Getting Started with NumPy

Author: ChatGPT & Tran Thu Le
Date: 17/02/2023

Contents:

NumPy is a Python library that stands for “Numerical Python”. In this tutorial, we will cover the basics of NumPy and show you how to use this library to work with matrices and vectors.

Installing NumPy

Before we can get started with NumPy, we need to install it. You can install NumPy using pip, the package installer for Python. Open a computer terminal and enter the following command:

```
pip install numpy
```

Importing NumPy

To use NumPy, we need to import it. We can do this using the following code:

```
import numpy as np
```

Here, we are importing NumPy and giving it an alias “np” to make it easier to reference the library later in our code.

Creating NumPy Arrays

The primary data structure in NumPy is the “ndarray”, or N-dimensional array. We can create an array in several ways.

From a List

We can create a NumPy array from a Python list. To do this, we can use the “np.array()” function. Here’s an example:

```
import numpy as np

list1 = [1, 2, 3, 4, 5]
arr1 = np.array(list1)
print(arr1)
```

Output:

```
[1 2 3 4 5]
```

From a Tuple

We can also create a NumPy array from a Python tuple. Here’s an example:

```
import numpy as np

tuple1 = (1, 2, 3, 4, 5)
arr1 = np.array(tuple1)
print(arr1)
```

Output:

```
[1 2 3 4 5]
```

Using NumPy Functions

NumPy provides several functions for creating arrays with specific values. Here are some examples:

```
import numpy as np

# Create an array of zeros
zeros_arr = np.zeros(5)
print(zeros_arr)

# Create an array of ones
ones_arr = np.ones(5)
print(ones_arr)

# Create an array of random values between 0 and 1
```

```
rand_arr = np.random.rand(5)
print(rand_arr)
```

Output:

```
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
[0.04271752 0.51870486 0.45002719 0.10837288 0.8461678 ]
```

Indexing and Slicing NumPy Arrays

We can access individual elements of a NumPy array using indexing. Indexing in NumPy is zero-based, meaning that the first element has an index of 0, the second element has an index of 1, and so on.

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])
print(arr1[0])
print(arr1[3])
```

Output:

```
1
4
```

We can also slice NumPy arrays to select a range of values. Slicing in NumPy is similar to slicing in Python lists.

```
# Slice elements 1 through 3
print(arr1[1:4])

# Slice elements 0 through 2
print(arr1[:3])

# Slice elements 2 through the end
print(arr1[2:])
```

Output:

```
[2 3 4]
[1 2 3]
[3 4 5]
```

NumPy Array Operations

NumPy provides many built-in functions that allow us to perform mathematical operations on arrays.

Basic Operations

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([6, 7, 8, 9, 10])

# Add two arrays
print(arr1 + arr2)

# Subtract two arrays
print(arr2 - arr1)

# Multiply two arrays
print(arr1 * arr2)

# Divide two arrays
print(arr2 / arr1)
```

Output:

```
[ 7  9 11 13 15]
[5 5 5 5 5]
[ 6 14 24 36 50]
[6.         3.5        2.66666667 2.25       2.        ]
```

Dot Product

We can also perform a dot product between two arrays using the “np.dot()” function.

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

dot_prod = np.dot(arr1, arr2)
print(dot_prod)
```

Output: 32

Sure, here's a section on working with matrices and reshaping in NumPy:

Matrices and Reshaping

NumPy allows us to create and manipulate matrices as well as other types of multidimensional arrays. We can create a matrix using the `np.array()` function and pass in a list of lists as an argument. Each inner list will represent a row of the matrix.

```
import numpy as np

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

Output:

```
 [4 5 6]
 [7 8 9]]
```

Matrix Operations

We can perform many of the same operations on matrices that we can on one-dimensional arrays, such as indexing and slicing.

```
# Select the element in the second row, third column
print(matrix[1, 2])

# Select the first two rows and all columns
print(matrix[:2, :])

# Select all rows and the last two columns
print(matrix[:, -2:])
```

Output:

```
6
[[1 2 3]
 [4 5 6]]
[[2 3]
 [5 6]
 [8 9]]
```

We can also perform mathematical operations on matrices using the same functions as we did with one-dimensional arrays.

```
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])

# Add two matrices
print(matrix1 + matrix2)

# Subtract two matrices
print(matrix2 - matrix1)

# Multiply two matrices
print(matrix1 * matrix2)

# Matrix multiplication
print(np.dot(matrix1, matrix2))
```

Output:

```
[[ 6  8]
 [10 12]]
[[4 4]
 [4 4]]
[[ 5 12]
 [21 32]]
[[19 22]
 [43 50]]
```

