1 Numpy

https://numpy.org/doc/stable/user/quickstart.html

https://cs231n.github.io/python-numpy-tutorial/#numpy

Numpy là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.

Cài đặt bằng pip install numpy

1.1 Arrays

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Numpy cũng cung cấp rất nhiều hàm để khởi tạo arrays

```
import numpy as np


a = np.zeros((2,2))   # Tạo một numpy array với tất cả phần tử là 0
print(a)              # "[[ 0.  0.]
                      #   [ 0.  0.]]"

b = np.ones((1,2))    # Tạo một numpy array với tất cả phần tử là 1
print(b)              # "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Tạo một mảng hằng
print(c)               # "[[ 7.  7.]
```

```
                        #   [ 7.  7.]]"

d = np.eye(2)         # Tạo một ma trận đơn vị 2 x 2
print(d)              # "[[ 1.  0.]
                      #   [ 0.  1.]]"

e = np.random.random((2,2))  # Tạo một mảng với các giá trị ngẫu nhiên
print(e)                     # Có thể là "[[ 0.91940167  0.08143941]
                             #             [ 0.68744134  0.87236687]]"
f = np.arange(10) # Tạo 1 numpy array với các phần tử từ 0 đến 9
print(f)          # "[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]"
```

1.2 Array indexing

Numpy cung cấp một số cách để truy xuất phần tử trong mảng

Slicing: Tương tự như list trong python, numpy arrays cũng có thể được cắt.

```
import numpy as np

# Khởi tạo numpy array có shape = (3, 4) có giá trị như sau:
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Sử dụng slicing để tạo numpy array b bằng cách lấy 2 hàng đầu tiên
# và cột 1, 2. Như vậy b sẽ có shape = (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# Một array tạo ra từ slicing sẽ có cùng địa chỉ với array gốc.
# Nếu thay đổi 1 trong 2 thì array còn lại cũng thay đổi.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] ở đây tương đương với a[0, 1]
print(a[0, 1])   # Prints "77"
```

Bạn cũng có thể kết hợp việc dùng slicing và dùng chỉ số. Tuy nhiên, cách làm đó sẽ cho ra một mảng mới có rank thấp hơn mảng gốc.

```
import numpy as np
```

```
# Tạo một numpy array có shape (3, 4) với giá trị như sau:
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Hai cách truy cập dữ liệu ở hàng giữa của mảng
# Dùng kết hợp chỉ số và slice -> được array mới có rank thấp hơn,
# Nếu chỉ dùng slice ta sẽ có 1 array mới có cùng rank
# với array gốc
row_r1 = a[1, :]    # Rank 1, hàng thứ 2 của a
row_r2 = a[1:2, :]  # Rank 2, vẫn là hàng thứ 2 của a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"

# Chúng ta có thể làm tương tự với cột của numpy array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # Prints "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # Prints "[[ 2]
                             #          [ 6]
                             #          [10]] (3, 1)"
```

Integer array indexing: Khi bạn truy xuất mảng dùng slicing, kết quả trả về sẽ là mảng con của mảng ban đầu, tuy nhiên sử dụng chỉ số mảng cho phép bạn xây dựng mảng tùy ý từ một mảng khác

```
import numpy as np

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

# Truy xuất mảng dùng chỉ số.
# Kết quả thu được là 1 mảng có shape (3,)
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# Sẽ thu được kết quả tương đương như trên với cách này:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# Bạn được phép sử dụng chỉ số mảng để
# truy xuất tới 1 phần tử
# của mảng gốc nhiều hơn 1 lần
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Một cách làm khác tương đương:
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

Một mẹo hữu ích dùng chỉ số mảng để chọn và thay đổi phần tử từ mỗi hàng của ma trận

```
import numpy as np


# Tạo một mảng mới từ đó ta sẽ chọn các phần tử
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print(a)  # prints "array([[ 1,  2,  3],
          #                [ 4,  5,  6],
          #                [ 7,  8,  9],
          #                [10, 11, 12]])"

