

## CH. 10. 딥러닝

# Section 01 딥러닝 개요

---

## □ 규칙기반의 한계

- 앞에서 구현한 OpenCV를 이용한 영상처리 알고리즘에 대해 생각해보자.
- 교통약자 보호 표지판을 인식하는 프로그램은 영상에서 SIFT 특징을 추출하고 빠른 매칭을 위해 kd 트리를 사용한다.
- 이 알고리즘은 모두 사람의 합리적 사고를 통해 개발되었다.
- 사람은 데이터를 면밀히 관찰하여 정교하게 알고리즘을 개발하고 꾸준히 개선한다.
- 사람이 만든 이런 특징과 매칭 알고리즘을 수작업(hand-crafted) 특징과 수작업 알고리즘이라고 한다.
- 수작업으로 개발한 결과는 대부분 규칙(rule)으로 표현한다.
- 고전적 컴퓨터 비전은 이런 과정으로 문제를 해결하기 때문에 규칙 기반(rule-based)이라고 한다.
- 규칙 기반을 사용하는 고전적 컴퓨터 비전은 사람의 노력으로 일정 수준의 성능은 달성하지만 그 이상을 돌파하지 못하는 결정적인 한계가 있다.

# Section 01 딥러닝 개요

---

## □ 규칙기반의 한계

- 컴퓨터 비전 방법론은 크게 규칙 기반과 기계학습(machine learning)으로 나눌 수 있다.
- 기계학습은 주어진 문제 도메인에서 데이터를 수집하고 모델을 학습하는 과정을 거쳐 문제를 해결한다.
- 기계학습 방법론에는 다양한 학습 모델이 있는데, 크게 신경망 모델과 신경망이 아닌 모델로 구분할 수 있다.
- 비신경망에 속하는 대표적 모델은 SVM, 결정 트리, 랜덤 포레스트다.
- 신경망에서는 대략 2010년 이전에는 얇은 신경망 모델을 사용했는데 이후에는 훨씬 많은 층을 배치한 깊은 신경망 모델로 발전한다.
- 깊은 신경망 구조를 설계하고 학습하고 예측에 사용하는 기술을 통틀어 딥러닝(deep learning)이라고 한다.
- 딥러닝은 뛰어난 성능으로 인해 다른 기계학습 모델을 제치고 컴퓨터 비전의 주류 기술로 자리잡았다.

# Section 01 딥러닝 개요

---

## □ 규칙기반의 한계

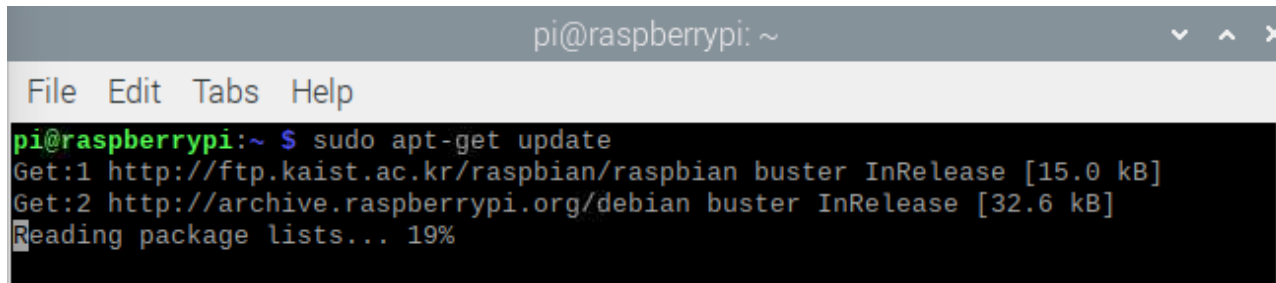
- 신경망은 얇은 모델로 출발하여 점점 깊어지며 발전해왔다.
- 이 장에서는 이런 과정에 따라 얇은 신경망인 퍼셉트론에서 시작해 다층 퍼셉트론과 깊은 다층 퍼셉트론 순으로 설명한다.
- 퍼셉트론은 단순하여 기본 개념과 연산을 이해하기 쉬울 뿐 아니라 여전히 딥러닝 모델의 주요 부품으로 활용되기 때문이다.
- 딥러닝의 영향력은 컴퓨터 비전을 넘어 아주 넓다.
- 자연어 처리(NLP: Natural Language Processing)는 영어나 한국어와 같은 자연어를 처리하여 언어 번역이나 챗봇 등을 만드는 기술이다.
- 딥러닝은 자연어 처리에서도 주류 기술로 활용된다.
- 지능 게임 분야에서도 주류 기술이 되었는데, 이세돌 기사를 이긴 알파고는 딥러닝 기술로 만들었다.
- 최근에 인공지능이 전 세계의 주목을 받게 된 중심에 딥러닝 기술이 있다.

# Section 01 딥러닝 개요

## □ 딥러닝 체험

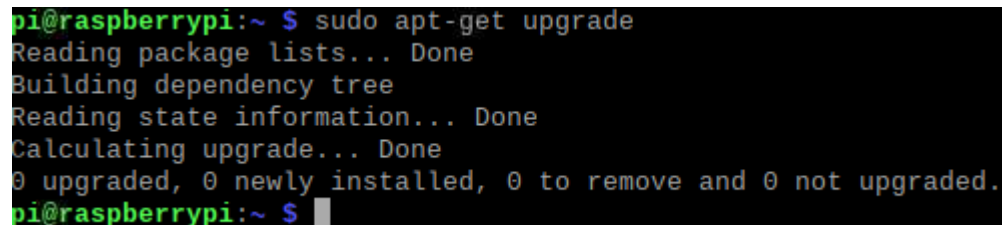
- 라즈베리파이에서 tensorflow를 설치하기 위해 터미널에 다음의 명령들을 입력한다.

```
$ sudo apt-get update
```



```
pi@raspberrypi:~ $ sudo apt-get update
Get:1 http://ftp.kaist.ac.kr/raspbian/raspbian buster InRelease [15.0 kB]
Get:2 http://archive.raspberrypi.org/debian buster InRelease [32.6 kB]
Reading package lists... 19%
```

```
$ sudo apt-get upgrade
```



```
pi@raspberrypi:~ $ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $
```

# Section 01 딥러닝 개요

## □ 딥러닝 체험

- 라즈베리파이에서 tensorflow를 설치하기 위해 터미널에 다음의 명령들을 입력한다.

```
$ sudo pip uninstall tensorflow
```

```
pi@raspberrypi:~ $ sudo pip uninstall tensorflow  
Skipping tensorflow as it is not installed.  
pi@raspberrypi:~ $
```

```
$ sudo pip3 uninstall tensorflow
```

```
pi@raspberrypi:~ $ sudo pip3 uninstall tensorflow  
Skipping tensorflow as it is not installed.  
pi@raspberrypi:~ $
```

```
$ sudo apt-get install gfortran
```

```
pi@raspberrypi:~ $ sudo apt-get install gfortran  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
gfortran is already the newest version (4:8.3.0-1+rpi2).  
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.  
pi@raspberrypi:~ $
```

# Section 01 딥러닝 개요

## □ 딥러닝 체험

- 라즈베리파이에서 tensorflow를 설치하기 위해 터미널에 다음의 명령들을 입력한다.

```
$ sudo apt-get install libhdf5-dev libc-ares-dev libeigen3-dev
pi@raspberrypi:~ $ sudo apt-get install libhdf5-dev libc-ares-dev libeigen3-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libeigen3-dev is already the newest version (3.3.7-1).
The following additional packages will be installed:
  hdf5-helpers libaec-dev libaec0 libhdf5-103 libhdf5-cpp-103 libsz2
Suggested packages:
  libhdf5-doc
The following NEW packages will be installed:
  hdf5-helpers libaec-dev libaec0 libc-ares-dev libhdf5-103 libhdf5-cpp-103
  libhdf5-dev libsz2
0 upgraded, 8 newly installed, 0 to remove and 0 not upgraded.
Need to get 153 kB/3,837 kB of archives.
After this operation, 13.9 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

# Section 01 딥러닝 개요

## □ 딥러닝 체험

- 라즈베리파이에서 tensorflow를 설치하기 위해 터미널에 다음의 명령들을 입력한다.

```
$ sudo apt-get install openmpi-bin libopenmpi-dev
```

```
pi@raspberrypi:~ $ sudo apt-get install openmpi-bin libopenmpi-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

```
$ sudo apt-get install liblapack-dev cython
```

```
pi@raspberrypi:~ $ sudo apt-get install liblapack-dev cython
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

```
$ sudo pip3 install keras_applications
```

```
pi@raspberrypi:~ $ sudo pip3 install keras_applications
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting keras_applications
```



# Section 01 딥러닝 개요

## □ 딥러닝 체험

- 라즈베리파이에서 tensorflow를 설치하기 위해 터미널에 다음의 명령들을 입력한다.

```
$ sudo pip3 install keras_preprocessing
```

```
pi@raspberrypi:~ $ sudo pip3 install keras_preprocessing
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting keras_preprocessing
  Downloading https://files.pythonhosted.org/packages/79/4c/7c3275a01e12ef9368a892926ab932b33b
b13d55794881e3573482b378a7/Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42kB)
    100% |#####| 51kB 465kB/s
Requirement already satisfied: six>=1.9.0 in /usr/lib/python3/dist-packages (from keras_prepro
cessing) (1.12.0)
Requirement already satisfied: numpy>=1.9.1 in /usr/lib/python3/dist-packages (from keras_prep
rocessing) (1.16.2)
```

```
$ sudo pip3 install -U --user six wheel mock
```

```
pi@raspberrypi:~ $ sudo pip3 install -U --user six wheel mock
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting six
  Downloading https://files.pythonhosted.org/packages/d9/5a/e7c31adbe875f2abbb91bd84cf2dc52d79
2b5a01506781dbcf25c91daf11/six-1.16.0-py2.py3-none-any.whl
Collecting wheel
```

# Section 01 딥러닝 개요

## □ 딥러닝 체험

- 라즈베리파이에서 tensorflow를 설치하기 위해 터미널에 다음의 명령들을 입력한다.

```
$ sudo -H pip3 install pybind11
```

```
pi@raspberrypi:~ $ sudo -H pip3 install h5py
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.8.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/lib/python3/dist-packages (from h5py) (1.16.2)
```

```
$ sudo -H pip3 install --upgrade setuptools
```

```
pi@raspberrypi:~ $ sudo -H pip3 install --upgrade setuptools
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting setuptools
  Downloading https://files.pythonhosted.org/packages/c2/8b/abb577ca6ab2c71814d535b1ed1464c5f4aaefela31bbeb85013eb9b2401/setuptools-66.1.1-py3-none-any.whl (1.3MB)
    99% |#####| 1.3MB 2.8MB/s eta 0:00:01
```

```
$ sudo pip3 install --upgrade https://github.com/lhelontra/tensorflow-on-arm/releases/download/v2.4.0/tensorflow-2.4.0-cp37-none-linux\_armv7l.whl (--no-cache-dir)
```

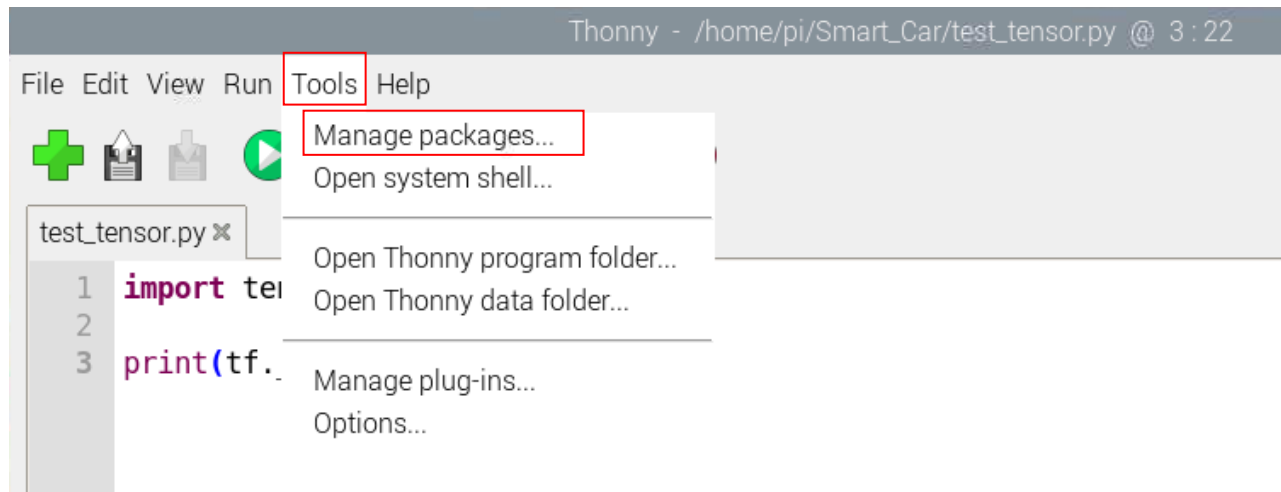
```
pi@raspberrypi:~ $ sudo wget https://github.com/lhelontra/tensorflow-on-arm/releases/download/v2.4.0/tensorflow-2.4.0-cp37-none-linux_armv7l.whl
--2023-01-28 10:13:38-- https://github.com/lhelontra/tensorflow-on-arm/releases/download/v2.4.0/tensorflow-2.4.0-cp37-none-linux_armv7l.whl
Resolving github.com (github.com)... 20.200.245.247
Connecting to github.com (github.com)|20.200.245.247|:443... connected.
```

