

# 인공신경망의 구현 - **Tensorflow 2**

지 준 교수

창의융합대학 AI 응용학과

Hansung University

# Contents

- **Language** : Python with Anaconda

- Numpy, Matplotlib, Pandas ...



- **IDE** (Integrated Development Environment : 통합개발환경)

- Local - Jupyter Notebook
- Online - <https://colab.research.google.com/>



- **Framework**

- Tensorflow 2.0 (with Keras)
- Pytorch



# Python using Anaconda



**Python is the most popular language** among AI developers

- Easy to learn and simple syntax
- a lot of frameworks and libraries such as Numpy, Scikit-learn, TensorFlow and PyTorch etc.



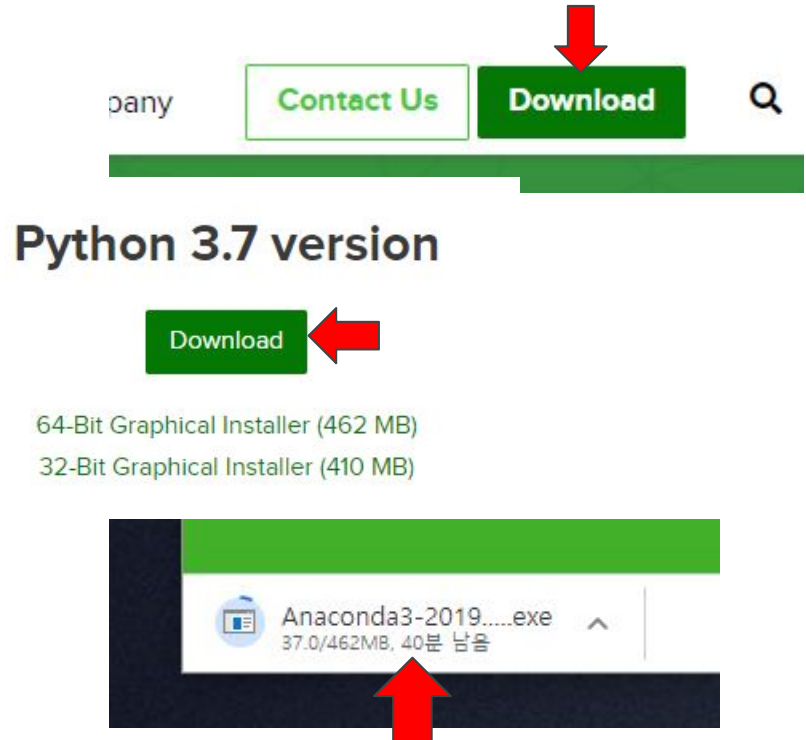
**Anaconda is the most popular standard platform** for Python data science including Machine Learning.



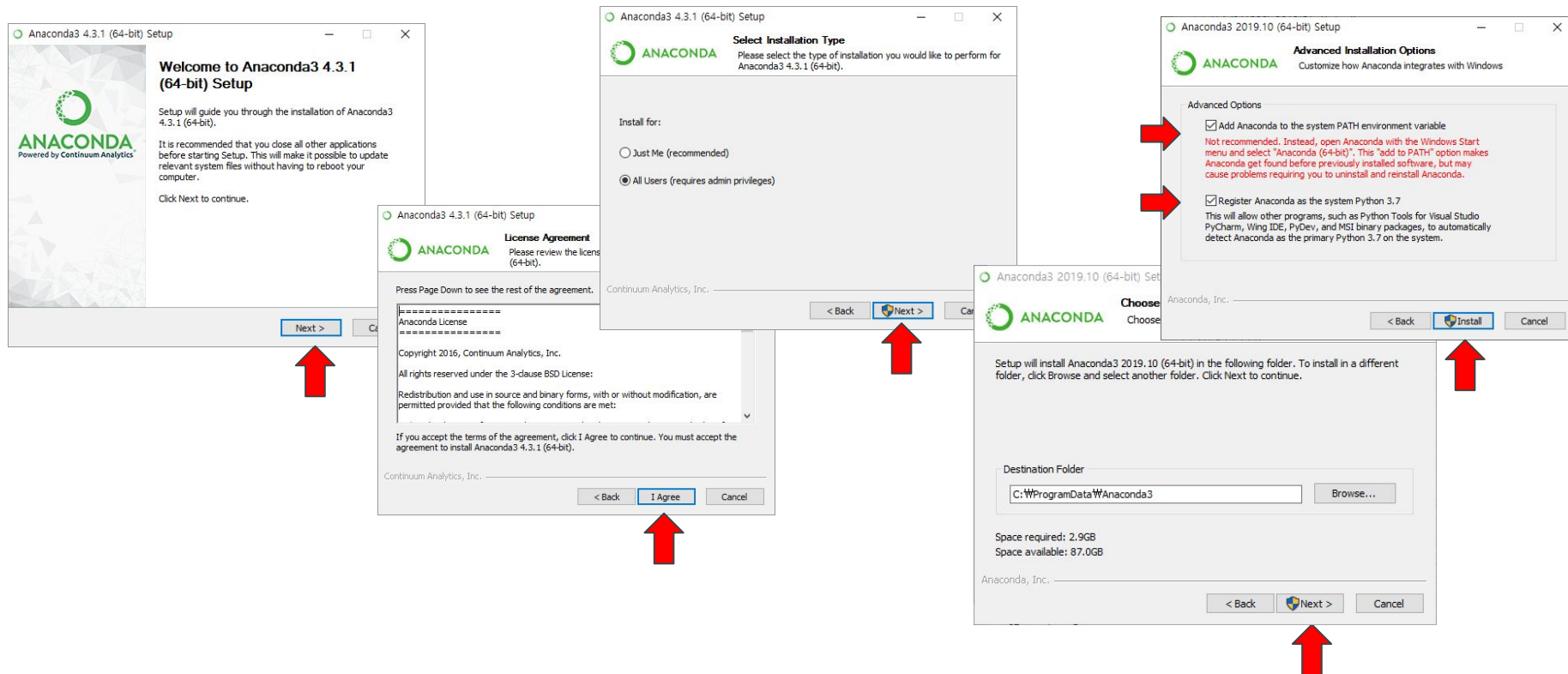
Recommend to install [Anaconda](#) or to use [CoLab](#) to start an AI project.

