NumPy

22-02-2024

By Ms. Anam Suri, NET-JRF

Introduction

- Stands for Numerical Python
- A python library used for working with arrays
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- It is an open source project and you can use it freely.
  - pip install numpy / conda install numpy

Why 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.
- NumPy is faster than lists:
  - NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.(also k/a locality of reference)
  - problem with python's list is its efficiency
  - Due to the way a python list is implemented, accessing items in a large list is computationally expensive
- To solve this issue, **NumPy comes into picture**

Cont…

- Overall, the memory layout of NumPy arrays enables efficient data access and manipulation, making them well-suited for numerical computations, scientific computing, and data analysis tasks where performance is critical.
- This efficiency is one of the reasons why NumPy is widely used in fields like machine learning, image processing, and computational physics.
- **NumPy** adds support for large, multidimensional arrays & matrices
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- All the elements of the array must be of the same type

Importing NumPy

- Once NumPy is installed, import it in your applications by adding the import keyword:
  - import numpy
- To create a NumPy array using numpy.array(), you call this function and pass a Python list (or any array-like object) as an argument. This function creates a new NumPy array from the given data.
- Example:
  - import numpy

```
arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

NumPy as np

- NumPy is usually imported under the np alias.
  - import numpy as np
- Example:

```
import numpy as np

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

print(arr)
```

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

NumPy Creating Arrays

- We can create a NumPy ndarray object by using the array() function.
- Example:
  - ```
    import numpy as np

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

    print(arr)

    print(type(arr))
    ```

- To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray.
- For example:
  - ```
    import numpy as np

    arr = np.array((1, 2, 3, 4, 5))

    print(arr)
    ```

NumPy creating Arrays cont..

- You can use **arange()** function to create an evenly spaced array with a given interval
- For example:

```
import numpy as np
a1 = np.arrange(10)   # creates a range from 0 to 9
print(a1)
print(a1.shape)
```

Note: In NumPy, the shape of an array is often represented as a tuple. In this case, (10,) indicates that there is one dimension with 10 elements.

Try, np.zeros, np.ones.

Dimensions in arrays

A dimension in arrays is one level of array depth (nested arrays).

nested array: are arrays that have arrays as their elements.

NumPy arrays can have multiple dimensions, allowing users to store data in multilayered structures

- 0D arrays -> Scalar (single element)
- 1D arrays -> Vector (a list of elements)
- 2D arrays -> Matrix (a spreadsheet of data)
- 3D arrays -> Tensors (Storing a color image)

Creating NumPy array from a list

- You can use the np alias to create ndarray of a list using the array() method.
  - li = [1,2,3,4]
    numpyArr = np.array(li) or
  - numpyArr = np.array([1,2,3,4])

Note: The resulting array looks the same as a list but is a NumPy object.

Try it yourself by checking the type.

Try creating a NumPy Array similarly from a Tuple.

Check the type to see if you have done it correctly!!

2D arrays

- An array that has 1-D arrays as its elements is called a 2-D array.
- These are often used to represent matrix or 2nd order tensors.

Example:

```
import numpy as np

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

print(arr)
```

3D arrays

- An array that has 2-D arrays (matrices) as its elements is called 3-D array. In other words, each element of the 3D array is itself a 2D array.
- These are often used to represent a 3rd order tensor.

Example:

```
import numpy as np

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

print(arr)
```

Checking number of dimensions

- NumPy Arrays provides the **ndim** attribute that returns an integer that tells us how many dimensions the array have.
- Example:

```
b = np.array([1, 2, 3, 4, 5])
print(b.ndim)
```

Create 0, 1, 2 and 3-dimensional arrays, and print their dimensions

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3],
[4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Higher Dimensional Arrays

- An array can have any number of dimensions.
- When the array is created, you can define the number of dimensions by using the ndmin argument.

Example:

```
import numpy as np

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

print(arr)
print('number of dimensions :', arr.ndim)
```

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.

Try it yourself!

- Get the second element from the following array.

```
import numpy as np

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

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.

- Create a two-dimensional array.
- Access the element on the first row, second column.

Solution

- We can create 2D arrays like:

list1 = [1, 2, 3, 4]

list2 = [5, 6, 7, 8]

arr = np.array([list1, list2])

- Or

arr2 = np.array([[1, 2, 3, 4], [5, 6, 7,8]])

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

Problem

- Create a two dimensional array with the following dimensions: (2,5)
- Access the element on the 2nd row, 5th column:

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:

```
import numpy as np

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

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

Example Explained
arr[0, 1, 2] prints the value 6.
And this is why:
The first number represents the first dimension, which contains two arrays:
[[1, 2, 3], [4, 5, 6]]
and:
[[7, 8, 9], [10, 11, 12]]
Since we selected 0, we are left with the first array:
[[1, 2, 3], [4, 5, 6]]
The second number represents the second dimension, which also contains two arrays:
[1, 2, 3]
and:
[4, 5, 6]
Since we selected 1, we are left with the second array:
[4, 5, 6]
The third number represents the third dimension, which contains three values:
4
5
6
Since we selected 2, we end up with the third value:
6

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

- 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])
  ```
- Try slicing elements from index 4 to the end of the array.
- Try slicing elements from the beginning to index 4 (not included)

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])
```

Try, returning every other element from the entire array

Slicing 2-D Arrays

For the two-dimensional arrays, the slicing syntax becomes [start:stop, start:stop]

- Example 1: 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 2: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 3: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])
```

Problem

- Given the following array, extract the last two rows and first two columns

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