What is NumPy in Python?

NumPy is an open source library available in Python, which helps in mathematical, scientific, engineering, and data science programming. It is a very useful library to perform mathematical and statistical operations in Python. It works perfectly for multi-dimensional arrays and matrix multiplication. It is easy to integrate with C/C++ and Fortran.

For any scientific project, NumPy is the tool to know. It has been built to work with the N-dimensional array, linear algebra, random number, Fourier transform, etc.

NumPy is a programming language that deals with multi-dimensional arrays and matrices. On top of the arrays and matrices, NumPy supports a large number of mathematical operations. In this part, we will review the essential functions that you need to know for the tutorial on TensorFlow'

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray , it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Which Language is NumPy written in?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Where is the NumPy Codebase?

The source code for NumPy is located at this github repository https://github.com/numpy/numpy

github: enables many people to work on the same codebase.

How to Install NumPy

To install NumPy library, please refer our tutorial How to install TensorFlow. NumPy is installed by default with Anaconda.

In remote case, NumPy not installed-

C:\Users*Your Name*>pip install numpy

Import NumPy and Check Version

The command to import numpy is:

```
import numpy as np
```

What is Python NumPy Array?

NumPy arrays are a bit like Python lists, but still very much different at the same time. For those of you who are new to the topic, let's clarify what it exactly is and what it's good for.

As the name kind of gives away, a NumPy array is a central data structure of the numpy library. The library's name is actually short for "Numeric Python" or "Numerical Python".

Creating a NumPy Array

Simplest way to create an array in Numpy is to use Python List

```
myPythonList = [1,9,8,3]
```

To convert python list to a numpy array by using the object np.array.

```
numpy_array_from_list = np.array(myPythonList)
```

To display the contents of the list

```
numpy_array_from_list
```

Output:

```
array([1, 9, 8, 3])
```

In practice, there is no need to declare a Python List. The operation can be combined.

```
a  = np.array([1,9,8,3])
```

NOTE: Numpy documentation states use of np.ndarray to create an array. However, this the recommended method.

You can also create a numpy array from a Tuple.

Mathematical Operations on an Array

You could perform mathematical operations like additions, subtraction, division and multiplication on an array. The syntax is the array name followed by operation (+.-,*,/) followed by the operand

Example:

```
numpy_array_from_list + 10
```

Output:

```
array([11, 19, 18, 13])
```

This operation adds 10 to each element of the numpy array.

Shape of Array

You can check the shape of the array with the object shape preceded by the name of the array. In the same way, you can check the type with dtypes.

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

(3,)
int64
```

An integer is a value without decimal. If you create an array with decimal, then the type will change to float.

```
#### Different type
b  = np.array([1.1,2.0,3.2])
print(b.dtype)

float64
```

2 Dimension Array

You can add a dimension with a ","coma

Note that it has to be within the bracket []

```
### 2 dimension
c = np.array([(1,2,3),
              (4,5,6)])
print(c.shape)
(2, 3)
```

3 Dimension Array

Higher dimension can be constructed as follow:

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

What is numpy.zeros()?

numpy.zeros() or np.zeros Python function is used to create a matrix full of zeroes. numpy.zeros() in Python can be used when you initialize the weights during the first iteration in TensorFlow and other statistic tasks.

numpy.zeros() function Syntax

```
numpy.zeros(shape, dtype=float, order='C')
```

Python numpy.zeros() Parameters

Here,

- **Shape**: is the shape of the numpy zero array
- **Dtype**: is the datatype in numpy zeros. It is optional. The default value is float64
- **Order**: Default is C which is an essential row style for numpy.zeros() in Python.

Python numpy.zeros() Example

```
import numpy as np
np.zeros((2,2))
```

Output:

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

Example of numpy zero with Datatype

```
import numpy as np
np.zeros((2,2), dtype=np.int16)
```

Output:

```
array([[0, 0],
       [0, 0]], dtype=int16)
```

numpy.reshape() function in Python

Python NumPy Reshape function is used to shape an array without changing its data. In some occasions, you may need to reshape the data from wide to long. You can use the np.reshape function for this.

Syntax of np.reshape()