# Anaconda - Install

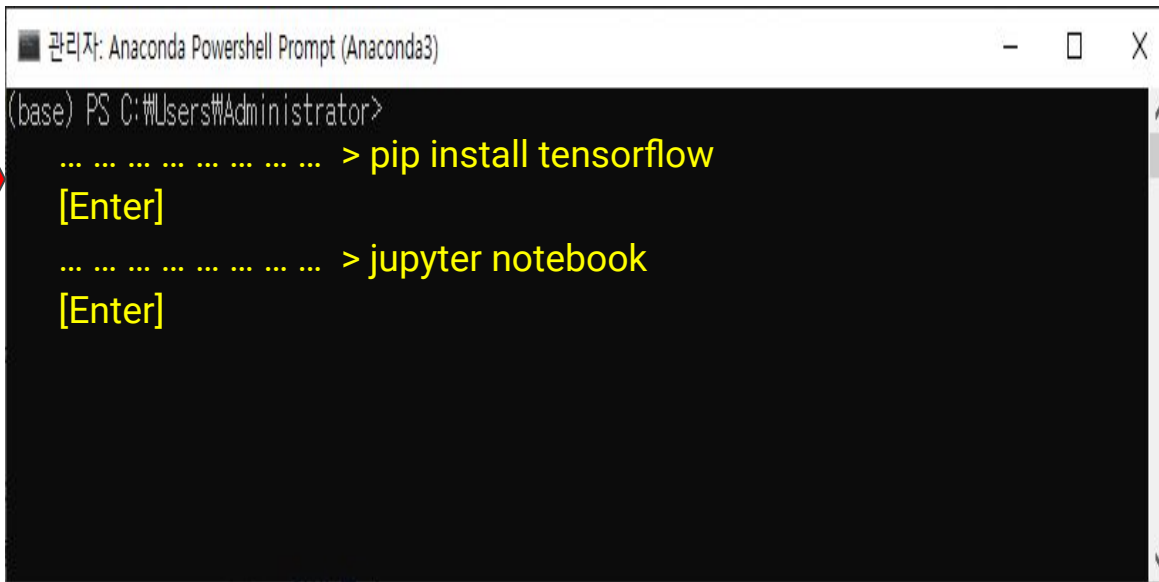
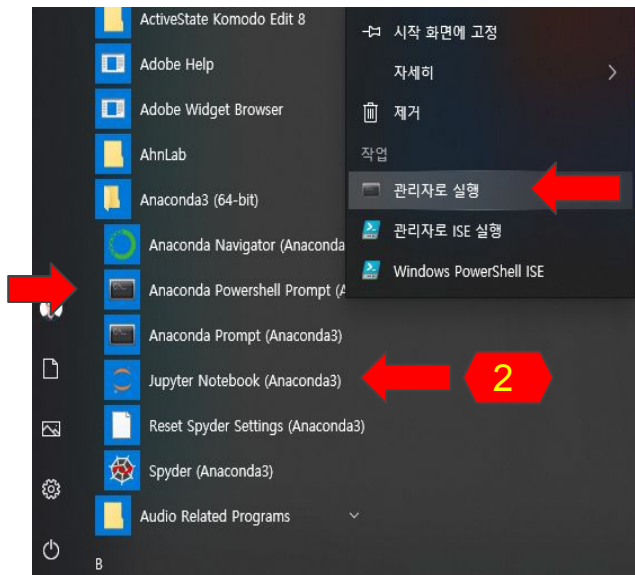
1. [Option] install [Chrome Browser](#)
2. visit <https://www.anaconda.com/>
3. Click “download” button
4. Click “Python 3.7 Version” (Download)
5. Install by clicking the downloaded file



# Anaconda - Install (continued)



# Anaconda - Install (continued)



**Warning** : When Click “Anaconda Powershell Prompt” use **right mouse button** and select **Run as Administrator** (관리자로 실행) as shown in the image above :

- 주의: 윈도우 사용자명은 반드시 영어로 되어있어야 합니다.
- 주의: 설치명령 일부변경

# Jupyter Notebook - Starting

The screenshot displays the Jupyter Notebook web interface in a browser. The main panel shows a file tree on the left with folders like '3D Objects', 'Contacts', 'Desktop', 'Documents', 'Downloads', 'Dropbox', 'erdasnet\_licensing', and 'Evernote'. A red arrow points to the 'New' button in the top right of the file tree, which has opened a dropdown menu. This menu includes options for 'Notebook: Python 3' and 'Other: Text File, Folder, Terminal'. Another red arrow points to the 'Folder' option. A 'Rename directory' dialog box is open in the foreground, prompting the user to 'Enter a new directory name:' with the text 'my folder' entered. The dialog has 'Cancel' and 'Rename' buttons. The top of the interface shows tabs for 'Files', 'Running', and 'Clusters', with 'Files' being the active tab. The browser address bar shows 'localhost:8888/tree#'. The Jupyter logo and 'Quit'/'Logout' buttons are visible at the top of the main panel.

Home x +

localhost:8888/tree#

jupyter

Files Running Clusters

Select items to perform actions on them.

Upload New

Notebook: Python 3

Other: Text File Folder Terminal

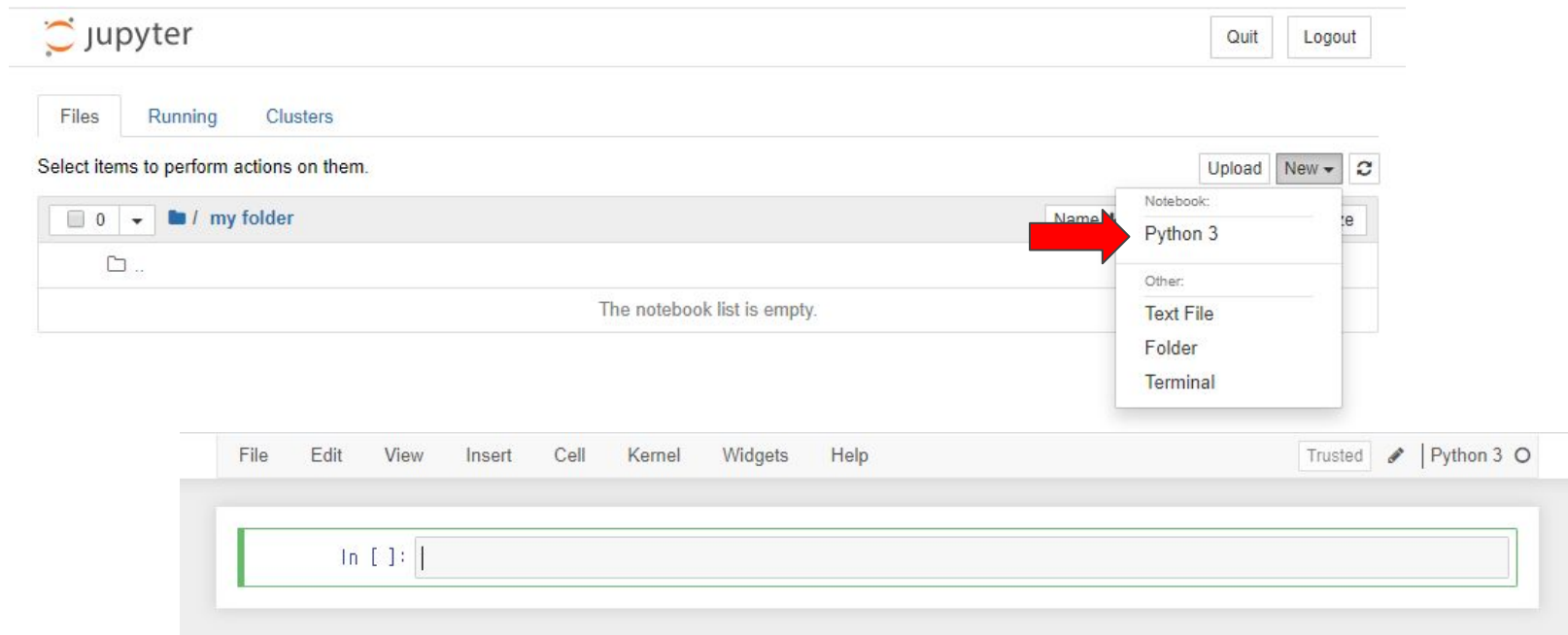
Rename directory

Enter a new directory name:

my folder

Cancel Rename

# Jupyter Notebook - Starting



The screenshot displays the Jupyter Notebook web interface. At the top, the Jupyter logo is on the left, and 'Quit' and 'Logout' buttons are on the right. Below this is a navigation bar with 'Files', 'Running', and 'Clusters' tabs. A message 'Select items to perform actions on them.' is shown. The main area features a file browser for '/ my folder' with a '0' icon and a 'Name' input field. A red arrow points to the 'New' button, which has opened a dropdown menu. The menu lists 'Notebook:' with 'Python 3' selected, and 'Other:' with 'Text File', 'Folder', and 'Terminal' options. Below the file browser, it says 'The notebook list is empty.' At the bottom, a toolbar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. On the right of the toolbar are 'Trusted', a pencil icon, and 'Python 3' with a refresh icon. The main workspace shows a code cell with 'In [ ]: |'.