```
$ sudo reboot
```

# Section 01 딥러닝 개요

## □ 딥러닝 체험

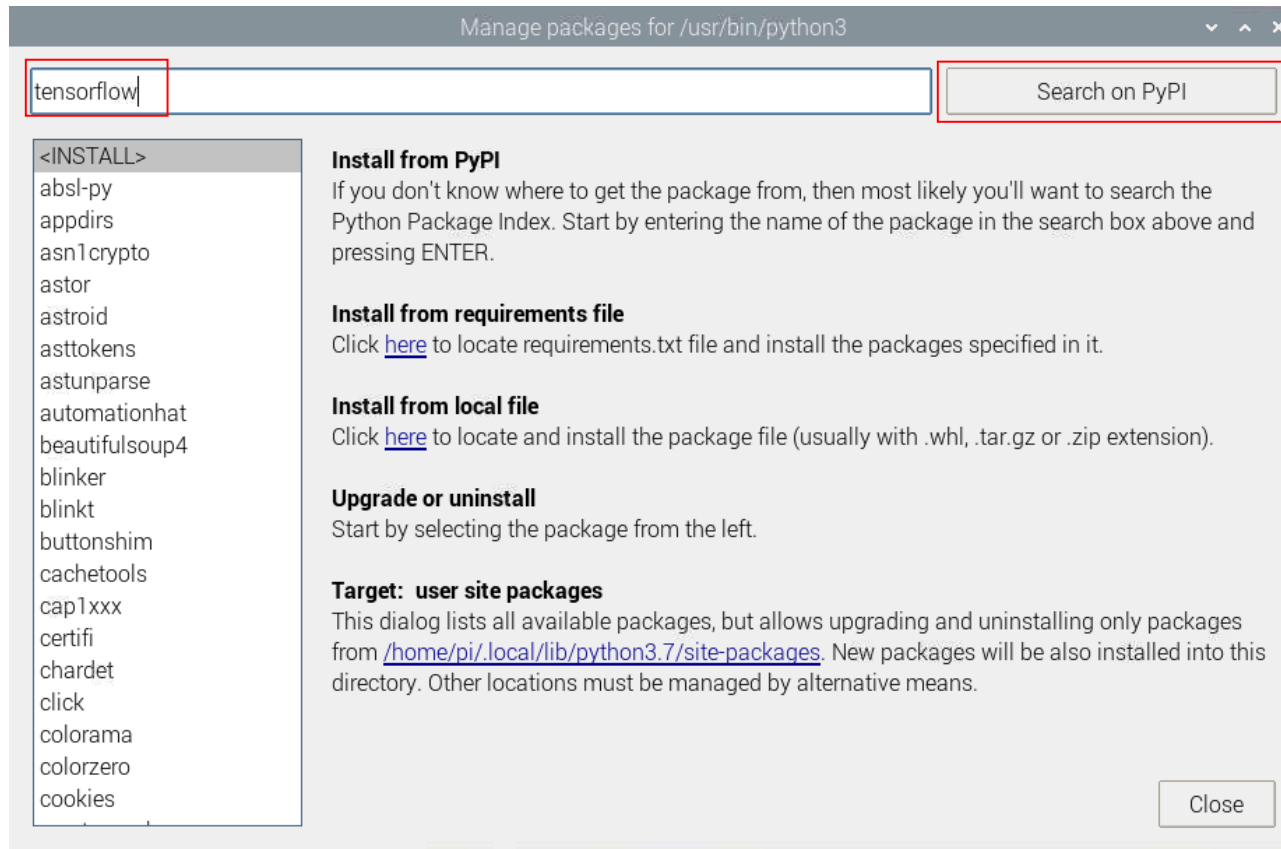
- Thonny python IDE를 실행하고 [Tools] -> [Manage packages]를 선택한다.



# Section 01 딥러닝 개요

## □ 딥러닝 체험

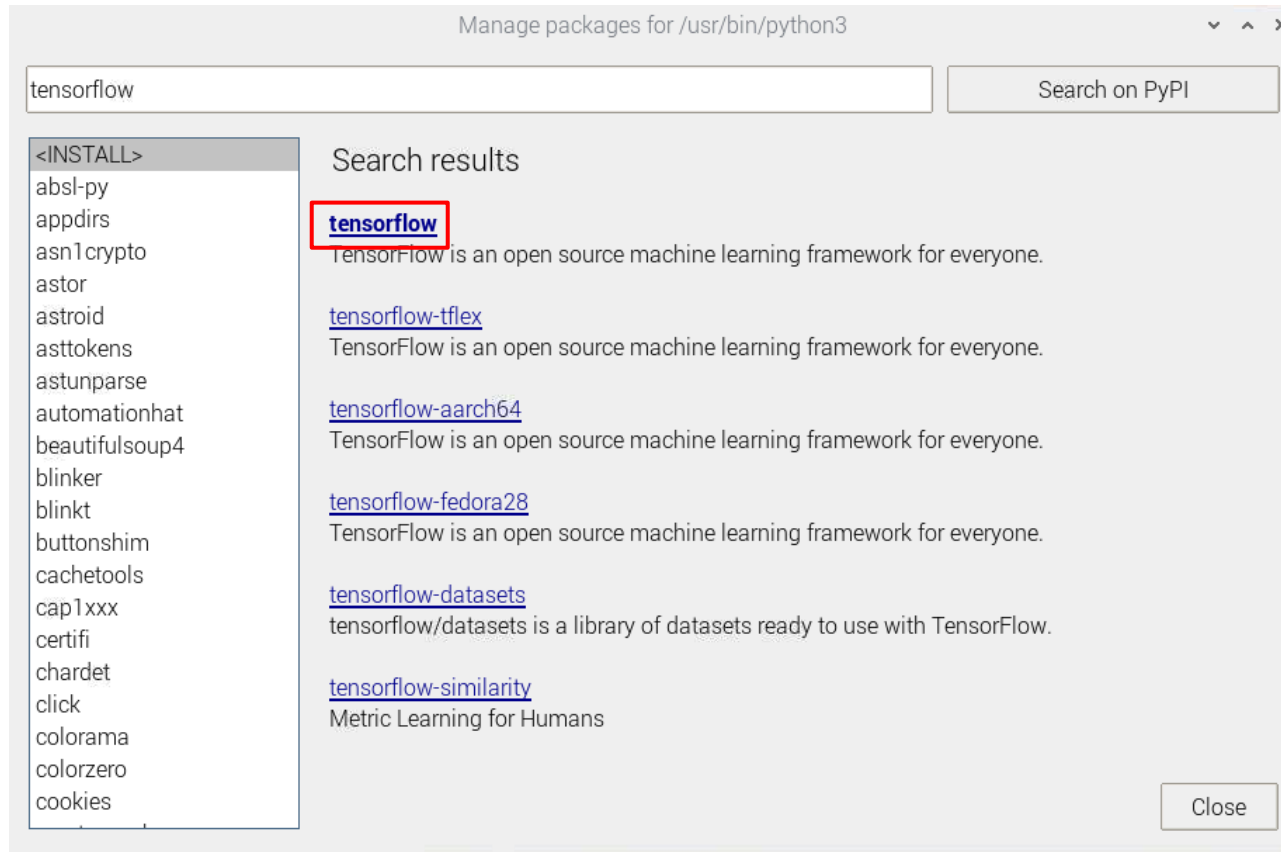
- 텍스트 입력창에 tensorflow를 입력하고 [Search on PyPI]를 클릭한다.



# Section 01 딥러닝 개요

## □ 딥러닝 체험

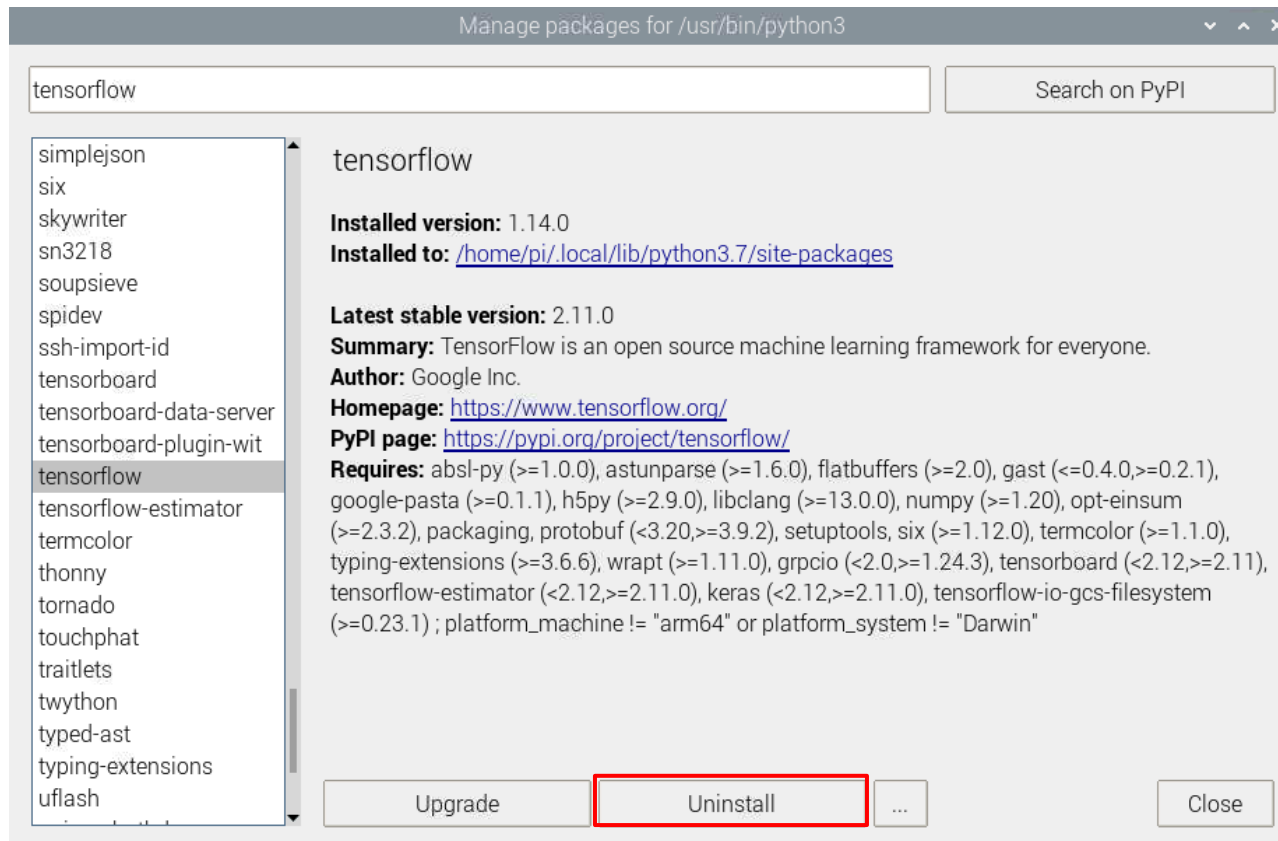
- 화면에서 tensorflow를 클릭한다.



# Section 01 딥러닝 개요

## □ 텐서플로우 설치

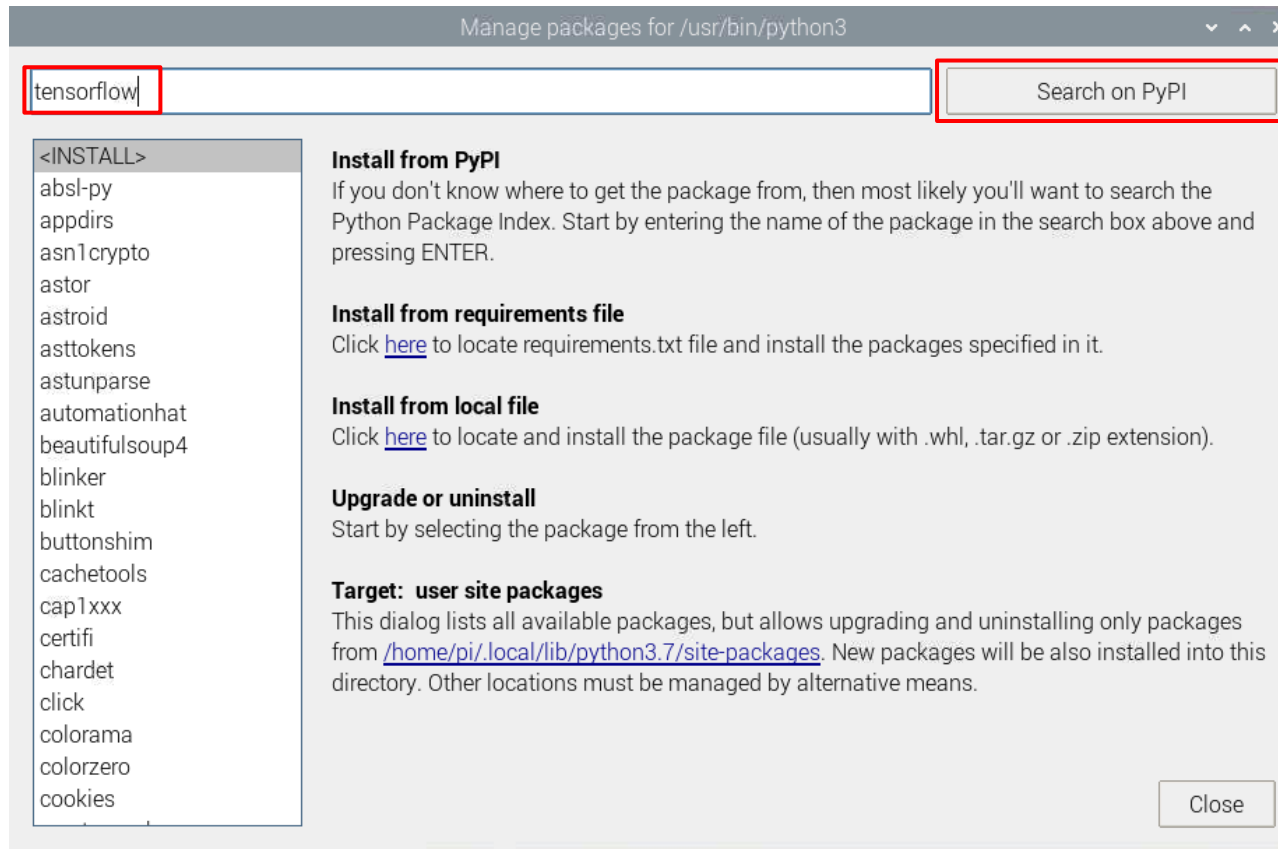
- 화면에서 [uninstall] 버튼을 클릭한다.



# Section 01 딥러닝 개요

## □ 딥러닝 체험

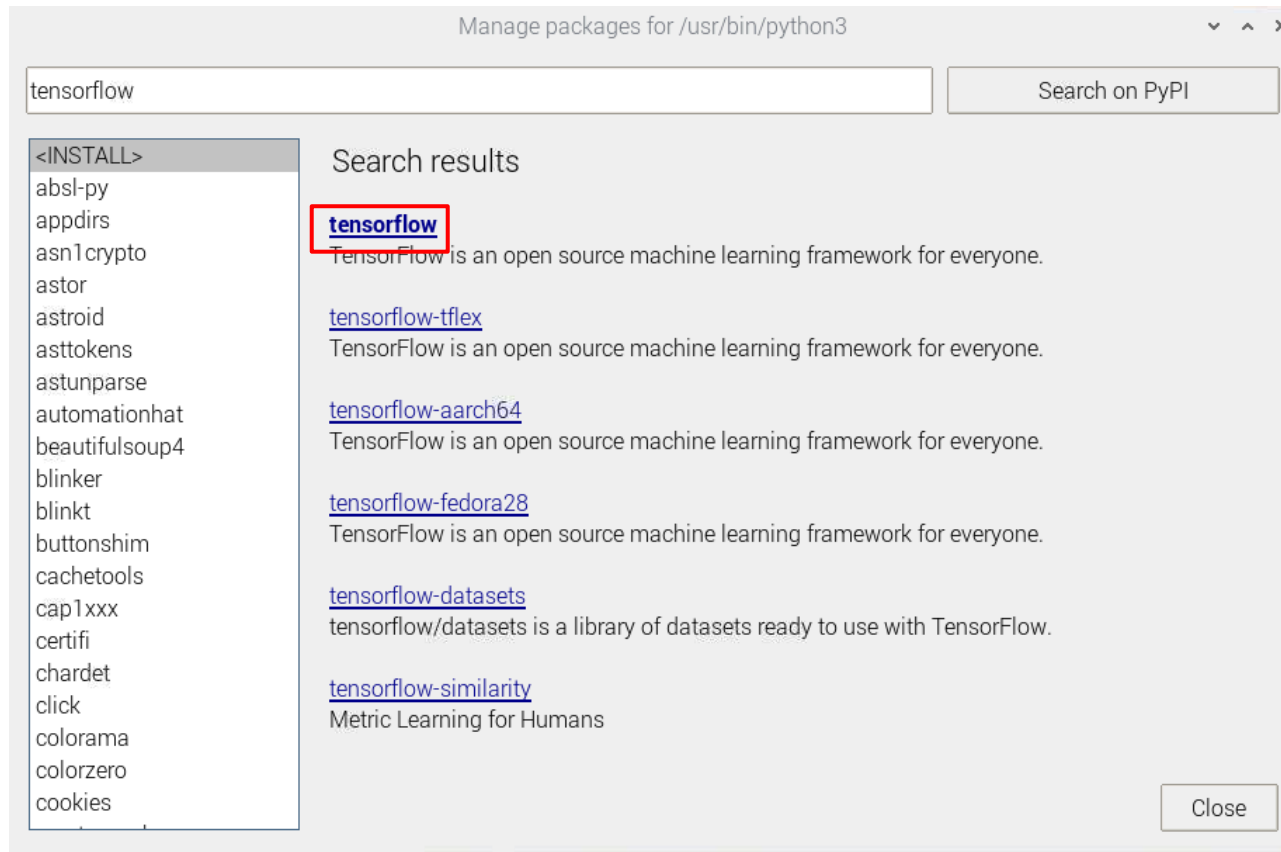
- 다시 텍스트 입력창에 tensorflow를 입력하고 [Search on PyPI]를 클릭한다.



# Section 01 딥러닝 개요

## □ 딥러닝 체험

- 화면에서 tensorflow를 클릭한다.

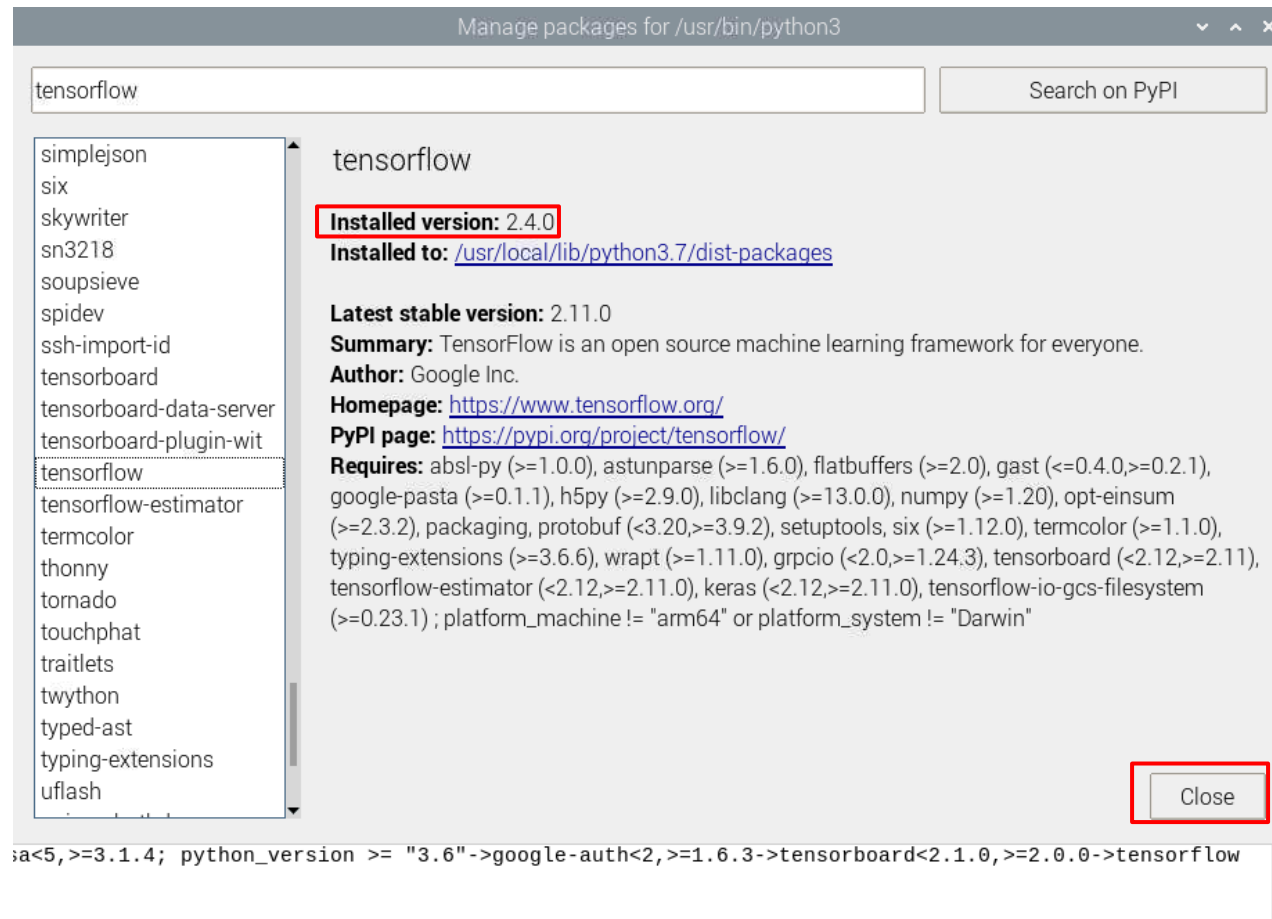




# Section 01 딥러닝 개요

## □ 딥러닝 체험

- 텐서플로우의 버전이 바뀌어 있는 것을 확인할 수 있다.



# Section 01 딥러닝 개요

---

## □ 딥러닝 체험

- 파이썬 셸에서 다음과 같이 입력한다.

