Arithmetic operations and Broadcasting

We have reached the last lesson in this Introduction to NumPy. In this last lesson we will see how NumPy does arithmetic operations on ndarrays. NumPy allows element-wise operations on ndarrays as well as matrix operations. In this lesson we will only be looking at element-wise operations on ndarrays. In order to do element-wise operations, NumPy sometimes uses something called *Broadcasting*. Broadcasting is the term used to describe how NumPy handles element-wise arithmetic operations with ndarrays of different shapes. For example, broadcasting is used implicitly when doing arithmetic operations between scalars and ndarrays.

Let's start by doing element-wise addition, subtraction, multiplication, and division, between ndarrays. To do this, NumPy provides a functional approach, where we use functions such as np.add(), or by using arithmetic symbols, such as +, that resembles more how we write mathematical equations. Both forms will do the same operation, the only difference is that if you use the function approach, the functions usually have options that you can tweak using keywords. It is important to note that when performing element-wise operations, the shapes of the ndarrays being operated on, must have the same shape or be broadcastable. We'll explain more about this later in this lesson. Let's start by performing element-wise arithmetic operations on rank 1 ndarrays:

Example 1. Element-wise arithmetic operations on 1-D arrays

```
# We create two rank 1 ndarrays

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

y = np.array([5.5,6.5,7.5,8.5])


# We print x

print()

print('x = ', x)


# We print y

print()

print('y = ', y)

print()


# We perfrom basic element-wise operations using arithmetic symbols and functions

print('x + y = ', x + y)

print('add(x,y) = ', np.add(x,y))
```

```
print()

print('x - y = ', x - y)

print('subtract(x,y) = ', np.subtract(x,y))

print()

print('x * y = ', x * y)

print('multiply(x,y) = ', np.multiply(x,y))

print()

print('x / y = ', x / y)

print('divide(x,y) = ', np.divide(x,y))
```

```
x = [1 2 3 4]

y = [ 5.5 6.5 7.5 8.5]

x + y = [ 6.5 8.5 10.5 12.5]
add(x,y) = [ 6.5 8.5 10.5 12.5]

x - y = [-4.5 -4.5 -4.5 -4.5]
subtract(x,y) = [-4.5 -4.5 -4.5 -4.5]

x * y = [ 5.5 13. 22.5 34. ]
multiply(x,y) = [ 5.5 13. 22.5 34. ]

x / y = [ 0.18181818 0.30769231 0.4 0.47058824]
divide(x,y) = [ 0.18181818 0.30769231 0.4 0.47058824]
```

We can also perform the same element-wise arithmetic operations on rank 2 ndarrays. Again, remember that in order to do these operations the shapes of the ndarrays being operated on, must have the same shape or be broadcastable.

Example 2. Element-wise arithmetic operations on a 2-D array (Same shape)

```
# We create two rank 2 ndarrays

X = np.array([1,2,3,4]).reshape(2,2)

Y = np.array([5.5,6.5,7.5,8.5]).reshape(2,2)


# We print X

print()

print('X = \n', X)
```

```
# We print Y
print()
print('Y = \n', Y)
print()


# We perform basic element-wise operations using arithmetic symbols and functions
print('X + Y = \n', X + Y)
print()
print('add(X,Y) = \n', np.add(X,Y))
print()
print('X - Y = \n', X - Y)
print()
print('subtract(X,Y) = \n', np.subtract(X,Y))
print()
print('X * Y = \n', X * Y)
print()
print('multiply(X,Y) = \n', np.multiply(X,Y))
print()
print('X / Y = \n', X / Y)
print()
print('divide(X,Y) = \n', np.divide(X,Y))
```

```
X =
[[1 2]
 [3 4]]

Y =
[[ 5.5 6.5]
 [ 7.5 8.5]]

X + Y =
[[ 6.5 8.5]
 [ 10.5 12.5]]
```

```
add(X,Y) =
[[ 6.5 8.5]
 [ 10.5 12.5]]

X - Y =
[[-4.5 -4.5]
 [-4.5 -4.5]]

subtract(X,Y) =
[[-4.5 -4.5]
 [-4.5 -4.5]]

X * Y =
[[ 5.5 13. ]
 [ 22.5 34. ]]

multiply(X,Y) =
[[ 5.5 13. ]
 [ 22.5 34. ]]

X / Y =
[[ 0.18181818 0.30769231]
 [ 0.4 0.47058824]]

divide(X,Y) =
[[ 0.18181818 0.30769231]
 [ 0.4 0.47058824]]
```

We can also apply mathematical functions, such as sqrt(x), to all elements of an ndarray at once.

Example 3. Additional mathematical functions

```
# We create a rank 1 ndarray

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


# We print x

print()

print('x = ', x)


# We apply different mathematical functions to all elements of x

print()

print('EXP(x) =', np.exp(x))

print()
```

```
print('SQRT(x) =',np.sqrt(x))

print()

print('POW(x,2) =',np.power(x,2)) # We raise all elements to the power of 2
```

x = [1 2 3 4]

EXP(x) = [2.71828183 7.3890561 20.08553692 54.59815003]

SQRT(x) = [1. 1.41421356 1.73205081 2.]

POW(x,2) = [1 4 9 16]

Another great feature of NumPy is that it has a wide variety of statistical functions. Statistical functions provide us with statistical information about the elements in an ndarray.

