A brief introduction to Python & Numpy

Deep Learning for Computer Vision

TA: Junming CHEN

COMP 4471 & ELEC4240 Lab 1
September 14

About Myself

- Name: Junming CHEN (Jeremy)
- Third year AI PhD student
- Research interests:
  - Video understanding & synthesis
  - Multi-modal learning & 3D vision
  - Domain adaptation & generalization
- Email
  - jeremy.junming.chen@connect.ust.hk

HKUST

HKUST CSE DEPARTMENT

Today's Tutorial

- Introduction to Python
  - Containers
  - Functions
  - Classes
- Introduction to Numpy
  - Basics
  - Useful functions
  - Common pitfalls

HKUST

HKUST CSE DEPARTMENT

An Introduction to Python

- What is Python?
  - Python is a high-level, dynamically typed multiparadigm programming language.
- Why do we learn Python?
  - Simplicity: Natural language style
  - Huge Community: Libraries and Frameworks
  - Most popular language used by AI researchers
- Now start your favorite python terminal
  - Recommend: Vscode, Jupyter Notebook
  - Environment management: Anaconda is all you need!

Python Basics

- Basic data types
  - Numbers: int, float, etc.
  - Booleans: True, False
    - and, or, not, !=, ==
  - Strings

```
In []: hw = 'hello' + ' world!'
```

    - replace(), split(), etc.

Python Containers

- Lists: ordered element sequence

```
In []: x = [1,2,3]
```

  - List could contain different types of elements

```
In []: x[0] = 'hello'
```

  - Basic functions

```
In []: x.append(4.0)
```

  - Loops

```
In []: for element in x:
           print(element)
```

Python Containers

- Dictionaries: (key, value) pairs

```
In []: d = {'cat': 'cute', 'dog': 'furry'}
```

  - Entry to value

```
In []: print(d['cat'])
```

  - Set an entry

```
In []: d['fish'] = 'wet'
```

  - Loops

```
In []: for animal, adj in d.items():
           print('A %s is %d' % (animal, adj))
```

Python Containers

- Sets: An unordered collection of distinct elements

```
In []: animals = {'cat', 'dog'}
```

  - Add / remove elements to set

```
In []: animals.add('fish')
In []: animals.remove('cat')
```

- Tuple: An (immutable) ordered list of values

```
In []: d = {(x, x + 1): x for x inrange(10)}
```

Python functions

- Define and call

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
```

- Default argument

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

Python classes

- Define

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)
```

- Construct an instance

```
g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

An Introduction to Numpy

- What is Numpy?
  - A scientific computing library for Python
  - All you need for Linear Algebra on CPU
  - Want linear algebra on GPU? PyTorch / TensorFlow / JAX
- Why do we learn Numpy?
  - N-dimensional array manipulation
    - E.g., your dataset (images), weights, and feature maps…
  - Deep learning is based on linear algebra and optimization

Numpy Basics

- Importing

```
In []: import numpy as np
```

- Array creation

```
In []: a = np.array([1, 2, 3])
```

- Shape: size of the array

```
In []: print(a.shape)
```

Array Creation

- Array creation (a 2 by 3 matrix)

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

- Shape: size of the array

```
In []: print(b.shape)
```

- Specify data type:

```
In []: b = np.array([[1,2,3],[4,5,6]],
dtype=np.int64)
In []: print(b.dtype)
```

Array creation: useful functions

- np.zeros()
- np.ones()
- np.full()
- np.eyes()
- np.random.random()
- Specify the shape with Python tuple

Array Indexing

- Fetching one number in the n-D array

```
In []: a = np.array([1, 2, 3])
```

- Use square brackets

```
In []: print(a[0], a[1], a[2])
```

Array Indexing

- Fetching one number in the n-D array

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

- Use square brackets

```
In []: print(b[0, 0], b[0, 1],
       b[1, 0])
```

HKUST

HKUST CSE DEPARTMENT

Array Indexing

- Fetching a sub-array of the n-D array

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

- Just like that for Python lists

```
In []: b = a[:2, 1:3]
```

- Warning:
  - Index starts at 0
  - Last index is not included
  - 1-2 rows, 2-3 columns

Array Indexing

- Fetching a sub-array of the n-D array

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

- Just like that for Python lists

```
In []: b = a[:2, 1:3]
```

- Warning:
  - Slicing does not copy arrays
  - `b[0, 0] = 77` change `a[0, 1]`

Integer Indexing

- Want to index many elements

