청주대학교
CHEONGJU UNIVERSITY

밑바닥부터 시작하는 딥러닝

CHAPTER 1

헬로 파이썬

박현준 (hyunjun@cju.ac.kr)

1.1 파이썬이란?

- 파이썬(Python)
  - 간단하고 배우기 쉬운 프로그래밍 언어
  - 오픈 소스, 무료로 자유롭게 이용
  - 프로그래밍 입문자에게 최적인 언어
  - 과학 분야, 특히 기계학습과 데이터 과학 분야에서 널리 쓰임

- 다양하고 강력한 라이브러리와 딥러닝 프레임워크
  - Numpy, SciPy, sklearn, Caffe, Tensorflow, Theano 등

1.2 파이썬 설치하기

- 파이썬 버전
  - 현재 파이썬은 2.x 와 3.x 라는 두 가지 버전이 공존
  - 3 버전이 나왔지만 아직 2 버전도 많이 이용
  - 파이썬을 처음 배울 때는 어느 버전을 설치할지 신중하게 선택 (하위 호환성 없음)
- 사용하는 외부 라이브러리
  - 외부 라이브러리는 최소한만 사용한다는 것이 기본 방침
  - 두 라이브러리는 예외
    - 넘파이: 수치 계산용 라이브러리
    - matplotlib: 그래프를 그려주는 라이브러리
- 아나콘다 배포판
  - 이 책에서는 아나콘다 Anaconda라는 배포판을 이용하기를 권함
    - 배포판: 한 번에 설치할 수 있도록 필요한 라이브러리 등을 하나로 정리해둔 것
  - 넘파이, matplotlib 등 데이터 분석에 중점을 둔 배포판
  - 아나콘다 설치방법 https://youtu.be/CNGBwFcmBZM

1.3 파이썬 인터프리터

```
$ python --version
Python 3.5.2 :: Anaconda 4.2.0 (x86_64)
```

- 이처럼 “Python 3.x.x”라 표시되면 파이썬 3가 제대로 설치된 것

```
$ python
Python 3.5.2 |Anaconda 4.2.0 (x86_64)| (default, Jul  2 2016, 17:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- 파이썬 인터프리터는 ‘대화 모드’라 하여, 개발자와 파이썬이 대화하듯 프로그래밍할 수 있음

```
>>> 1 + 2
3
```

- 이처럼 파이썬 인터프리터에서는 대화식으로 프로그래밍할 수 있음

1.3.1 산술 연산

- 덧셈과 곱셈 등의 산술 연산은 다음과 같이 할 수 있음

```
>>> 1 - 2
-1
>>> 4 * 5
20
>>> 7 / 5
1.4
>>> 3 ** 2
9
```

1.3.2 자료형

- 자료형(data type): 데이터의 성질을 나타내는 것

```
>>> type(10)
<class 'int'>
>>> type(2.718)
<class 'float'>
>>> type("hello")
<class 'str'>
```

1.3.3 변수

- x와 y 등의 알파벳을 사용하여 변수(variable)를 정의
- 변수를 사용하여 계산하거나 변수에 다른 값을 대입할 수 있음

```
>>> x = 10      # 초기화
>>> print(x)    # x의 값 출력
10
>>> x = 100     # 변수에 값 대입
>>> print(x)
100
>>> y = 3.14
>>> x * y
```

1.3.4 리스트

- [ ] 안의 수를 인덱스(색인)라 하며 인덱스는 0부터 시작
- 원소에 접근할 때는 a[0]

```
>>> a = [1, 2, 3, 4, 5] # 리스트 생성
>>> print(a)  # 리스트의 내용 출력
[1, 2, 3, 4, 5]
>>> len(a)    # 리스트의 길이 출력
5
>>> a[0]      # 첫 원소에 접근
1
>>> a[4]      # 다섯 번째 원소에 접근
5
>>> a[4] = 99 # 값 대입
>>> print(a)
[1, 2, 3, 4, 99]
```

1.3.5 딕셔너리

- 리스트는 인덱스 번호로 0, 1, 2, … 순으로 값 저장
- 딕셔너리(dictionary)는 키 key와 값 value을 한 쌍으로 저장

```
>>> me = {'height':180}      # 딕셔너리 생성
>>> me['height']             # 원소에 접근
180
>>> me['weight'] = 70        # 새 원소 추가
>>> print(me)
{'weight': 70, 'height': 180}
```

1.3.6 bool

- 파이썬에는 bool(불 혹은 불리언)이라는 자료형
- True(참)와 False (거짓)라는 두 값 중 하나를 취함

```
>>> hungry = True       # 배가 고프다.
>>> sleepy = False      # 졸리지 않다.
>>> type(hungry)
<class 'bool'>
>>> not hungry
False
>>> hungry and sleepy   # 배가 고프다 그리고 졸리지 않다.
False
>>> hungry or sleepy    # 배가 고프다 또는 졸리지 않다.
True
```

1.3.7 if문

- 들여쓰기는 지난 조건(if hungry)이 충족될 때 실행되는 코드를 표현

```
>>> hungry = True
>>> if hungry:
...     print("I'm hungry")
...
I'm hungry
>>> hungry = False
>>> if hungry:
...     print("I'm hungry")      # 들여쓰기는 공백 문자로
... else:
...     print("I'm not hungry")
...     print("I'm sleepy")
...
I'm not hungry
I'm sleepy
```

1.3.8 for문

- 반복(루프) 처리에는 for 문 사용

```
>>> for i in[1, 2, 3]:
...     print(i)
...
1
2
3
```

1.3.9 함수

- 특정 기능을 수행하는 일련의 명령들을 묶어 하나의 함수(function)로 정의

```
>>> def hello():
...     print("Hello World!")
...
>>> hello()
Hello World!

