

Fundamentals of Machine Learning

Lab Assignment-7

Name: Hitha Choudhary G

USN: 22BTRAD015

Branch: CSE in AI&DE

Implementing Numpy.

- A Python package called NumPy is used to work with arrays.
- It also includes functions for working with matrices, the Fourier transform, and linear algebra.
- Multidimensional array and matrix data structures are available in the NumPy library (more details on these will be provided in later sections).
- It offers techniques for effectively working with ndarray, a homogenous n-dimensional array object. Numerous mathematical operations on arrays can be carried out with NumPy.
- It provides an extensive library of high-level mathematical functions that operate on these arrays and matrices, and it enhances Python with strong data structures that ensure effective calculations with arrays and matrices.

Creating a Numpy array: We import numpy library as np. To create a numpy array we use the function “np.array” and print the array.

Code:

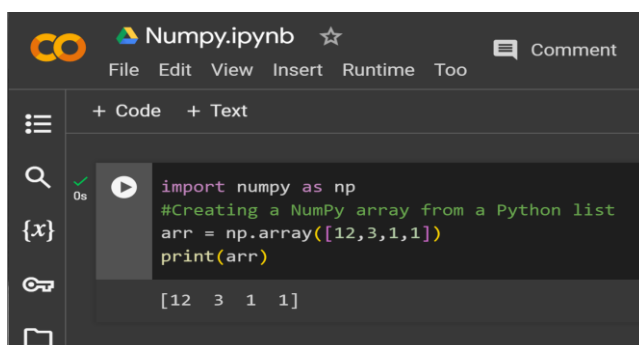
```
import numpy as np

#Creating a NumPy array from a Python list

arr = np.array([12,3,1,1])

print(arr)
```

Output:

A screenshot of a Jupyter Notebook interface. The title bar shows 'Numpy.ipynb' with a star icon and a 'Comment' button. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', and 'Tools'. The main area has a sidebar on the left with icons for file operations and a central code editor. The code editor contains the following Python code:

```
import numpy as np
#Creating a NumPy array from a Python list
arr = np.array([12,3,1,1])
print(arr)
```

The output of the code is displayed below the code cell as a 1D array:

```
[12  3  1  1]
```

Operations: Operations in numpy are known as element wise operations. Some of the operations include addition, multiplication of a matrix, division and many more.

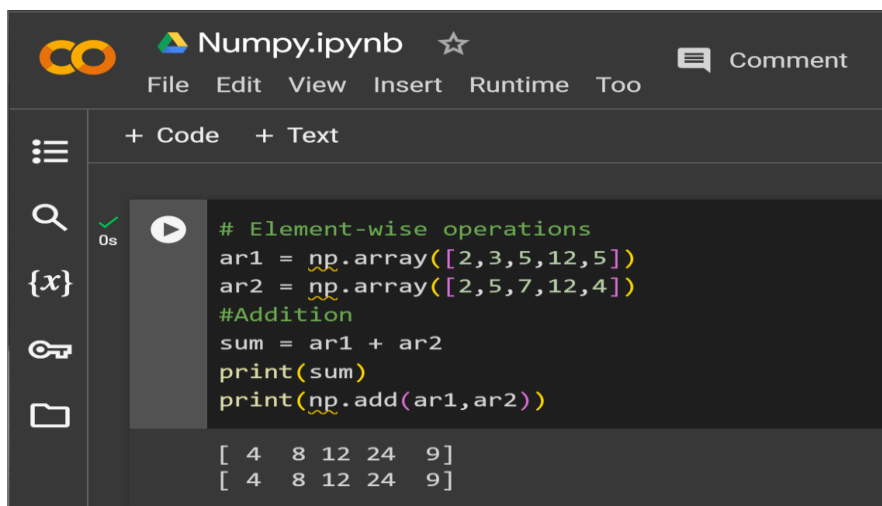
In the below example we will be taking two arrays and performing addition using “+” where we can get the sum of both the arrays.

Using numpy we can do the sum of both the arrays using “np.add()”.