```
In []: print(np.array(
[a[0, 0], a[1, 1],
 a[2, 0], a[2, 1]]))
```

- Concise way: use less brackets

```
In []: print(a[[0, 1, 2, 2],
              [0, 1, 0, 1]])
```

Boolean Indexing

- Select elements that satisfy certain conditions

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

```
In []: bool_idx = (a > 2)
```

```
In []: print(a[bool_idx])
```

Boolean Indexing

- Select elements that satisfy certain conditions

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

```
In []: bool_idx = (np.sqrt(a) > 2)
```

```
In []: print(a[bool_idx])
```

Array arithmetic

- Assume `x` and `y` have the same shape

```
In []: print(x + y)
```

```
In []: print(x - y)
```

```
In []: print(x*y)
```

```
In []: print(x/y)
```

```
In []: print(np.sqrt(x))
```

Array arithmetic

- How to do matrix multiplication?

```
In []: print(np.dot(x, y)
```

```
In []: print(x.dot(y)) # Simpler: x@y
```

- Transpose

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

```
In []: print(x.T)
```

Broadcasting

- Assume we want to deduct a scalar *mean* value for each row element

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

```
In []: for i in range(a.shape[0]):
           a[i, :] = a[i, :] - mean
```

- The concise (and faster) way

```
In []: a = a - mean
```

Broadcasting

- Assume we want to deduct a three-dimension *mean* vector for each row

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

```
In []: mean = np.sum(a, axis = 0)/2
       print(mean)
```

```
Out []: [2, 3.5, 4.5]
```

Broadcasting

- Assume we want to deduct a three-dimension *mean* vector for each row

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

```
In []: for i in range(a.shape[0]):
           a[i, :] = a[i, :] - mean
```

- The concise (and faster) way

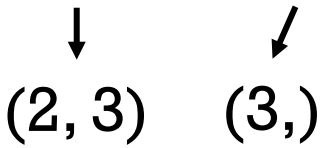
```
In []: a = a - mean
```

Broadcasting: how it works?

- Basically we are dealing with two arrays with different shapes

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

```
In []: a = a - mean
```



Broadcasting: how it works?

- Behind Idea: Broadcast one row and do element-wise subtraction

```
In []: a = a - mean
```

(2, 3) (3,)

1	2	3
4	5	6

-

2	3.5	4.5

Broadcasting: how it works?

- Broadcast one row and do element-wise subtraction

```
In []: a = a - mean
```

(2, 3) (3,)

1	2	3
4	5	6

-

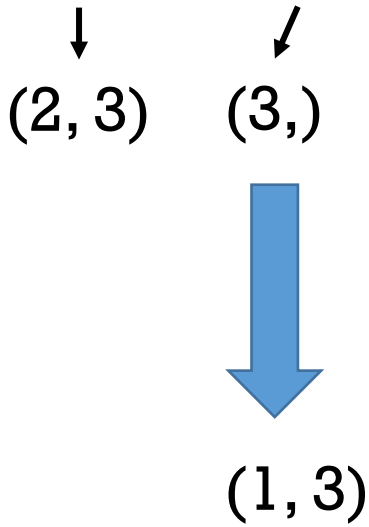
2	3.5	4.5
2	3.5	4.5

Broadcast

Broadcasting: general rules

- When the two arrays have different shape, append 1 to the smaller array

```
In []: a = a - mean
```



HKUST

HKUST CSE DEPARTMENT

Broadcasting: general rules

- Only two cases are compatible in broadcasting
  - They have the same size
  - One of the size is 1

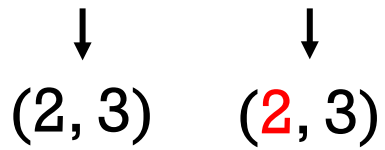
```
In []: a = a - mean
```

(2, 3) (1, 3)

Broadcasting: general rules

- Only two cases are allowed for each dimension:
  - They have the same size
  - One of the size is 1
- Broadcasting brings each dimension to be the maximum possible size