```
>> import tensorflow as tf  
>> print(tf.__version__)
```

```
Python 3.7.3 (/usr/bin/python3)
```

```
>>> import tensorflow as tf
```

```
>>> print(tf.__version__)
```

```
2.4.0
```

```
>>> |
```

---

# Section 01 딥러닝 개요

---

## □ 데이터와 텐서

- 딥러닝 모델 학습은 고성능의 컴퓨팅 자원이 필요하기 때문에, 라즈베리 파이에서의 실습은 어렵다.
- 따라서 앞으로의 모델 학습은 구글 코랩을 이용해서 수행하겠다.
- 주행하여 획득한 데이터의 학습모델을 만들기 위해서 파이썬의 tensorflow, keras를 사용한다.
- 구글 코랩 (colab)은 웹상에서 파이썬 개발환경을 제공해준다.
- 연산을 위한 CPU뿐만 아니라 GPU, TPU까지 딥러닝 연산을 위한 고성능의 기능을 제공한다.
- PC에서 동일한 환경을 구성하기 위해서는 높은 비용이 발생하고 라이브러리 설치가 복잡하여 손쉽게 사용할 수 있는 구글 코랩 서비스를 이용하여 딥러닝 학습모델을 생성한다.

# Section 01 딥러닝 개요

## □ 데이터와 텐서

- 구글 코랩을 사용하는 방법에 대해 알아보자.
- 구글에서 “colab”을 검색 후 아래 사이트에 접속한다.
- 또는 <https://colab.research.google.com> 주소를 입력하여 접속한다.
- 브라우저는 [크롬] 브라우저의 사용을 추천한다.



colab



전체

이미지

동영상

쇼핑

지도

더보기

도구

검색결과 약 61,800,000개 (0.29초)

<https://colab.research.google.com> > ...

**Colab - Google**

Colaboratory(줄여서 'Colab'이라고 함)을 통해 브라우저 내에서 Python 스크립트를 작성하고 실행할 수 있습니다. 구성이 필요하지 않음; 무료로 GPU 사용 ...

<https://colab.research.google.com> > ...

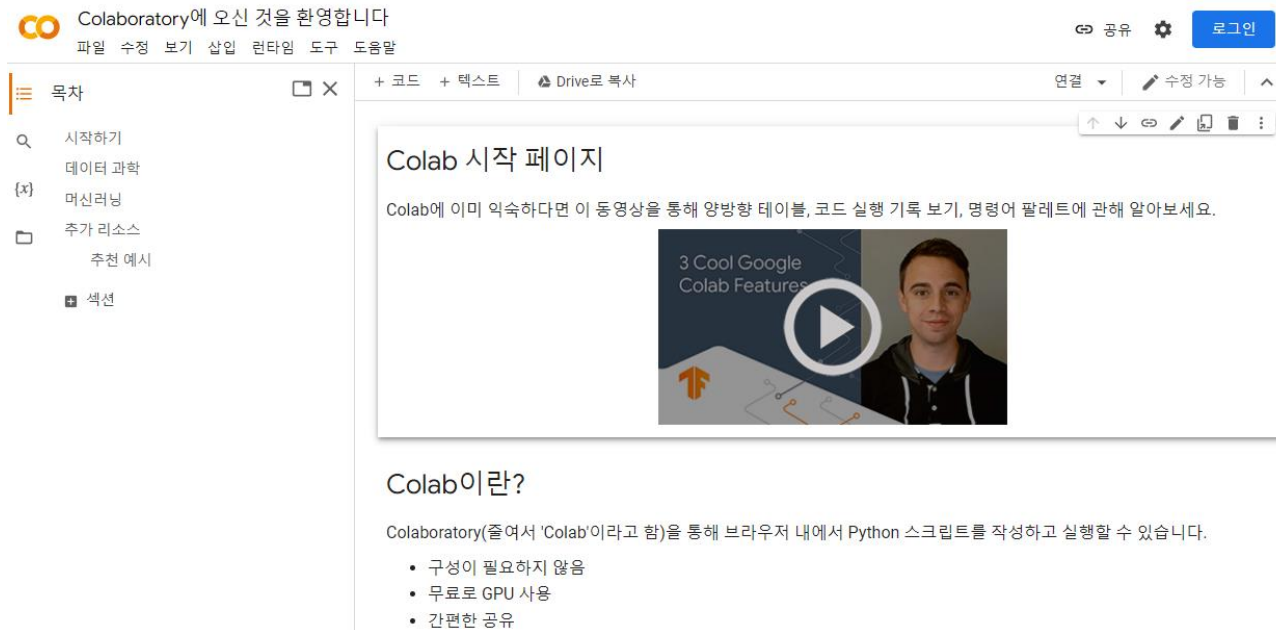
**Google Colab**

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your own Colab ...

# Section 01 딥러닝 개요

## □ 데이터와 텐서

- 코랩을 사용하기 위해서는 구글 계정이 필요하다.
- 구글 계정으로 로그인 후 코랩 메인페이지에 접속한다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

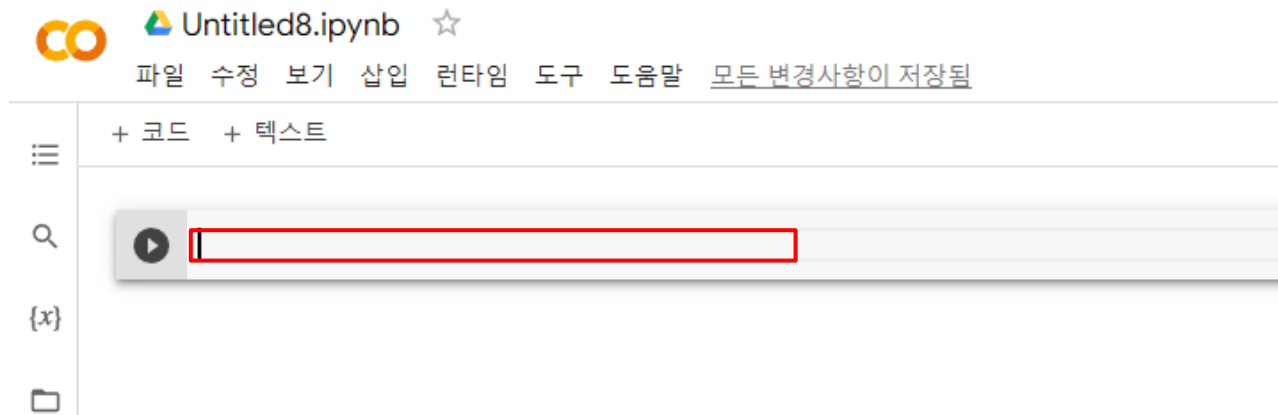
- [파일] -> [새 노트]를 열어 새로운 파이썬 프로젝트를 시작한다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

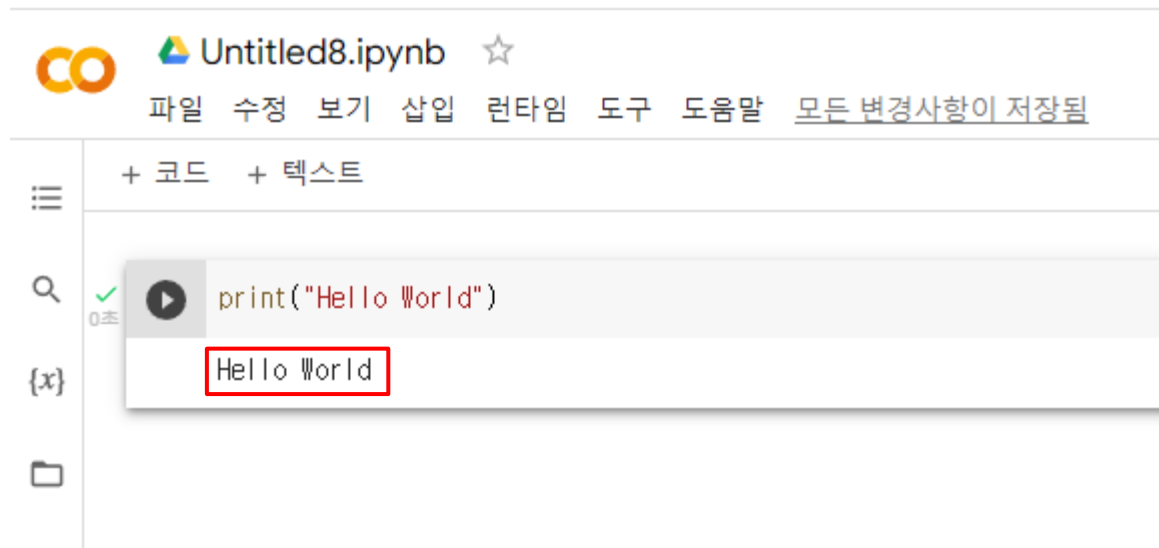
- 빨간 네모 박스부분에 코드를 작성하고 ▶ 실행 버튼을 누르면 코드의 실행이 가능하다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

- “Hello World” 를 출력하는 코드를 실행하면 다음과 같이 코드 아래에 결과가 출력된다.

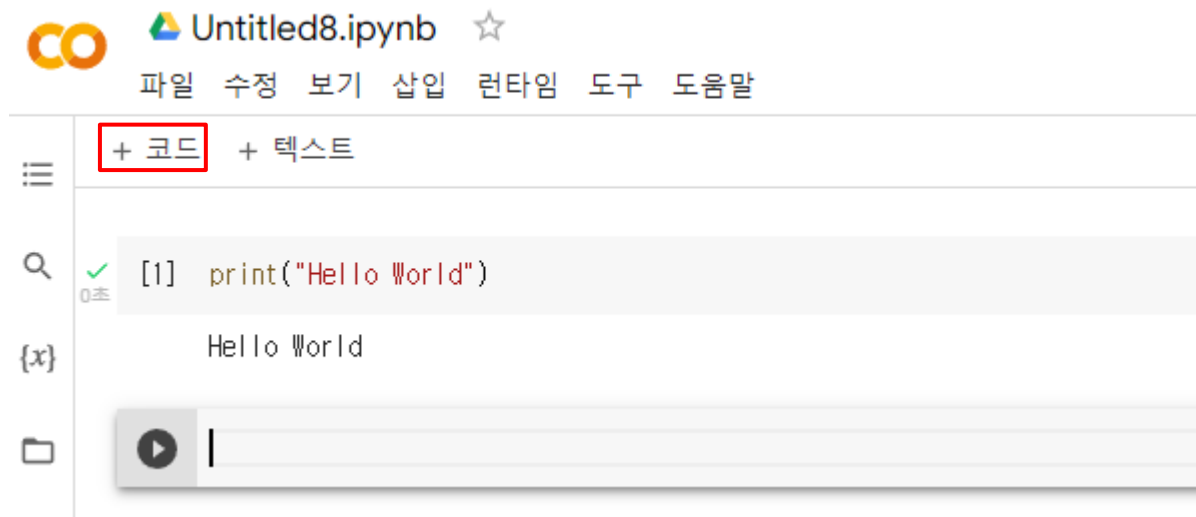




# Section 01 딥러닝 개요

## □ 데이터와 텐서

- [+코드]를 눌러 아래쪽에 [셀]을 추가할 수 있다.
- 코드를 입력하는 부분을 [셀]이라고 한다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

- 아래쪽에 생성된 [셀]에 파이썬 코드를 작성하여 실행하였다.



CO Untitled8.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

[1] `print("Hello World")`

0초

Hello World

[2] `a = 10`  
`b = 20`  
`c = a + b`  
`print(c)`

0초

30

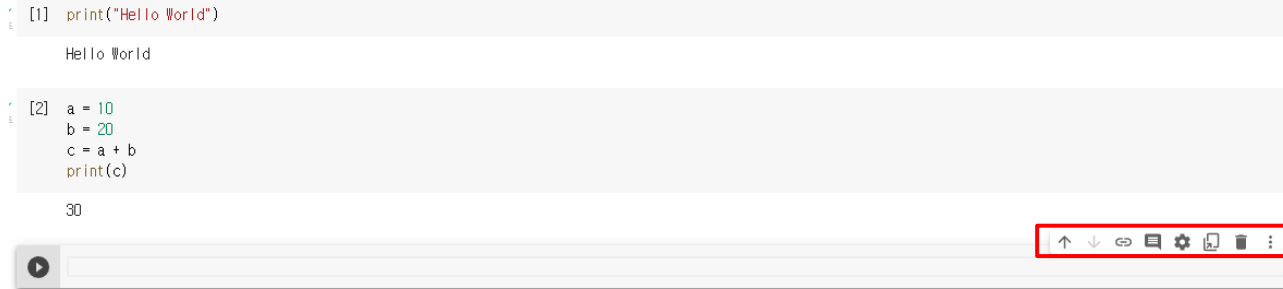
▶

# Section 01 딥러닝 개요

## □ 데이터와 텐서

- [셀] 오른쪽 위를 보면 [셀]의 추가 작업을 할 수 있다.

```
[1] print("Hello World")  
Hello World  
  
[2] a = 10  
    b = 20  
    c = a + b  
    print(c)  
30
```



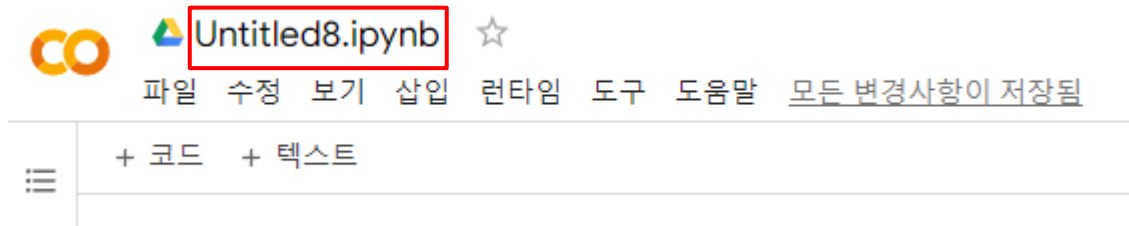
- 화살표를 눌러 [셀]을 위로 올리거나 내릴 수 있다.
- [셀]의 삭제, 댓글 등의 기능이 있다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

- 이름 부분을 클릭하여 이름을 변경할 수 있다.
- 코랩의 확장자는 .ipynb 이다.



- 코랩의 장점은 [셀]을 추가하면서 코드를 하나하나 테스트하고 다음 단계로 진행할 수 있다.
- 이전 [셀]에서 저장된 변수 등은 [런타임]이 초기화되지 전까지 값을 가지고 있다.

# Section 01 딥러닝 개요

## □ 데이터와 텐서

- 다음과 같은 코드를 추가하여 실행해보자.
- [2] 번 [셀]에서 저장되었던  $a$ ,  $b$  변수의 값이 현재 실행중인 셀에서 사용이 가능하다.
- [1], [2] 등의 번호는 [셀]을 실행한 순서의 번호이다.

✓  
0초

[1] `print("Hello World")`

▶ Hello World

✓  
0초

[2] `a = 10`  
`b = 20`  
`c = a + b`  
`print(c)`

30

✓  
0초

▶ `d = a - b`  
`print(d)`

-10

# Section 01 딥러닝 개요

## □ 데이터와 텐서

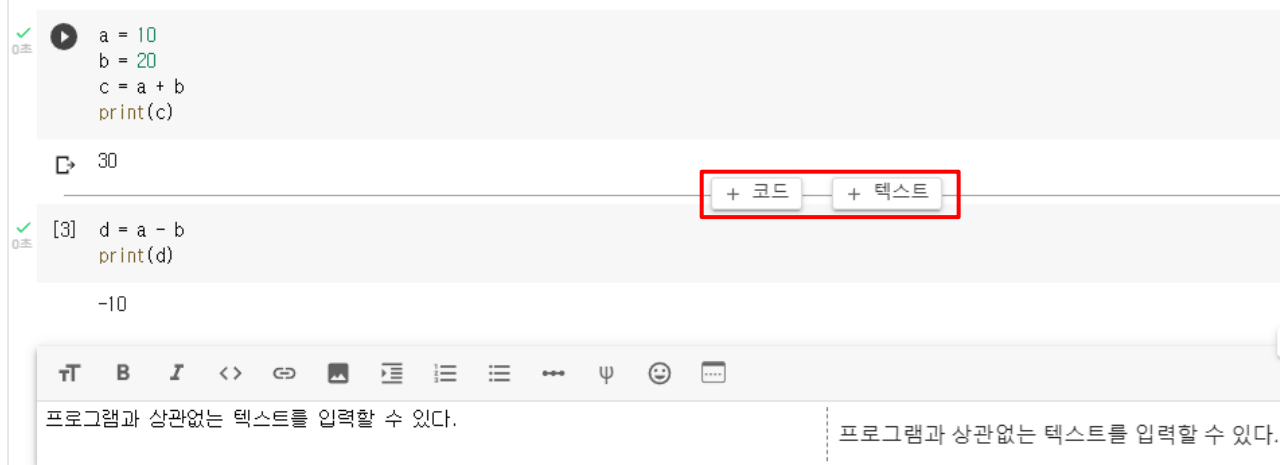
- [+ 텍스트]를 눌러 텍스트를 입력할 수도 있다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

- [셀]과 [셀] 중간에 코드나 텍스트를 추가하고 싶다면 중간에 마우스를 가져다 대면 [+코드] [+텍스트]를 추가할 수 있는 버튼이 생성된다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

- 코랩은 구글 드라이브와 연동하여 데이터를 읽고 쓸 수 있다.
- 딥러닝을 위한 데이터는 데이터의 개수와 용량이 크기 때문에 구글 드라이브와 연동하여 사용하는 경우가 많다.
- 우리가 획득한 주행사진 데이터도 개수가 많아 구글 드라이브를 사용한다.
- 구글 드라이브 연동 테스트를 위해 [파일] -> [새 노트] 버튼을 클릭하여 새 노트를 생성한다.

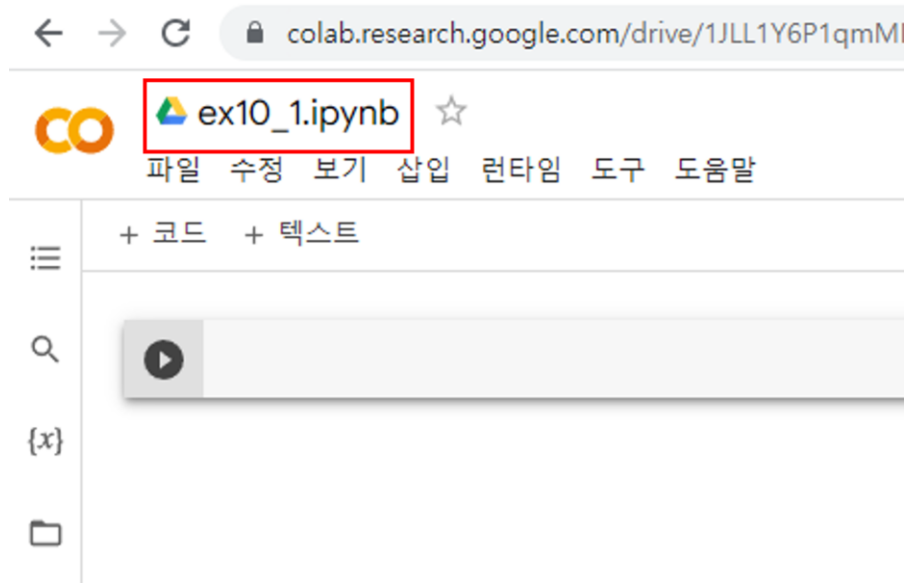




# Section 01 딥러닝 개요

## □ 데이터와 텐서

- [ex10\_1]의 이름으로 변경한다.



# Section 01 딥러닝 개요

## □ 데이터와 텐서

- ex10\_1.ipynb

