Data Science Workshop-1

ITER, SOA University

Centre for Data Science
SOA University

All about numpy

- `ndarray`:- Multidimensional array providing fast array oriented arithmetic operations
- Mathematical functions for fast operations on entire arrays of data(without writing loops)
- Tools for reading/writing array data to disk and working with memory mapped files.(A technique that allows a part of the virtual address space to be associated with a file logically, which will increase the performance)
- Linear algebra, random number generation
- A **C API** for connecting NumPy with libraries written in C,C++ or FORTRAN(interface means contract of service between two applications)

Contents

ndarrays are faster

```
import numpy as np
L = np.arange(1000000)
M = list(range(1000000))
%timeit L1 = L * 2
```

```
1.25 ms ± 22.1 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%timeit M1 = [x * 2 for x in M]
```

```
62.2 ms ± 1.8 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
from datetime import datetime as dt
now1 = dt.now()
M1 = [x * 2 for x in M]
now2=dt.now()
print((now2-now1))
```

```
0:00:00.063089
```

```
now1 = dt.now()
L1 = L * 2
now2=dt.now()
print((now2-now1))
```

```
0:00:00.000862
```

Creating arrays in numpy

- An ndarray is a generic multidimensional container for homogeneous data; that is, all of the elements must be the same type
- The easiest way to create an ndarray is to use the array function in numpy module.
- Nested sequences, like a list of equal-length lists, will be converted into a multidimensional array

```
data = np.array([1.5, -0.1, 3])
print(data,"and it's type is",type(data))
```

```
[ 1.5 -0.1  3. ] and it's type is <class 'numpy.ndarray'>
```

```
#difference data types converted to same
data=np.array(['Hello',1])
print(data)
```

```
['Hello' '1']
```

```
data=np.array([[1,2,6.5],['Hello',4.7,'ITER']])
print(data)
```

```
[['1' '2' '6.5']
 ['Hello' '4.7' 'ITER']]
```

Shape and dimension of ndarray

- `arr.ndim`: function return the number of dimensions of an array.
- `arr.shape`: shape of an array is the number of elements in each dimension.
- `arr.size`: Try this and see what you are getting.

```
data=np.array([[1,2,6.5],['Hello',4.7,'ITER']])
print(data)
```

```
[['1' '2' '6.5']
 ['Hello' '4.7' 'ITER']]
```

```
data.ndim
```

```
2
```

```
data.shape
```

```
(2, 3)
```

Other functions for creating new arrays

- `numpy.zeros` and `numpy.ones` create arrays of 0s or 1s, respectively, with a given length or shape.
- `numpy.empty` creates an array without initializing its values to any particular value.
- To create a higher dimensional array with these methods, pass a tuple for the shape.

```
x=np.zeros(10)
x
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
y=np.ones(7)
y
```

```
array([1., 1., 1., 1., 1., 1., 1.])
```

```
np.zeros((3, 6))
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

```
np.empty((2, 3, 2))
```

```
array([[[1.37335591e-311, 2.47032823e-322],
        [0.00000000e+000, 0.00000000e+000],
        [1.16709769e-312, 2.42336543e-057]],

       [[4.75778252e-090, 6.88903061e-042],
        [8.26772458e-072, 8.37820819e+169],
        [3.99910963e+252, 2.17564768e-076]]])
```

Creating new arrays

- `arange`: Like the built-in range but returns an ndarray instead of a list
  - syntax: `numpy.arange(start = 0, stop, step = 1, dtype = None)`
- `linspace()`: function is used to create an array of evenly spaced numbers within a specified range

```
import numpy as np
np.arange(5)
```

```
array([0, 1, 2, 3, 4])
```

```
#generate an array starting from 10 to 20(exclusive)
#with step size 5
np.arange(10,20,5)
```

```
array([10, 15])
```

```
# generate 5 elements between 10 and 20
np.linspace(10, 20 ,5)
```

```
array([10. , 12.5, 15. , 17.5, 20. ])
```

Creating new arrays

- Produce an array of the given shape and data type with all values set to the indicated "fill value"
- `numpy.full(shape, fill value)`
- `eye/identity` Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere)
- Return a new array of given shape and type, filled with fill value.