Note - Most of the statistical operations can be done using either a function or an equivalent method. For example, both numpy.mean function and numpy.ndarray.mean method will return the arithmetic mean of the array elements along the given axis.

Let's see some examples showing a variety of statistical operations:

Example 4. Statistical functions

```
# We create a 2 x 2 ndarray

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


# We print x

print()

print('X = \n', X)

print()


print('Average of all elements in X:', X.mean())

print('Average of all elements in the columns of X:', X.mean(axis=0))

print('Average of all elements in the rows of X:', X.mean(axis=1))

print()

print('Sum of all elements in X:', X.sum())

print('Sum of all elements in the columns of X:', X.sum(axis=0))

print('Sum of all elements in the rows of X:', X.sum(axis=1))

print()
```

```
print('Standard Deviation of all elements in X:', X.std())
print('Standard Deviation of all elements in the columns of X:', X.std(axis=0))
print('Standard Deviation of all elements in the rows of X:', X.std(axis=1))
print()
print('Median of all elements in X:', np.median(X))
print('Median of all elements in the columns of X:', np.median(X,axis=0))
print('Median of all elements in the rows of X:', np.median(X,axis=1))
print()
print('Maximum value of all elements in X:', X.max())
print('Maximum value of all elements in the columns of X:', X.max(axis=0))
print('Maximum value of all elements in the rows of X:', X.max(axis=1))
print()
print('Minimum value of all elements in X:', X.min())
print('Minimum value of all elements in the columns of X:', X.min(axis=0))
print('Minimum value of all elements in the rows of X:', X.min(axis=1))
```

```
X =
[[1 2]
 [3 4]]

Average of all elements in X: 2.5
Average of all elements in the columns of X: [ 2. 3.]
Average of all elements in the rows of X: [ 1.5 3.5]

Sum of all elements in X: 10
Sum of all elements in the columns of X: [4 6]
Sum of all elements in the rows of X: [3 7]

Standard Deviation of all elements in X: 1.11803398875
Standard Deviation of all elements in the columns of X: [ 1. 1.]
Standard Deviation of all elements in the rows of X: [ 0.5 0.5]

Median of all elements in X: 2.5
Median of all elements in the columns of X: [ 2. 3.]
Median of all elements in the rows of X: [ 1.5 3.5]

Maximum value of all elements in X: 4
Maximum value of all elements in the columns of X: [3 4]
Maximum value of all elements in the rows of X: [2 4]
```

Minimum value of all elements in X: 1
Minimum value of all elements in the columns of X: [1 2]
Minimum value of all elements in the rows of X: [1 3]

Finally, let's see how NumPy can add single numbers to all the elements of an ndarray without the use of complicated loops.

Example 5. Change value of all elements of an array

```
# We create a 2 x 2 ndarray

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


# We print x

print()

print('X = \n', X)

print()


print('3 * X = \n', 3 * X)

print()

print('3 + X = \n', 3 + X)

print()

print('X - 3 = \n', X - 3)

print()

print('X / 3 = \n', X / 3)
```

```
X =
[[1 2]
 [3 4]]

3 * X =
[[ 3 6]
 [ 9 12]]

3 + X =
[[4 5]
 [6 7]]

X - 3 =
[[-2 -1]
 [ 0 1]]
```

```
X / 3 =
[[ 0.33333333 0.66666667]
 [ 1. 1.33333333]]
```

In the examples above, NumPy is working behind the scenes to broadcast 3 along the ndarray so that they have the same shape. This allows us to add 3 to each element of X with just one line of code.

Subject to certain constraints, Numpy can do the same for two ndarrays of different shapes, as we can see below.

Example 6. Arithmetic operations on 2-D arrays (Compatible shape)

```
# We create a rank 1 ndarray

x = np.array([1,2,3])


# We create a 3 x 3 ndarray

Y = np.array([[1,2,3],[4,5,6],[7,8,9]])


# We create a 3 x 1 ndarray

Z = np.array([1,2,3]).reshape(3,1)


# We print x

print()

print('x = ', x)

print()


# We print Y

print()

print('Y = \n', Y)

print()


# We print Z

print()

print('Z = \n', Z)
```

```
print()


print('x + Y = \n', x + Y)
print()
print('Z + Y = \n',Z + Y)
```

```
x = [1 2 3]

Y =
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Z =
[[1]
 [2]
 [3]]

x + Y =
[[ 2 4 6]
 [ 5 7 9]
 [ 8 10 12]]

Z + Y =
[[ 2 3 4]
 [ 6 7 8]
 [10 11 12]]
```

As before, NumPy is able to add 1 x 3 and 3 x 1 ndarrays to 3 x 3 ndarrays by broadcasting the smaller ndarrays along the big ndarray so that they have compatible shapes. In general, NumPy can do this provided that the smaller ndarray, such as the 1 x 3 ndarray in our example, can be expanded to the shape of the larger ndarray in such a way that the resulting broadcast is unambiguous.

Make sure you check out the NumPy Documentation for more information on Broadcasting and its rules:
Broadcasting

Glossary of Mathematical Functions

- Refer to this list of NumPy Mathematical Functions to find the one you need.

Supporting Materials

- Arithmetic operations and Broadcasting