# Tạo một mảng các chỉ số
b = np.array([0, 2, 0, 1])

# Lấy 1 phần tử từ mỗi hàng của a dùng với chỉ số ở mảng b
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"

# Thay đổi một phần tử từ mỗi hàng của a dùng với chỉ số ở mảng b
a[np.arange(4), b] += 10

print(a)  # prints "array([[11,  2,  3],
          #                [ 4,  5, 16],
          #                [17,  8,  9],
          #                [10, 21, 12]])
```

Boolean array indexing: Cho phép bạn chọn ra các phần tử tùy ý của một mảng, thường được sử dụng để chọn ra các phần tử thỏa mãn điều kiện nào đó

```
import numpy as np


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

bool_idx = (a > 2)   # Tìm các phần tử lớn hơn 2;
                     # Trả về 1 numpy array of Booleans có shape như mảng
a
                     # và giá trị tại mỗi phần tử là
                     # True nếu phần tử của a tại đó > 2,
                     # False cho trường hợp ngược lại.
```

```
print(bool_idx)      # Prints "[[False False]
                     #          [ True  True]
                     #          [ True  True]]"

# Sử dụng một boolean array indexing để lấy
# các phần tử thỏa mãn điều kiện nhất định trong a
# Ví dụ ở đây in ra các giá trị của a > 2
# sử dụng array bool_idx đã tạo
print(a[bool_idx])  # Prints "[3 4 5 6]"

# Một cách ngắn gọn hơn:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

1.3 Datatypes

Mỗi numpy array là một lưới các phần tử cùng kiểu dữ liệu. Numpy cung cấp một tập hợp lớn các kiểu dữ liệu số mà bạn có thể sử dụng để xây dựng các mảng. Numpy cố gắng đoán một kiểu dữ liệu khi bạn tạo một mảng, nhưng các hàm xây dựng các mảng thường cũng bao gồm một đối số tùy chọn để chỉ định rõ ràng kiểu dữ liệu

```
import numpy as np


x = np.array([1, 2])   # Để numpy xác định kiểu dữ liệu
print(x.dtype)         # Prints "int64"

x = np.array([1.0, 2.0])   # Để numpy xác định kiểu dữ liệu
print(x.dtype)             # Prints "float64"

x = np.array([1, 2], dtype=np.int64)   # Chỉ định kiểu dữ liệu củ thể cho
mảng
print(x.dtype)                         # Prints "int64"
```

1.4 Array math

Ghép, cộng, nhân, hoán vị chỉ với một dòng code. Dưới đây là một số ví dụ về các phép toán số học và nhân khác nhau với các mảng Numpy

```
import numpy as np


x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Tổng của 2 mảng, cả 2 cách cho cùng một kết quả
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Hiệu 2 mảng
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Tính tích từng phần tử của x nhân với từng phần tử của y
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Tương tự thương của từng phần x chia với từng phần tử y
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Bình phương từng phần tử trong x
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

Để nhân 2 ma trận hoặc nhân vector với ma trận trong numpy, chúng ta sử dụng hàm **dot**

```
import numpy as np


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

v = np.array([9,10])
```

```
w = np.array([11, 12])

# Tích trong của 2 vector; Cả 2 đều cho kết quả là 219
print(v.dot(w))
print(np.dot(v, w))

# Nhân ma trận với vector; cả 2 đều cho mảng rank 1: [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Ma trận với ma trận; cả 2 đều cho mảng rank 2
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Numpy cung cấp nhiều hàm hữu ích để thực hiện tính toán trên mảng; một trong những hàm hữu ích nữa là **sum**

```
import numpy as np


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

print(np.sum(x))  # Tổng các phần tử của mảng; prints "10"
print(np.sum(x, axis=0))  # Tính tổng theo từng cột; prints "[4 6]"
print(np.sum(x, axis=1))  # Tính tổng theo từng hàng; prints "[3 7]"
```

2 Matplotlib