**Numpy 1**

**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++](https://www.guru99.com/cpp-tutorial.html) 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](https://www.guru99.com/tensorflow-tutorial.html)’

**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](https://www.guru99.com/download-install-tensorflow.html). 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

Above code renames the Numpy namespace to np. This permits us to prefix Numpy function, methods, and attributes with ” np ” instead of typing ” numpy.” It is the standard shortcut you will find in the numpy literature

**alias:** In Python alias are an alternate name for referring to the same thing.

To check your installed version of NumPy, use the below command:

print (np.\_\_version\_\_)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**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](https://www.guru99.com/python-list-comprehension-sort-examples.html)

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 the 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)

| **Objective** | **Code** |
| --- | --- |
| Create array | array([1,2,3]) |
| print the shape | array([.]).shape |

**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)

**What is numpy.ones()?**

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

**Python numpy.ones() Syntax**

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

**Python numpy.ones() Parameters**

Here,

* **Shape**: is the shape of the np.ones [Python Array](https://www.guru99.com/python-arrays.html)
* **Dtype**: is the datatype in numpy ones. It is optional. The default value is float64
* **Order**: Default is C which is an essential row style.

**Python numpy.ones() 2D Array with Datatype Example**

import numpy as np

np.ones((1,2,3), dtype=np.int16)

**Output:**

array([[[1, 1, 1],

[1, 1, 1]]], 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:**

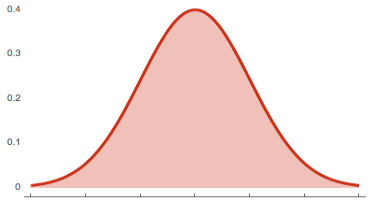
## Generate random nmber from normal distribution

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 4.68731125]

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 defined interval. For instance, you want to create values from 1 to 10; you can use np.arange() in Python function.

**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 parenthesis. It will change the step.

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, 4.55555556, 5. ])

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.