```
1 import tensorflow as tf
2 import tensorflow.keras.datasets as ds
3 import matplotlib.pyplot as plt
4
5 (x_train, y_train), (x_test, y_test) = ds.mnist.load_data()
6 print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
7 plt.figure(figsize=(24, 3))
8 plt.suptitle("MNIST", fontsize=30)
9 for i in range(10):
10     plt.subplot(1, 10, i+1)
11     plt.imshow(x_train[i], cmap='gray')
12     plt.xticks([]); plt.yticks([])
13     plt.title(str(y_train[i]), fontsize=30)
14
```

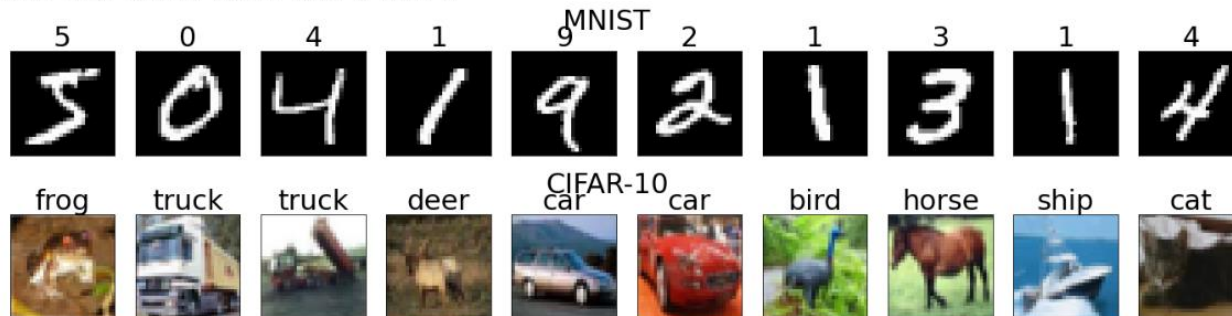
# Section 01 딥러닝 개요

## □ 데이터와 텐서

- ex10\_1.ipynb

```
15 (x_train, y_train), (x_test, y_test) = ds.cifar10.load_data()
16 print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
17 class_names = ['airplane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
18 plt.figure(figsize=(24, 3))
19 plt.suptitle('CIFAR-10', fontsize=30)
20 for i in range(10):
21     plt.subplot(1, 10, i+1)
22     plt.imshow(x_train[i])
23     plt.xticks([]); plt.yticks([])
24     plt.title(class_names[y_train[i, 0]], fontsize=30)
```

↳ (60000, 28, 28) (60000,) (10000, 28, 28) (10000,)  
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)

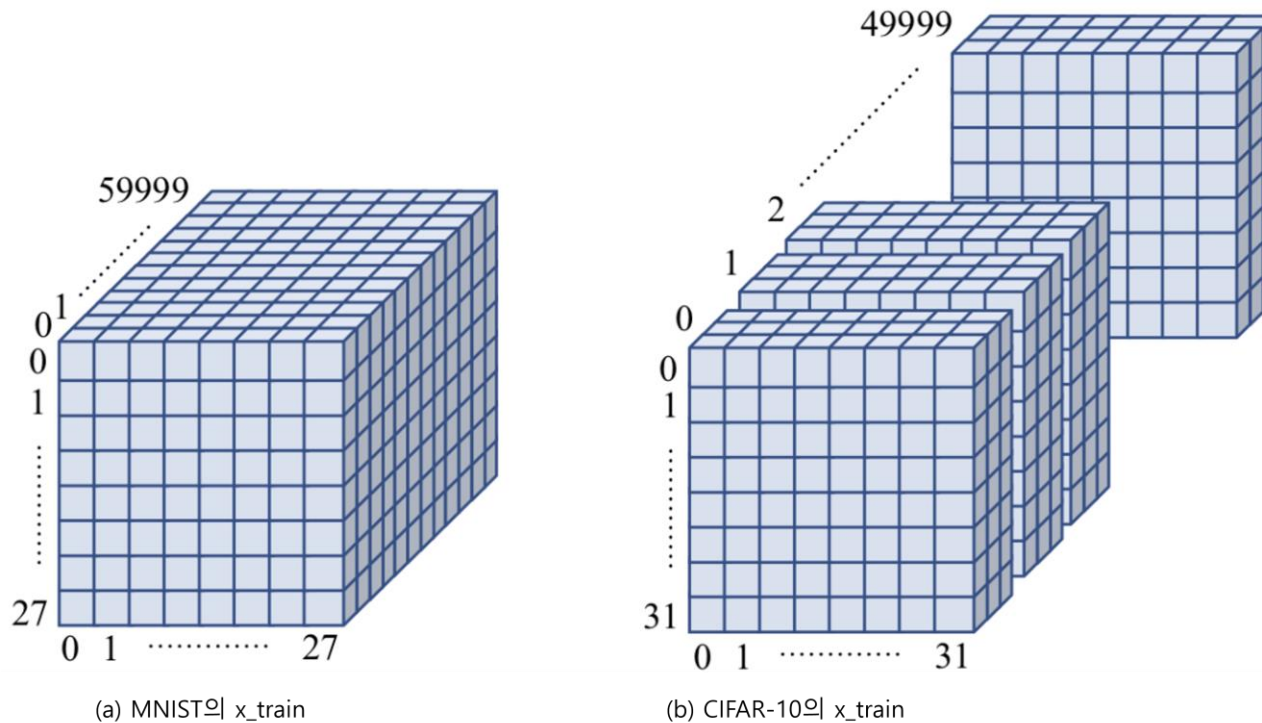


# Section 01 딥러닝 개요

## □ 데이터와 텐서

- ex10\_1.ipynb

### ➤ 예제 10-1의 데이터셋의 구조

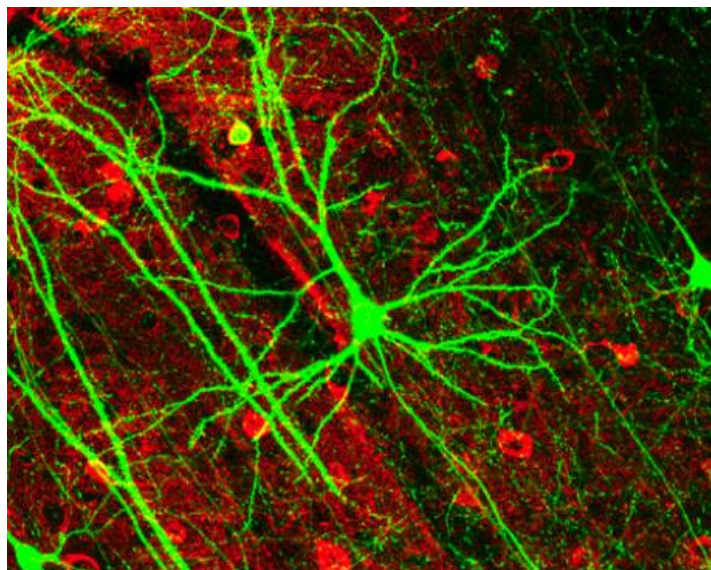


# Section 01 딥러닝 개요

---

## □ 딥러닝이란?

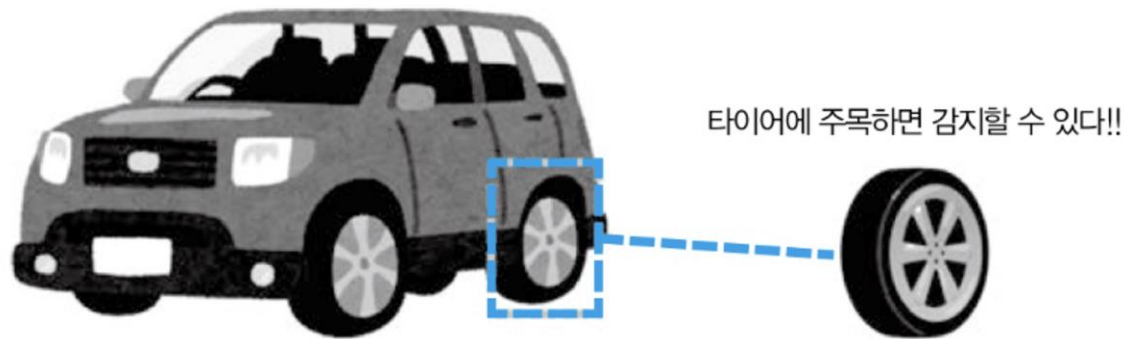
- 딥러닝은 동물의 신경망을 참고로 한 심층 신경망 모델을 사용하여 데이터의 분류나 회귀를 실시한다.
- 실제의 신경망에서 착상한 심층 신경망이지만 뇌의 신경망 재현을 목표로 하지 않고 순수하게 정밀도를 높이는 연구가 활발히 이루어지고 있다.
- 딥러닝은 기계학습의 한 파트이다.



# Section 01 딥러닝 개요

## □ 딥러닝이란?

- 딥러닝이 주목받는 이유는 일손이 많이 가는 작업이 자동화되며 정확도가 높기 때문이다.
- 예를 들어 자동차 감지 작업을 생각해보자.
- 기존의 기술을 사용할 때는 인간이 자동차의 검출에 중요한 특징(feature)을 미리 정하고 이를 정점적으로 파악할 수 있는 모델을 고려한다.
- 그에 반해 딥러닝은 그러한 특징을 자동으로 찾아낸다.  
차를 감지하는 경우

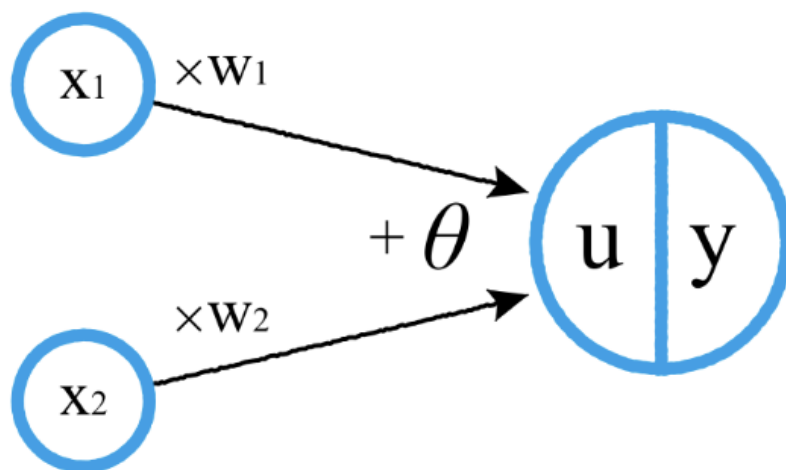


기존의 기계학습 → '자동차의 특징인 타이어에 주목하라!'고 인간이 프로그램에 지시  
딥러닝 → 프로그램이 차를 검출하는 데 필요한 특징을 자동으로 학습

# Section 01 딥러닝 개요

## □ 딥러닝이란?

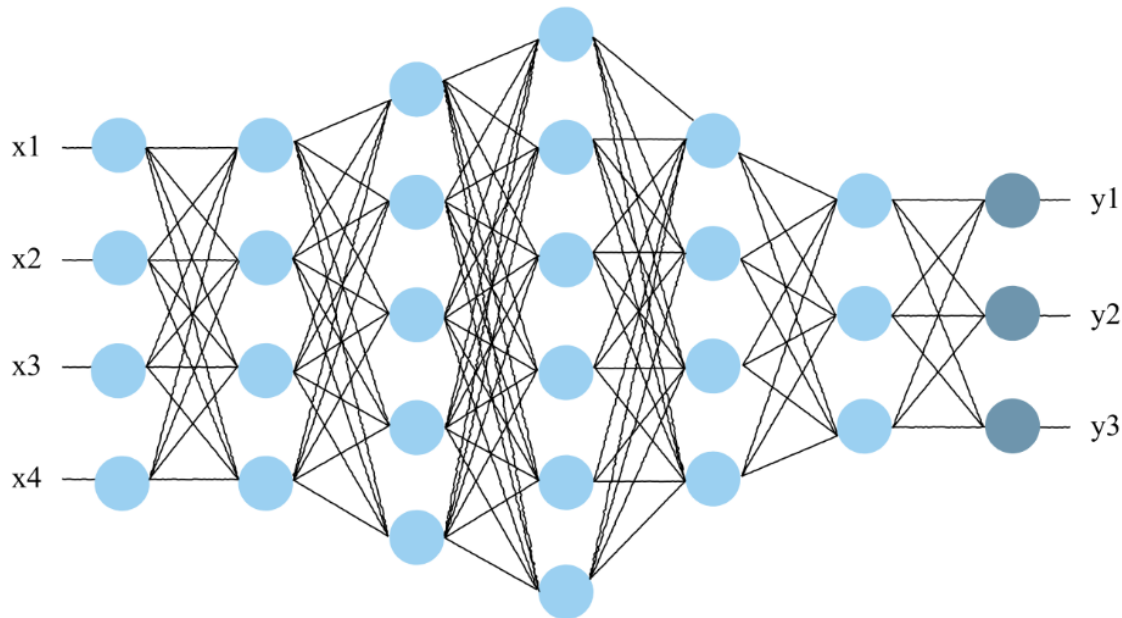
- 그림 10-19는 신경망의 기본이 되는 뉴런(neuron)이다.
- $x_1, x_2$ 가 입력,  $w_1, w_2$ 가 가중치 파라미터이다.
- $w_1x_1 + w_2x_2$ 의 값이 임계값  $\theta$ 보다 높으면 뉴런이 발화(spiking, firing)하여 1을 출력하고, 그렇지 않으면 0을 출력하는 모델이다.



# Section 01 딥러닝 개요

## □ 딥러닝이란?

- 이 뉴런만으로는 복잡한 문제를 해결할 수 없다.
- 그러나 그림 10-20처럼 여러 층을 구축하여 복잡한 문제를 다룰 수 있게 되었다.
- 이것이 심층 신경망이다.
- 심층이라고 하는 이유는 층이 깊게 쌓인 구조를 갖기 때문이다.



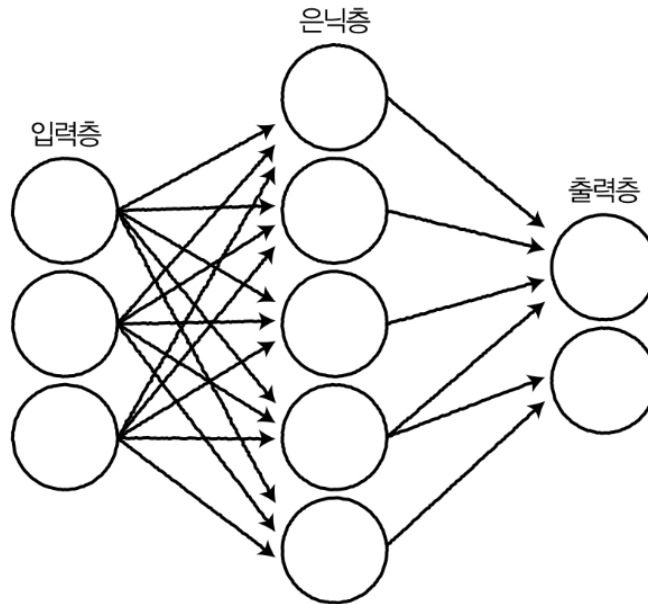


# Section 01 딥러닝 개요

## □ 딥러닝을 이용한 분류의 흐름

### • 네트워크 모델 작성하기

- 여러 뉴런을 묶은 층을 거듭해서 심층 네트워크를 구축한다.
- 그러면 처음에는 각 뉴런이 입력에 대해 무작위로 반응하여 엉터리 값을 출력하게 된다.

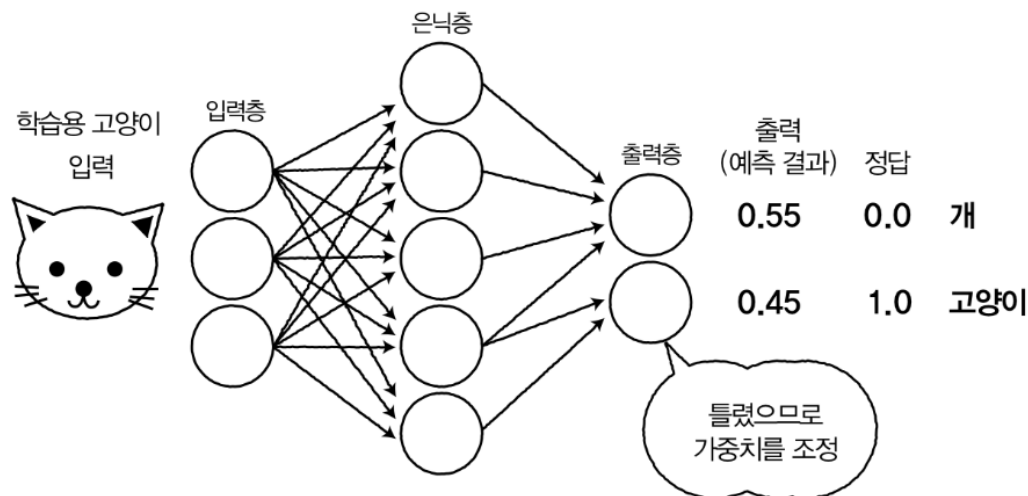


# Section 01 딥러닝 개요

## □ 딥러닝을 이용한 분류의 흐름

### • 모델에 훈련용 데이터를 부여하고 학습시키기

- 모델은  $x$ 를 입력받아  $y$ 를 출력한다.
- 이때 출력  $y$ 와 정답 데이터(지도 라벨)  $T$  사이의 차이  $\Delta E$ 를 작게 하기 위해 오차역전파법(backpropagation)을 사용해 자동으로 각 뉴런의 가중치를 조정한다.
- 데이터  $x$ (다량의 이미지 등)와 정답 데이터  $T$ 를 제공하여 반복적으로 가중치가 조정되면 점차적으로 구하려는 출력값을 얻을 수 있게 된다.
- 학습이 잘 진행되면 적절한 예측 값을 반환하는 모델이 된다.

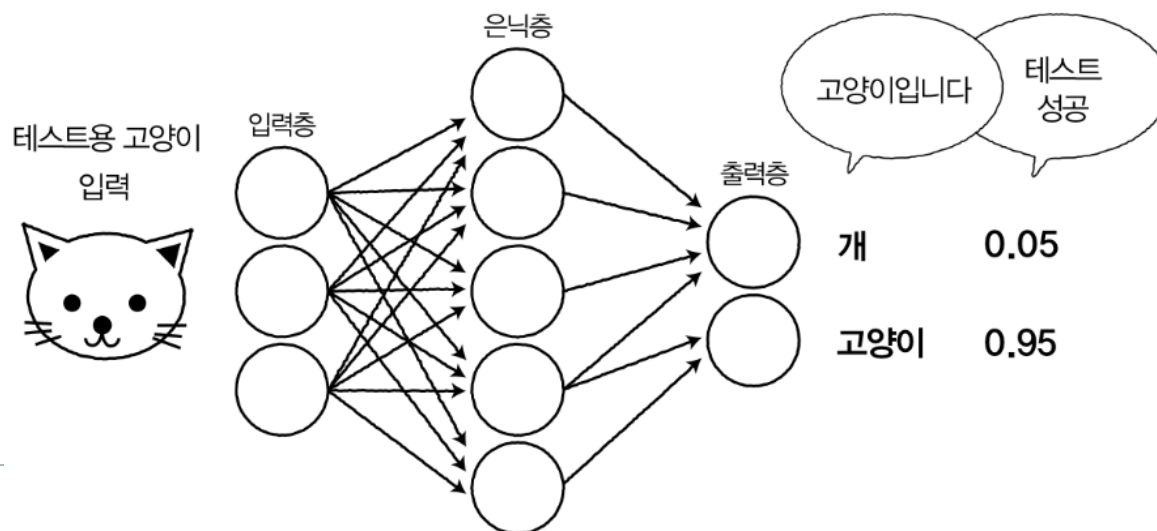


# Section 01 딥러닝 개요

## □ 딥러닝을 이용한 분류의 흐름

### • 분류할 데이터를 모델에 전달하기

- 딥러닝 모델의 학습이 완료되어 학습이 끝난 모델이 완성되었다.
- 이제는 학습된 모델을 사용하는 추론(inference) 단계를 고려한다.
- 추론 단계에서는 실제로 학습이 끝난 모델을 사용하여 예측할 데이터를 모델에 전달하고 추론한다.
- 예를 들어 그림과 같은 이미지를 입력했을 때 고양이일 확률이 95%로 나왔다.
- 그러면 이미지가 고양이라고 판단할 수 있다.



## Section 02 필기체 숫자의 분류

---

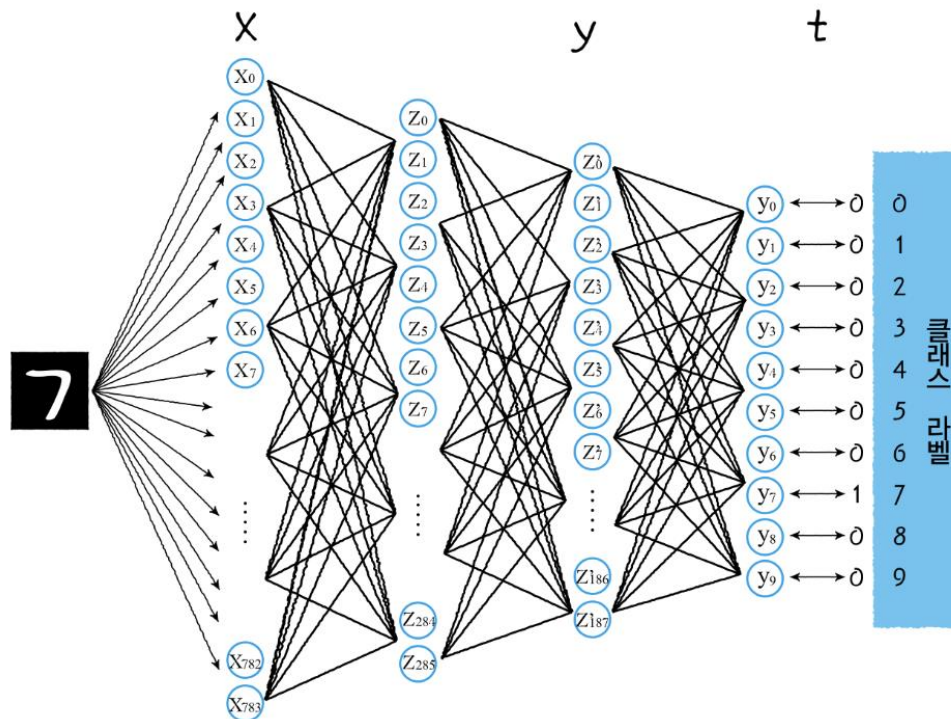
### □ 분류의 흐름

- 여기서는 Keras라는 파이썬 라이브러리를 사용하여 신경망 모델을 실제로 구현해보자.
- 딥러닝 입문으로 자주 등장하는 필기체 숫자분류이다.
- 작업의 흐름은 다음과 같다.
  1. 데이터 준비
  2. 신경망모델 구축
  3. 모델에 데이터를 전달해서 학습시킴
  4. 모델의 분류 정확도 평가
- 마지막으로 실제 필기체 숫자의 이미지를 전달하여 예측된 값을 확인한다.

## Section 02 필기체 숫자의 분류

### □ 심층 신경망

- 여기에서 만들 신경망은 모든 뉴런이 이전 층의 뉴런에 결합하는 전결합층이 2개뿐인 단순한 네트워크 구조로 되어 있다.
- 이렇게 어느 정도 깊이 있는 신경망을 심층 신경망이라고 한다.



## Section 02 필기체 숫자의 분류

### □ 심층 신경망

- 입력을 맡은 층을 입력층, 출력하는 층을 출력층, 입력층과 출력층 사이의 층을 은닉층이라고 한다.
- 여기서 소개할 모델은 입력에  $28 \times 28$ 의 흑백 이미지를 일차원 배열로 평탄화(flattening)한 784차원 벡터를 전달한다.
- 출력은 10차원의 벡터이다. 세로로 늘어선 벡터 하나하나의 요소를 노드라고 하며, 그 차원 수를 노드수라고 한다.
- 필기체 숫자를 0~9의 연속된 값으로 분류하는 것이 아니라 0~9의 10개 클래스로 분류하는 것이 자연스럽기 때문에 출력 유닛의 수는 1 이 아니라 10이 된다.
- 정답이 7인 이미지 데이터에 대한 지도 데이터  $t$ 는 클래스라벨이 7인 곳만 값이 1이고, 그 외에는 0이 된다.
- 이러한 데이터를 원-핫 벡터(one-hot encoding/vector)라고 한다.

## Section 02 필기체 숫자의 분류

### □ 데이터 준비

- 필기체 숫자 데이터셋으로는 MNIST 데이터셋을 사용한다.
- MNIST에는 방대한 수의 필기체 숫자 이미지와 각각의 이미지에 대한 0~9로 표시된 정답 라벨이 포함되어 있다.
- MNIST은 Yann LeCun의 웹사이트<sup>3</sup>에 공개되어 있으며, Keras 코드를 실행하여 쉽게 다운로드할 수 있다.

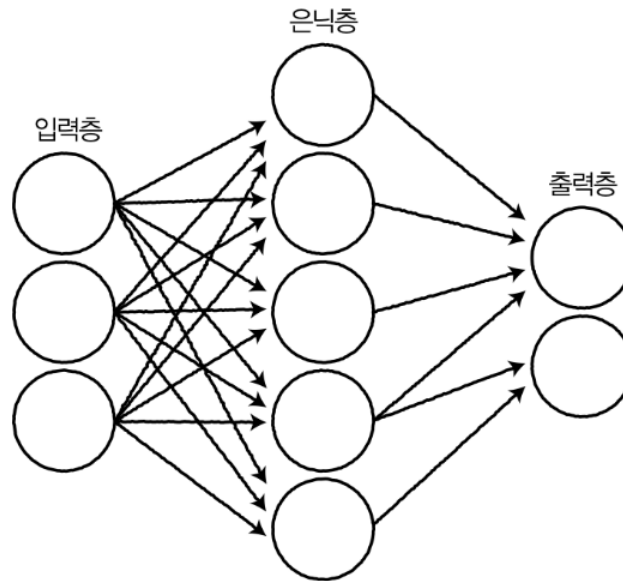
```
import tensorflow.keras.datasets as ds
(x_train, y_train), (x_test, y_test) = ds.mnist.load_data()
```

- 위 코드를 처음 실행하면 인터넷에서 데이터를 다운로드한다.
- 두 번째 이후부터는 PC에 미리 저장(캐싱)된 데이터를 읽어 수행한다.
- x는 대량의 이미지 데이터, y는 대량의 지도 라벨 데이터를 의미한다.
- train은 모델 학습용 데이터, test는 모델 성능 평가시 사용하는 데이터이다.
- train과 test는 모두 일반적인 데이터로서 본질적인 차이는 없다.

## Section 02 필기체 숫자의 분류

### □ 모델 생성

- 먼저 Keras로 모델을 관리하는 인스턴스를 만들고, `add()` 메서드로 한 층씩 정의한다.





# Section 02 필기체 숫자의 분류

## □ 모델 생성

- 먼저 인스턴스를 만든다.