<https://www.dataquest.io/blog/jupyter-notebook-tutorial/>



# Jupyter Notebook - document in markdown

# This is a level 1 heading

## This is a level 2 heading

This is some plain text that forms a paragraph.

Add emphasis via **\*\*bold\*\*** and \_\_bold\_\_ or *\*italic\** and \_italic\_

Paragraphs must be separated by an empty line.

\* Sometimes we want to include lists.

\* Which can be indented.

1. Lists can also be numbered.

2. For ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: `foo()` and code blocks use triple backticks:

```
'''
```

```
bars()
```

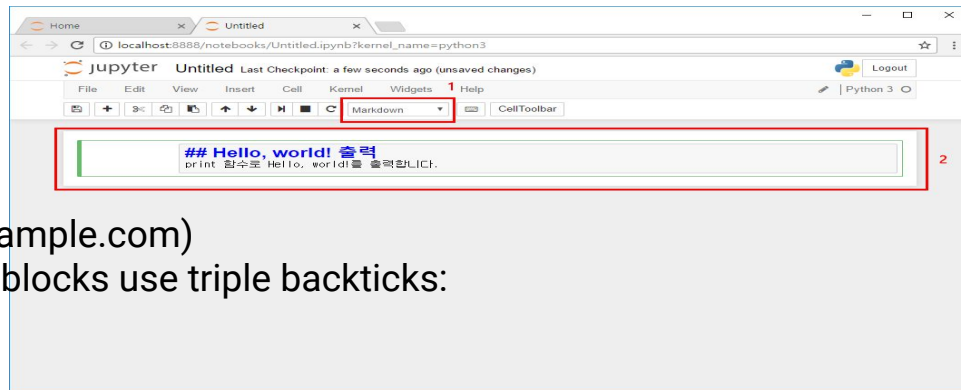
```
'''
```

Or can be indented by 4 spaces:

```
foo()
```

And finally, adding images is easy:

![Alt text](https://www.citypng.com/public/uploads/preview/-11596996178apaxd1pez2.png)



# Jupyter Notebook - code

The screenshot shows a web browser window with a Jupyter Notebook titled "Untitled". The URL is `localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3`. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. A red box labeled "1" highlights the "Run" button (a play icon) in the toolbar. Below the toolbar, the notebook content area displays the text "Hello, world! 출력" and "print 함수로 Hello, world!를 출력합니다." Below this text, a code cell is shown with the code `In [ ]: print('Hello, world!')`. A red box labeled "2" highlights the code input area of this cell.

Home x Untitled x

localhost:8888/notebooks/Untitled.ipynb?kernel\_name=python3

jupyter Untitled Last Checkpoint: 8 minutes ago (unsaved changes) Logout

File Edit View Insert Cell 1 Kernel Widgets Help

+ < > Run CellToolbar

Code

Python 3

Hello, world! 출력

print 함수로 Hello, world!를 출력합니다.

In [ ]: `print('Hello, world!')` 2

주피터-1 노트북에서 인공지능망은 훈련

## Ex) Handwritten digits recognition (in core) -

```
import tensorflow as tf
from tensorflow import keras

(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\n테스트 정확도:', test_acc)
```

## Ex) Handwritten digits recognition (in detail)

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
print(tf.__version__)
```

```
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)
```

```
plt.figure()
```

```
plt.imshow(train_images[9], cmap='gray')
```

```
plt.title(train_labels[9])
```

```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

디자이너

가즈=1271

9번째 이미지

2271

0~1로 노멀라이징

```
plt.figure(figsize=(6,7))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(train_images[i], cmap='gray')
    plt.title(train_labels[i], fontsize=16)
```

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
model.summary()
```

모델 만들기  
28x28 의 input (1번째 레이어)  
128개의 노드로 구성된 히든 레이어 (2번째)  
0~9 의 10개의 노드를 가진 출력 레이어  
모델 정보보기

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print("\n테스트 정확도:", test_acc)
```

cost  
훈련  
테스트

```
predictions = model.predict(test_images)
```

i=3 34-172 222

```
print(predictions[i])
```

```
np.argmax(predictions[i])
```

0~9 사이의 각각의 확률

```
plt.figure(figsize=(2,2))
```

```
plt.imshow(test_images[i], cmap='gray')
```

```
plt.xticks([])
```

```
plt.yticks([])
```

```
plt.title(np.argmax(predictions[i]))
```

## Ex) Handwritten digits recognition (Option) v.2

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)

plt.figure(figsize=(6,7))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(train_images[i], cmap='gray')
    plt.title(train_labels[i])
```

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
model.summary()
keras.utils.plot_model(model, show_shapes=True)
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=1)
print('\n테스트 정확도:', test_acc)
```

```
predictions = model.predict(test_images)
plt.imshow(test_images[9], cmap='gray')
plt.title(np.argmax(predictions[9]))
```

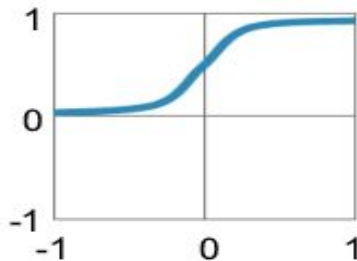


# Activations ( <https://keras.io/activations/> )

활성화 함수

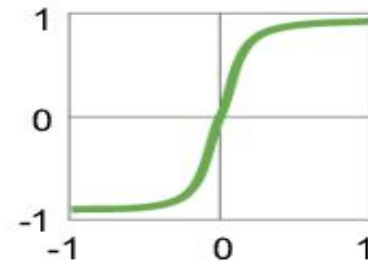
Traditional  
Non-Linear  
Activation  
Functions

Sigmoid



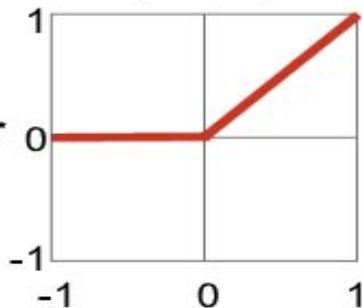
$$y = 1 / (1 + e^{-x})$$

Hyperbolic Tangent



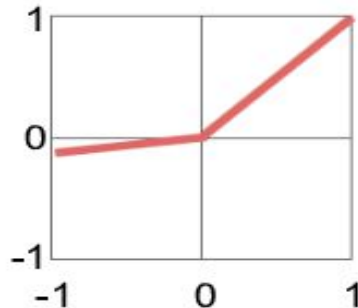
$$y = (e^x - e^{-x}) / (e^x + e^{-x})$$

Rectified Linear Unit  
(ReLU)