Code:

```
# Element-wise operations
ar1 = np.array([2,3,5,12,5])
ar2 = np.array([2,5,7,12,4])
#Addition
sum = ar1 + ar2
print(sum)
print(np.add(ar1,ar2))
```

Output:



```
# Element-wise operations
ar1 = np.array([2,3,5,12,5])
ar2 = np.array([2,5,7,12,4])
#Addition
sum = ar1 + ar2
print(sum)
print(np.add(ar1,ar2))
```

```
[ 4  8 12 24  9]
[ 4  8 12 24  9]
```

Linear Algebra operations in numpy: NumPy's Linear Algebra module offers several methods for using linear algebra on any numpy array. Matrix rank, determinant, trace, and eigenvalues can all be found. Additionally, we can obtain matrix exponentiation, vector products of matrices, etc.

Using the numpy linear algebra module, we can also solve tensor or linear equations.

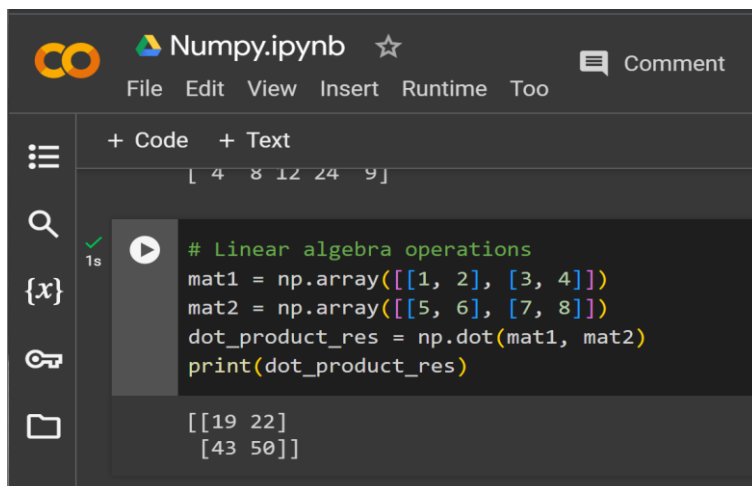
We can find the dot product of the matrix by using "np.dot()". We are considering two matrix as mat1 and mat2 in the below example to find the dot product between them.

Code:

```
# Linear algebra operations

mat1 = np.array([[1, 2], [3, 4]])
mat2 = np.array([[5, 6], [7, 8]])
dot_product_res = np.dot(mat1, mat2)
print(dot_product_res)
```

Output:

A screenshot of a Jupyter Notebook interface. The top bar shows the 'Numpy.ipynb' title and a star icon. Below it is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', and 'Tools'. On the left is a sidebar with icons for a menu, search, variables, keyboard shortcuts, and a file explorer. The main area has a tab labeled '+ Code' and '+ Text'. The code cell contains the following Python code:

```
# Linear algebra operations
mat1 = np.array([[1, 2], [3, 4]])
mat2 = np.array([[5, 6], [7, 8]])
dot_product_res = np.dot(mat1, mat2)
print(dot_product_res)
```

The output of the code is displayed below the code cell:

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

Mathematical operations using numpy: Numerous diverse mathematical procedures are included in NumPy. It offers common trigonometric functions as well as arithmetic operation and complex number handling functions.

We will use the "np.sin()" function, which accepts an array as an argument, to output the sine values of the supplied array. Similarly, we will use the "np.tan()" function, which accepts an array as an input, to display the tan values of the provided array.

Code:

```
array=np.array([12,4,6])  
  
# Mathematical functions  
  
sin = np.sin(array)  
  
tan = np.tan(array)  
  
print("Sin values:")  
  
print(sin)  
  
print("Tan values:")  
  
print(tan)
```

Output:

```
array=np.array([12,4,6])  
# Mathematical functions  
sin = np.sin(array)  
tan = np.tan(array)  
print("Sin values:")  
print(sin)  
print("Tan values:")  
print(tan)  
  
Sin values:  
[-0.53657292 -0.7568025 -0.2794155 ]  
Tan values:  
[-0.63585993  1.15782128 -0.29100619]
```

Numpy indexing and slicing:

Numpy indexing is a technique for locating a specific element within an array. The index values of the array's elements range from 0 to one less than the array's length. To obtain a specific element located at a given index, we shall utilize numpy for indexing.

Extracting elements from an array inside a specified range is known as **Numpy Slicing** in a Numpy array. It publishes an array's elements starting at one supplied index and going all the way to another. The elements that are present from a to b-1 are printed if we specify the range as a to b. For instance, it prints the components present from 3 to 5 if we set the range to 3 to 6.