```
model = Sequential()
```

- 다음처럼 add () 메서드를 사용하여 모델을 한 층씩 정의한다.
- 유닛 수가 128인 전결합층을 정의한다.

```
model.add(Dense(128))
```

- 각 전결합층의 출력은 다음처럼 활성화 함수를 적용한다.
- 이는 본래 동물의 신경 발화에 해당하는 구조이다.
- 시그모이드 함수 sigmoid나 ReLU 함수 relu 등을 설정할 수 있다.

```
model.add(Activation("sigmoid"))
```

- 마지막으로 컴파일 메서드 compile ()을 이용하여 어떠한 학습을 실시할지 설정하면 모델 생성이 종료된다.

```
model.compile(optimizer=sgd, loss="categorical_crossentropy", metrics=["acc"])
```

# Section 02 필기체 숫자의 분류

## □ 모델 생성

- ex10\_2.ipynb

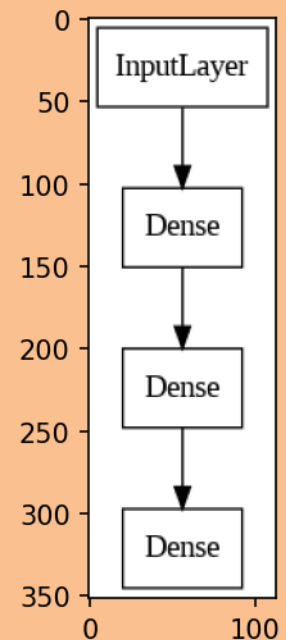
```
1 import tensorflow as tf
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation
8 from tensorflow.keras.utils import plot_model
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
```

# Section 02 필기체 숫자의 분류

## □ 모델 생성

- ex10\_2.ipynb

```
19 model = Sequential()
20
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24
25 model.add(Dense(10, activation='softmax'))
26
27 model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['acc'])
28
29 plot_model(model, "model125.png", show_layer_names=False)
30
31 image = plt.imread("model125.png")
32 plt.figure(dpi=150)
33 plt.imshow(image)
34 plt.show()
```



## Section 02 필기체 숫자의 분류

---

### □ 모델 생성

- **예제 10-2의 6~9행은 models, layers, optimizers 모듈에서 필요한 클래스를 불러온다.**
- **이들 세 모듈은 텐서플로 프로그래밍의 기초에 해당하므로 주의 깊게 살피고 기억해야 한다.**
  - **models 모듈** : Sequential과 functional API의 두 모델을 제공한다. 다층 퍼셉트론처럼 왼쪽에서 오른쪽으로 계산이 한 줄기로 흐르는 경우에 Sequential을 쓴다.
  - **layers 모듈** : 여러 가지 층을 제공한다. 다층 퍼셉트론을 구성하는 완전 연결층은 Dense 클래스로 쌓는다.
  - **optimizers 모듈** : 학습 알고리즘이 사용하는 옵티마이저 함수를 제공한다. SGD, Adam, AdaGrad, RMSprop 등이 있다.

## Section 02 필기체 숫자의 분류

### □ 모델 학습

- 모델에 훈련 데이터를 전달하여 학습을 실시한다.
- 다음처럼 fit () 메서드를 사용한다.

```
model.fit(x_train, y_train, verbose=1, epochs=3)
```

- x\_train, y\_train은 각각 학습용 입력 데이터와 지도 데이터이다.
- verbose에 지정한 숫자로 학습의 진척 상황 표시를 조정할 수 있다.
- verbose=1 로 지정하면 학습 등의 진척을 출력하고, verbose=0으로 지정하면 진척을 출력하지 않는다.
- epochs에는 동일한 데이터셋으로 몇 번 반복 학습할지 지정한다.
- fit() 메서드는 학습용 데이터(트레이닝 데이터)를 순서대로 모델에 입력하고, 출력 및 지도 데이터 간의 차이가 작아지도록 각 뉴런의 가중치를 조금씩 갱신한다.
- 이에 따라 오차가 감소하고, 모델의 예측 정확도가 향상된다.

# Section 02 필기체 숫자의 분류

## □ 모델 학습

- ex10\_3.ipynb

```
1 import tensorflow as tf
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation
8 from tensorflow.keras.utils import plot_model
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
```

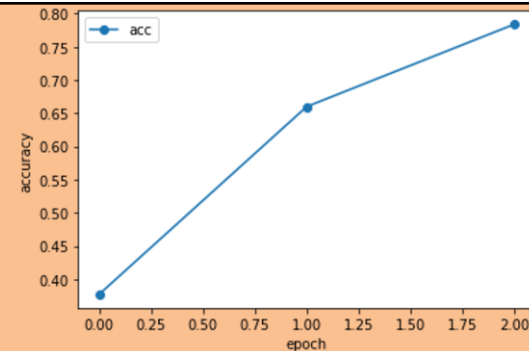
# Section 02 필기체 숫자의 분류

## □ 모델 학습

- ex10\_3.ipynb

Epoch 1/3  
1875/1875 [=====] - 7s 3ms/step - loss: 2.1443 - acc: 0.3774  
Epoch 2/3  
1875/1875 [=====] - 7s 4ms/step - loss: 1.5061 - acc: 0.6598  
Epoch 3/3  
1875/1875 [=====] - 6s 3ms/step - loss: 0.9325 - acc: 0.7837

```
19 model = Sequential()
20
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24
25 model.add(Dense(10, activation='softmax'))
26
27 model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['acc'])
28 history = model.fit(x_train, y_train, verbose=1, epochs=3)
29
30 plt.plot(history.history['acc'], label='acc', ls='-', marker='o')
31 plt.ylabel("accuracy")
32 plt.xlabel("epoch")
33 plt.legend(loc='best')
34 plt.show()
```



## Section 02 필기체 숫자의 분류

### □ 모델 평가

- 훈련 데이터로 학습을 수행하여 모델의 튜닝이 성공적으로 진행되었다.
- 그러나 모델이 훈련 데이터에만 통용되도록 학습해버린 가능성(과적합이라고 한다)도 존재하므로 이것만으로는 모델의 성능을 제대로 평가할 수 없다.
- 그래서 여기서는 학습에 이용하지 않았던 테스트 데이터를 사용해서 모델로 분류하고, 모델의 평가를 실시한다.
- 모델에 테스트 데이터를 전달했을 때의 분류 정확도를 일반화 정확도라고 한다.
- 일반화 정확도의 계산은 `evaluate ()` 메서드를 사용한다.