Reshaping Arrays

NumPy also provides a convenient way to reshape arrays. We can use the `np.reshape()` function to change the shape of an array to any size as long as the total number of elements remains the same.

```
arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])

# Reshape the array to a 2x4 matrix
matrix1 = np.reshape(arr1, (2, 4))
print(matrix1)

# Reshape the array to a 2x2x2 cube
cube = np.reshape(arr1, (2, 2, 2))
print(cube)
```

Output:

```
[[1 2 3 4]
 [5 6 7 8]]
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

Transposing Matrices

We can also transpose matrices, which means to switch the rows and columns of the matrix. We can do this using the `.T` attribute of the matrix.

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matrix1)

# Transpose the matrix
matrix1_T = matrix1.T
print(matrix1_T)
```

Element-wise Operations

NumPy provides a variety of functions for performing element-wise operations on arrays. Element-wise operations allow us to apply an operation to each element in an array

individually, rather than to the entire array as a whole.

Basic Operations

We can perform basic arithmetic operations on arrays, such as addition, subtraction, multiplication, and division. These operations are applied element-wise.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Add two arrays
print(arr1 + arr2)

# Subtract two arrays
print(arr2 - arr1)

# Multiply two arrays
print(arr1 * arr2)

# Divide two arrays
print(arr2 / arr1)
```

Output:

```
[5 7 9]
[3 3 3]
[ 4 10 18]
[4.  2.5 2.]
```

We can also perform element-wise operations on multidimensional arrays.

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = np.array([[2, 3, 4], [5, 6, 7], [8, 9, 10]])

# Add two matrices
print(matrix1 + matrix2)

# Subtract two matrices
print(matrix2 - matrix1)

# Multiply two matrices
print(matrix1 * matrix2)

# Divide two matrices
print(matrix2 / matrix1)
```

Output:

```
[[ 3  5  7]
 [ 9 11 13]
 [15 17 19]]
```

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
[[ 2  6 12]
 [20 30 42]
 [56 72 90]]
[[2.         1.5        1.33333333]
 [1.25       1.2        1.16666667]
 [1.14285714 1.125      1.11111111]]
```

Broadcasting

NumPy also allows us to perform element-wise operations on arrays of different shapes and sizes using a technique called broadcasting. Broadcasting allows us to apply an operation to two arrays with different shapes, as long as the shapes can be aligned in a way that makes sense.

```
arr1 = np.array([1, 2, 3])
scalar = 2

# Add a scalar to an array
print(arr1 + scalar)

# Multiply an array by a scalar
print(arr1 * scalar)
```

Output:

```
[3 4 5]
[2 4 6]
```

In this example, the scalar value is broadcast to match the shape of the array, allowing us to perform the element-wise operation.

Universal Functions

NumPy provides many universal functions that can be used to perform element-wise operations on arrays. These functions are optimized for speed and can be used on arrays of any size.

```
arr1 = np.array([1, 2, 3])

# Compute the square root of each element in the array
print(np.sqrt(arr1))

# Compute the exponential of each element in the array
print(np.exp(arr1))

# Compute the sine of each element in the array
print(np.sin(arr1))
```

```
# Compute the logarithm of each element in the array
print(np.log(arr1))
```

Reshaping Arrays

In NumPy, we can reshape an array into a different size or shape using the `reshape()` function. This can be useful when working with data that is in a particular format or shape.

```
arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Reshape the array to a 3x3 matrix
matrix1 = arr1.reshape(3, 3)

print(matrix1)
```

Output:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In this example, we first created a one-dimensional array with nine elements. We then reshaped the array into a 3x3 matrix using the `reshape()` function. Note that the total number of elements in the original array must match the total number of elements in the new shape.

We can also use the `reshape()` function to convert a matrix into a one-dimensional array.

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Convert the matrix to a one-dimensional array
arr1 = matrix1.reshape(1, 9)

print(arr1)
```

Output:

```
[[1 2 3 4 5 6 7 8 9]]
```

In this example, we first created a 3x3 matrix. We then used the `reshape()` function to convert the matrix to a one-dimensional array with nine elements.

Transposing Arrays

In NumPy, we can transpose an array by rearranging its dimensions. Transposing can be useful when working with matrices and other types of data that are organized in rows and columns.

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Transpose the matrix
```

```
matrix2 = matrix1.T

print(matrix2)
```

Output:

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

In this example, we first created a 3x3 matrix. We then transposed the matrix using the `.T` attribute to produce a new matrix with the rows and columns interchanged.

Conclusion

In this tutorial, we covered the basics of NumPy and showed you how to use this library to manipulate and analyze numerical data. We covered how to install NumPy, import it into Python, create NumPy arrays, index and slice arrays, and perform mathematical operations on arrays. NumPy is a powerful library that is widely used in the scientific computing and data analysis communities, and we hope that this tutorial has helped you get started with using it.