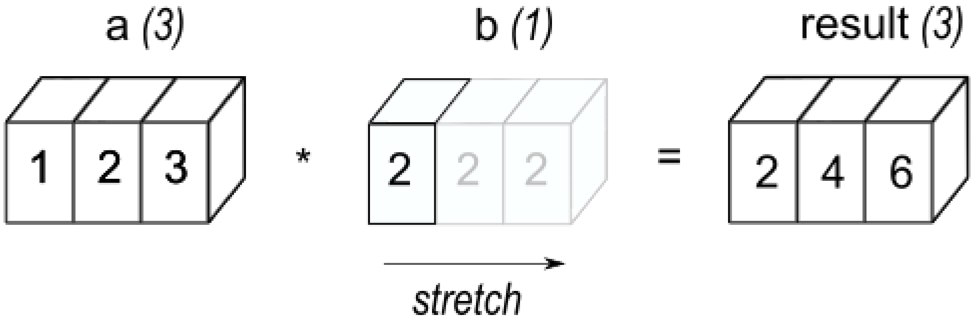

Figure 1: A scalar is broadcast to match the shape of the 1-d array it is being multiplied to.

General Broadcasting Rules

- When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing (i.e., rightmost) dimensions and works its way left.
- Two dimensions are compatible when
  1. they are equal, or
  2. one of them is 1

- If these conditions are not met, a `ValueError: operands could not be broadcast together` exception is thrown.
- Arrays do not need to have the same number of dimensions, but still, the operations can be performed by NumPy thanks to broadcasting.
- **Broadcasting is possible if the following rules are satisfied —**
  - Array with a smaller **ndim** than the other is prepended with '1' in its shape.
  - The size in each dimension of the output shape is the maximum of the input sizes in that dimension.
  - An input can be used in a calculation, if its size in a particular dimension matches the output size or its value is exactly 1.
  - If an input has a dimension size of 1, the first data entry in that dimension is used for all calculations along that dimension.
- Example:

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

Broadcastable Arrays

- A set of arrays is said to be **broadcastable** if one of the following is true and produces a valid result:
  - Arrays have exactly the same shape.
  - Arrays have the same number of dimensions, and the length of each dimension is either a common length or 1.
  - Array having too few dimensions can have its shape prepended with a dimension of length 1 so that the above-stated property is true.
- For example, if `a.shape` is (5,1), `b.shape` is (1,6), `c.shape` is (6,) and `d.shape` is () so that `d` is a scalar, then `a,b,c,` and `d` are all broadcastable to dimension (5,6); and
  - `a` acts like a (5,6) array where `a[:,0]` is broadcast to the other columns,
  - `b` acts like a (5,6) arrays where `b[0,:]` is broadcast to the other rows,
  - `c` acts like a (5,6) array where `c[:]` is broadcast to every row, and finally,
  - `d` acts like a (5,6) array where the single value is repeated in the entire array.
- Demonstration:

```
A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4

A      (2d array):  5 x 4
```

```
B      (1d array):      4
Result (2d array):  5 x 4

A      (3d array):  15 x 3 x 5
B      (3d array):  15 x 1 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 1
Result (3d array):  15 x 3 x 5
```

- Examples that do not broadcast

```
A      (1d array):  3
B      (1d array):  4 # trailing dimensions do not match

A      (2d array):      2 x 1
B      (3d array):  8 x 4 x 3 # second from last dimensions mismatched
```

Programming examples demonstrating the broadcasting.

Example 1

```
import numpy as np
a = np.array([[0,0,0],[10,10,10],[20,20,20],[30,30,30]])
b = np.array([1,2,3])

print('First array:')
print(a)
print('\n')

print('Second array:')
print(b)
print('\n')

print('First Array + Second Array')
print(a + b)
```

The output of this program would be as follows —

```
First array:
[[ 0  0  0]
 [ 10 10  10]
 [ 20 20  20]
 [ 30 30  30]]
```

```
Second array:
[ 1 2 3]

First Array + Second Array
[[ 1 2 3]
 [ 11 12 13]
 [ 21 22 23]
 [ 31 32 33]]
```

The following figure demonstrates how array **b** is broadcast to become compatible with **a**.

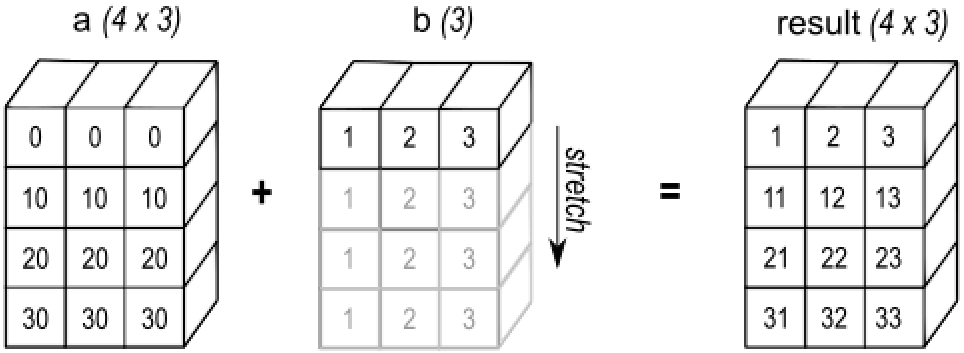

Figure 2: A 1-d array with shape (3) is stretched to match the 2-d array of shape (4, 3) it is being added to, and the result is a 2-d array of shape (4, 3).