```
score = model.evaluate(x_test, y_test, verbose=1)
```

- `x_test`, `y_test`는 평가용 입력 데이터와 지도 데이터이다.
- `evaluate()` 메서드는 손실 함수의 값과 정확도를 얻을 수 있으며, 위 예제의 경우 모두 `score`에 저장된다.



# Section 02 필기체 숫자의 분류

## □ 모델 평가

- ex10\_4.ipynb

```
1 import tensorflow as tf
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation
8 from tensorflow.keras.utils import plot_model
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
```

# Section 02 필기체 숫자의 분류

## □ 모델 평가

- ex10\_4.ipynb

```
19 model = Sequential()
20
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24
25 model.add(Dense(10, activation='softmax'))
26
27 model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['acc'])
28 model.fit(x_train, y_train, verbose=1, epochs=3)
29
30 score = model.evaluate(x_test, y_test, verbose=1)
31
32 print("evaluate loss: {0[0]}\nevaluate acc: {0[1]}".format(score))
```

```
Epoch 1/3
1875/1875 [=====] - 6s 3ms/step - loss: 2.1283 - acc: 0.4214
Epoch 2/3
1875/1875 [=====] - 7s 4ms/step - loss: 1.4453 - acc: 0.6831
Epoch 3/3
1875/1875 [=====] - 6s 3ms/step - loss: 0.8983 - acc: 0.7816
313/313 [=====] - 1s 3ms/step - loss: 0.7376 - acc: 0.8130
evaluate loss: 0.7376301884651184
evaluate acc: 0.8130000233650208
```

## Section 02 필기체 숫자의 분류

### □ 모델에 의한 분류

- model의 predict() 메서드로 예측치를 얻을 수 있다.
- 예를 들어 x\_test의 첫사진 1장의 숫자를 예측하려면 다음과 같이 기술한다.
- predict는 여러 장의 이미지를 인수로 받아들이는 것을 가정하고 있으므로 사진 1 장을 예측하는 경우에는 차원에 주의해야 한다.

```
pred = np.argmax(model_predict(x_test[0]))  
print("예측치 : " + str(pred))
```

- predict () 메서드의 출력은 10차원이므로 argmax () 함수로 가장 큰 값을 반환하는 뉴런의 위치를 취득하고 있다.

# Section 02 필기체 숫자의 분류

## □ 모델에 의한 분류

- ex10\_5.ipynb

```
1 import tensorflow as tf
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation
8 from tensorflow.keras.utils import plot_model
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
```

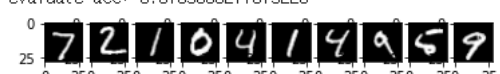
# Section 02 필기체 숫자의 분류

## □ 모델에 의한 분류

- ex10\_5.ipynb

```
19 model = Sequential()
20
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24
25 model.add(Dense(10, activation='softmax'))
26
27 model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['acc'])
28 model.fit(x_train, y_train, verbose=1, epochs=3)
29
30 score = model.evaluate(x_test, y_test, verbose=1)
31
32 print("evaluate loss: {0[0]}\nevaluate acc: {0[1]}".format(score))
33
34 for i in range(10):
35     plt.subplot(1, 10, i+1)
36     plt.imshow(x_test[i].reshape((28, 28)), "gray")
37 plt.show()
```

Epoch 1/3  
1875/1875 [=====] - 8s 4ms/step - loss: 2.1533 - acc: 0.3927  
Epoch 2/3  
1875/1875 [=====] - 6s 3ms/step - loss: 1.5162 - acc: 0.6739  
Epoch 3/3  
1875/1875 [=====] - 7s 4ms/step - loss: 0.9199 - acc: 0.7836  
313/313 [=====] - 1s 2ms/step - loss: 0.7527 - acc: 0.8159  
evaluate loss: 0.7526882886886597  
evaluate acc: 0.8159000277519226



## Section 03 딥러닝 튜닝

---

### □ 하이퍼파라미터

- 딥러닝을 사용하면 분류 또는 회귀 알고리즘을 간단한 코드로 구현할 수 있으므로 매우 편리하다.
- 또한 신경망 모델은 여러 경우에 적용할 수 있어 범용적이다.
- 그러나 네트워크를 구성할 때 사람이 조정해야 하는 파라미터가 존재한다.
- 이를 하이퍼파라미터라고 한다.

## Section 03 딥러닝 튜닝

### □ 하이퍼파라미터

- **예제 10-6은 필기체 숫자 분류 코드를 약간 변경하고, 몇 가지 파라미터를 명시한 전형적인 딥러닝 코드이다.**
- ex10\_6.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3
4 from tensorflow.keras.datasets import mnist
5 from tensorflow.keras.models import Sequential
6 from tensorflow.keras.layers import Dense, Activation, Dropout
7 from tensorflow.keras.optimizers import SGD
8 from tensorflow.keras.utils import to_categorical
9
10 (x_train, y_train), (x_test, y_test) = mnist.load_data()
11 x_train = x_train.reshape(60000, 784)
12 x_test = x_test.reshape(10000, 784)
13 x_train = x_train.astype(np.float32) / 255.0
14 x_test = x_test.astype(np.float32) / 255.0
15 y_train = to_categorical(y_train, 10)
16 y_test = to_categorical(y_test, 10)
```

# Section 03 딥러닝 튜닝

## □ 하이퍼파라미터

- ex10\_6.ipynb

```
17
18 model = Sequential()
19
20 model.add(Dense(256, input_dim=784, activation="sigmoid"))
21
22 model.add(Dense(128, activation='sigmoid'))
23 model.add(Dropout(rate=0.5))
24
25 model.add(Dense(10, activation='softmax'))
26
27 model.compile(optimizer=SGD(learning_rate=0.01), loss='categorical_crossentropy', metrics=['acc'])
28 model.fit(x_train, y_train, batch_size=32, verbose=1, epochs=10)
29
30 score = model.evaluate(x_test, y_test, verbose=1)
31
32 print("evaluate loss: {0[0]}\nevaluate acc: {0[1]}".format(score))
```

```
Epoch 1/10
1875/1875 [=====] - 7s 4ms/step - loss: 2.3063 - acc: 0.1511
Epoch 2/10
1875/1875 [=====] - 7s 4ms/step - loss: 1.9428 - acc: 0.3375
Epoch 3/10
1875/1875 [=====] - 7s 4ms/step - loss: 1.4167 - acc: 0.5238
Epoch 4/10
1875/1875 [=====] - 7s 4ms/step - loss: 1.0813 - acc: 0.6401
Epoch 5/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.8864 - acc: 0.7123
Epoch 6/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.7695 - acc: 0.7546
Epoch 7/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.6891 - acc: 0.7850
Epoch 8/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.6339 - acc: 0.8034
Epoch 9/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.5910 - acc: 0.8196
Epoch 10/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.5571 - acc: 0.8304
evaluate loss: 0.42317187786102295
evaluate acc: 0.8806999921798706
```



## Section 03 딥러닝 튜닝

---

### □ 네트워크 구조

- 네트워크 구조(은닉층 수, 은닉층의 유닛 수)는 자유롭게 설정할 수 있다.
- 일반적으로 은닉층 수 및 은닉층의 유닛 수를 많이 하면 다양한 함수를 표현할 수 있게 된다.
- 그러나 은닉층이 많아지면 입력층에 가까운 가중치를 적절하게 갱신하기 어렵고 학습이 좀처럼 진행되지 않는다.
- 은닉층의 유닛 수가 많아지면 중요성이 낮은 특징량을 추출해버려 과적합(일반화 성능이 낮아진 상태)되기 쉬워질 수도 있다.
- 그래서 적절한 네트워크 구조를 설정할 필요가 있다.
- 네트워크 구조는 이론을 구축해서 정하기 어려우며, 실제로는 이미 구현된 유사한 예를 참고하는 등 경험에 근거해 결정하는 경향이 있다.

# Section 03 딥러닝 튜닝

## □ 네트워크 구조

- **예제 10-7은 네트워크 구조를 변경했을 때 모델의 정확도를 평가하는 코드이다.**
- ex10\_7.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3
4 from tensorflow.keras.datasets import mnist
5 from tensorflow.keras.models import Sequential
6 from tensorflow.keras.layers import Dense, Activation, Dropout
7 from tensorflow.keras.optimizers import SGD
8 from tensorflow.keras.utils import to_categorical
9
10 (x_train, y_train), (x_test, y_test) = mnist.load_data()
11 x_train = x_train.reshape(60000, 784)
12 x_test = x_test.reshape(10000, 784)
13 x_train = x_train.astype(np.float32) / 255.0
14 x_test = x_test.astype(np.float32) / 255.0
15 y_train = to_categorical(y_train, 10)
16 y_test = to_categorical(y_test, 10)
```

# Section 03 딥러닝 튜닝

## □ 네트워크 구조

- ex10\_7.ipynb

```
17
18 model = Sequential()
19 model.add(Dense(256, input_dim=784, activation="sigmoid"))
20
21 def funcA():
22     model.add(Dense(128, activation='sigmoid'))
23
24 def funcB():
25     model.add(Dense(128, activation='sigmoid'))
26     model.add(Dense(128, activation='sigmoid'))
27     model.add(Dense(128, activation='sigmoid'))
28
29 def funcC():
30     model.add(Dense(1568, activation='sigmoid'))
31
32 # 다음 중 두 줄을 주석 처리할 것
33 #-----
34 funcA()
35 #funcB()
36 #funcC()
```

## Section 03 딥러닝 튜닝

### □ 네트워크 구조

- ex10\_7.ipynb

```
37 #-----
38
39 model.add(Dropout(rate=0.5))
40 model.add(Dense(10, activation='softmax'))
41
42 model.compile(optimizer=SGD(learning_rate=0.01), loss='categorical_crossentropy', metrics=['acc'])
43 model.fit(x_train, y_train, batch_size=32, verbose=1, epochs=3)
44
45 score = model.evaluate(x_test, y_test, verbose=0)
46 print("evaluate loss: {0[0]}\nevaluate acc: {0[1]}".format(score))
```

```
Epoch 1/3
1875/1875 [=====] - 8s 4ms/step - loss: 2.3010 - acc: 0.1566
Epoch 2/3
1875/1875 [=====] - 6s 3ms/step - loss: 1.9271 - acc: 0.3437
Epoch 3/3
1875/1875 [=====] - 7s 4ms/step - loss: 1.4239 - acc: 0.5193
evaluate loss: 1.0546293258666992
evaluate acc: 0.7581999897956848
```

## Section 03 딥러닝 튜닝

### □ 드롭아웃

- 드롭아웃(dropout)은 과적합을 방지하여 모델의 정확도를 높이는 방법 중 하나이다.
- 드롭아웃을 사용하면 유닛의 일부가 학습할 때마다 무작위로 제거된다.
- 따라서 신경망은 특정 뉴런의 존재에 의존할수 없게 되어 보다 범용적인(학습 데이터 외에도 통용되기 쉬운) 특징을 학습한다.
- 결과적으로 학습 데이터에 대한 과적합을 방지할 수 있다.
- 드롭아웃은 다음과 같이 사용한다.

```
model.add(Dropout(rate=0.5))
```

- 여기서 rate는 제거할 유닛의 비율이다.
- 드롭아웃을 사용하는 위치, 인수( rate)는 모두 하이퍼파라미터이다.

# Section 03 딥러닝 튜닝

## □ 드롭아웃

- **예제 10-8은 드롭아웃을 적용한 코드이다.**
- ex10\_8.ipynb

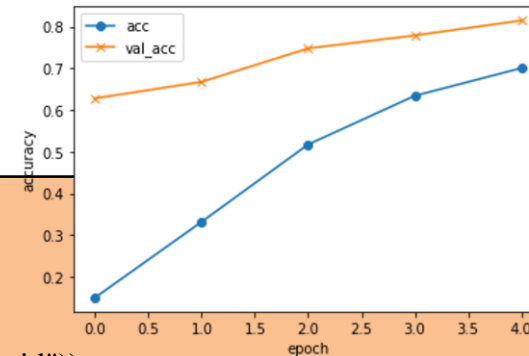
```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation, Dropout
8 from tensorflow.keras.optimizers import SGD
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
```

Epoch 1/5  
 1875/1875 [=====] - 8s 4ms/step - loss: 2.3036 - acc: 0.1502 - val\_loss: 2.1025 - val\_acc: 0.6280  
 Epoch 2/5  
 1875/1875 [=====] - 8s 4ms/step - loss: 1.9529 - acc: 0.3319 - val\_loss: 1.5552 - val\_acc: 0.6676  
 Epoch 3/5  
 1875/1875 [=====] - 7s 3ms/step - loss: 1.4265 - acc: 0.5175 - val\_loss: 1.0575 - val\_acc: 0.7478  
 Epoch 4/5  
 1875/1875 [=====] - 8s 4ms/step - loss: 1.0901 - acc: 0.6341 - val\_loss: 0.8260 - val\_acc: 0.7789  
 Epoch 5/5  
 1875/1875 [=====] - 8s 4ms/step - loss: 0.9098 - acc: 0.7006 - val\_loss: 0.6920 - val\_acc: 0.8147

## Section 03 딥러닝 튜닝

### □ 드롭아웃

- ex10\_8.ipynb