```
np.full((2, 3), 10)
```

```
array([[10, 10, 10],
       [10, 10, 10]])
```

```
res = np.identity(4)
res
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
arr2 = np.eye(4)
print(arr2)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

```
arr2 = np.eye(4,k=1)
print(arr2)
```

```
[[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]]
```

Data types for ndarrays

- The data type or dtype is a special object containing information about data.
- numpy tries to infer a good data type for the array that it creates.
- You can explicitly convert or cast an array from one data type to another using ndarray's astype method.
- A string which cannot be converted to float64, if we use astype method, a Value Error will be raised

converting data types for ndarrays

```
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)
print('arr1',arr1,'\n','arr2',arr2)
```

```
arr1 [1. 2. 3.]
 arr2 [1 2 3]
```

```
print(arr1.dtype)
print(arr2.dtype)
```

```
float64
int32
```

```
arr = np.array([1, 2, 3, 4, 5])
print(arr,arr.dtype)
```

```
[1 2 3 4 5] int32
```

```
arr = arr.astype(np.float64)
print(arr,arr.dtype)
```

```
[1. 2. 3. 4. 5.] float64
```

Arithmetic With NumPy arrays

- Batch operations on data can be performed in numpy without writing any for loops. This is known as vectorization.
- Any arithmetic operations between equal-size arrays apply the operation element-wise:

```
arr1 = np.array([[1., 2., 3.], [4., 5., 6.]]);arr1
```

```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

```
arr2 = np.array([[1., 1., 1.], [2., 2., 2.]]);arr2
```

```
array([[1., 1., 1.],
       [2., 2., 2.]])
```

```
arr1+arr2
```

```
array([[2., 3., 4.],
       [6., 7., 8.]])
```

```
arr1*arr2
```

```
array([[ 1.,  2.,  3.],
       [ 8., 10., 12.]])
```

Arithmetic operations

- Arithmetic operations with scalars propagate the scalar argument to each element in the array
- Comparisons between arrays of the same size yield Boolean arrays.

```
7*arr2
```

```
array([[ 7.,  7.,  7.],
       [14., 14., 14.]])
```

```
arr1
```

```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

```
arr2**2
```

```
array([[1., 1., 1.],
       [4., 4., 4.]])
```

```
arr2
```

```
array([[1., 1., 1.],
       [2., 2., 2.]])
```

```
7/arr2
```

```
array([[7. , 7. , 7. ],
       [3.5, 3.5, 3.5]])
```

```
arr1>arr2
```

```
array([[False,  True,  True],
       [ True,  True,  True]])
```