Code:

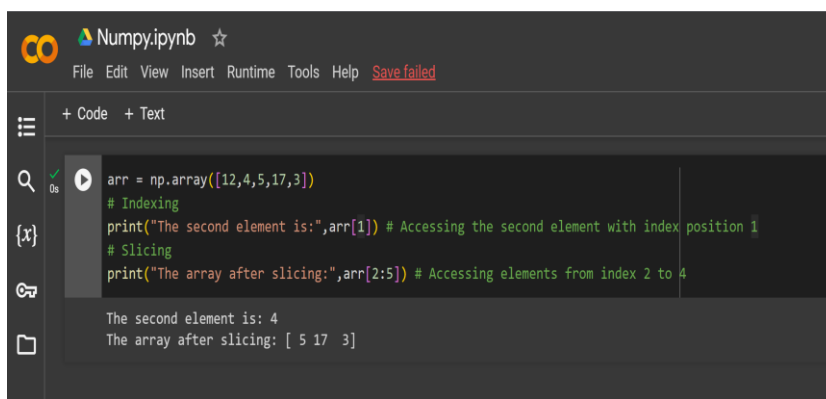
```
arr = np.array([12,4,5,17,3])
```

```
# Indexing
```

```
print("The second element is:",arr[1]) # Accessing the second element with  
index position 1
```

```
# Slicing
```

```
print("The array after slicing:",arr[2:5]) # Accessing elements from index 2 to 4
```

Output:

```
arr = np.array([12,4,5,17,3])  
# Indexing  
print("The second element is:",arr[1]) # Accessing the second element with index position 1  
# Slicing  
print("The array after slicing:",arr[2:5]) # Accessing elements from index 2 to 4  
  
The second element is: 4  
The array after slicing: [ 5 17  3]
```

Array shape and reshaping of an array: We can get the number of rows and columns in array using “.shape” and we can reshape the dimensions of an array by using “.reshape”.

Code:

```
ar1 = np.array([12,4,5,17,3])
```

```
# Shape of an array
```

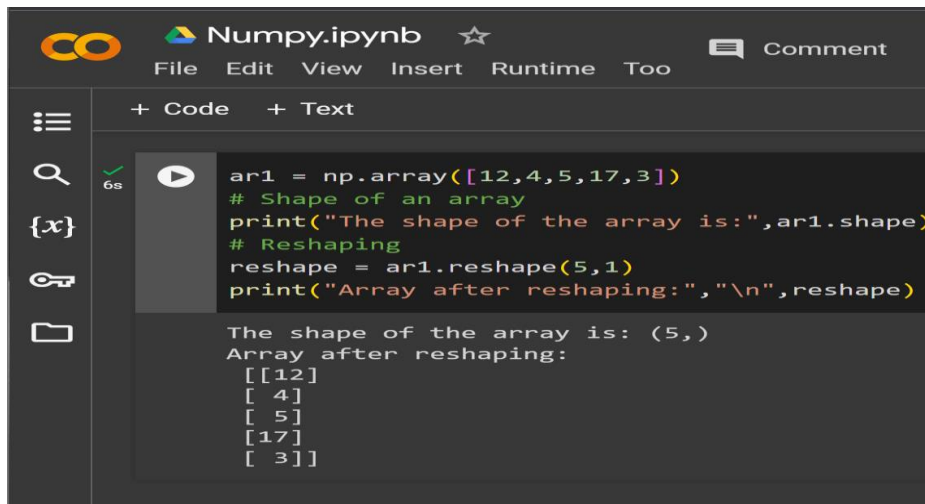
```
print("The shape of the array is:",ar1.shape)
```

```
# Reshaping
```

```
reshape = ar1.reshape(5,1)
```

```
print("Array after reshaping:", "\n", reshape)
```

Output:



```
ar1 = np.array([12,4,5,17,3])
# Shape of an array
print("The shape of the array is:",ar1.shape)
# Reshaping
reshape = ar1.reshape(5,1)
print("Array after reshaping:", "\n", reshape)
```

The shape of the array is: (5,)
Array after reshaping:
[[12]
[4]
[5]
[17]
[3]]

These are all the different operations using numpy.

Github Link: <https://github.com/hithachoudhary/machinelearning>