>>> def hello(object):
...     print("Hello " + object + "!")
...
>>> hello("cat")
Hello cat!
```

1.4.1 파일로 저장하기

- 텍스트 편집기를 열고 hungry.py 파일 작성

```
print("I'm hungry!")
```

- hungry.py 파일 실행 방법

```
$ cd ~/deep-learning-from-scratch/ch01  # 디렉터리로 이동
$ python hungry.py
I'm hungry!
```

1.4.2 클래스

- 클래스(class): 개발자가 직접 정의한 독자적인 자료형
- 클래스 정의에는 _ _init_ _라는 특별한 메서드가 있는데, 클래스를 초기화하는 방법을 정의. 이 초기화용 메서드를 생성자(constructor)라고 함

```
class Man:
    def __init__(self, name):
        self.name = name
        print("Initialized!")

    def hello(self):
        print("Hello " + self.name + "!")

    def goodbye(self):
        print("Good-bye " + self.name + "!")

m = Man("David")
m.hello()
m.goodbye()
```

1.5.1 넘파이 가져오기

- 넘파이는 외부 라이브러리, '외부'는 표준 파이썬에는 포함되지 않는다는 것
- 넘파이 라이브러리를 쓸 수 있도록 가져와야 import 해야 함

```
>>> import numpy as np
```

1.5.2 넘파이 배열 생성하기

- 넘파이 배열을 만들 때는 np.array() 메서드 이용
- np.array()는 파이썬의 리스트를 인수로 받아 넘파이 라이브러리가 제공하는 특수한 형태의 배열(numpy.ndarray) 반환

```
>>> x = np.array([1.0, 2.0, 3.0])
>>> print(x)
[1. 2. 3.]
>>> type(x)
<class 'numpy.ndarray'>
```

1.5.3 넘파이의 산술 연산


```
>>> x = np.array([1.0, 2.0, 3.0])
>>> y = np.array([2.0, 4.0, 6.0])
>>> x + y  # 원소별 덧셈
array([ 3.,  6.,  9.])
>>> x - y
array([-1., -2., -3.])
>>> x * y  # 원소별 곱셈
array([ 2.,  8., 18.])
>>> x / y
array([0.5, 0.5, 0.5])
```

- '원소별': element-wise, '원소별 곱셈': element-wise product
- 주의할 점
  - 배열 x 와 y 의 원소 수가 같다는 것(둘 다 원소를 3 개씩 갖는 1 차원 배열)
  - 배열 x 와 y 의 원소 수가 같다면 산술 연산은 각 원소에 대해서 행해 짐
  - 원소 수가 다르면 오류가 발생하니 원소 수 맞추기가 중요

1.5.4 넘파이의 N차원


```
>>> A = np.array([[1, 2], [3, 4]])
>>> print(A)
[[1 2]
 [3 4]]
>>> A.shape
(2, 2)
>>> A.dtype
dtype('int64')
```

- 행렬의 형상(shape): 행렬의 크기, ex) A.shape = A 행렬의 크기 = 2X2
- 행렬 원소의 자료형(dtype), ex) A.dtype = A행렬 원소의 자료형 = int64

```
>>> B = np.array([[3, 0], [0, 6]])
>>> A + B
array([[ 4,  2],
       [ 3, 10]])
>>> A * B
array([[ 3,  0],
       [ 0, 24]])
```

- 형상이 같은 행렬끼리면 행렬의 산술 연산도 원소별 연산

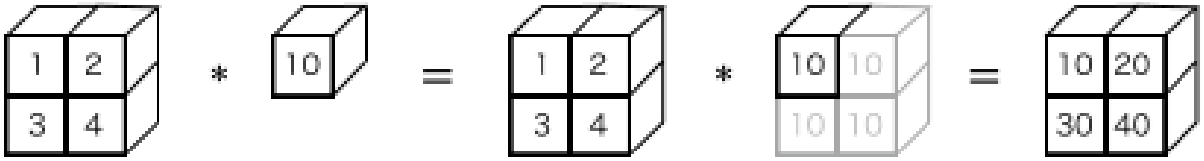
1.5.4 넘파이의 N차원

- 수학에서
  - 1차원 배열은 벡터(vector)
  - 2차원 배열은 행렬(matrix)
  - 벡터와 행렬을 일반화 한 것을 텐서(tensor)라고 함

1.5.5 브로드캐스트

- 넘파이에서는 형상이 다른 배열끼리도 계산 가능

그림 1-1 브로드캐스트의 예 : 스칼라값인 10이 2×2 행렬로 확대된다.



다른 예를 살펴봅시다.

