Python
CH09: Numpy

전종준 교수

서울시립대학교 통계학과/통계데이터사이언스대학원

Contents

- 배열의 개념과 특성

- 배열을 이용한 이미지 처리

- 행렬과 변환

배열의 개념과 특성

배열(array)이란?

- 배열은 **값들을 일정한 순서로 나열한 자료 구조**
- 파이썬의 리스트도 배열과 비슷하게 작동하지만, NumPy의 배열은 **더 정해진 규칙과 효율성**을 가지고 있음
- 가장 간단한 배열은 숫자의 **줄(row)**

```
# 리스트처럼 생긴 1차원 배열
[10, 20, 30, 40]
```

- **2차원 배열**은 줄(row)과 칸(column)이 있는 표 형태입니다:

```
# 2행 3열짜리 2차원 배열
[[1, 2, 3],
  [4, 5, 6]]
```

- NumPy는 이런 배열들을 빠르게 만들고 계산할 수 있도록 도와줌

배열의 더하기와 빼기

- NumPy 배열은 **반복문 없이도** 수학 연산을 빠르게 할 수 있음
- 배열 간의 더하기, 빼기는 **원소별(element-wise)**로 수행

예시: 두 배열을 더하거나 뺄 수 있어요.

```
1  import numpy as np
2
3  a = np.array([10, 20, 30])
4  b = np.array([1, 2, 3])
5
6  print(a + b)  # [11 22 33]
7  print(a - b)  # [9 18 27]
```

- 각 자리의 숫자끼리 연산이 됨

1차원 배열을 2차원으로 바꾸기 (reshape)

- NumPy의 `reshape()` 함수는 배열의 모양을 바꿔줍니다.
- 기본적으로는 C-style (행 우선, row-major) 순서를 따릅니다.

예시: 1차원 → 2행 3열로 변형

```
1  import numpy as np
2
3  a = np.array([1, 2, 3, 4, 5, 6])
4  print(a.reshape((2, 3)))
```

결과 (기본: C-order):

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

reshape: C vs F 순서 차이

- `order='C'`: 행 우선 (기본값)
- `order='F'`: 열 우선 (Fortran 스타일)

같은 데이터, 다른 정렬 방식:

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

print(a.reshape((2, 3), order='C'))  # 행 우선
print(a.reshape((2, 3), order='F'))  # 열 우선
```

결과:

```
# C-order
[[1 2 3]
 [4 5 6]]

# F-order
[[1 3 5]
 [2 4 6]]
```

- 순서의 차이는 데이터 해석이나 저장 시 중요한 역할을 합니다.

브로드캐스팅(Broadcasting) 이해하기

- **브로드캐스팅**은 크기가 다른 배열끼리 연산할 수 있도록 도와주는 NumPy의 기능입니다.
- 작은 배열이 큰 배열의 크기에 맞게 **자동으로 확장**되어 연산됩니다.

예시: 1차원 배열 + 스칼라

```
1  import numpy as np
2
3  a = np.array([10, 20, 30])
4  print(a + 5)  # [15 25 35]
```

브로드캐스팅 예제: (3,3) + (3,1)

- (3, 3) 배열에 (3, 1) 배열을 더하면 각 행(row)에 값이 더해집니다.
- 열 방향으로 자동 복제(broadcast)되어 연산이 수행됩니다.

예제 코드:

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])   # shape (3,3)

B = np.array([[10],
              [20],
              [30]])        # shape (3,1)

C = A + B
print(C)
```

브로드캐스팅 예제: (3,3) + (1,3)

- (3, 3) 배열에 (1, 3) 배열을 더하면 각 열(column)에 값이 더해집니다.
- 행 방향으로 자동 복제되어 연산이 수행됩니다.