Example 2

```
import numpy as np

a = np.array([0,0,0],[10,10,10],[20,20,20],[30,30,30])
b = np.array([1,2,3,4])

print(a.shape)
#Output: (4,3)
print(b.shape)
#Output: (4,)

print(a+b)
# ValueError: operands could not be broadcast together with shapes (4,3) (4,)
```

Example 3

```
import numpy as np

x = np.array([0,10,20,30])
xx = x.reshape(4,1)
```

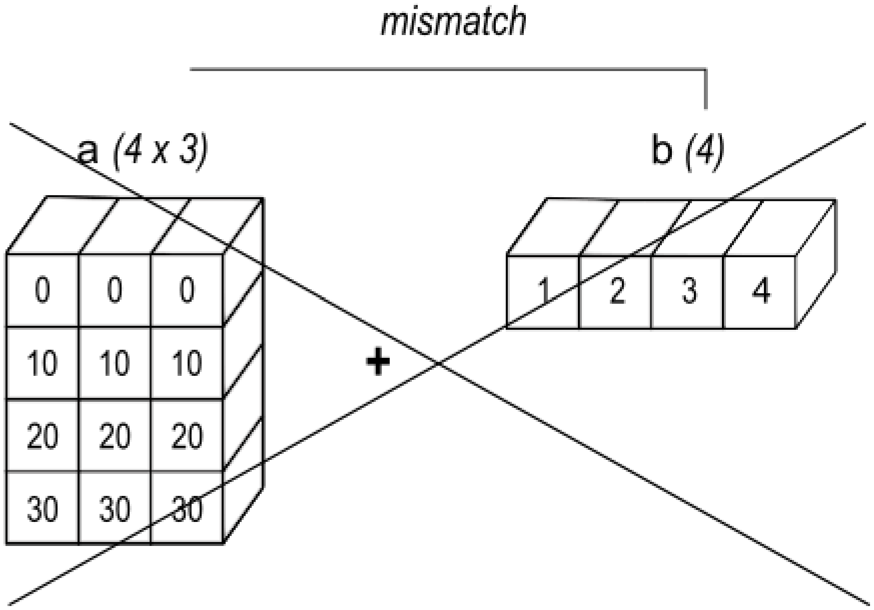

Figure 3: A huge cross over the 2-d array of shape (4, 3) and the 1-d array of shape (4) shows that they can not be broadcast due to mismatch of shapes and thus produce no result.

```
y = np.ones(5)
yy = np.array([1,2,3])
z = np.ones((3,4))

print(x.shape)
#Output: (4,)

print(y.shape)
#Output: (5,)

print(x + y)
#Output: ValueError: operands could not be broadcast together with shapes (4,) (5,)

print(xx.shape)
#Output: (4, 1)

print(y.shape)
#Output: (5,)

print((xx + yy).shape)
#Output: (4, 4)

print(xx + yy)
'''Output:
[[  1  2  3],
 [ 11 12 13],
 [ 21 22 23],
 [ 31 32 33]]
'''

print(x.shape)
#Output: (4,)

print(z.shape)
#Output: (3, 4)

print((x + z).shape)
#Output: (3, 4)

print(x + z)
'''
Output:
[[ 1.  11.  21.  31.],
 [ 1.  11.  21.  31.],
 [ 1.  11.  21.  31.]]
'''
```

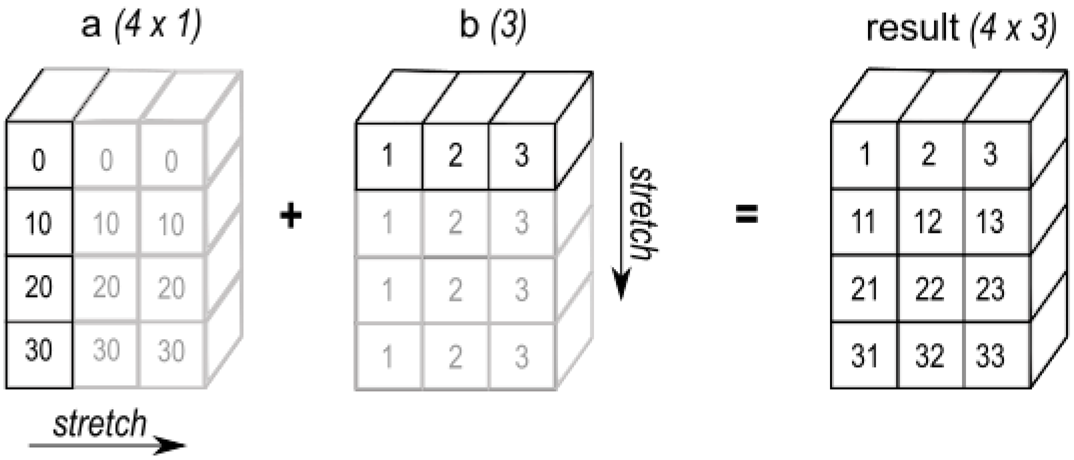

Figure 4: A 2-d array of shape (4, 1) and a 1-d array of shape (3) are stretched to match their shapes and produce a resultant array of shape (4, 3).

The first part of the code is equivalent to the below also:

```
x = np.array([0, 10, 20, 30])
y = np.array([1, 2, 3])
xx = x[:, np.newaxis] #creates a newaxis and makes the array into two-dimensional
print(xx+y)
'''
Output:
```

```
[[  1   2   3],
 [ 11  12  13],
 [ 21  22  23],
 [ 31  32  33]]
'''
```