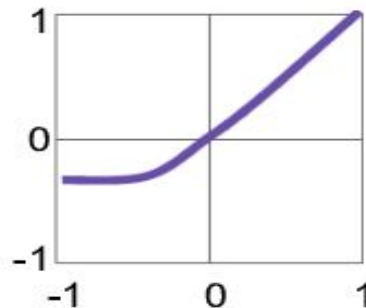
$$y = \max(0, x)$$

Leaky ReLU



$$y = \max(\alpha x, x)$$

Exponential LU

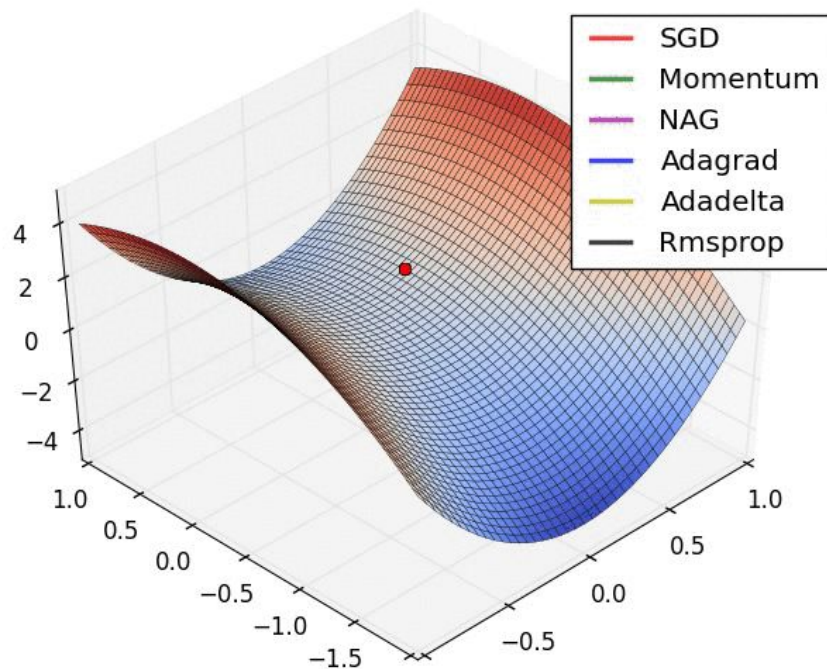
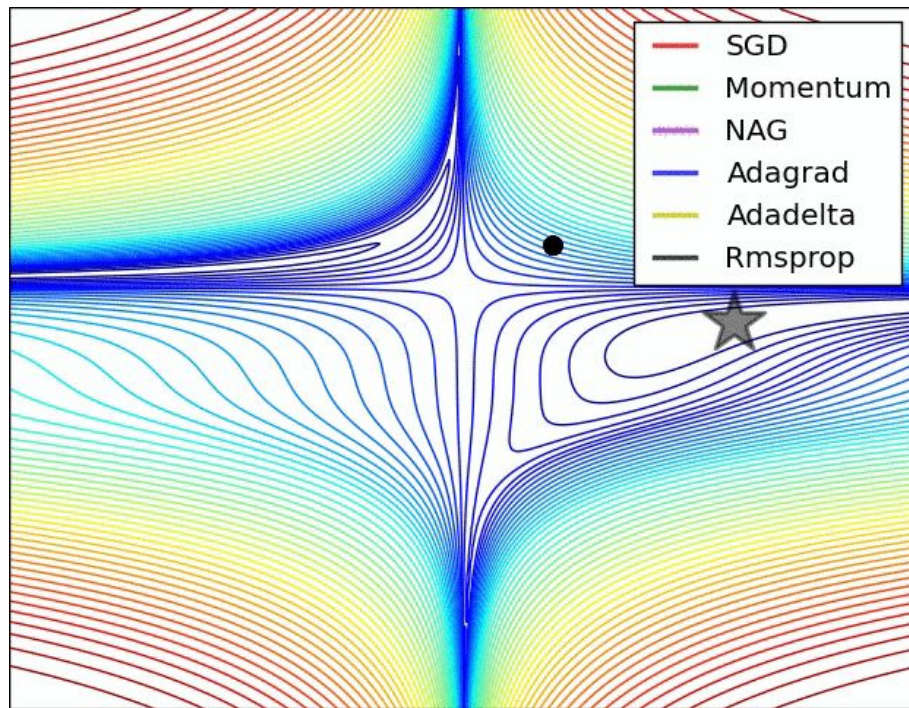


$$y = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

$\alpha$  = small const. (e.g. 0.1)

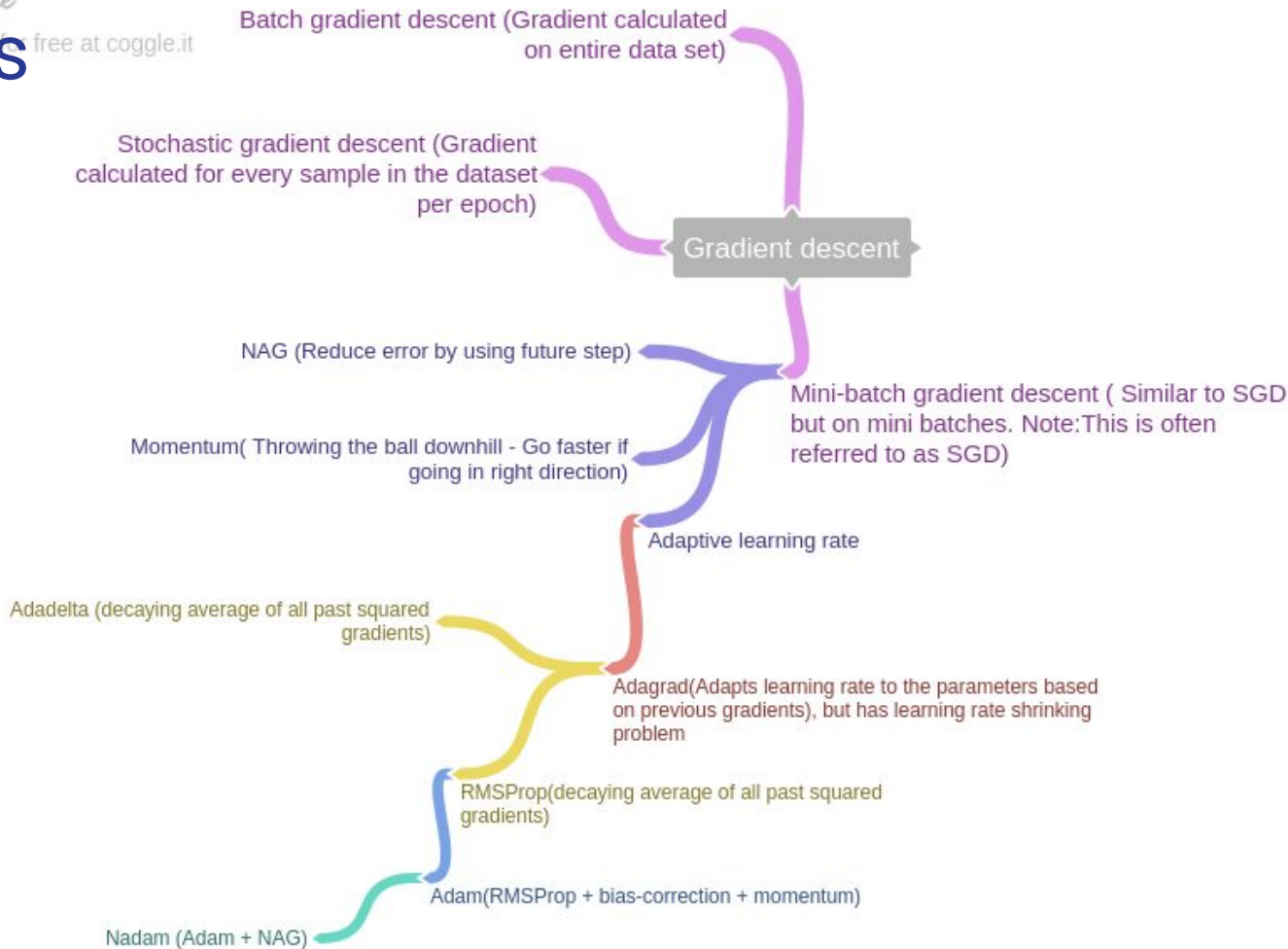
Modern  
Non-Linear  
Activation  
Functions