예제 코드:

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])    # shape (3,3)

B = np.array([[100, 200, 300]])  # shape (1,3)

C = A + B
print(C)
```

열마다 스칼라 곱: (3,3) * (1,3)

- (3,3) 배열과 (1,3) 배열의 곱 → 각 열에 대응하는 값으로 곱해짐

예제:

```
1  import numpy as np
2
3  A = np.array([[1, 2, 3],
4                [4, 5, 6],
5                [7, 8, 9]])      # shape (3,3)
6
7  scalars = np.array([[1, 10, 100]])  # shape (1,3)
8
9  C = A * scalars
10 print(C)
```

행마다 스칼라 곱: (3,3) * (3,1)

- (3,3) 배열과 (3,1) 배열의 곱 → 각 행에 대응하는 값으로 곱해짐

예제:

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])      # shape (3,3)

scalars = np.array([[1],
                    [10],
                    [100]])    # shape (3,1)

C = A * scalars
print(C)
```

브로드캐스팅(Broadcasting) 이해하기

예시: (3, 1) 배열 + (3,) 배열

```
a = np.array([[1], [2], [3]])   # shape (3, 1)
b = np.array([10, 20, 30])      # shape (3,)

print(a + b)
# 결과:
# [[11 21 31]
#  [12 22 32]
#  [13 23 33]]
```

- a는 열방향, b는 행방향으로 복사되어 연산됩니다!

배열을 이용한 이미지 처리

NumPy 배열과 주요 메서드

- 배열 속성: `shape`, `dtype`, `ndim`, `size`
- 연산 메서드: `mean()`, `max()`, `min()`, `sum()`, `reshape()`
- axis 개념:
  - `axis=0`: 행 방향 (세로 기준)
  - `axis=1`: 열 방향 (가로 기준)

예시:

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a.mean(axis=0))  # 각 열 평균
print(a.mean(axis=1))  # 각 행 평균
```

PIL 이미지 객체와 NumPy 배열

- `PIL.Image` 객체는 시각적 이미지 표현에 적합
- 주요 속성:
  - `img.mode` : 'L', 'RGB' 등
  - `img.size` : (너비, 높이)
  - `img.format` : JPEG, PNG 등
- `np.array(img)`로 변환해야 픽셀을 수치로 조작 가능

흑백 이미지 불러오기

- mode='L'로 열면 2차원 배열 (gray scale)

예시:

```
from PIL import Image
import numpy as np

img = Image.open('gray_sample.png').convert('L')
gray = np.array(img)

print(gray.shape)
print(gray.max(), gray.min())
```

흑백 이미지 시각화

- matplotlib.pyplot.imshow()로 출력 가능
- cmap='gray'로 회색조 표현

```
import matplotlib.pyplot as plt

plt.imshow(gray, cmap='gray')
plt.colorbar()
plt.show()
```

imshow()의 좌표계 이해하기

- `plt.imshow()`는 이미지를 행렬(배열)처럼 다룹니다.
- 즉, (행, 열) → (y, x) 순서로 해석됩니다.
- 원점 (0,0)은 **왼쪽 위**입니다.

예시 배열:

```
import numpy as np
import matplotlib.pyplot as plt

a = np.array([[0, 1],
              [2, 3]])

plt.imshow(a, cmap='gray')
plt.colorbar()
plt.show()
```

시각적 위치:

(0,0) = 0	(0,1) = 1
(1,0) = 2	(1,1) = 3

이미지는 이렇게 표시됩니다:

- 위쪽 행이 먼저 나타남
- 왼쪽 → 오른쪽으로 열 진행

컬러 이미지 불러오기

- 컬러 이미지는 보통 RGB 모드로 구성된 3차원 배열입니다.
- `PIL.Image` 객체를 `'RGB'` 모드로 변환한 뒤 NumPy 배열로 바꿉니다.

```
from PIL import Image
import numpy as np

img = Image.open("dog.jpg").convert("RGB") # 이미지가 RGB인경우 불필요
img_array = np.array(img)

print(img_array.shape)  # (높이, 너비, 3)
print(img_array.dtype)  # 보통 uint8
```

- 마지막 축은 채널(R, G, B)이며 각 값은 0 255 범위입니다.

컬러 이미지와 채널 분리

- RGB 이미지는 3차원 배열 (H, W, C)
- 채널 분리 예시1 (채널이 분리되고 크기는 (H,W)가 됨)
  - R: `img[:, :, 0]`
  - G: `img[:, :, 1]`
  - B: `img[:, :, 2]`
- 채널 분리 예시2 (채널이 분리되고 크기는 (H,W,1)가 됨)
  - R: `img[:, :, 0:1]`
  - G: `img[:, :, 1:2]`
  - B: `img[:, :, 2:3]`

colormap (cmap) 설명

- cmap은 숫자 데이터를 색으로 표현하는 방식
- 주로 단일 채널(R, G, B 또는 gray)에 사용
- 자주 쓰는 cmap:
  - 'gray', 'viridis', 'hot', 'Reds', 'cool'