```
>>> A = np.array([[1, 2], [3, 4]])
>>> B = np.array([10, 20])
>>> A * B
array([[ 10, 40],
       [ 30, 80]])
```

- 브로드캐스트(broadcast): 스칼라 값이 행렬의 형상만큼 확대된 후 연산이 이뤄지는 것

1.5.6 원소 접근


```
>>> X = np.array([[51, 55], [14, 19], [0, 4]])
>>> print(X)
[[51 55]
 [14 19]
 [ 0  4]]
>>> X[0]        # 0행
array([51, 55])
>>> X[0][1]     # (0, 1) 위치의 원소
55
```

for 문으로도 각 원소에 접근할 수 있습니다.

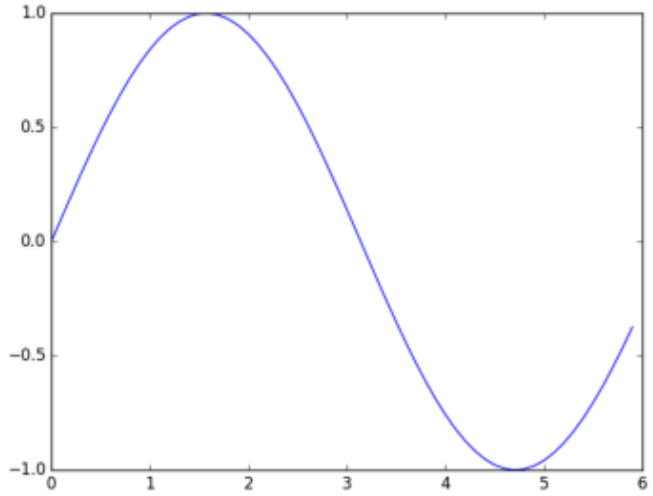
```
>>> for row in X:
...     print(row)
...
[51 55]
[14 19]
[0 4]
```

1.6.1 단순한 그래프 그리기


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

# 데이터 준비
x = np.arange(0, 6, 0.1)    # 0에서 6까지 0.1 간격으로 생성
y = np.sin(x)

# 그래프 그리기
plt.plot(x, y)
plt.show()
```



- 넘파이의 arange 메서드로 [0 , 0.1 , 0.2 , ..., 5.8 , 5.9]라는 데이터를 생성하여 변수 x 에 할당
- x의 각 원소에 넘파이의 sin 함수인 np.sin()을 적용하여 변수 y에 할당
- x 와 y를 인수로 plt.plot 메서드를 호출하여 그래프 생성
- plt.show()를 호출하여 그래프를 화면에 출력

1.6.2 pyplot의 기능

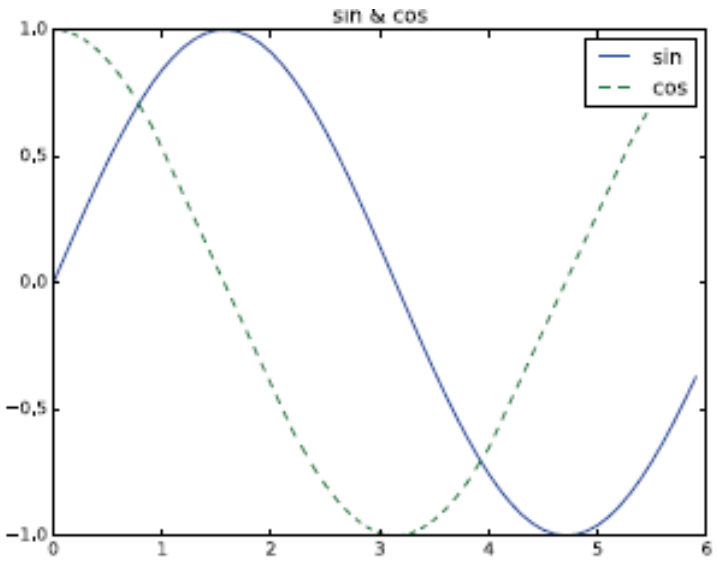

```
import numpy as np
import matplotlib.pyplot as plt

# 데이터 준비
x = np.arange(0, 6, 0.1) # 0에서 6까지 0.1 간격으로 생성
y1 = np.sin(x)
y2 = np.cos(x)

# 그래프 그리기
plt.plot(x, y1, label="sin")

# cos 함수는 점선으로 그리기
plt.plot(x, y2, linestyle = "--", label="cos")

plt.xlabel("x") # x축 이름
plt.ylabel("y") # y축 이름
plt.title('sin & cos')
plt.legend()
plt.show()
```



1.6.3 이미지 표시하기

- 이미지를 읽을 때는 matplotlib.image 모듈의 imread ()메서드 이용
- 이미지를 표시해주는 메서드 imshow()

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

# 이미지 읽어오기
img = imread('../dataset/cactus.png')
plt.imshow(img)

plt.show()
```

그림 1-5 이미지 표시하기