```
numpy.reshape(a, newShape, order='C')
```

Here,

a: Array that you want to reshape

newShape: The new desires shape

Order: Default is C which is an essential row style.

Example of NumPy Reshape

```
import numpy as np
e  = np.array([(1,2,3), (4,5,6)])
print(e)
e.reshape(3,2)
```

Output:

```
 // Before reshape
[[1 2 3]
 [4 5 6]]
```

```
//After Reshape
array([[1, 2],
        [3, 4],
        [5, 6]])
```

numpy.flatten() in Python

Python NumPy Flatten function is used to return a copy of the array in one-dimension. When you deal with some neural network like convnet, you need to flatten the array. You can use the np.flatten() functions for this.

Syntax of np.flatten()

```
numpy.flatten(order='C')
```

Here,
Order: Default is C which is an essential row style.

Example of NumPy Flatten

```
e.flatten()
```

Output:

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

Generate Random Numbers using NumPy

To generate random numbers for Gaussian distribution, use:

```
numpy.random.normal(loc, scale, size)
```

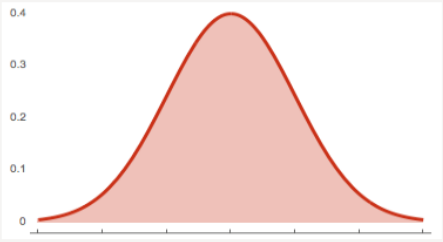
Here,

- **Loc**: the mean. The center of distribution
- **Scale**: standard deviation.
- **Size**: number of returns

Example:

```
## Generate random nmber from normal distribution
normal_array = np.random.normal(5, 0.5, 10)
print(normal_array)
[5.56171852 4.84233558 4.65392767 4.946659   4.85165567 5.61211317 4.46704244 5.22675736 4.49888936 4
```

If plotted the distribution will be similar to following plot



Example to Generate Random Numbers using NumPy

What is numpy.arange()?

numpy.arange() is an inbuilt numpy function that returns an ndarray object containing evenly spaced values within a

Syntax:

```
numpy.arange(start, stop, step, dtype)
```

Python NumPy arange Parameters:

- **Start**: Start of interval for np.arange in Python function.
- **Stop**: End of interval.
- **Step**: Spacing between values. Default step is 1.
- **Dtype**: Is a type of array output for NumPy arange in Python.

Example:

```
import numpy np
np.arange(1, 11)
```

Output:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Example:

If you want to change the step in this NumPy arange function in Python example, you can add a third number in the pa

```
import numpy np
np.arange(1, 14, 4)
```

Output:

```
array([ 1,  5,  9, 13])
```

NumPy Linspace Function

Linspace gives evenly spaced samples.

Syntax:

```
numpy.linspace(start, stop, num, endpoint)
```

Here,

- **Start:** Starting value of the sequence
- **Stop:** End value of the sequence
- **Num:** Number of samples to generate. Default is 50
- **Endpoint:** If True (default), stop is the last value. If False, stop value is not included.

Example:

For instance, it can be used to create 10 values from 1 to 5 evenly spaced.

```
import numpy as np
np.linspace(1.0, 5.0, num=10)
```

Output:

```
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,       3.22222222, 3.66666667, 4.11111111, 4.5555555
```

If you do not want to include the last digit in the interval, you can set endpoint to false

```
np.linspace(1.0, 5.0, num=5, endpoint=False)
```

Output:

```
array([1. , 1.8, 2.6, 3.4, 4.2])
```

LogSpace NumPy Function in Python

LogSpace returns even spaced numbers on a log scale. Logspace has the same parameters as np.linspace.

Syntax:

```
numpy.logspace(start, stop, num, endpoint)
```

Example:

```
np.logspace(3.0, 4.0, num=4)
```

Output:

```
array([ 1000. ,  2154.43469003,  4641.58883361, 10000.        ])
```

Finaly, if you want to check the memory size of an element in an array, you can use itemsize

```
x = np.array([1,2,3], dtype=np.complex128)
x.itemsize
```

Output:

```
16
```

Each element takes 16 bytes.

Numpy 2

NumPy Array Indexing

Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np

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

print(arr[0])
```