```
In []: a = a - mean
```



Broadcasting: general rules

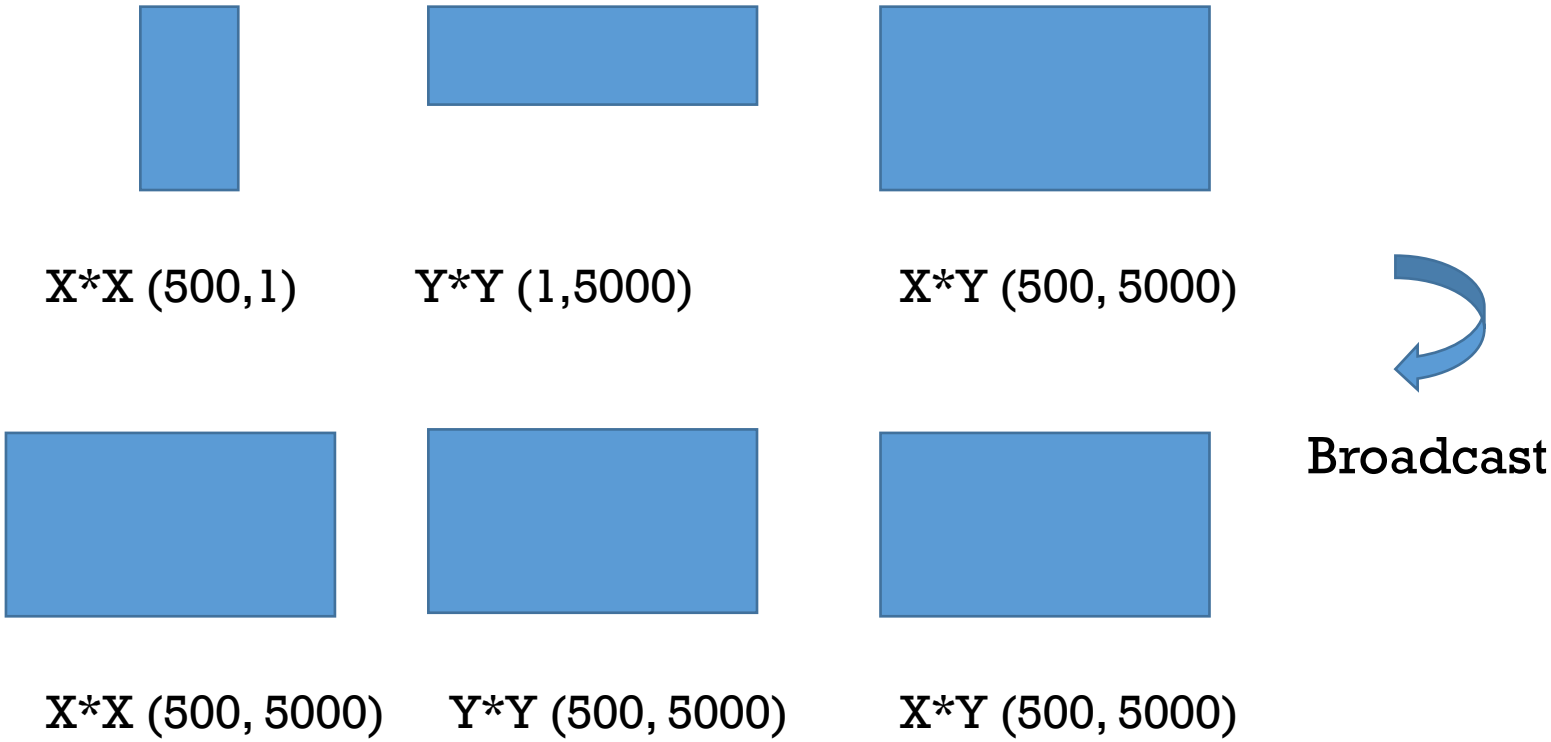
- When operating on two arrays, NumPy compares their shapes element-wise. Two dimensions are compatible when:
  - They are equal, or
  - One of them is 1
- Broadcasting brings each dimension to be the maximum possible size
- Exercise: add arrays of shapes (2, 3, 4) and (3, 4)

  -> (1, 3, 4) -> (2, 3, 4)

Broadcasting: general rules

- More exercises: whether those array shapes are compatible for broadcasting
  - (3,6,8)
    (6,8)
  - (3,6,8)
    (3,6)
  - (3,6,8)
    (3,6,1)
  - (3,1,8)
    (1,6,1)

Use broadcast



$$dist(\boldsymbol{X}, \boldsymbol{Y}) = \sqrt{(\boldsymbol{X} - \boldsymbol{Y})^2} = \sqrt{\boldsymbol{X}^2 + \boldsymbol{Y}^2 - 2\boldsymbol{XY}}$$

Other packages

- Scipy
  - Optimization, eigenvalue problems, algebraic equations, differential equations
- PIL / OpenCV
  - Image manipulation
- Matplotlib
  - For data visulization
- Documentation and Google are your friends

Q&A

HKUST CSE
DEPARTMENT