FTDS // PYTHON: NUMPY

Hacktiv8 DS
Curriculum
Team

Phase 0
Day 4 AM
2021

Phase 0

2021

Contents

WEEK 1

Python: NumPy

NumPy Introduction

NumPy (Numerical Python) is an open-source Python library that’s used in almost every field of science and engineering. It’s the universal standard for working with numerical data in Python, and it’s at the core of the scientific Python and PyData ecosystems.

The NumPy library contains multidimensional array and matrix data structures. It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it. NumPy can be used to perform a wide variety of mathematical operations on arrays.

WEEK 1

Python: NumPy

Installing NumPy

If you already have Python, you can install NumPy with:

conda install numpy

or

pip install numpy

In order to start using NumPy and all of the functions available in NumPy, you'll need to import it. This can be easily done with this import statement:

import numpy as np

WEEK 1

Python: NumPy

What is an array?

An array is a central data structure of the NumPy library. It's a grid of values and it contains information about the raw data, how to locate an element, and how to interpret an element. It has a grid of elements that can be indexed in various ways.

The rank of the array is the number of dimensions. The shape of the array is a tuple of integers giving the size of the array along each dimension.

a = np.array([[1 , 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

We can access the elements in the array using square brackets. When you're accessing elements, remember that indexing in NumPy starts at 0.

In NumPy, dimensions are called axes. This means that if you have a 2D array that looks like this:
[[0., 0., 0.],

WEEK 1

Python: NumPy

Array & ndarray

You might occasionally hear an array referred to as a “ndarray,” which is shorthand for “N-dimensional array.” An N-dimensional array is simply an array with any number of dimensions. You might also hear 1-D, or one-dimensional array, 2-D, or two-dimensional array, and so on.

The NumPy ndarray class is used to represent both matrices and vectors. A vector is an array with a single column, while a matrix refers to an array with multiple columns.

In NumPy, dimensions are called axes. This means that if you have a 2D array that looks like this:

[[0., 0., 0.],

[1., 1., 1.]]

WEEK 1

Python: NumPy

Creating Array

To create a NumPy array, you can use the function np.array(). For example:

```
import numpy as np
a = np.array([1, 2, 3])
print(a)

# you can easily create an array filled with 0s or 1s:
np.zeros(6)
np.ones(6)

# You can create an array with a range of elements:
print(np.arange(4))
print(np.arange(0,10,2)) # (start, stop, step)
```

WEEK 1

Python: NumPy

List Vs Numpy

Numpy has a faster speed than lists and we can also perform operations directly with numpy. Here is an illustration of the difference between list and numpy:

```
height= [1.73, 1.68, 1.71, 1.86, 1.87]
weight= [65.4, 59.2, 63.2, 88.4, 68.7]

#Body Mass Index (BMI)
weight/height **2   # error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-ec0b55479873> in <module>()
      3
      4 #Body Mass Index (BMI)
----> 5 weight/height **2   # error

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

```
[1] import numpy as np
    height= [1.73, 1.68, 1.71, 1.86, 1.87]
    weight= [65.4, 59.2, 63.2, 88.4, 68.7]
    np_height= np.array(height)
    np_weight= np.array(weight)

    np_weight/np_height **2

    array([21.85171573, 20.97505669, 21.61348791, 25.55208695, 19.64597215])
```

WEEK 1

Python: NumPy

Add, Remove, and Sort

You can add elements to your array any time with np.append() :

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
np.append(arr, [1,2])
```

You can delete an element with np.delete():

```
np.delete(arr, 1)
```

Sorting an element is simple with np.sort():

```
arr = np.array([2, 1, 5, 3, 7, 4, 6, 8])
np.sort(arr)
```

WEEK 1

Python: NumPy

Shape and Size

- **ndarray.ndim will tell you the number of axes, or dimensions, of the array.**
- **ndarray.size will tell you the total number of elements of the array. This is the product of the elements of the array's shape.**
- **ndarray.shape will display a tuple of integers that indicate the number of elements stored along each dimension of the array.**