Example

Get the second element from the following array.

```
import numpy as np

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

print(arr[1])
```

Example

Get third and fourth elements from the following array and add them.

```
import numpy as np

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

print(arr[2] + arr[3])
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

Example

Access the element on the first row, second column:

```
import numpy as np
```

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

print('2nd element on 1st row: ', arr[0, 1])
```

Example

Access the element on the 2nd row, 5th column:

```
import numpy as np

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

print('5th element on 2nd row: ', arr[1, 4])
```

Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example

Access the third element of the second array of the first array:

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

print(arr[0, 1, 2])
```

Negative Indexing

Use negative indexing to access an array from the end.

Example

Print the last element from the 2nd dim:

```
import numpy as np

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

print('Last element from 2nd dim: ', arr[1, -1])

NumPy Array Slicing

Slicing arrays

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this:

```
[start:end]
```

.

We can also define the step, like this:

```
[start:end:step]
```

.

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

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

print(arr[1:5])
```

Note: The result *includes* the start index, but *excludes* the end index.

Example

Slice elements from index 4 to the end of the array:

```
import numpy as np

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

print(arr[4:])
```

Example

Slice elements from the beginning to index 4 (not included):

```
import numpy as np

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

print(arr[:4])
```

Negative Slicing

Use the minus operator to refer to an index from the end:

Example

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np

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

print(arr[-3:-1])
```

STEP

Use the

`step`

value to determine the step of the slicing:

Example

Return every other element from index 1 to index 5:

```
import numpy as np

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

print(arr[1:5:2])
```

Example

Return every other element from the entire array:

```
import numpy as np

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

print(arr[::2])
```

Slicing 2-D Arrays

Example

From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np

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

print(arr[1, 1:4])
```

Note: Remember that *second element* has index 1.

Example

From both elements, return index 2:

```
import numpy as np

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

```
print(arr[0:2, 2])
```

Example

From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:

```
import numpy as np

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

print(arr[0:2, 1:4])
```

NumPy Data Types

Data Types in Python

By default Python have these data types:

- strings

  - used to represent text data, the text is given under quote marks. e.g. "ABCD"

- integer

  - used to represent integer numbers. e.g. -1, -2, -3

- float

  - used to represent real numbers. e.g. 1.2, 42.42

- boolean

  - used to represent True or False.

- complex

  - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

Checking the Data Type of an Array

The NumPy array object has a property called

`dtype`

that returns the data type of the array:

Example

Get the data type of an array object:

```
import numpy as np

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

print(arr.dtype)
```

Example

Get the data type of an array containing strings:

```
import numpy as np

arr = np.array(['apple', 'banana', 'cherry'])

print(arr.dtype)
```

NumPy Sorting Arrays

Sorting Arrays

Sorting means putting elements in an *ordered sequence*.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called

```
sort()
```

, that will sort a specified array.

Example

Sort the array:

```
import numpy as np

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

print(np.sort(arr))
```

Note: This method returns a copy of the array, leaving the original array unchanged.

You can also sort arrays of strings, or any other data type:

Example

Sort the array alphabetically:

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

Example:

```
import numpy as np
normal_array = np.random.normal(5, 0.5, 10)
print(normal_array)
```

Output:

```
[5.56171852 4.84233558 4.65392767 4.946659   4.85165567 5.61211317 4.46704244 5.22675736 4.49888936 4.68731125]
```

Example of NumPy Statistical function

```
### Min
print(np.min(normal_array))

### Max
print(np.max(normal_array))

### Mean
print(np.mean(normal_array))

### Median
print(np.median(normal_array))

### Sd
print(np.std(normal_array))
```

Output:

```
4.467042435266913
5.612113171990201
4.934841002270593
4.846995625786663
0.3875019367395316
```

Statistical Functions in Python

NumPy has quite a few useful statistical functions for finding minimum, maximum, percentile standard deviation and variance, etc from the given elements in the array. The functions are explained as follows −

Numpy is equipped with the robust statistical function as listed below

Function

Min
Max
Mean
Median
Standard deviation
Consider the following Array: