

Deep Learning

1~3차시. 머신러닝, 딥러닝 복습

머신러닝, 딥러닝 개념 복습

TensorFlow, Keras

간단한 Classification

Deep learning

Deep Learning

시스템 세팅

- Google Colab을 통한 시스템 세팅



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

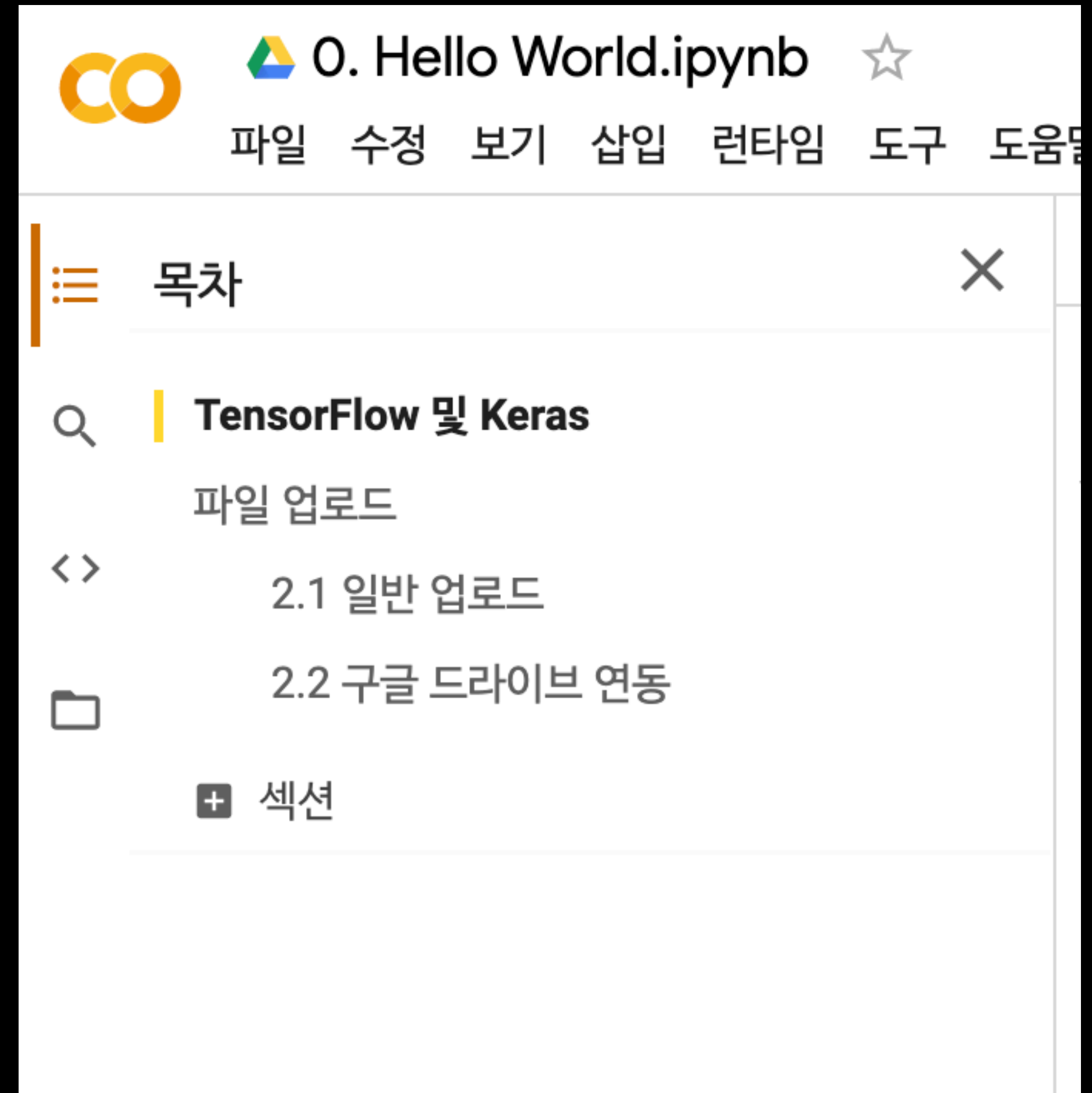
Colab 세팅

- TensorFlow 버전 확인
- Keras 버전 확인



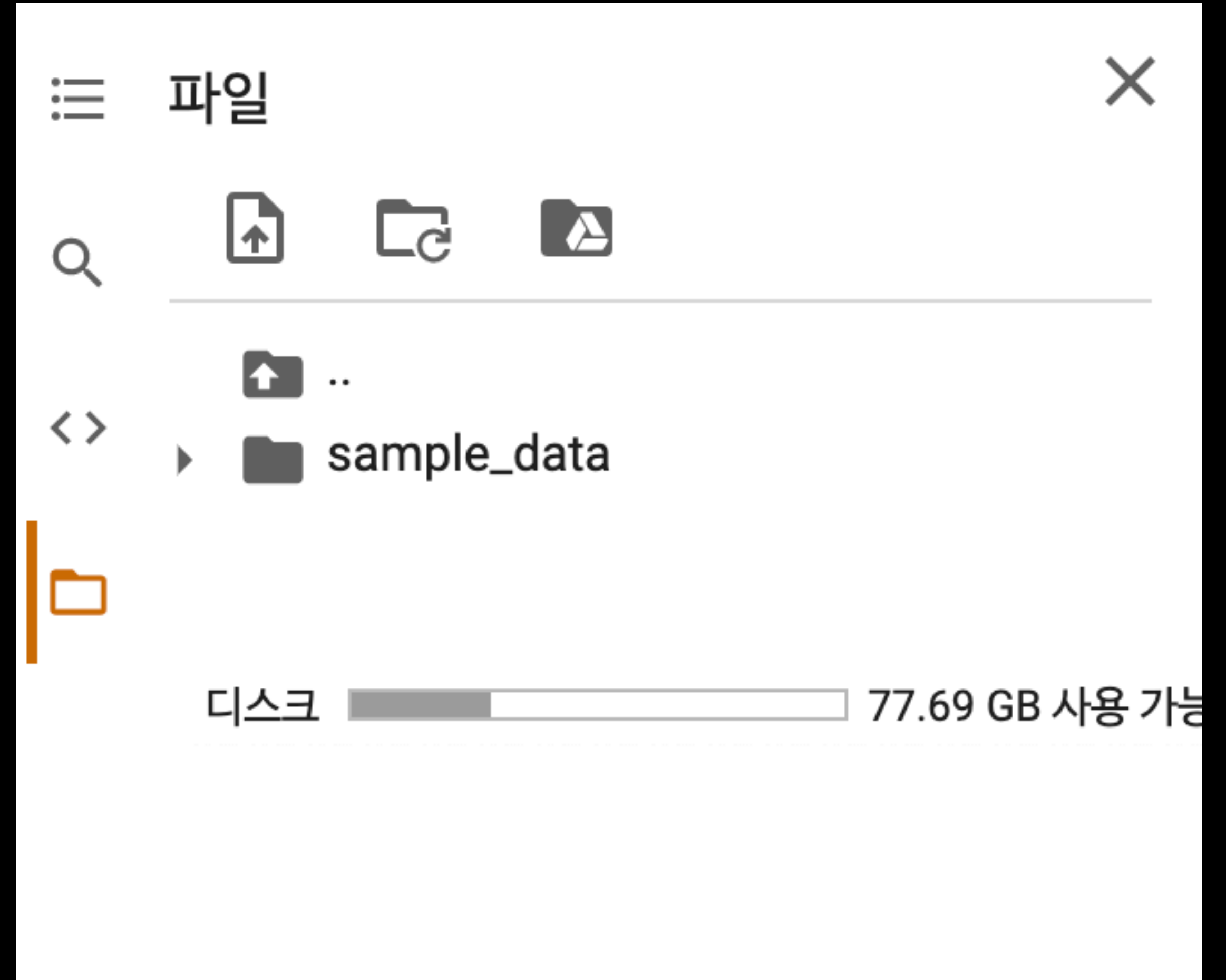
Colab 세팅

- 왼쪽의 최상단 아이콘이 “목차”
 - Markdown 형식처럼, #으로 구분되어 표시



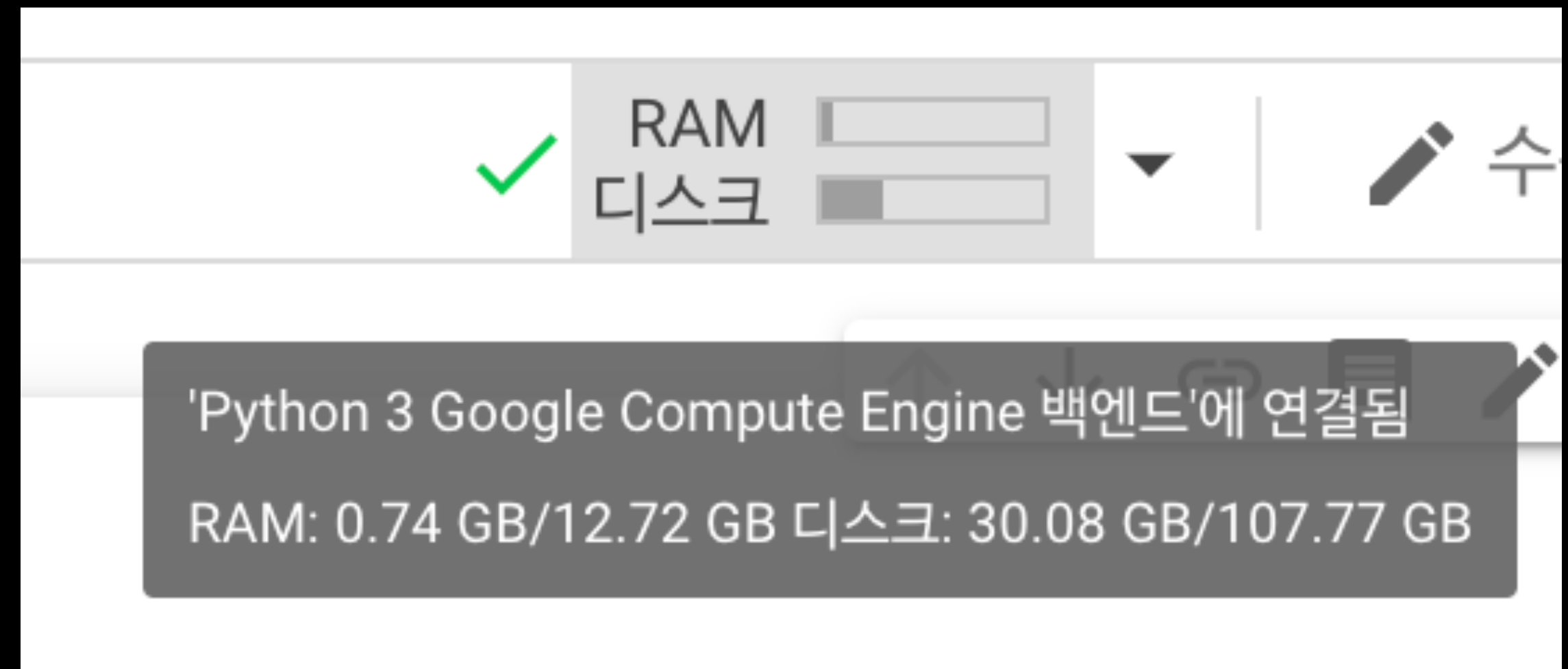
Colab 세팅

- 왼쪽의 최하단 아이콘이 “파일”
 - 연결된 세션의 파일을 보여주며,
 - 하단에는 할당된 디스크의 사용 공간



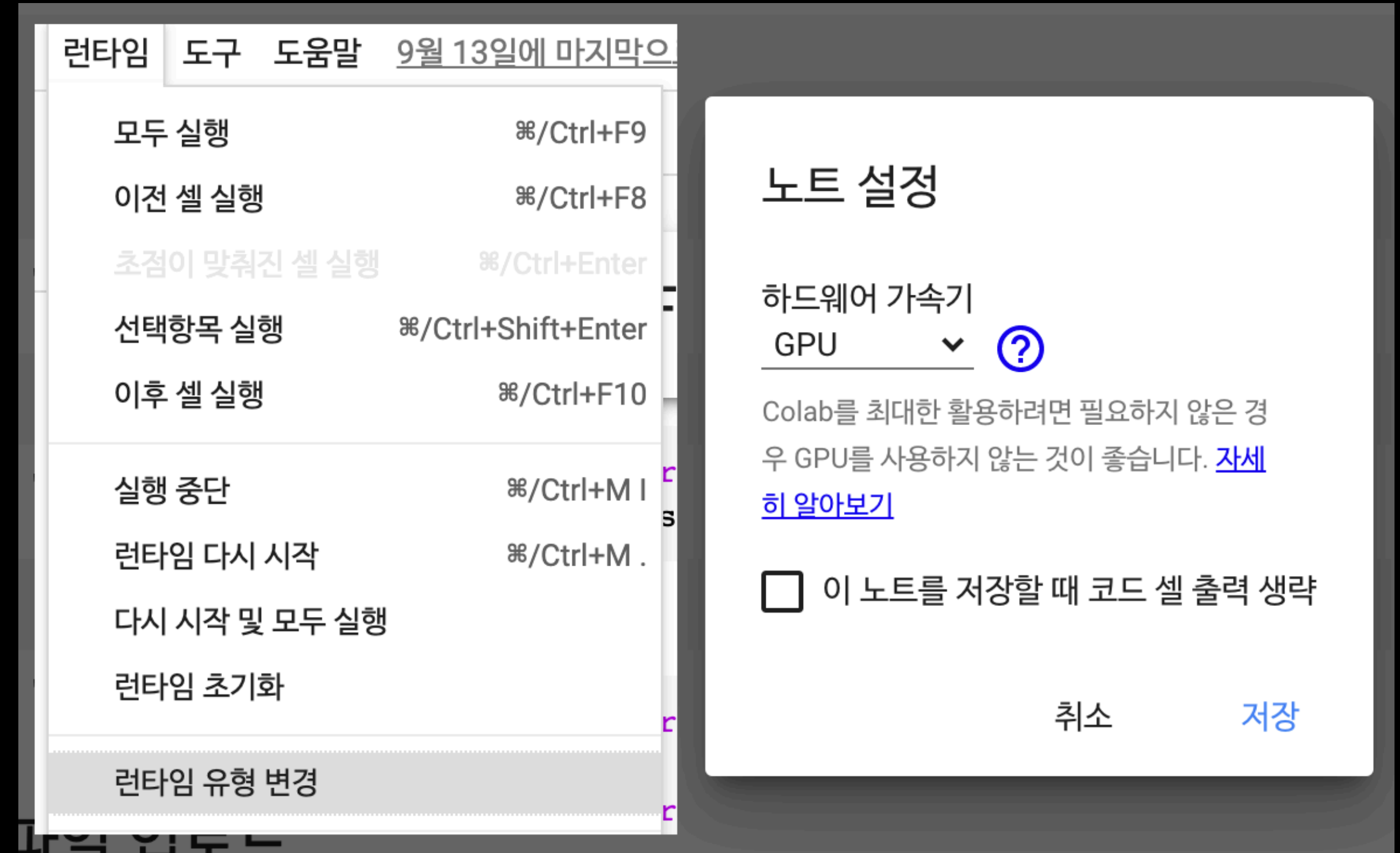
Colab 세팅

- 오른쪽에는 할당된 컴퓨터의 정보
 - 램, 디스크 정보



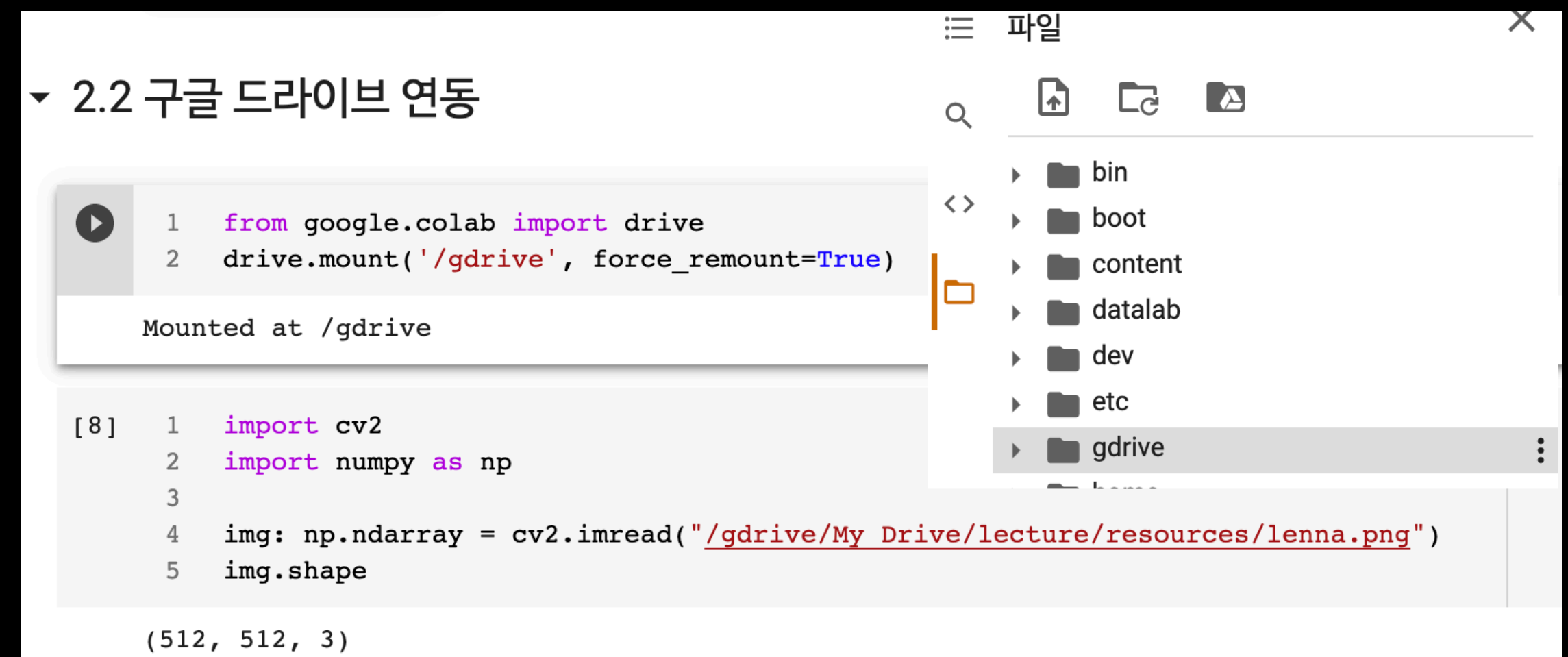
Colab 세팅

- 기본적으로 CPU에서 실행되며,
 - 런타임 - 런타임 유형 변경 메뉴에서
- GPU, TPU, None 선택 가능



Colab 세팅

- 구글 드라이브 마운트 확인
 - /gdrive로 마운트 됨을 확인 가능



The screenshot displays the Google Colab environment. At the top, a tab titled '2.2 구글 드라이브 연동' (2.2 Google Drive Connection) is active. Below the tab, a code cell is shown with the following Python code:

```
1 from google.colab import drive
2 drive.mount('/gdrive', force_remount=True)
```

Below the code, a message indicates 'Mounted at /gdrive'. To the right of the code cell, a file explorer sidebar is visible, showing a directory structure with folders like 'bin', 'boot', 'content', 'datalab', 'dev', 'etc', and 'gdrive'. The 'gdrive' folder is highlighted. Below the code cell, another code cell is shown with the following Python code:

```
[8] 1 import cv2
    2 import numpy as np
    3
    4 img: np.ndarray = cv2.imread("/gdrive/My Drive/lecture/resources/lenna.png")
    5 img.shape
```

The output of the second code cell is displayed as '(512, 512, 3)'.

