20190802107 **DNN-LAB1**

Name: Sia Vashist

LAB EXPERIMENT- 01

- **AIM: Introduction to various libraries required in DNN.**

1. NUMPY

Python's primary library for scientific computing is called Numpy. It offers a multidimensional array object with outstanding speed as well as capabilities for interacting with these arrays.

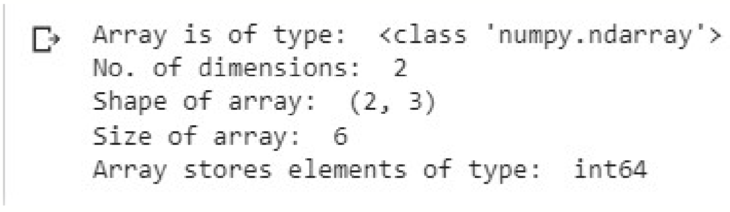
The array object in NumPy is known as ndarray, and it has lots of supporting functions that make using ndarray very simple.

Arrays

A tuple of nonnegative integers serves as the index for a numpy array, which is a grid of identically typed items. The rank of an array is determined by the number of dimensions; an array's shape is a tuple of numbers indicating the size of the array along each axis.

- basic array characteristics

```
import numpy as np
# Creating array object
arr = np.array( [[ 2, 4, 6],
        [ 4, 16, 64]] )
# Printing type of arr object
print("Array is of type: ", type(arr))
# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)
# Printing shape of array
print("Shape of array: ", arr.shape)
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

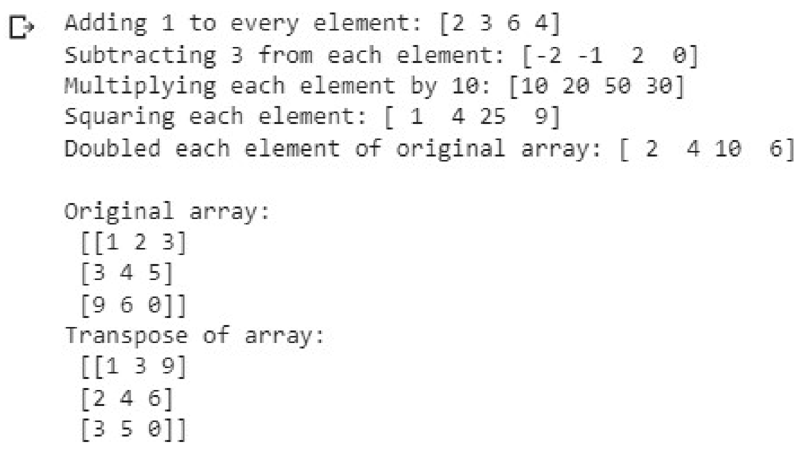


```
Array is of type:  <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
```

20190802107 **DNN-LAB1**

- basic operations on single array

```
import numpy as np
a = np.array([1, 2, 5, 3])

# add 1 to every element
print ("Adding 1 to every element:", a+1)

# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)

# multiply each element by 10
print ("Multiplying each element by 10:", a*10)

# square each element
print ("Squaring each element:", a**2)

# modify existing array
a *= 2
print ("Doubled each element of original array:", a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print ("\nOriginal array:\n", a)
print ("Transpose of array:\n", a.T)
```



```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
 [[1 2 3]
 [3 4 5]
 [9 6 0]]
Transpose of array:
 [[1 3 9]
 [2 4 6]
 [3 5 0]]
```

- **Applications:**
  Probability & Statistics (numpy.random.hypergeometric)

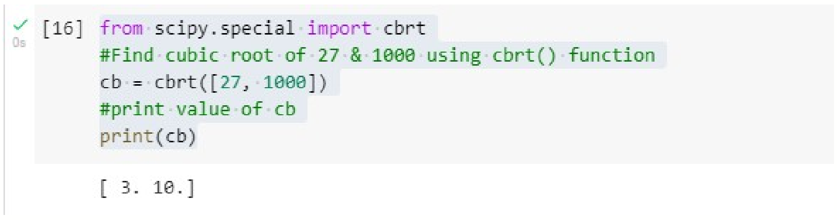
  A sample is taken from a hypergeometric distribution using the numpy.random.hypergeometric(ngood, nbad, nsample, size=None) function. The binomial distribution, which is applicable when choosing from a finite population with replacement or an infinite population without replacement, is first understood in order to comprehend hypergeometric distribution, which is employed when choosing from a finite population without replacement.

2. SCIPY

SciPy is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines, such as routines for numerical integration and optimization.

- Cubic Root using SciPy

```
from scipy.special import cbrt
#Find cubic root of 27 & 1000 using cbrt() function
cb = cbrt([27, 1000])
#print value of cb
print(cb)
```

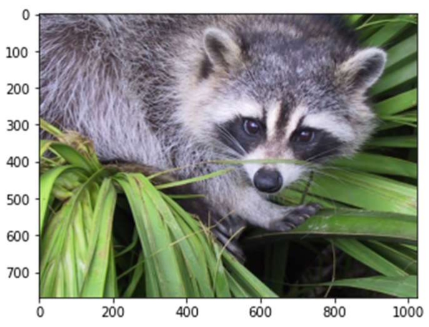


```
[16] from scipy.special import cbrt
     #Find cubic root of 27 & 1000 using cbrt() function
     cb = cbrt([27, 1000])
     #print value of cb
     print(cb)

     [ 3. 10.]
```

- Image Processing with SciPy – scipy.ndimage

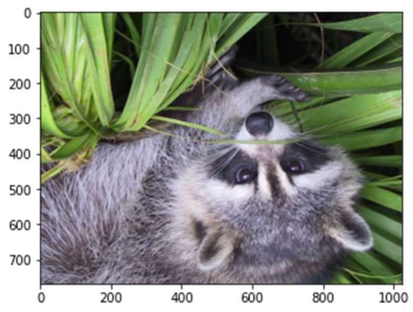
▪ SciPy Image Processing provides Geometrics transformation (rotate, crop, flip), image filtering (sharp and de nosing), display image, image segmentation, classification and features extraction.
▪ **MISC Package** in SciPy contains prebuilt images which can be used to perform image manipulation task

```
from scipy import misc
from matplotlib import pyplot as plt
import numpy as np
#get face image of panda from misc package
panda = misc.face()
#plot or show image of face
plt.imshow( panda )
plt.show()
```



Flipping down the image:

```
#Flip Down using scipy misc.face image
flip_down = np.flipud(misc.face())
plt.imshow(flip_down)
plt.show()
```



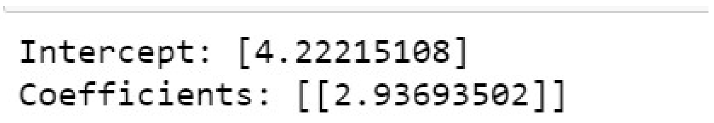
3. *Scikit-learn (Sklearn)*

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

- **Python code using scikit-learn (sklearn) to fit a linear regression model:**

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

# Generate sample data
np.random.seed(0)
X = np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Fit a linear regression model to the data
reg = LinearRegression().fit(X, y)

# Print the intercept and coefficients
print("Intercept:", reg.intercept_)
print("Coefficients:", reg.coef_)
```

- **OUTPUT:**



```
Intercept: [4.22215108]
Coefficients: [[2.93693502]]
```

- **APPLICATIONS:**

Scikit-learn (Sklearn) is a popular machine learning library in Python and has a wide range of applications in different fields such as:

  - Regression,
  - Classification,
  - Clustering,
  - Dimensionality Reduction,
  - Model Selection,
  - Preprocessing

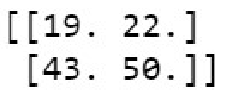
4. THEANO:

Theano is a Python library for fast numerical computation that was primarily developed for deep learning. Theano is known for its ability to optimize and accelerate numerical computations, making it ideal for deep learning tasks.

- **Python code using Theano to perform matrix multiplication:**

```
import theano
import theano.tensor as T

# Define two matrices
x = T.matrix("x")
y = T.matrix("y")

# Define the dot product of the two matrices
z = T.dot(x, y)

# Compile the function
f = theano.function([x, y], z)

# Generate some test data
a = [[1, 2], [3, 4]]
b = [[5, 6], [7, 8]]

# Call the function
c = f(a, b)

# Print the result
print(c)
```

- **OUTPUT**



```
[[19. 22.]
 [43. 50.]]
```

- **Applications of Theano:**
  - Deep Learning: Theano is widely used for building and training deep neural networks, as it can efficiently perform the numerical computations required for training deep models.
  - Scientific Computing: Theano can be used for a wide range of scientific computing tasks, including numerical optimization, linear algebra, and signal processing.
  - High-Performance Computing: Theano has the ability to run computations on GPUs, making it ideal for high-performance computing tasks.

5. *TENSORFLOW:*

TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is used for machine learning and deep learning applications.

- **Python code using Tensorflow for a simple calculation:**

```
import tensorflow as tf

# Define two constant tensors
a = tf.constant(6, name="constant_a")
b = tf.constant(3, name="constant_b")

# Define a tensor for the sum of two constants
c = tf.add(a, b, name="add_c")

# Create a session to run the computation
with tf.Session() as sess:
    # Run the computation and print the result
    result = sess.run(c)
    print("The result of 6 + 3 is: ", result)
```

- **Application of TensorFlow:**
  - Image classification
  - Object detection
  - Natural language processing (NLP)
  - Speech recognition
  - Recommender systems
  - Generative models, etc.

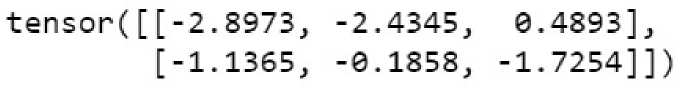
6. *PyTorch:*

PyTorch is a popular open-source machine learning library used for developing and training deep learning models. It is based on the Torch library and provides functionality for tensor computations with strong GPU acceleration.

- **Python code that that creates a tensor and performs a simple operation: using PyTorch:**

```
import torch

# Create a tensor with shape (2, 3) and random values
x = torch.randn((2, 3))

# Multiply all elements in the tensor by 2
y = 2 * x

print(y)
```

- **OUTPUT**



```
tensor([[-2.8973, -2.4345,  0.4893],
        [-1.1365, -0.1858, -1.7254]])
```

- **Applications of PyTorch:**
  - Computer Vision: Object detection, image classification, semantic segmentation, etc.
  - Natural Language Processing: Text classification, language translation, text generation, etc.
  - Generative Models: Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), etc.
  - Reinforcement Learning: Training agents for games, robotics, etc.
  - Time-series forecasting, etc.

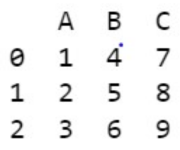
7. *Pandas:*

Pandas is a popular open-source library for data analysis in Python, it is not specifically required for Deep Neural Network models, but it is often used in the preparation and preprocessing of the data before it is used to train a neural network.

- **Python code using Pandas:**

```
import pandas as pd

# Creating a sample dataframe
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]})

# Displaying the dataframe
print(df)
```

- Output:



```
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9
```

- **Applications of Pandas:**
  - Data cleaning and preprocessing.
  - Data manipulation and transformation.
  - Data analysis and exploration.
  - Data visualization.
  - Data aggregation and summarization.

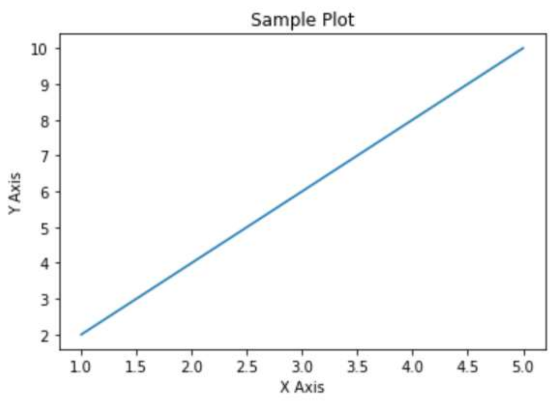
8. *Matplotlib:*

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It is not required for Deep Neural Networks, but it is often used for visualizing and plotting data and the results of training a neural network.

- **Python code for using MATPLOTLIB:**

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Plotting the data
plt.plot(x, y)

# Adding labels and title to the plot
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Sample Plot')

# Show the plot
plt.show()
```

- **OUTPUT:**



- **APPLICATIONS:**

  - Data visualization and plotting of 1D and 2D data.
  - Creation of bar charts, line charts, scatter plots, histograms, and many other types of visualizations.
  - Plotting and visualization of the results of training deep neural network models.
  - Customization of the plots with labels, titles, legends, and other annotations.
  - Generation of plots for publication quality reports and scientific papers.

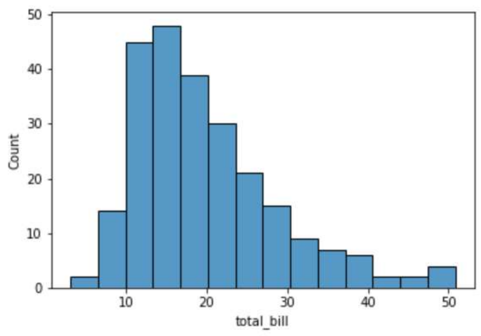
9. *Seaborn:*

Seaborn is a data visualization library for the Python programming language, built on top of Matplotlib. It is not required for Deep Neural Networks, but it is often used for data visualization and plotting. Seaborn provides more advanced and visually appealing statistical graphics compared to Matplotlib.

- **Python code for using SEABORN:**

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Plotting the data using seaborn
sns.lineplot(x, y)

# Adding labels and title to the plot
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Sample Plot')

# Show the plot
plt.show()
```

- **OUTPUT:**



- **APPLICATIONS:**
  - Data visualization and plotting of 1D and 2D data.
  - Creation of advanced and visually appealing statistical graphics such as heatmaps, violin plots, and pair plots.
  - Plotting and visualization of the results of training deep neural network models.
  - Generation of plots for publication quality reports and scientific papers.
  - Easily and quickly creating plots with built-in themes and color palettes.