# Optimizers ( <https://keras.io/optimizers/> )



# Optimizers

coggle  
made for free at coggle.it



# Loss Functions ( <https://keras.io/losses/> )

1. Regression Loss Functions
  1. Mean Squared Error Loss
  2. Mean Squared Logarithmic Error Loss
  3. Mean Absolute Error Loss
2. Binary Classification Loss Functions
  1. Binary Cross-Entropy
  2. Hinge Loss
  3. Squared Hinge Loss
3. Multi-Class Classification Loss Functions
  1. Multi-Class Cross-Entropy Loss
  2. Sparse Multiclass Cross-Entropy Loss
  3. Kullback Leibler Divergence Loss

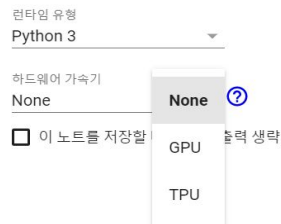
필요자하는 문제에 따라  
고른다

# Google CoLab - Start

Google Colab is a free Jupyter notebook environment provided by Google where you can use free GPUs and TPUs.

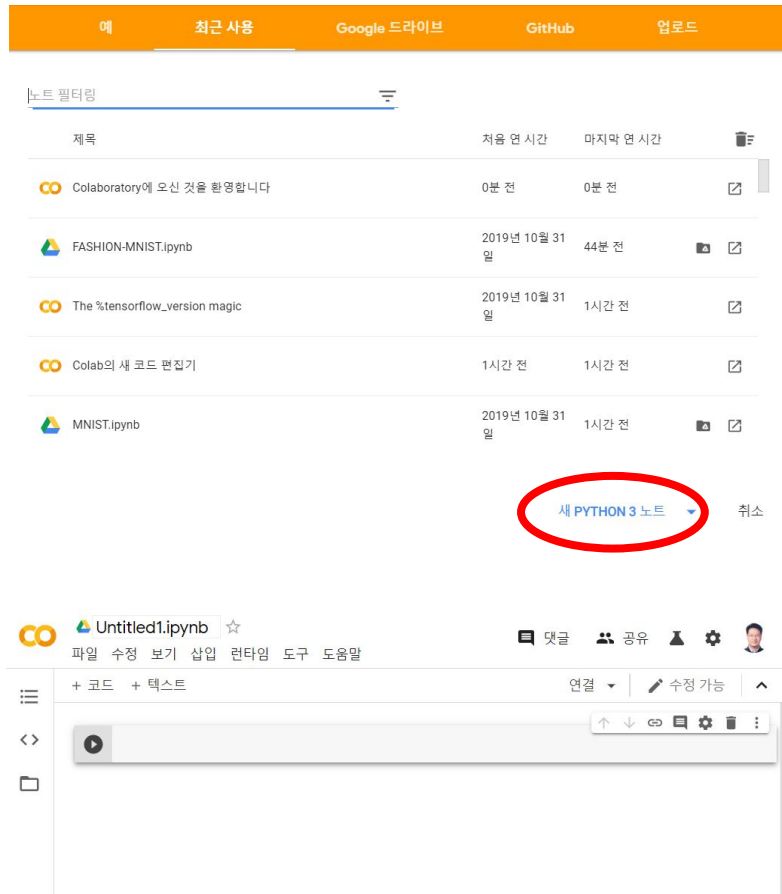
- go to this link <https://colab.research.google.com>
- On opening the website you will see a pop-up containing following tabs – 예 / 최근사용 / Google 드라이브 / GitHub / 업로드
- create a new Jupyter notebook by clicking New Python3 Notebook
- Click the “수정/런타임” dropdown menu. Select “노트 설정/런타임 유형변경”  
an select anything (GPU, CPU, None)  
you want

노트 설정



취소

저장



# Examples using Google CoLab

1. Hand-written digits recognition :

[https://colab.research.google.com/drive/1J5DOjzt4lyv8HDN0dA\\_4EYKbCJVa\\_9TE](https://colab.research.google.com/drive/1J5DOjzt4lyv8HDN0dA_4EYKbCJVa_9TE)

2. Fashion-items recognition :

[https://colab.research.google.com/drive/1-2RXklyrFtb3A\\_SC6PW2G6lpO4xBJ3ia](https://colab.research.google.com/drive/1-2RXklyrFtb3A_SC6PW2G6lpO4xBJ3ia)

3. Tensorflow Tutorial : <https://www.tensorflow.org/tutorials>

4. Tensorflow Dataset : <https://www.tensorflow.org/datasets/catalog/overview>

# (ex #1) Hand-written digits recognition

## 첫 번째 신경망 훈련하기: 기초적인 분류 문제 - 초보자용

이 튜토리얼에서는 손으로 쓴 숫자를 인식하는 신경망 모델을 훈련합니다. 상세 내용을 모두 이해하지 못해도 괜찮습니다. 여기서는 완전한 텐서플로(TensorFlow) 2.0 프로그램을 빠르게 살펴 보겠습니다. 자세한 내용은 앞으로 배우면서 더 설명합니다.

우선 텐서플로 모델을 만들고 훈련할 수 있는 고수준 API인 **tf.keras**를 사용합니다.

```
# tensorflow 2.0 을 선택합니다.  
# %tensorflow_version 2.x  
  
# tensorflow와 tf.keras를 임포트합니다  
import tensorflow as tf  
from tensorflow import keras  
  
# 헬퍼(helper) 라이브러리를 임포트합니다  
import numpy as np  
import matplotlib.pyplot as plt  
  
print(tf.__version__) # 임포트된 Tensorflow 버전을 확인합니다.
```

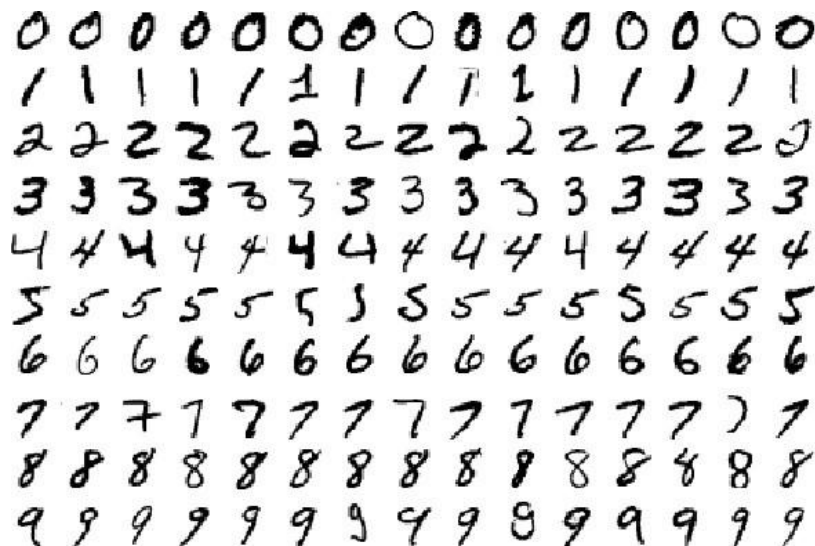
2.2.0-rc1



## MNIST 데이터셋 임포트하기

10개의 라벨과 70,000개의 흑백 이미지로 구성된 MNIST 데이터셋을 사용하겠습니다. 이미지는 해상도 (28x28 픽셀)가 낮고 다음처럼 숫자를 나타냅니다.