```

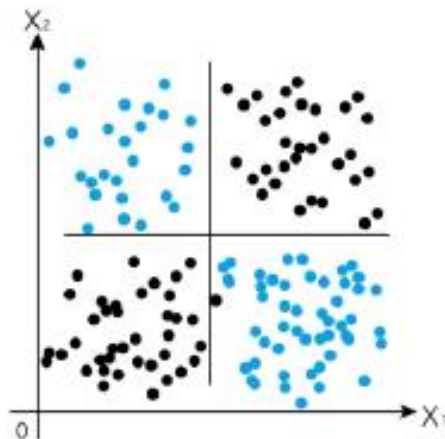
18
19 model = Sequential()
20
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24 model.add(Dropout(rate=0.5))
25
26 model.add(Dense(10, activation='softmax'))
27
28 model.compile(optimizer=SGD(learning_rate=0.01), loss='categorical_crossentropy', metrics=['acc'])
29 history = model.fit(x_train, y_train, batch_size=32, verbose=1, epochs=5, validation_data=(x_test, y_test))
30
31 plt.plot(history.history["acc"], label="acc", ls="-", marker="o")
32 plt.plot(history.history["val_acc"], label="val_acc", ls="-", marker="x")
33 plt.ylabel("accuracy")
34 plt.xlabel("epoch")
35 plt.legend(loc="best")
36 plt.show()

```

## Section 03 딥러닝 튜닝

### □ 활성화 함수

- 활성화 함수는 주로 전결합층 뒤에 적용하는 함수로, 뉴런의 발화에 해당한다.
- 전결합층에서는 입력을 선형 변환한 것을 출력하지만, 활성화함수를 이용함으로써 비선형성을 갖게 한다.
- 활성화 함수를 사용하지 않을 경우 그림과 같이 단일 직선으로 분리할 수 없는(선형 분리가 불가능한) 데이터는 분류할 수 없음이 수학적으로 밝혀져 있다.





# Section 03 딥러닝 튜닝

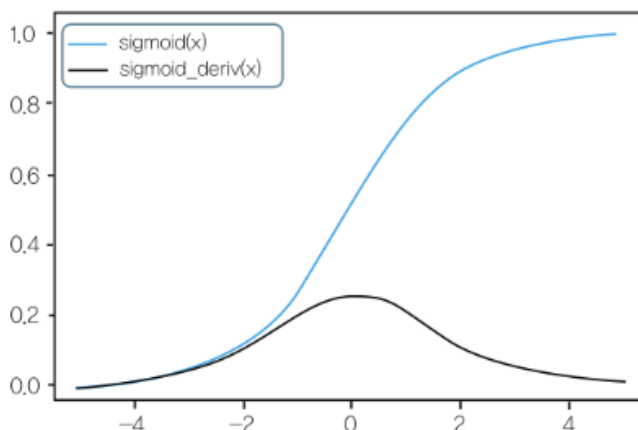
## □ 활성화 함수

### • 시그모이드 함수

- 활성화 함수로 사용되는 함수 중 하나로 시그모이드 함수(sigmoid function)가 있다.

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

- 파란색 그래프는 시그모이드 함수고, 검은 그래프는 시그모이드 함수의 도함수(미분했을 때의 함수) 이다.



## Section 03 딥러닝 튜닝

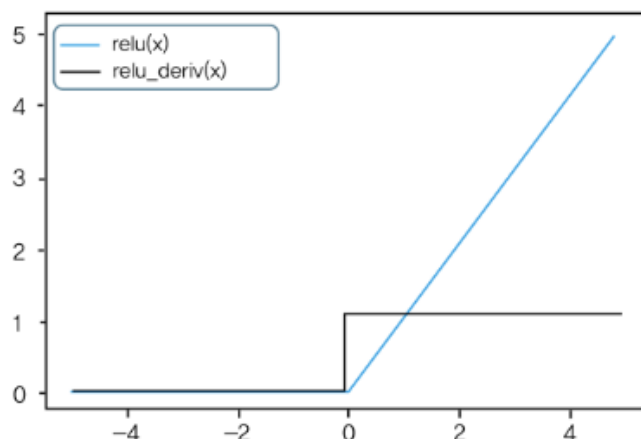
### □ 활성화 함수

- ReLU 함수

- 활성화 함수로 자주 사용되는 ReLU 함수(렐루 또는 램프 함수로 불림)에 대해 설명한다.
- ReLU는 Rectified Linear Unit의 약자이며, 다음과 같은 함수이다.

$$\text{ReLU}(x) = \begin{cases} 0 & (x < 0) \\ x & (x \geq 0) \end{cases}$$

- 파란색 그래프가 ReLU 함수고, 검은 그래프가 ReLU 함수의 도함수이다.



## Section 03 딥러닝 튜닝

---

### □ 손실 함수

- 학습시 모델의 출력과 지도 데이터의 차이(잘못된 상태)를 평가하는 함수를 손실 함수(오차 함수)라고 한다.
- 손실 함수에는 제곱오차(Squared error)와 교차 엔트로피 오차(cross-entropy error) 등이 사용된다.
- 이 손실 함수를 최소화하도록 오차역전파법(backpropagation)으로 각 층의 가중치가 갱신된다.

# Section 03 딥러닝 튜닝

---

## □ 손실 함수

### • 제곱 오차

- 제곱오차는 최소제곱법으로 통계학 등 다양한 분야에서 사용되는 오차 함수이다.

$$E = \sum_{i=1}^N (t_i - y_i)^2$$

- 연속값 평가가 뛰어나므로 주로 회귀 모델의 오차함수로 사용된다.
- 위 식에서  $y_i$ 와  $t_i$ 는 각각 예측 라벨과 정답 라벨을 나타낸다.

## Section 03 딥러닝 튜닝

### □ 손실 함수

#### • 교차 엔트로피 오차

- 교차 엔트로피 오차(cross entropy error, CEE)는 이항 분류의 평가에 특화되어 있으므로 주로 분류 모델의 오차함수로 사용된다.

$$E = \sum_{i=1}^N (-t_i \log y_i - (1 - t_i) \log(1 - y_i))$$

- 그러면 이 함수가 어떤 특성을 갖는지 살펴보자.
- (i)  $t_i \ll y_i$  일 때  $-t_i \log y_i$  는 거의 0이고,  $-(1 - t_i) \log(1 - y_i)$  는 양의 무한대이다.
- (ii)  $t_i \gg y_i$  일 때  $-t_i \log y_i$  는 양의 무한대이고,  $-(1 - t_i) \log(1 - y_i)$  는 거의 0이다.
- (iii)  $t_i \approx y_i$  일 때  $-t_i \log y_i$  가  $-(1 - t_i) \log(1 - y_i)$  는  $0.69... \sim 0$  의 값을 취하며, 간단한 계산으로 구할 수 있다.
- 따라서  $-t_i \log y_i - (1 - t_i) \log(1 - y_i)$  는  $|t_i - y_i|$  가 클 때 매우 큰 값을 반환하고,  $|t_i - y_i|$  가 작을 때 0에 가까운 값을 취한다는 것을 알 수 있다.
- 분류 학습에서 예측 라벨  $y_i$ 와 정답 라벨  $t_i$ 의 값은 가까울수록 좋으므로 이 함수는 유용하다.

## Section 03 딥러닝 튜닝

### □ 손실 함수

#### • 교차 엔트로피 오차

- 교차 엔트로피 오차(cross entropy error, CEE)는 이항 분류의 평가에 특화되어 있으므로 주로 분류 모델의 오차함수로 사용된다.

$$E = \sum_{i=1}^N (-t_i \log y_i - (1 - t_i) \log(1 - y_i))$$

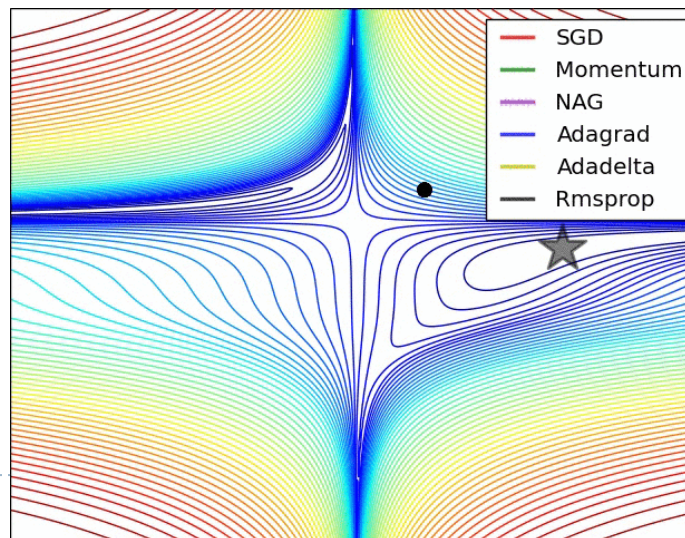
- 그러면 이 함수가 어떤 특성을 갖는지 살펴보자.

- ①  $t_i \ll y_i$  일 때  $-t_i \log y_i$  는 거의 0이고,  $-(1 - t_i) \log(1 - y_i)$  는 양의 무한대이다.
  - ②  $t_i \gg y_i$  일 때  $-t_i \log y_i$  는 양의 무한대이고,  $-(1 - t_i) \log(1 - y_i)$  는 거의 0이다.
  - ③  $t_i \approx y_i$  일 때  $-t_i \log y_i$  가  $-(1 - t_i) \log(1 - y_i)$  는  $0.69... \sim 0$ 의 값을 취하며, 간단한 계산으로 구할 수 있다.
- 따라서  $-t_i \log y_i - (1 - t_i) \log(1 - y_i)$  는  $|t_i - y_i|$  가 클 때 매우 큰 값을 반환하고,  $|t_i - y_i|$  가 작을 때 0에 가까운 값을 취한다는 것을 알 수 있다.
  - 분류 학습에서 예측 라벨  $y_i$ 와 정답 라벨  $t_i$ 의 값은 가까울수록 좋으므로 이 함수는 유용하다.

## Section 03 딥러닝 튜닝

### □ 최적화 함수

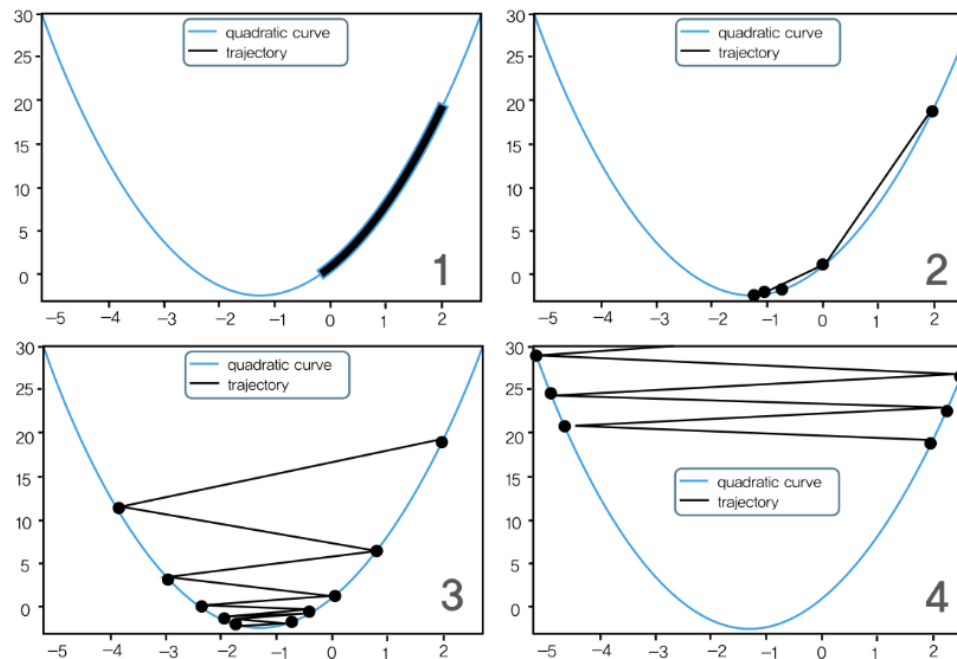
- 가중치 갱신은 오차 함수를 각 가중치로 미분한 값을 바탕으로 갱신해야 할 방향과 어느 정도로 갱신할지 결정한다.
- 최적화 함수는 미분에 의해 구한 값을 학습 속도, epoch 수, 과거의 가중치 갱신량 등을 근거로 어떻게 가중치 갱신에 반영할지 정하는 것이다.
- 최적화 함수는 하이퍼파라미터이다.
- 그림과 같이 최적화 함수는 여러 종류가 있으며, 올바르게 선택하지 않으면 학습에 많은 시간이 걸릴 수 있다.



## Section 03 딥러닝 튜닝

### □ 학습률

- 학습률(learning rate)은 각 층의 가중치를 한 번에 어느 정도 변경할지 결정하는 하이퍼파라미터이다.
- 그림은 최소화하려는 모델과 학습률이 미치는 영향을 표시한 것이다.
- 우측 상단의 점이 기본값이다.





# Section 03 딥러닝 튜닝

## □ 학습률

- ex10\_9.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation, Dropout
8 from tensorflow.keras.optimizers import SGD
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
19 model = Sequential()
20
```

# Section 03 딥러닝 튜닝

## □ 학습률

- ex10\_9.ipynb

```
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24 model.add(Dropout(rate=0.5))
25
26 model.add(Dense(10, activation='softmax'))
27
28 def funcA():
29     global lr
30     lr = 0.01
31
32 def funcB():
33     global lr
34     lr = 0.1
35
36 def funcC():
37     global lr
38     lr = 1.0
39
```

# Section 03 딥러닝 튜닝

## □ 학습률

- ex10\_9.ipynb

```
40 # 다음 중 두 줄을 주석 처리할 것
41 #-----
42 funcA()
43 #funcB()
44 #funcC()
45 #-----
46
47 model.compile(optimizer=SGD(learning_rate=lr), loss='categorical_crossentropy', metrics=['acc'])
48 history = model.fit(x_train, y_train, batch_size=32, verbose=1, epochs=3)
49
50 score = model.evaluate(x_test, y_test, verbose=0)
51 print("evaluate loss: {0[0]}\nevaluate acc: {0[1]}".format(score))
```

```
Epoch 1/3
1875/1875 [=====] - 8s 4ms/step - loss: 2.2982 - acc: 0.1544
Epoch 2/3
1875/1875 [=====] - 6s 3ms/step - loss: 1.9339 - acc: 0.3442
Epoch 3/3
1875/1875 [=====] - 7s 4ms/step - loss: 1.3754 - acc: 0.5402
evaluate loss: 1.0082892179489136
evaluate acc: 0.7498999834060669
```

## Section 03 딥러닝 튜닝

### □ 미니배치 학습

- 한 번에 전달하는 데이터 수를 배치 크기(batch size)라고 하며, 이 또한 하이퍼파라미터이다.
- 한 번에 여러 데이터를 전달했을 때 모델은 각 데이터의 손실과 손실 함수의 기울기(가중치를 어떻게 갱신할 것인가)를 구하지만 가중치 갱신은 구해진 기울기의 평균으로 한 번만 실시된다.
- 복수의 데이터를 이용하여 가중치를 갱신하면 극단적으로 바뀐 데이터의 영향을 덜 받게 되고, 병렬 계산이 가능하기 때문에 계산 시간이 단축된다.
- 하지만 극단적인 가중치의 갱신이 발생하지 않게 되어 손실 함수의 국소해(local minima)에서 벗어나지 못할 수 있다.
- 편향된 데이터가 많을 때는 배치 크기를 크게 하고, 유사한 데이터가 많을 때는 배치 크기를 작게 하는 등 배치 크기를 잘 조정해야 한다.
- 배치 크기를 1로 하는 방식을 온라인 학습(확률적 경사하강법), 배치 크기를 전체 데이터 수로 지정하는 방식을 배치 학습(경사하강법), 이들의 중간이 되는 방식을 미니배치 학습이라고 한다.

# Section 03 딥러닝 튜닝

## □ 미니배치 학습

- ex10\_10.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation, Dropout
8 from tensorflow.keras.optimizers import SGD
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
19 model = Sequential()
20
```

# Section 03 딥러닝 튜닝

## □ 미니배치 학습

- ex10\_10.ipynb

```
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24 model.add(Dropout(rate=0.5))
25
26 model.add(Dense(10, activation='softmax'))
27
28 model.compile(optimizer=SGD(learning_rate=0.1), loss='categorical_crossentropy', metrics=['acc'])
29
30 def funcA():
31     global batch_size
32     batch_size = 16
33
34 def funcB():
35     global batch_size
36     batch_size = 32
37
38 def funcC():
39     global batch_size
40     batch_size = 64
```

# Section 03 딥러닝 튜닝

## □ 미니배치 학습

- ex10\_10.ipynb

```
41
42 # 다음 중 두 줄을 주석 처리할 것
43 #-----
44 #funcA()
45 #funcB()
46 funcC()
47 #-----
48
49 history = model.fit(x_train, y_train, batch_size=batch_size, verbose=1, epochs=3)
50 score = model.evaluate(x_test, y_test, verbose=0)
51 print("evaluate loss: {0[0]}\nevaluate acc: {0[1]}".format(score))
```

```
Epoch 1/3
938/938 [=====] - 4s 4ms/step - loss: 1.5297 - acc: 0.4701
Epoch 2/3
938/938 [=====] - 4s 4ms/step - loss: 0.6689 - acc: 0.7879
Epoch 3/3
938/938 [=====] - 5s 5ms/step - loss: 0.5048 - acc: 0.8465
evaluate loss: 0.3634733259677887
evaluate acc: 0.892799973487854
```

## Section 03 딥러닝 튜닝

---

### □ 반복 학습

- 일반적으로 모델의 정확도를 높이기 위해서는 동일한 훈련 데이터를 사용하여 여러 번 학습시킨다.
- 이것을 반복 학습이라고 한다.
- 학습할 횟수는 epoch 수고, 이것도 하이퍼파라미터이다.
- epoch 수를 높인다고 해서 모델의 정확도가 계속 오르는 것은 아니다.
- 정확도는 중간부터 높아지지 않을 뿐만 아니라 반복 학습으로 손실 함수를 최소화시키려 하면 과학습이 일어난다.
- 적절한 시기에 학습을 중단할 필요가 있다.



# Section 03 딥러닝 튜닝

## □ 반복 학습

- ex10\_11.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation, Dropout
8 from tensorflow.keras.optimizers import SGD
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
19 model = Sequential()
20
```

# Section 03 딥러닝 튜닝

## □ 반복 학습

- ex10\_11.ipynb

```
21 model.add(Dense(256, input_dim=784, activation="sigmoid"))
22
23 model.add(Dense(128, activation='sigmoid'))
24 model.add(Dropout(rate=0.5))
25
26 model.add(Dense(10, activation='softmax'))
27
28 model.compile(optimizer=SGD(learning_rate=0.1), loss='categorical_crossentropy', metrics=['acc'])
29
30 def funcA():
31     global epochs
32     epochs = 5
33
34 def funcB():
35     global epochs
36     epochs = 10
37
38 def funcC():
39     global epochs
40     epochs = 60
41
```

# Section 03 딥러닝

## □ 반복 학습

### • ex10\_11.ipynb

```
Epoch 1/10
1875/1875 [=====] - 9s 4ms/step - loss: 0.8223 - acc: 0.7590 - val_loss: 0.3597 - val_acc: 0.8982
Epoch 2/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.3325 - acc: 0.9027 - val_loss: 0.2904 - val_acc: 0.9140
Epoch 3/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.2821 - acc: 0.9178 - val_loss: 0.2554 - val_acc: 0.9263
Epoch 4/10
1875/1875 [=====] - 8s 4ms/step - loss: 0.2512 - acc: 0.9267 - val_loss: 0.2351 - val_acc: 0.9330
Epoch 5/10
1875/1875 [=====] - 10s 5ms/step - loss: 0.2248 - acc: 0.9338 - val_loss: 0.2122 - val_acc: 0.9384
Epoch 6/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.2025 - acc: 0.9408 - val_loss: 0.1917 - val_acc: 0.9438
Epoch 7/10
1875/1875 [=====] - 8s 4ms/step - loss: 0.1830 - acc: 0.9464 - val_loss: 0.1716 - val_acc: 0.9497
Epoch 8/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.1663 - acc: 0.9515 - val_loss: 0.1607 - val_acc: 0.9525
Epoch 9/10
1875/1875 [=====] - 8s 4ms/step - loss: 0.1523 - acc: 0.9556 - val_loss: 0.1457 - val_acc: 0.9569
Epoch 10/10
1875/1875 [=====] - 8s 4ms/step - loss: 0.1394 - acc: 0.9591 - val_loss: 0.1364 - val_acc: 0.9591
```

```
42 # 다음 중 두 줄을 주석 처리할 것
```

```
43 #-----
```

```
44 #funcA()
```

```
45 funcB()
```

```
46 #funcC()
```

```
47 #-----
```

```
48
```

```
49 history = model.fit(x_train, y_train, batch_size=32, verbose=1, epochs=epochs, validation_data=(x_test, y_test))
```

```
50
```

```
51 plt.plot(history.history["acc"], label="acc", ls="-", marker="o")
```

```
52 plt.plot(history.history["val_acc"], label="val_acc", ls="-", marker="x")
```

```
53 plt.ylabel('accuracy')
```

```
54 plt.xlabel('epoch')
```

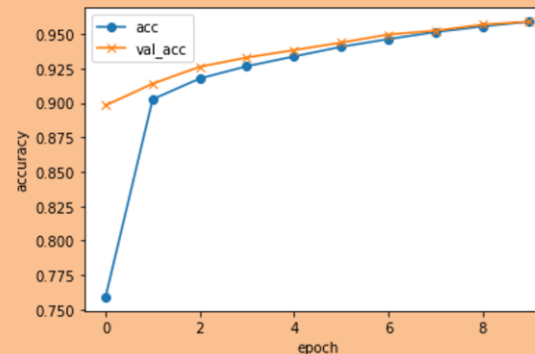
```
55 plt.legend(loc="best")
```

```
56 plt.show()
```

```
57
```

```
58 score = model.evaluate(x_test, y_test, verbose=0)
```

```
59 print("evaluate loss: {0[0]}\nevaluate acc: {0[1]}".format(score))
```



# Section 03 딥러닝 튜닝

## □ 우편번호 인식기 v1.0

- 여기서는 사람이 필기한 우편번호를 인식하는 프로그램을 만든다.
- 먼저, 학습 모델을 만들어보자.
- ex10\_12.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Activation, Dropout
8 from tensorflow.keras.optimizers import Adam
9 from tensorflow.keras.utils import to_categorical
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train = x_train.reshape(60000, 784)
13 x_test = x_test.reshape(10000, 784)
14 x_train = x_train.astype(np.float32) / 255.0
15 x_test = x_test.astype(np.float32) / 255.0
```

# Section 03 딥러닝 튜닝

## □ 우편번호 인식기 v1.0

- ex10\_12.ipynb