For example:

```
array_example = np.array([[0, 1, 2, 3],[4, 5, 6, 7]])
print(array_example)
array_example.ndim #output: 2
array_example.size #output: 8
array_example.shape #output: (3, 2, 4)
```

WEEK 1

Python: NumPy

Reshape

Using np.reshape() will give a new shape to an array without changing the data. When you use the reshape method, the array you want to produce needs to have the same number of elements as the original array. For example:

```
a = np.arange(6)
print(a) #[0 1 2 3 4 5]

b = a.reshape(3,2)
print(b)
#output:
[[0 1]
 [2 3]
 [4 5]]
```

WEEK 1

Python: NumPy

Convert 1D to 2D

Using np.newaxis will increase the dimensions of your array by one dimension when used once. This means that a 1D array will become a 2D array, a 2D array will become a 3D array, and so on. For example, if you start with this array:

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

# You can use np.newaxis to add a new axis:
a2 = a[np.newaxis]
print(a2.shape)
print(a2)
```

```
for n in x:
    print(n)
```

WEEK 1

Python: NumPy

Convert 1D to 2D

You can convert a 1D array to a row vector by inserting an axis along the first dimension:

```
row_vector = a[np.newaxis, :]
print(row_vector.shape)
print(row_vector)
```

for a column vector, You can insert an axis along the second dimension:

```
col_vector = a[:, np.newaxis]
print(col_vector.shape)
print(col_vector)
```

WEEK 1

Python: NumPy

Convert 1D to 2D

You can also expand an array by inserting a new axis at a specified position with np.expand_dims. For example:

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

# You can use np.expand_dims to add an axis at index position 1 with:
b = np.expand_dims(a, axis=1)
b.shape

# You can add an axis at index position 0 with:
c = np.expand_dims(a, axis=0)
c.shape
```

WEEK 1

Python: NumPy

Indexing and Slicing

You can index and slice NumPy arrays in the same ways you can slice Python lists. For example:

```
data = np.array([1,2,3])
print(data)
print(data[0])
print(data[1])
print(data[0:2])
print(data[1:])
print(data[-2:])
a = np.array([[1 , 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(a[a>=5])
```

WEEK 1

Python: NumPy

Basic array operations

Basic operations are simple with NumPy. If you want to find the sum of the elements in an array, you'd use sum(). This works for 1D arrays, 2D arrays, and arrays in higher dimensions:

```
a = np.array([1, 2, 3, 4])
# sum all of the elements in the array
a.sum()
b = np.array([[1, 1], [2, 2]])

# You can sum the rows
b.sum(axis=0) #output: array([3, 3])

# You can sum the columns
b.sum(axis=1) #array([2, 4])
```

WEEK 1

Python: NumPy

More Array Operations

NumPy also performs aggregation functions. In addition to min, max, and sum, you can easily run mean to get the average, prod to get the result of multiplying the elements together, std to get the standard deviation, and more. For example:

```
A = np.array([[0.45053314, 0.17296777, 0.34376245, 0.5510652],
              [0.54627315, 0.05093587, 0.40067661, 0.55645993],
              [0.12697628, 0.82485143, 0.26590556, 0.56917101]])

print(A)
A.sum()
A.min()
A.min(axis=0)
A.max()
A.max(axis=1)
A.std()
```

WEEK 1

Python: NumPy

Transposing and Reshaping

A common need when dealing with matrices is the need to rotate them. This is often the case when we need to take the dot product of two matrices and need to align the dimension they share. NumPy arrays have a convenient property called T to get the transpose of a matrix. For example:

```
data_col = np.array([[1, 2, 3, 4, 5, 6]]).T
print(data_col)
data_col.reshape(2, 3)
```

WEEK 1

Python: NumPy

Flatten N-Dimensional Array

There are two popular ways to flatten an array: .flatten() and .ravel(). The primary difference between the two is that the new array created using ravel() is actually a reference to the parent array. This means that any changes to the new array will affect the parent array as well. Since ravel does not create a copy, it's memory efficient. For example:

```
arrflat = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(arrflat)

# You can use flatten to flatten your array into a 1D array.
arrflat.flatten()
```

Hacktiv8 DS
Curriculum
Team

External References

Colab Link

Visit Here