MNIST 데이터셋을 로드하여 준비합니다. 네트워크를 훈련하는데 60,000개의 흑백 숫자 이미지를 사용합니다. 그 다음 네트워크가 얼마나 정확하게 이미지를 분류하는지 10,000개의 이미지로 평가합니다. MNIST 데이터셋은 텐서플로에서 바로 임포트하여 로드할 수 있습니다:



```
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>  
11493376/11490434 [=====] - 0s 0us/step  
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)



## 데이터 탐색

모델을 훈련하기 전에 데이터셋 구조를 살펴보죠. 다음 코드는 훈련 세트에 **60,000**개의 이미지가 있다는 것을 보여줍니다. 각 이미지는 **28x28** 픽셀로 표현됩니다:

```
train_images.shape
```

```
(60000, 28, 28)
```

**비슷하게** 훈련 세트에는 **60,000**개의 레이블이 있습니다:

```
len(train_labels)
```

```
60000
```

각 레이블은 **0**과 **9**사이의 정수입니다:

```
train_labels
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

테스트 세트에는 **10,000**개의 이미지가 있습니다. 이 이미지도 **28x28** 픽셀로 표현됩니다:

테스트 세트는 **10,000**개의 이미지에 대한 레이블을 가지고 있습니다:

```
print(test_images.shape)
```

```
len(test_labels)
```

```
(10000, 28, 28)
```

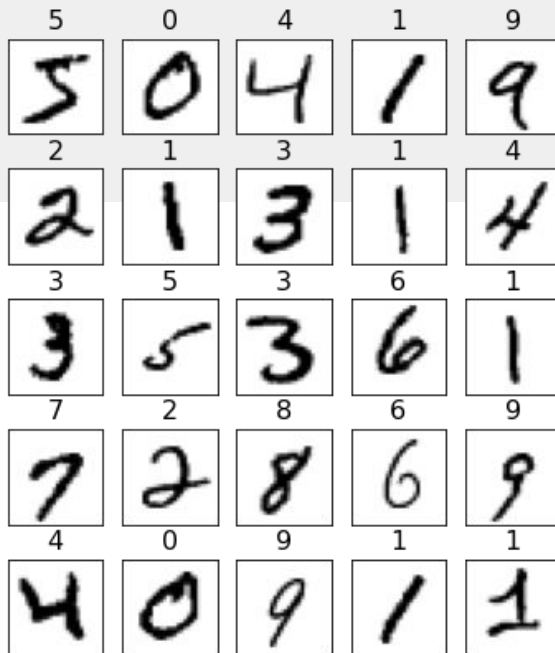
```
10000
```

## 데이터 전처리

네트워크를 훈련하기 전에 데이터를 전처리해야 합니다. 훈련 세트에 있는 첫 번째 이미지를 보면 픽셀 값의 범위가 0~255 사이라는 것을 알 수 있습니다:

```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
plt.figure(figsize=(6,7))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.title(train_labels[i], fontsize=16)
```



층을 차례대로 쌓아 `tf.keras.Sequential` 모델을 만듭니다. 훈련에 사용할 옵티마이저(optimizer)와 손실 함수를 선택합니다:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    # keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# model.summary()
# keras.utils.plot_model(model, show_shapes=True)
```

모델을 훈련하고 평가합니다:

```
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\n테스트 정확도:', test_acc)
```

```
Epoch 1/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.2601 - accuracy: 0.9255
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0458 - accuracy: 0.9857
313/313 - 0s - loss: 0.0813 - accuracy: 0.9755
테스트 정확도: 0.9754999876022339
```

훈련된 이미지 분류기는 이 데이터셋에서 약 **97.7%**의 정확도를 달성합니다. 더 자세한 내용은 [TensorFlow 튜토리얼](#)을 참고하세요.

테스트 세트의 정확도가 훈련 세트의 정확도보다 조금 낮습니다. 훈련 세트의 정확도와 테스트 세트의 정확도 사이의 차이는 과적합(**overfitting**) 때문입니다. 과적합은 머신러닝 모델이 훈련 데이터보다 새로운 데이터에서 성능이 낮아지는 현상을 말합니다.

## 예측 만들기

훈련된 모델을 사용하여 이미지에 대한 예측을 만들 수 있습니다.

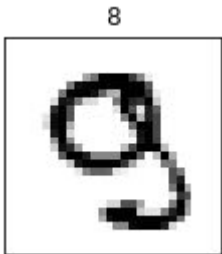
```
predictions = model.predict(test_images)
i=151
predictions[i]
```

```
array([1.8292204e-04, 7.0453643e-10, 1.2538026e-02, 3.4020868e-01,
       1.2377453e-10, 1.6020803e-03, 3.1261842e-09, 6.3065016e-07,
       3.8707632e-01, 2.5839132e-01], dtype=float32)
```

이 예측은 **10**개의 숫자 배열로 나타납니다. 이 값은 **10**개의 패션 품목에 상응하는 모델의 신뢰도(**confidence**)를 나타냅니다. 가장 높은 신뢰도를 가진 레이블을 찾아보죠:

```
plt.figure(figsize=(2,2))
plt.imshow(test_images[i], cmap=plt.cm.binary)
plt.xticks([])
plt.yticks([])
plt.title(np.argmax(predictions[i]))
```

```
Text(0.5, 1.0, '8')
```



```
predict = np.argmax(predictions, axis=1)
compare = np.equal(predict, test_labels)
wrong = np.where(compare == False)
wrong
```

```
(array([ 149,  151,  217,  247,  259,  321,  340,  381,  445,  448,  478,
        . . .
        9891, 9904, 9941]),)
```

## (ex #2) Fashion items recognition

### 첫 번째 신경망 훈련하기: 기초적인 분류 문제

이 튜토리얼에서는 운동화나 셔츠 같은 패션 아이템 이미지를 분류하는 신경망 모델을 훈련합니다. 상세 내용을 모두 이해하지 못해도 괜찮습니다. 여기서는 완전한 텐서플로(TensorFlow) 프로그램을 빠르게 살펴 보겠습니다. 자세한 내용은 앞으로 배우면서 더 설명합니다.

여기에서는 텐서플로 모델을 만들고 훈련할 수 있는 고수준 API인 **tf.keras**를 사용합니다.