```
16 y_train = to_categorical(y_train, 10)
17 y_test = to_categorical(y_test, 10)
18
19 model = Sequential()
20
21 model.add(Dense(1024, input_dim=784, activation="relu"))
22 model.add(Dense(512, activation='relu'))
23 model.add(Dense(512, activation='relu'))
24 model.add(Dense(10, activation='softmax'))
25
26 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
27 history = model.fit(x_train, y_train, batch_size=128, verbose=2, epochs=50, validation_data=(x_test, y_test))
28
29 print("정확도=", model.evaluate(x_test, y_test, verbose=0)[1] * 100)
30
31 model.save('model_trained.h5')
32
```

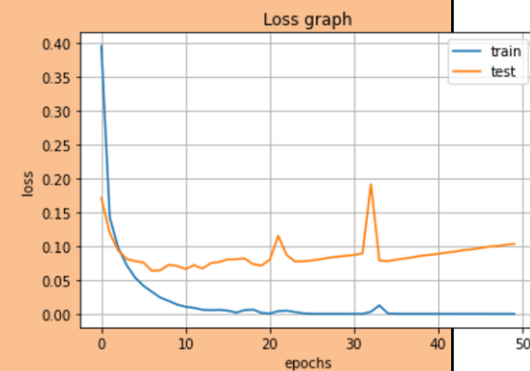
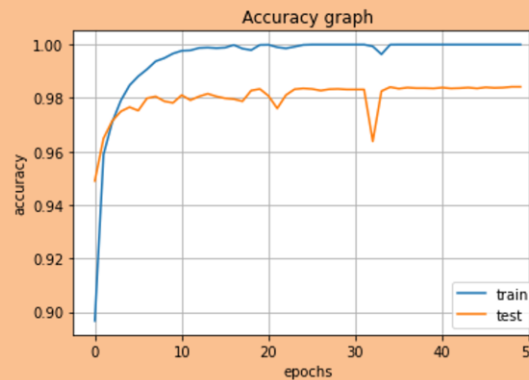
# Section 03 딥러닝 튜닝

## □ 우편번호 인식기 v1.0

- ex10\_12.ipynb

469/469 - 15s - loss: 9.1760e-06 - acc: 1.0000 - val\_loss: 0.1001 - val\_acc: 0.9839 - 15s/epoch - 32ms/step  
Epoch 49/50  
469/469 - 15s - loss: 7.3259e-06 - acc: 1.0000 - val\_loss: 0.1019 - val\_acc: 0.9842 - 15s/epoch - 32ms/step  
Epoch 50/50  
469/469 - 18s - loss: 6.1219e-06 - acc: 1.0000 - val\_loss: 0.1032 - val\_acc: 0.9842 - 18s/epoch - 38ms/step  
정확률 = 98.4200007629395

```
33 plt.plot(history.history['acc'])
33 plt.plot(history.history['acc'])
34 plt.plot(history.history['val_acc'])
35 plt.title('Accuracy graph')
36 plt.xlabel('epochs')
37 plt.ylabel('accuracy')
38 plt.legend(['train', 'test'])
39 plt.grid()
40 plt.show()
41
42 plt.plot(history.history['loss'])
43 plt.plot(history.history['val_loss'])
44 plt.title('Loss graph')
45 plt.xlabel('epochs')
46 plt.ylabel('loss')
47 plt.legend(['train', 'test'])
48 plt.grid()
49 plt.show()
```



## Section 03 딥러닝 튜닝

---

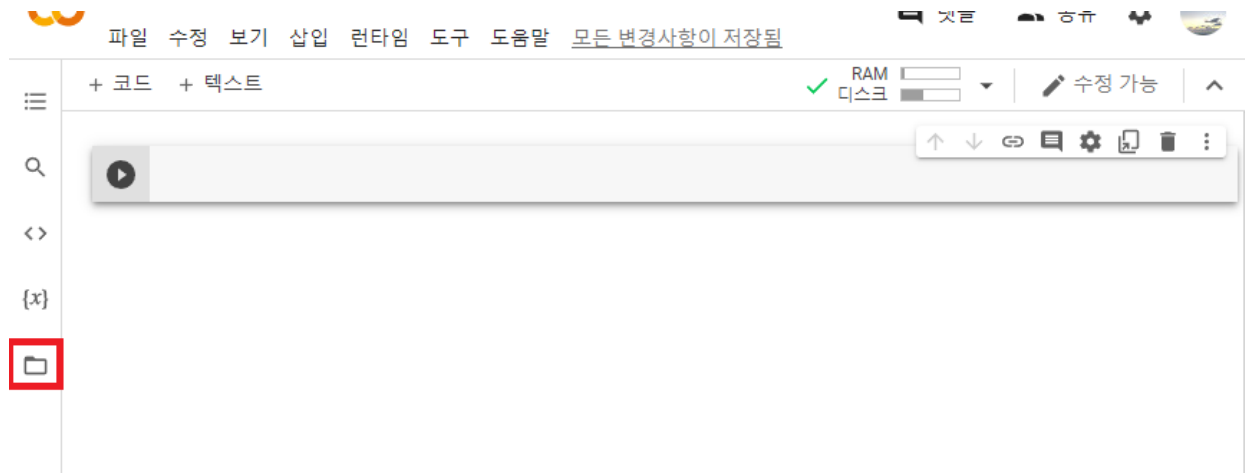
### □ 우편번호 인식기 v1.0

- 예제 10-12에서 눈 여겨 볼 곳은 31행의 `save` 함수다.
- 이 함수는 `fit` 함수로 학습을 마친 신경망 모델의 구조 정보와 가중치 값을 지정한 파일에 저장한다.
- 파일 포맷은 대용량의 과학 데이터를 계층적으로 저장할 때 주로 쓰는 `h5`다.
- 딥러닝에서는 학습에 시간이 많이 걸리기 때문에 학습을 마치면 `save`로 신경망 정보를 파일에 저장해두고 필요할 때 `load_model` 함수로 불러다 쓰곤 한다.
- 우편번호 인식기 프로그램에서 이 파일을 불러다 쓴다.

## Section 03 딥러닝 튜닝

### □ 우편번호 인식기 v1.0

- 그럼 구글 코랩에서 생성된 딥러닝 모델을 확인해보자.
- 구글 코랩 노트에 왼쪽中间的 '파일'버튼을 클릭한다.

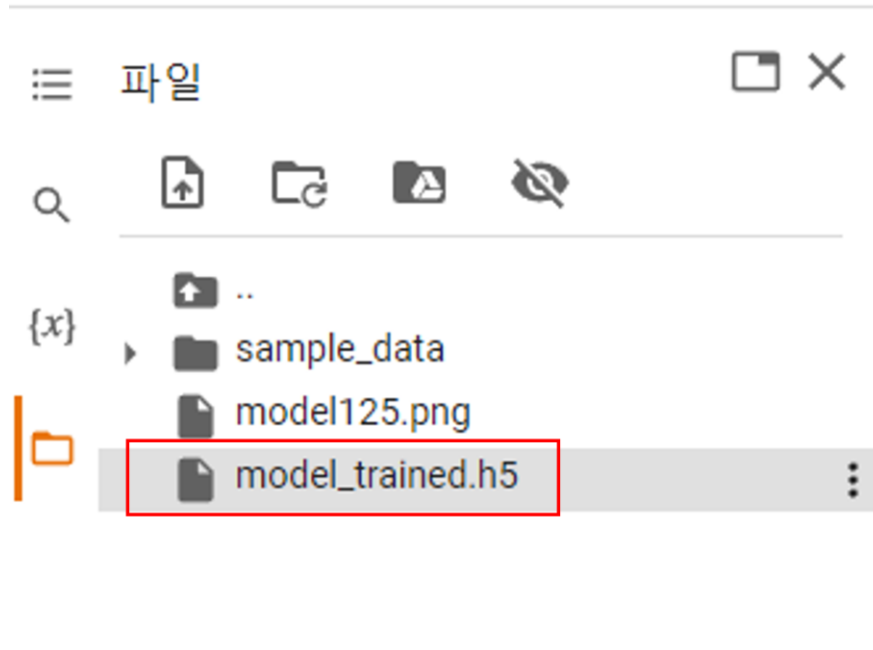




# Section 03 딥러닝 튜닝

## □ 우편번호 인식기 v1.0

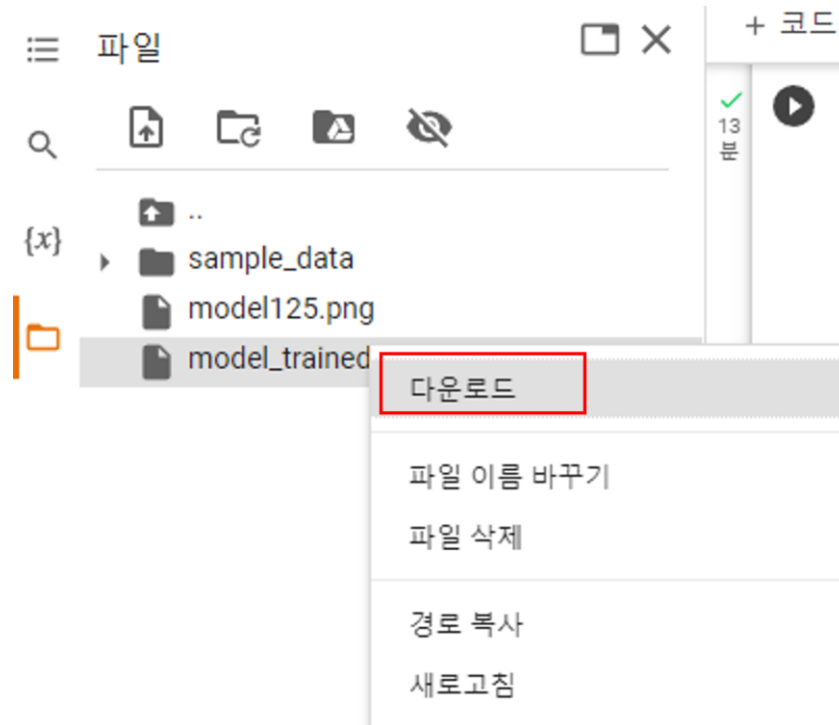
- 파일 저장 공간에 ex10\_12.ipynb에서 생성한 딥러닝 모델이 있는 것을 확인할 수 있다.



## Section 03 딥러닝 튜닝

### □ 우편번호 인식기 v1.0

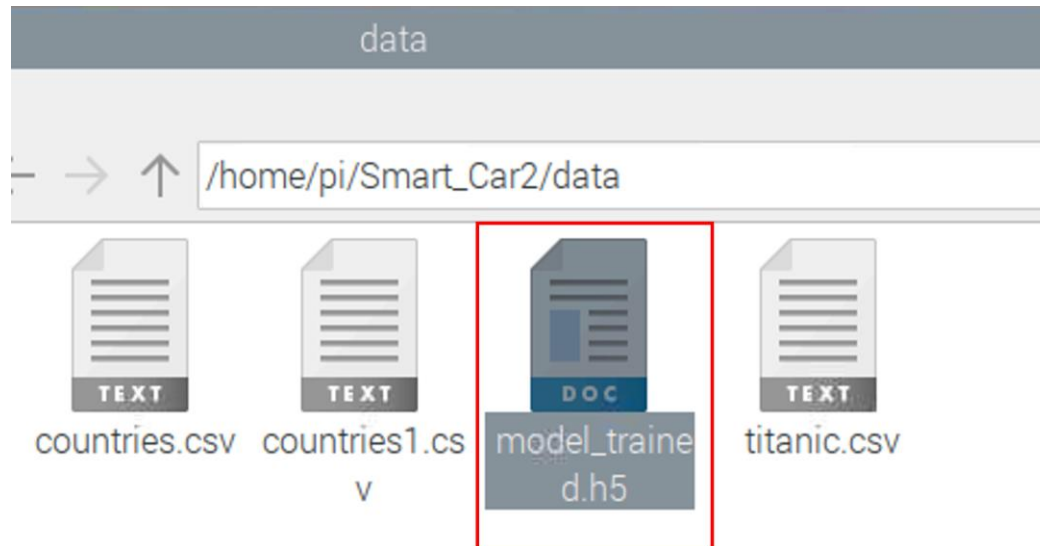
- 생성한 딥러닝 모델 파일을 다운받자.
- model\_trained.h5 파일을 선택하고 마우스 오른쪽 버튼을 클릭하고, 다운로드를 클릭한다.



## Section 03 딥러닝 튜닝

### □ 우편번호 인식기 v1.0

- 파일을 다운로드하면, 다운로드 폴더에 model\_trained.h5파일이 있을 것이다.
- 이제 이 파일을 라즈베리 파이의 /home/pi/Smart\_Car2/data 폴더로 이동시킨다.



## Section 03 딥러닝 튜닝

### □ 우편번호 인식기 v1.0

- **예제 10-13은 마우스로 그린 숫자를 인식하는 코드이다.**
- Ex10\_13.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5
6 model = tf.keras.model.load_model('./data/model_trained.h5')
7
8 def reset():
9     global img
10    img = np.ones((200, 520, 3), dtype=np.uint8) * 255
11    for i in range(5):
12        cv2.rectangle(img, (10 + i * 100, 50), (10 + (i + 1) * 100, 150), (0, 0, 255))
13    cv2.putText(img, 'e:erase s:show r:recognition q:quit', (10, 40), cv2.FONT_HERSHEY_SIMPLEX,
14                0.8, (255, 0, 0), 1)
```

# Section 03 딥러닝 튜닝

## □ 우편번호 인식기 v1.0

- **예제** 10-13은 마우스로 그린 숫자를 인식하는 코드이다.
- Ex10\_13.ipynb

```
15 def grab_numerals():
16     numerals = []
17     for i in range(5):
18         roi = img[51:149, 11 + i * 100:9 + (i + 1) * 100, 0]
19         roi = 255 - cv2.resize(roi, (28, 28), interpolation = cv2.INTER_CUBIC)
20         numerals.append(roi)
21     numerals = np.array(numerals)
22     return numerals
23
24 def show():
25     numerals = grab_numerals()
26     plt.figure(figsize = (25, 5))
27     for i in range(5):
28         plt.subplot(1, 5, i + 1)
29         plt.imshow(numerals[i], cmap='gray')
30         plt.xticks([]); plt.yticks([])
31     plt.show()
32
```

# Section 03 딥러닝 튜닝

## □ 우편번호 인식기 v1.0

- **예제 10-13은 마우스로 그린 숫자를 인식하는 코드이다.**
- Ex10\_13.ipynb

```
33 def recognition():
34     numerals = grab_numerals()
35     numerals = numerals.reshape(5, 784)
36     epochs = 10
37     numerals = numerals.astype(np.float32) / 255.0
38     pred = model.predict(numerals)
39     class_id = np.argmax(pred, axis=1)
40     for i in range(5):
41         cv2.putText(img, str(class_id[i]), (50 + i * 100, 180), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 1)
42
43     BrushSize = 4
44     LColor = (0, 0, 0)
45
46     def writing(event, x, y, flags, param):
47         if event == cv2.EVENT_LBUTTONDOWN:
48             cv2.circle(img, (x, y), BrushSize, LColor, -1)
49         elif event == cv2.EVENT_MOUSEMOVE and flags == cv2.EVENT_FLAG_LBUTTON:
50             cv2.circle(img, (x, y), BrushSize, LColor, -1)
51
```

## Section 03 딥러닝 튜닝

### □ 우편번호 인식기 v1.0

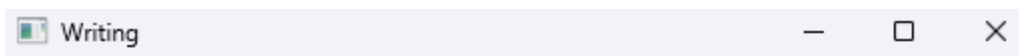
- **예제 10-13은 마우스로 그린 숫자를 인식하는 코드이다.**
- Ex10\_13.ipynb

```
52 reset()
53 cv2.namedWindow("Writing")
54 cv2.setMouseCallback("Writing", writing)
55 while(True):
56     cv2.imshow('Writing', img)
57     key = cv2.waitKey(1)
58     if key == ord('e'):
59         reset()
60     elif key == ord('s'):
61         show()
62     elif key == ord('r'):
63         recognition()
64     elif key == ord('q'):
65         break
66
67 cv2.destroyAllWindows()
```

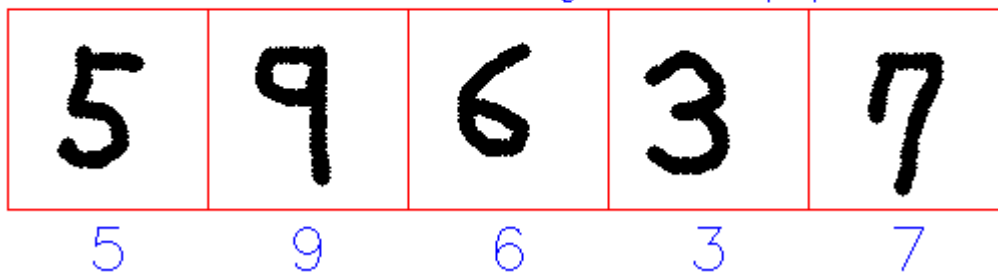
## Section 03 딥러닝 튜닝

### □ 우편번호 인식기 v1.0

- 예제 10-13의 사용자 인터페이스를 파악하자.
- 5개의 빨간색 박스는 마우스로 숫자를 써 넣는 곳이다.
- e는 박스를 지우고, s는 박스에서 숫자를 떼내어 명암 영상으로 표시하고, r은 인식을 하고, q는 끝내는 명령어다.



e:erase s:show r:recognition q:quit





## Section 03 딥러닝 튜닝

---

### □ 우편번호 인식기 v1.0

- 실행 결과를 보면, 틀린 샘플이 보일 수 있다.
- 쓰고 인식하는 일을 반복해보면 상당히 많이 틀려 ex10\_12.py에서 측정한 98.42%에 크게 미치지 못한다는 사실을 알 수 있다.
- 미국 사람이 종이에 쓴 숫자를 스캐너로 수집한 MNIST와 화면에 마우스로 쓴 패턴이 달라 차이가 발생한다.
- 다음 장에서는 컨볼루션 신경망으로 정확률을 크게 향상한 버전<sup>2</sup>를 만든다.

---

# Q&A