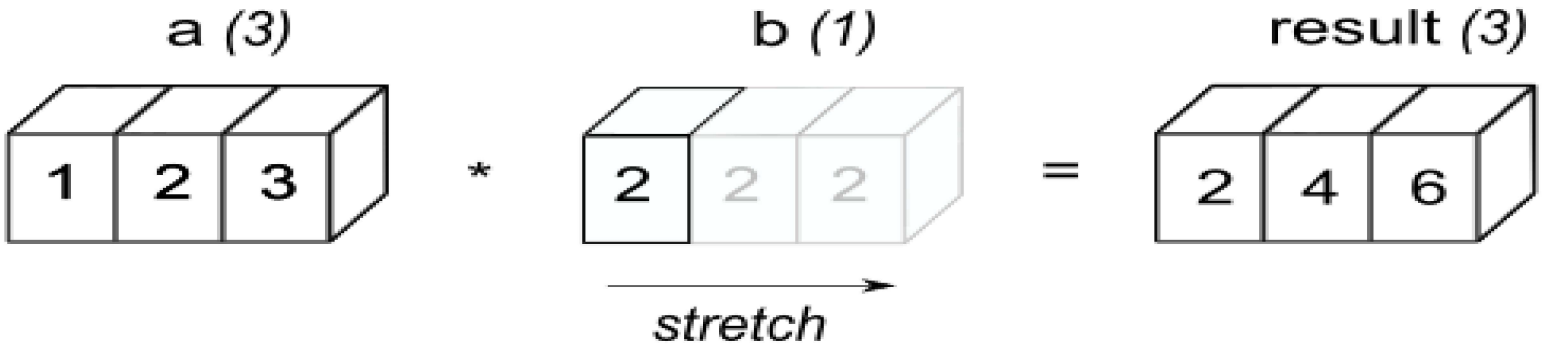
Let's solve

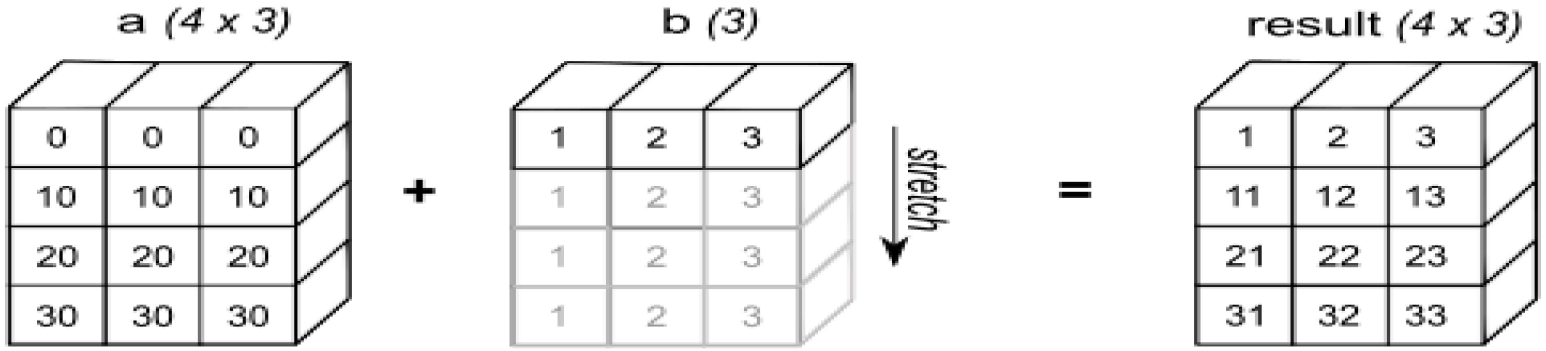
check point

- Create an ndarray object with float data type. Convert this to integer data type. Observe the changes in the data.
- Create identity matrix of order 4, and another matrix of order 4 with sub-diagonal elements 1 and super-diagonal elements 1, and other element zero. Compare the elements of each matrix.

