IDS-unit-3

14 June 2025 20:20

1. What is broadcasting in NumPy, and how does it facilitate operations on arrays of different shapes?
7. Describe different methods of reshaping NumPy arrays. How does array broadcasting work?

Broadcasting arrays
• Python lists do not support direct vectorizing arithmetic operations.
• NumPy offers a faster vectorized array operation compared to Python list loop-based operations.
• Broadcasting functionality checks a set of rules for applying binary functions, such as addition, subtraction, and multiplication, on different shapes of an array.

```
# Create NumPy Array
arr1 = np.arange(1,5).reshape(2,2)
print(arr1)
Output: [[1 2] [3 4]]

# Create another NumPy Array
arr2 = np.arange(5,9).reshape(2,2)
print(arr2)
Output:
[[5 6]
[7 8]]


# Add two matrices
print(arr1+arr2)
Output: [[ 6 8]
[10 12]]
```

In all three preceding examples, we can see the addition of two arrays of the same size. This concept is known as broadcasting:

```
# Multiply two matrices
print(arr1*arr2)
Output: [[ 5 12]
[21 32]]

# Add a scaler value
print(arr1 + 3)
Output: [[4 5]
[6 7]]

# Multiply with a scalar value
print(arr1 * 3)
Output:
[[ 3 6]
[ 9 12]]
```

In all three preceding examples, we can see the addition of two arrays of the same size. This concept is known as broadcasting:

==========

2. Explain various techniques for reshaping NumPy arrays. How does broadcasting operate in this context?

Manipulating array shapes :

• Let's learn some new Python functions of NumPy, such as
reshape(),
flatten(),
ravel(),
transpose(), and
resize():

• reshape() will change the shape of the array:

```
# Create an array

arr = np.arange(12)
print(arr)
```

Output: [0 1 2 3 4 5 6 7 8 9 10 11]

# Reshape the array dimension
new_arr=arr.reshape(4,3)
print(new_arr)
Output: [[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]]

# Reshape the array dimension
new_arr2=arr.reshape(3,4)
print(new_arr2)
Output:
array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]])

• flatten() transforms an n-dimensional array into a one-dimensional array:
Create an array

arr=np.arange(1,10).reshape(3,3)
print(arr)
Output:
[[1 2 3]
[4 5 6]
[7 8 9]]

print(arr.flatten())

Output:
[1 2 3 4 5 6 7 8 9]

• The ravel() function is similar to the flatten() function.
It also transforms an n-dimensional array into a one dimensional array.
The main difference is that flatten() returns the actual array while ravel() returns the reference of the original array.
The ravel() function is faster than the flatten() function because it does not occupy extra memory:

print(arr.ravel())
Output:
[1, 2, 3, 4, 5, 6, 7, 8, 9]

• The transpose() function is a linear algebraic function that transposes the given two-dimensional matrix.
The word transpose means converting rows into columns and columns into rows:
Transpose the matrix

print(arr.transpose())
Output:
[[1 4 7]
[2 5 8]
[3 6 9]]

• The resize() function changes the size of the NumPy array.
It is similar to reshape() but it changes the shape of the original array:

# resize the matrix
arr.resize(1,9)
print(arr)
Output:[[1 2 3 4 5 6 7 8 9]]

==========

3. Discuss the different indexing methods available in NumPy, including slicing, boolean indexing, and fancy indexing. Provide relevant examples.