```
plt.imshow(img_array[:,:,0:1], cmap='Reds')
plt.title("Generated Grayscale Image")
plt.colorbar()
plt.show()
```

Green 채널을 cmap = 'viridis' 로 Blue 채널을 cmap='cool'로 확인해보자

배열 슬라이싱과 조작

- 이미지 배열도 일반 배열처럼 인덱싱, 슬라이싱이 가능합니다.
- 자주 쓰는 조작 예시:
  - 특정 영역 자르기: `img[50:100, 50:100]`
  - 상하 반전: `np.flipud(img)`
  - 좌우 반전: `np.fliplr(img)`
  - 회전 (90도): `np.rot90(img)`

```
img = img_array.copy()
cropped = img[50:100, 50:100,:]
flipped_ud = np.flipud(img)
plt.imshow(flipped_ud)
flipped_lr = np.fliplr(img)
rotated = np.rot90(img)
```

조건 기반 이미지 필터링

- 예: R 채널이 150 이상인 픽셀만 남기기
- 마스크를 활용해 조건을 만족하지 않는 영역 제거

```
red = img[:, :, 0]                          # R 채널 추출
mask = red > 150                            # 조건 마스크

filtered = img.copy()
filtered[~mask] = 0                         # not 의 의미 조건을 만족하지 않으면 검정 처리
```

- 마스크는 `True/False`로 구성된 2D 배열
- 색상이 강한 영역만 남기고 나머지를 제거할 수 있음

마스크 시각화

- 마스크는 True/False 값을 갖는 2D 배열입니다.
- 이를 시각화하면 조건을 만족하는 영역이 어디인지 볼 수 있습니다.

```
plt.imshow(mask, cmap='gray')
plt.title("R > 150인 영역 마스크")
plt.colorbar()
plt.show()
```

- 밝은 영역 = 조건을 만족함 (True)
- 어두운 영역 = 조건을 만족하지 않음 (False)

필터링된 이미지 출력

- 마스크를 적용한 이미지를 다시 시각화하면, 조건을 만족하는 부분만 남은 이미지를 확인할 수 있습니다.

```
plt.imshow(filtered)
plt.title("R > 150 조건을 만족하는 영역만 남김")
plt.show()
```

- 원래 이미지에서 R 채널이 약한 영역은 검정색(0)으로 제거됨
- 시각적으로 필터링 결과를 확인할 수 있음

RGB → Grayscale 변환

- 컬러 이미지는 (H, W, 3)의 형태를 가짐 (R, G, B)
- 각 채널의 값을 평균내면 1채널 흑백 이미지로 변환 가능
- 밝기 = (R + G + B) / 3

```
gray = np.mean(img_array, axis=2)

print(gray.shape)  # (H, W)
print(gray.dtype)  # float64
```

- 결과는 2차원 배열이며, float형으로 출력됩니다.

Grayscale 이미지 시각화

- Grayscale 이미지도 `imshow()`로 출력 가능
- 반드시 `cmap='gray'`를 지정

```
plt.imshow(gray, cmap='gray')
plt.title("RGB → Grayscale 변환 결과")
plt.colorbar()
plt.show()
```

요약: 이미지와 NumPy 배열

- 이미지 = 숫자로 이루어진 2차원 또는 3차원 배열
- PIL.Image → `np.array()`로 변환하여 수치 조작 가능
- 배열 슬라이싱, 반전, 회전, 조건 필터링 등을 배움
- R, G, B 채널 분리 및 Grayscale 변환도 실습함
- `imshow()` 시 좌표계는 (행, 열) 기준이며, `cmap`으로 시각화 방법 지정 가능

퀴즈: 배열과 이미지

- **Q1.** RGB 이미지에서 초록(G) 채널만 추출하는 코드로 옳은 것은?
  1. `img[:, :, 0]`
  2. `img[:, :, 1]`
  3. `img[:, :, 2]`

- **Q2.** 아래 코드의 결과 shape은?