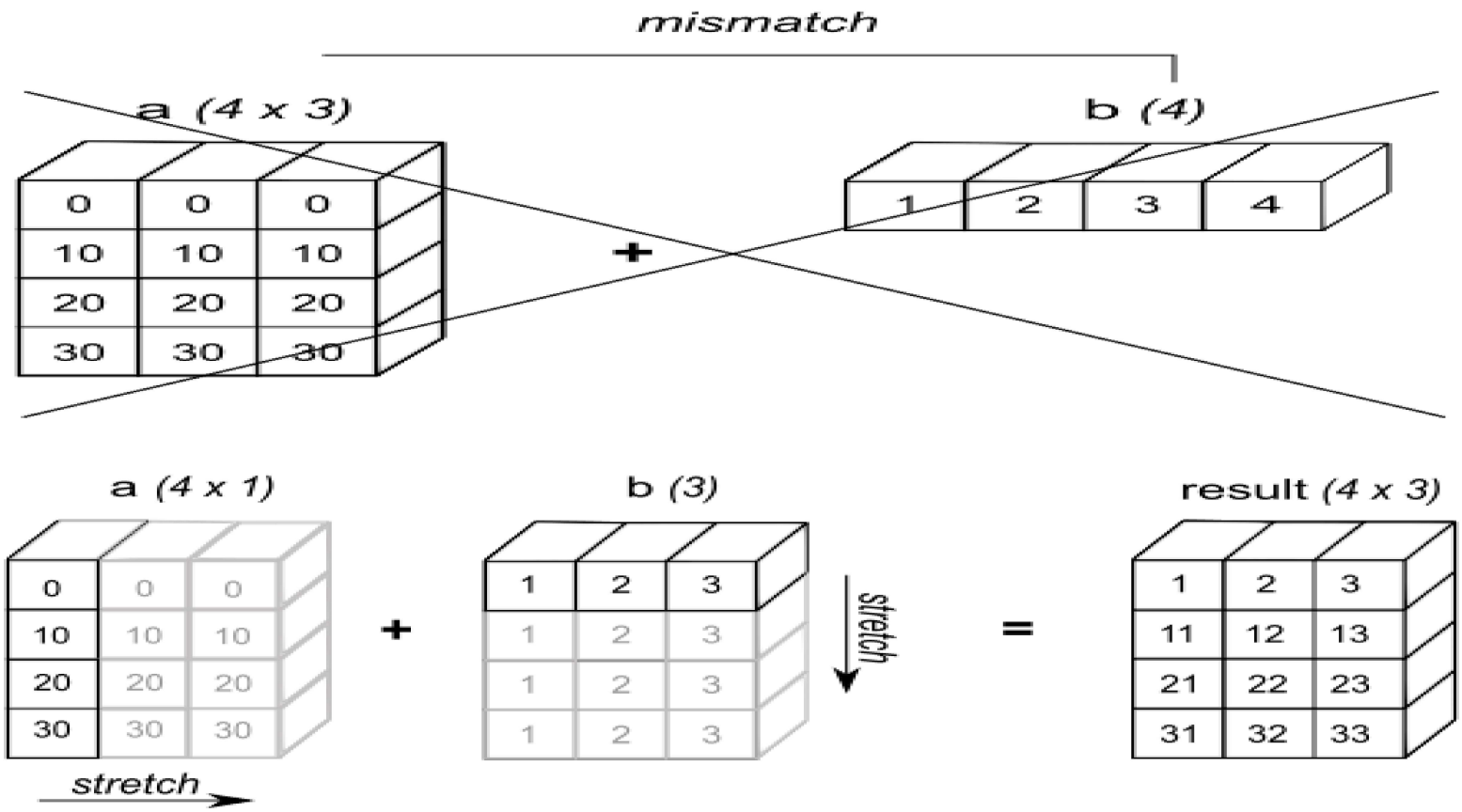
Numpy Broadcasting

- Broadcasting refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations.
- The smaller array is broadcast to the size of the larger array so that they have compatible shapes.