6. How does Boolean indexing function in NumPy? Illustrate its use with examples.

Boolean and fancy indexing

• Indexing techniques help us to select and filter elements from a NumPy array.
• Boolean indexing uses a Boolean expression in the place of indexes (in square brackets) to filter the NumPy array.
This indexing returns elements that have a true value for the Boolean expression

```
# Create NumPy Array
arr = np.arange(21,41,2)
print("Orignial Array:\n",arr)

# Boolean Indexing
print("After Boolean Condition:",arr[arr>30])
```

Output:
Orignial Array: [21 23 25 27 29 31 33 35 37 39]
After Boolean Condition: [31 33 35 37 39

• Fancy indexing is a special type of indexing in which elements of an array are selected by an array of indices.
• This means we pass the array of indices in brackets.
• Fancy indexing also supports multi-dimensional arrays.
• This will help us to easily select and modify a complex multi-dimensional set of arrays.

```
# Create NumPy Array
arr = np.arange(1,21).reshape(5,4)
print("Orignial Array:\n",arr)

# Selecting 2nd and 3rd row
indices = [1,2]
print("Selected 1st and 2nd Row:\n", arr[indices])

# Selecting 3nd and 4th row
indices = [2,3]
print("Selected 3rd and 4th Row:\n", arr[indices])
```

Output:
Orignial Array: [[1 2 3 4]
[5 6 7 8]
[9 10 11 12]
[13 14 15 16]
[17 18 19 20]]

Selected 1st and 2nd Row:
[[5 6 7 8]
[9 10 11 12]]

Selected 3rd and 4th Row:
[[9 10 11 12]
[13 14 15 16]]

==========

4. Explain the concept of matrix decomposition using Singular Value Decomposition (SVD). Why is it important in data analysis?

Decomposing a matrix using svd
• Matrix decomposition is the process of splitting a matrix into parts. It is also known as matrix factorization.
• There are lots of matrix decomposition methods available such as
1. lower-upper (LU) decomposition
2. QR decomposition (where Q is orthogonal and R is upper-triangular)
3. Cholesky decomposition, and
4. SVD.

• Eigenanalysis decomposes a matrix into vectors and values.
• SVD decomposes a matrix into the following parts: singular vectors and singular values.
• SVD is widely used in signal processing, computer vision, natural language processing (NLP), and machine learning—for example, topic modeling and recommender systems where SVD is widely accepted and implemented in real-life business solutions.

$$A = U\Sigma V^T$$

Here, A is a m x n left singular matrix, Σ is a n x n diagonal matrix, V is a m x n right singular matrix, and VT is the transpose of the V.
The numpy.linalg subpackage offers the svd() function to decompose a matrix.

```
# import required libraries
import numpy as np
from scipy.linalg import svd

# Create a matrix
mat=np.array([[5, 3, 1],[5, 3, 0],[1, 0, 5]])

# Perform matrix decomposition using SVD

U, Sigma, V_transpose = svd(mat)
print("Left Singular Matrix:",U)
print("Diagonal Matrix: ", Sigma)
print("Right Singular Matrix:", V_transpose)
```

Output:
Left Singular Matrix: [[-0.70097269 -0.06420281 -0.7102924]
[-0.6748668 -0.26235919 0.68972636]
[-0.23063411 0.9628321 0.14057828]]

Diagonal Matrix: [8.42757145 4.89599358 0.07270729]

Right Singular Matrix: [[-0.84363943 -0.48976369 -0.2200092]
[-0.13684207 -0.20009952 0.97017237]
[0.51917893 -0.84858218 -0.10179157]]

==========

5. What does it signify if the determinant of a matrix is zero? Explain its implications in linear algebra.

Determinant
- It is a scalar value that is calculated from a square matrix.
- The determinant is a fundamental operation that helps us in the inverse matrix and in solving linear equations.
- Determinants are only calculated for square matrices. A square matrix has an equal number of rows and columns.
- The numpy.linalg subpackage provides the det() function for calculating the determinant of a given input matrix

```
# Import numpy
import numpy as np
# Create matrix using NumPy
mat=np.mat([[2,4],[5,7]])
print("Matrix:\n",mat)
# Calculate determinant
print("Determinant:",np.linalg.det(mat))
```

Output: Matrix: [[2 4] [5 7]]

Determinant: -5.999999999999998

==========

6. How does Boolean indexing function in NumPy? Illustrate its use with examples.