```
# tensorflow 2.0 을 선택합니다.  
%tensorflow_version 2.x  
  
# tensorflow와 tf.keras를 임포트합니다  
import tensorflow as tf  
from tensorflow import keras  
  
# 헬퍼(helper) 라이브러리를 임포트합니다  
import numpy as np  
import matplotlib.pyplot as plt  
  
print(tf.__version__)
```

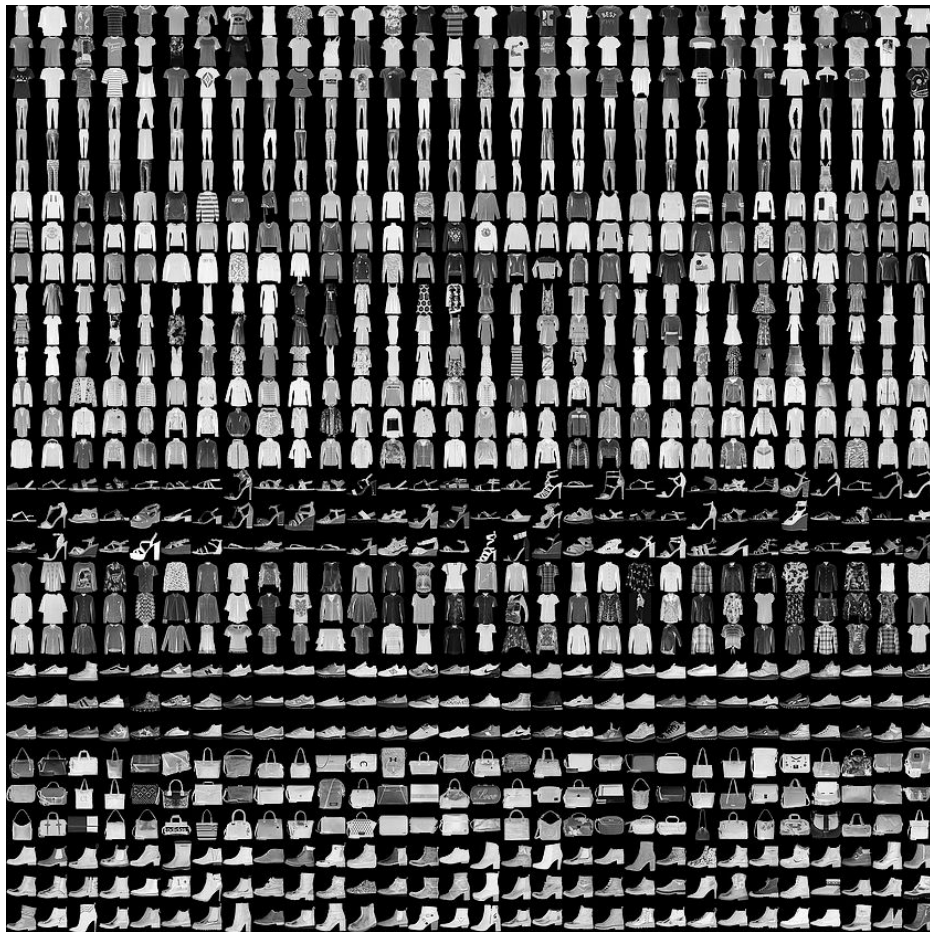
2.2.0-rc2

## 패션 MNIST 데이터셋 임포트하기

10개의 범주(category)와 70,000개의 흑백 이미지로 구성된 패션 MNIST 데이터셋을 사용하겠습니다. 이미지는 해상도(28x28 픽셀)가 낮고 다음처럼 개별 옷 품목을 나타냅니다:

패션 MNIST는 컴퓨터 비전 분야의 "Hello, World" 프로그램적인 고전 MNIST 데이터셋을 대신해서 자주 사용됩니다.

네트워크를 훈련하는데 60,000개의 이미지를 사용합니다. 그 다음 네트워크가 얼마나 정확하게 이미지를 분류하는지 또 다른 10,000개의 이미지로 평가합니다. 패션 MNIST 데이터셋은 텐서플로에서 바로 임포트하여 적재할 수 있습니다:



```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Downloading data from

<https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz>

32768/29515 [=====] - 0s 0us/step

. . .

Downloading data from

<https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz>

4423680/4422102 [=====] - 0s 0us/step

`load_data()` 함수를 호출하면 네 개의 넘파이(NumPy) 배열이 반환됩니다:

- `train_images`와 `train_labels` 배열은 모델 학습에 사용되는 훈련 세트입니다.
- `test_images`와 `test_labels` 배열은 모델 테스트에 사용되는 테스트 세트입니다.

이미지는 28x28 크기의 넘파이 배열이고 픽셀 값은 0과 255 사이입니다. 레이블(label)은 0에서 9까지의 정수 배열입니다. 이 값은 이미지에 있는 옷의 클래스(class)를 나타냅니다:

각 이미지는 하나의 레이블에 매핑되어 있습니다. 데이터셋에 클래스 이름이 들어있지 않기 때문에 나중에 이미지를 출력할 때 사용하기 위해 별도의 변수를 만들어 저장합니다:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal',
               'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

레이블	클래스
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



## 데이터 탐색

모델을 훈련하기 전에 데이터셋 구조를 살펴보죠. 다음 코드는 훈련 세트에 **60,000**개의 이미지가 있다는 것을 보여줍니다. 각 이미지는 **28x28** 픽셀로 표현됩니다:

```
train_images.shape
```

```
(60000, 28, 28)
```

비슷하게 훈련 세트에는 **60,000**개의 레이블이 있습니다: 각 레이블은 **0**과 **9**사이의 정수입니다:

```
print(len(train_labels))  
train_labels
```

```
60000  
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

테스트 세트에는 **10,000**개의 이미지가 있습니다. 이 이미지도 **28x28** 픽셀로 표현됩니다: 테스트 세트는 **10,000**개의 이미지에 대한 레이블을 가지고 있습니다:

```
print(test_images.shape)  
len(test_labels)
```

```
(10000, 28, 28)  
10000
```

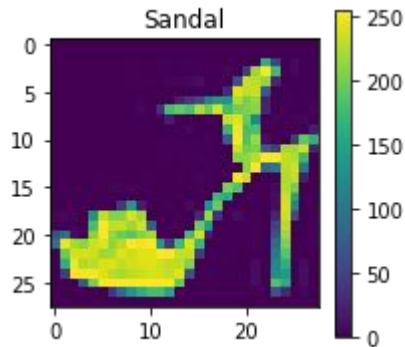
## 데이터 전처리

네트워크를 훈련하기 전에 데이터를 전처리해야 합니다. 훈련 세트에 있는 첫 번째 이미지를 보면 픽셀 값의 범위가 0~255 사이라는 것을 알 수 있습니다:

```
plt.figure(figsize=(3,3))
plt.imshow(train_images[9])
plt.title(class_names[train_labels[9]])
plt.colorbar()
```

신경망 모델에 주입하기 전에 이 값의 범위를 0~1 사이로 조정하겠습니다. 이렇게 하려면 255로 나누어야 합니다. 훈련 세트와 테스트 세트를 동일한 방식으로 전처리하는 것이 중요합니다:

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```



훈련 세트에서 처음 25개 이미지와 그 아래 클래스 이름을 출력해 보죠. 데이터 포맷이 올바른지 확인하고 네트워크 구성과 훈련할 준비를 마칩니다.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(train_images[i],
               cmap=plt.cm.binary)
    plt.title(class_names[train_labels[i]],
              fontsize=14)
```



## 모델 구성

신경망 모델을 만들려면 모델의 층을 구성한 다음 모델을 컴파일합니다.

### 층 설정

신경망의 기본 구성 요소는 층(layer)입니다. 층은 주입된 데이터에서 표현을 추출합니다. 아마도 문제를 해결하는데 더 의미있는 표현이 추출될 것입니다.