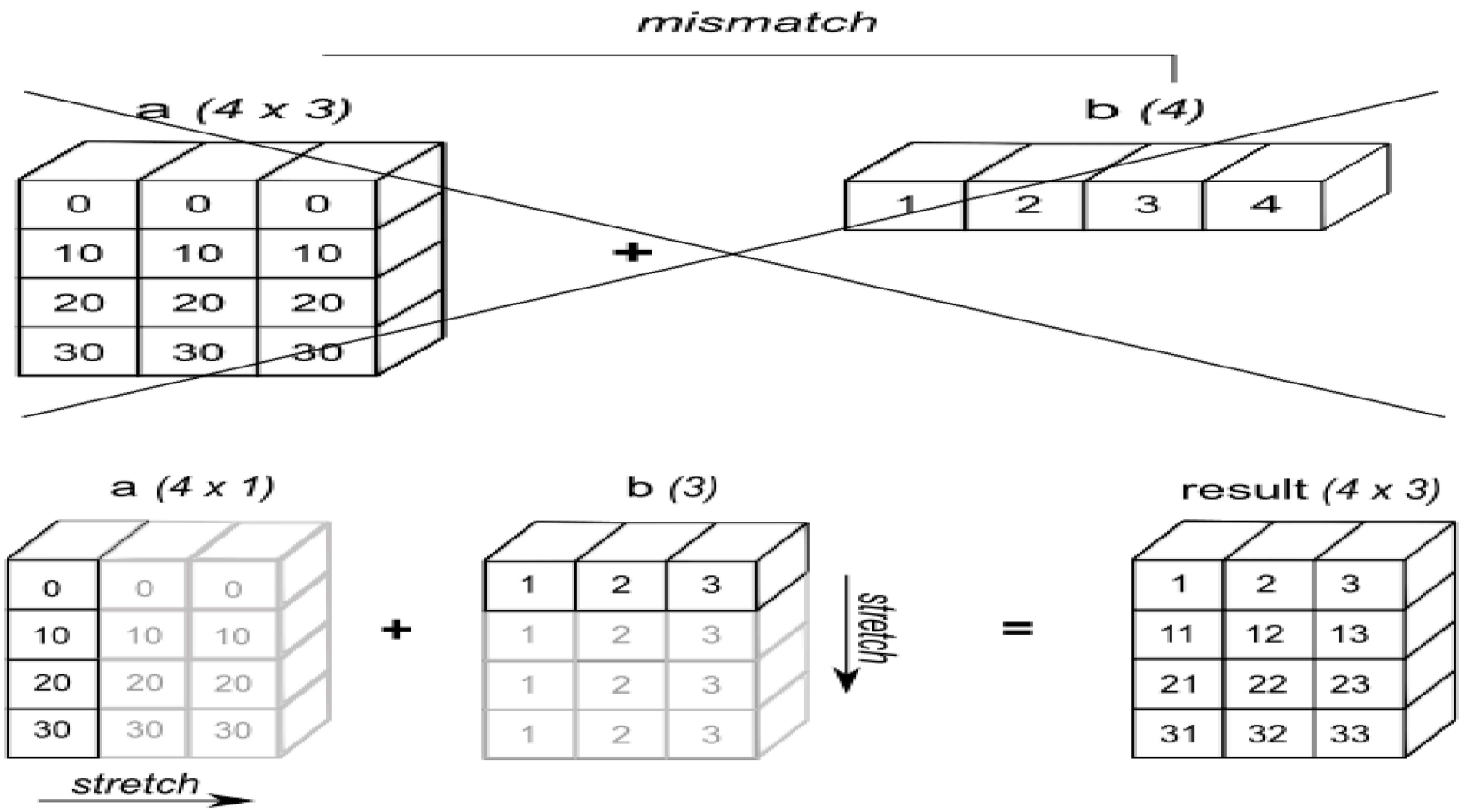
Numpy Broadcasting



Check Point

Take a 3-dimensional array of shape $4 \times 7 \times 1$. Find all possible 2-dimensional and 3-dimensional array shapes which are compatible with the above array.

Numpy Broadcasting



Check Point

Take a 3-dimensional array of shape $4 \times 7 \times 1$. Find all possible 2-dimensional and 3-dimensional array shapes which are compatible with the above array. Answer $1 \times 1 \times k$, $1 \times 7 \times k$, $4 \times 1 \times k$, 1, 1×1, 4×1, 4×7

NumPy Broadcasting

- Two dimensions are compatible(broadcast-able) when they are equal, or one of them is 1.

```
1st      (4d array):  8 x 1 x 6 x 1
2nd      (3d array):      7 x 1 x 5


3rd      (1d array):  7
4th      (1d array):  4

5th      (2d array):  4 x 7
6th      (2d array):  4 x 2

7th      (2d array):  1 x 7
8th      (2d array):  4 x 7
```