```
gray = np.mean(img_array, axis=2)
```

  1. (H, W, 1)
  2. (H, W)
  3. (H,)

- **Q3.** `np.flipud()` 함수는 어떤 방향으로 반전하나요?
  1. 좌우
  2. 상하
  3. 90도 회전

행렬과 변환

수업 목표

- 이미지는 숫자(배열)로 이루어진 데이터
- 숫자들을 곱하거나 더하면 어떤 일이 일어날까?
- 오늘은 **선형 변환(linear transformation)**을 통해 데이터를 바꾸는 과정을 확인

열벡터의 선형 변환

- 열벡터 $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$: 2차원 입력데이터
- 행렬 $\mathrm{A} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$: 변환연산자
- 결과 $\mathrm{A}\mathbf{x} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$: 2차원 출력데이터

```
import numpy as np
x = np.array([[1], [2]])
A = np.array([[2, 0], [0, 3]])
result = A @ x
print(result)
```

이 결과에서 A 는 2행 2열 행렬로서 2차원 입력데이터를 2차원 출력데이데이터로 변환하였음

행렬 곱셈은 어떻게 작동할까?

- 행렬의 각 행과 열벡터를 곱해 더하면 결과 벡터가 됩니다.

```
A = np.array([[1, 2],
              [3, 4]])
x = np.array([[5], [6]])
result = A @ x  # [[17], [39]]
```

- 결과: $\begin{bmatrix} 1*5+2*6 \\ 3*5+4*6 \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$

열벡터의 선형 변환

- 열벡터 $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$: 3차원 입력데이터
- 행렬 $\mathrm{A} = \begin{bmatrix} 2 & 0 & 1 \\ 0 & 3 & 1 \end{bmatrix}$: 변환연산자

```
import numpy as np
x = np.array([[1], [2], [3]])
A = np.array([[2, 0, 1], [0, 3, 1]])
result = A @ x
print(result)
```

result 를 구하고 이 결과에서 A를 변환연산자로서 설명해보자.

행렬 곱 연습 1: (2×3) × (3×1)

- 아래 행렬 A, B를 곱하여 결과 행렬 C를 손으로 계산해보세요.

$$\mathsf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad \mathsf{B} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

$$\mathsf{C} = \mathsf{A} \times \mathsf{B} \quad (\text{결과 shape: } 2 \times 1)$$

- $\mathsf{C}_{0,0} = 1 \times 1 + 2 \times 0 + 3 \times (-1) =$ _____
- $\mathsf{C}_{1,0} = 4 \times 1 + 5 \times 0 + 6 \times (-1) =$ _____

행렬 곱 연습 2: (2×2) × (2×2)

$$P = \begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix}, \quad Q = \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}$$

$$P \times Q = ? \quad (\text{shape: } 2 \times 2)$$

- $C_{0,0} = 1 \times 3 + (-1) \times 0 = \_\_\_\_\_$
- $C_{0,1} = 1 \times 2 + (-1) \times 1 = \_\_\_\_\_$
- $C_{1,0} = 2 \times 3 + 0 \times 0 = \_\_\_\_\_$
- $C_{1,1} = 2 \times 2 + 0 \times 1 = \_\_\_\_\_$

행렬 곱 연습 3: (3×2) × (2×3)

$$A = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$C = A \times B \quad (\text{결과 shape: } 3 \times 3)$$

예: $C_{0,0} = 1 \times 2 + 0 \times 1 =$ _____, $C_{0,1} = 1 \times 0 + 0 \times 1 =$ _____

나머지 항목도 스스로 계산해보세요!

이미지 배열에 행렬 곱 적용

- 이미지를 2D 배열로 보고 행렬을 곱해봅시다.

```
img = np.array([[10, 20, 30],
                [40, 50, 60],
                [70, 80, 90]])
W = np.array([[0, 0, 1],
              [0, 1, 0],
              [1, 0, 0]])
transformed = W @ img
```

- 위 행렬은 행 순서를 반전시키는 효과를 낸다

flatten된 이미지와 dot product

- 이미지를 1차원 벡터로 펼치고 가중치 벡터와 내적해봅니다.

```
img = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
x = img.flatten().reshape(-1, 1)
w = np.array([[1, 0, 0, 0, 0, 0, 0, 0, -1]])
result = w @ x
```

- flatten 연산은 2D array 를 1D로 변환
- 이는 한 줄의 가중합을 만드는 연산입니다 (뉴런의 연산과 유사).

실습 퀴즈

- 아래의 행렬과 이미지를 곱한 결과를 예측해보세요.

```
W = np.array([[0, 1, 0],
              [1, 0, 0],
              [0, 0, 1]])
img = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
result = W @ img
```

- 행이 어떻게 바뀔지 손으로 계산해보세요!