대부분 딥러닝은 간단한 층을 연결하여 구성됩니다. `tf.keras.layers.Dense`와 같은 층들의 가중치(parameter)는 훈련하는 동안 학습됩니다.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

이 네트워크의 첫 번째 층인 `tf.keras.layers.Flatten`은 2차원 배열(28 x 28 픽셀)의 이미지 포맷을  $28 * 28 = 784$  픽셀의 1차원 배열로 변환합니다. 이 층은 이미지에 있는 픽셀의 행을 펼쳐서 일렬로 늘립니다. 이 층에는 학습되는 가중치가 없고 데이터를 변환하기만 합니다.

픽셀을 펼친 후에는 두 개의 `tf.keras.layers.Dense` 층이 연속되어 연결됩니다. 이 층을 밀집 연결(densely-connected) 또는 완전 연결(fully-connected) 층이라고 부릅니다. 첫 번째 Dense 층은 128개의 노드(또는 뉴런)를 가집니다. 두 번째 (마지막) 층은 10개의 노드의 소프트맥스(softmax) 층입니다. 이 층은 10개의 확률을 반환하고 반환된 값의 전체 합은 1입니다. 각 노드는 현재 이미지가 10개 클래스 중 하나에 속할 확률을 출력합니다.

# 모델 컴파일

모델을 훈련하기 전에 필요한 몇 가지 설정이 모델 컴파일 단계에서 추가됩니다:

- 손실 함수(**Loss function**)-훈련 하는 동안 모델의 오차를 측정합니다. 모델의 학습이 올바른 방향으로 향하도록 이 함수를 최소화해야 합니다.
- 옵티마이저(**Optimizer**)-데이터와 손실 함수를 바탕으로 모델의 업데이트 방법을 결정합니다.
- 지표(**Metrics**)-훈련 단계와 테스트 단계를 모니터링하기 위해 사용합니다. 다음 예에서는 올바르게 분류된 이미지의 비율인 정확도를 사용합니다.

```
model.compile( optimizer='adam',  
               loss='sparse_categorical_crossentropy',  
               metrics=['accuracy'] )
```

## 모델 훈련

신경망 모델을 훈련하는 단계는 다음과 같습니다:

1. 훈련 데이터를 모델에 주입합니다-이 예에서는 **train\_images**와 **train\_labels** 배열입니다.
2. 모델이 이미지와 레이블을 매핑하는 방법을 배웁니다.
3. 테스트 세트에 대한 모델의 예측을 만듭니다-이 예에서는 **test\_images** 배열입니다. 이 예측이 **test\_labels** 배열의 레이블과 맞는지 확인합니다.

훈련을 시작하기 위해 **model.fit** 메서드를 호출하면 모델이 훈련 데이터를 학습합니다:

```
model.fit(train_images, train_labels, epochs=5)
```

Train on 60000 samples

Epoch 1/5

60000/60000 [=====] - 7s 122us/sample - loss: 0.4770 - accuracy: 0.8277

...

Epoch 5/5

60000/60000 [=====] - 7s 108us/sample - loss: 0.2827 - accuracy: 0.8945

<tensorflow.python.keras.callbacks.History at 0x7fb2906fc198>

모델이 훈련되면서 손실과 정확도 지표가 출력됩니다. 이 모델은 훈련 세트에서 약 **0.88(88%)** 정도의 정확도를 달성합니다.

## 정확도 평가

그다음 테스트 세트에서 모델의 성능을 비교합니다:

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\n테스트 정확도:', test_acc)
```

```
313/313 - 0s - loss: 2.4344 - accuracy: 0.0616
```

```
테스트 정확도: 0.06159999966621399
```

테스트 세트의 정확도가 훈련 세트의 정확도보다 조금 낮습니다. 훈련 세트의 정확도와 테스트 세트의 정확도 사이의 차이는 과적합(**overfitting**) 때문입니다. 과적합은 머신러닝 모델이 훈련 데이터보다 새로운 데이터에서 성능이 낮아지는 현상을 말합니다.

## 예측 만들기

훈련된 모델을 사용하여 이미지에 대한 예측을 만들 수 있습니다.

```
predictions = model.predict(test_images)
```

여기서는 테스트 세트에 있는 각 이미지의 레이블을 예측했습니다. 첫 번째 예측을 확인해 보죠:

```
predictions[0]
```

```
array([0.09425194, 0.07598469, 0.14607379, 0.12054448, 0.19235411,  
       0.10052773, 0.0466389 , 0.07125824, 0.08701146, 0.0653547 ],  
      dtype=float32)
```

```
np.argmax(predictions[0])
```

4

모델은 이 이미지가 앵클 부츠(class\_name[9])라고 가장 확신하고 있습니다. 이 값이 맞는지 테스트 레이블을 확인해 보죠:

```
class_names[test_labels[0]]
```

'Ankle boot'



10개 클래스에 대한 예측을 모두 그래프로 표현해 보겠습니다:

```
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img, cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})."
               .format(class_names[predicted_label],
                       100*np.max(predictions_array),
                       class_names[true_label]),
               color=color)

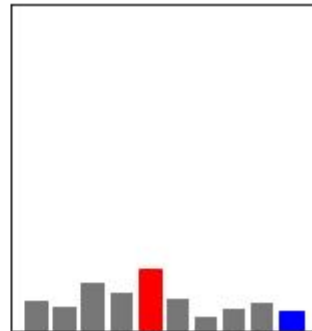
def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)
    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

0번째 원소의 이미지, 예측, 신뢰도 점수 배열을 확인해 보겠습니다.

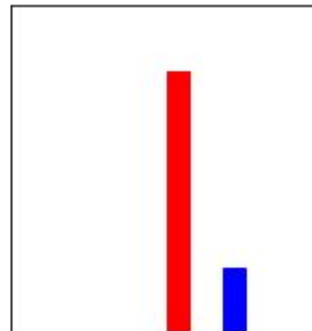
```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.show()
```



Coat 19% (Ankle boot)



Sandal 80% (Sneaker)



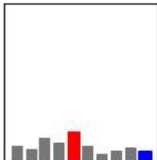
몇 개의 이미지의 예측을 출력해 보죠. 올바르게 예측된 레이블은 파란색이고 잘못 예측된 레이블은 빨강색입니다. 숫자는 예측 레이블의 신뢰도 퍼센트(100점 만점)입니다. 신뢰도 점수가 높을 때도 잘못 예측할 수 있습니다.

```
# 처음 x 개의 테스트 이미지와 예측 레이블, 진짜 레이블을 출력합니다
# 올바른 예측은 파랑색으로 잘못된 예측은 빨강색으로 나타냅니다
```

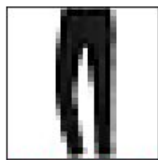
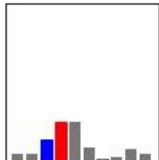
```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
plt.show()
```



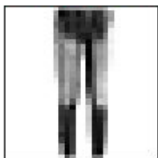
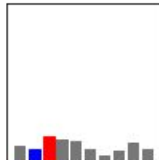
Coat 19% (Ankle boot)



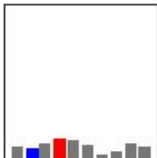
Dress 25% (Pullover)



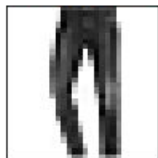
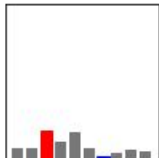
Pullover 16% (Trousers)



Dress 15% (Trousers)



Pullover 20% (Shirt)



Pullover 16% (Trousers)