Colab

세션



구글 드라이브 마운트

세션이 계속 유지되는 것이 아닌, 활동이 없으면 끊기는 등 연결의 불안정성.

트레이닝 결과를 중간에 저장해, 구글 드라이브에 올리고, 다시 계속 트레이닝



유료 결제 (<https://colab.research.google.com/signup>)

9.99 달러 / 달

세션은 여전히 끊길 수 있다.

런타임 연결이 끊어짐

활동이 없어 런타임 연결이 종료되었습니다. [자세히 알아보기](#)

닫기

다시 연결

간단한 예제

0. Hello World.ipynb

딥러닝 시스템



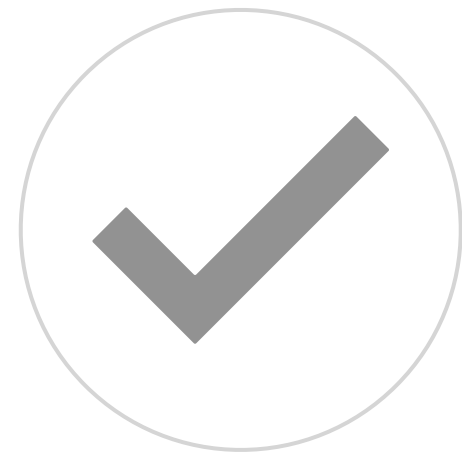
Google Colab



딥러닝 용 컴퓨터 마련

RTX 3080, 3090, ...

NVLink는 3090만 지원하니 유의

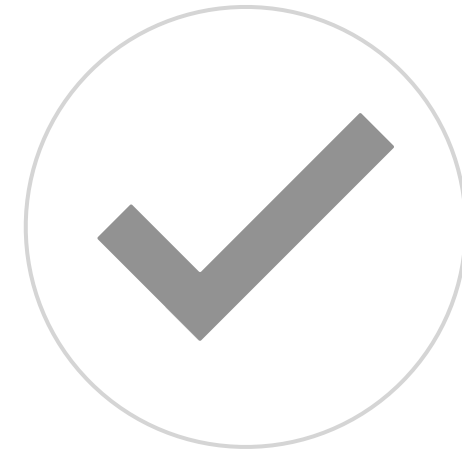


클라우드 서비스 사용

Amazon, MS, Google Cloud TPU

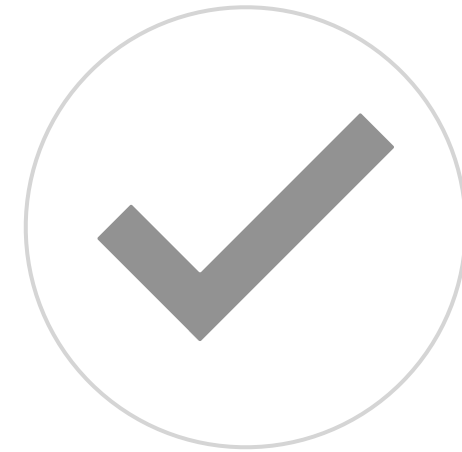
딥러닝 로컬 시스템

딥러닝 용 컴퓨터를 따로 마련하는 경우



Native 환경

리눅스를 기준으로 말하자면,
Virtualenv 등을 통해 여러 가상 환경을 만들어서 실행



Docker 활용

Docker나 쿠버네티스 활용 가능

지원이 종료된 예전의 프로젝트를 돌려볼 수 있음.



Keras는 TF 2.0부터 TF에서 공식 지원

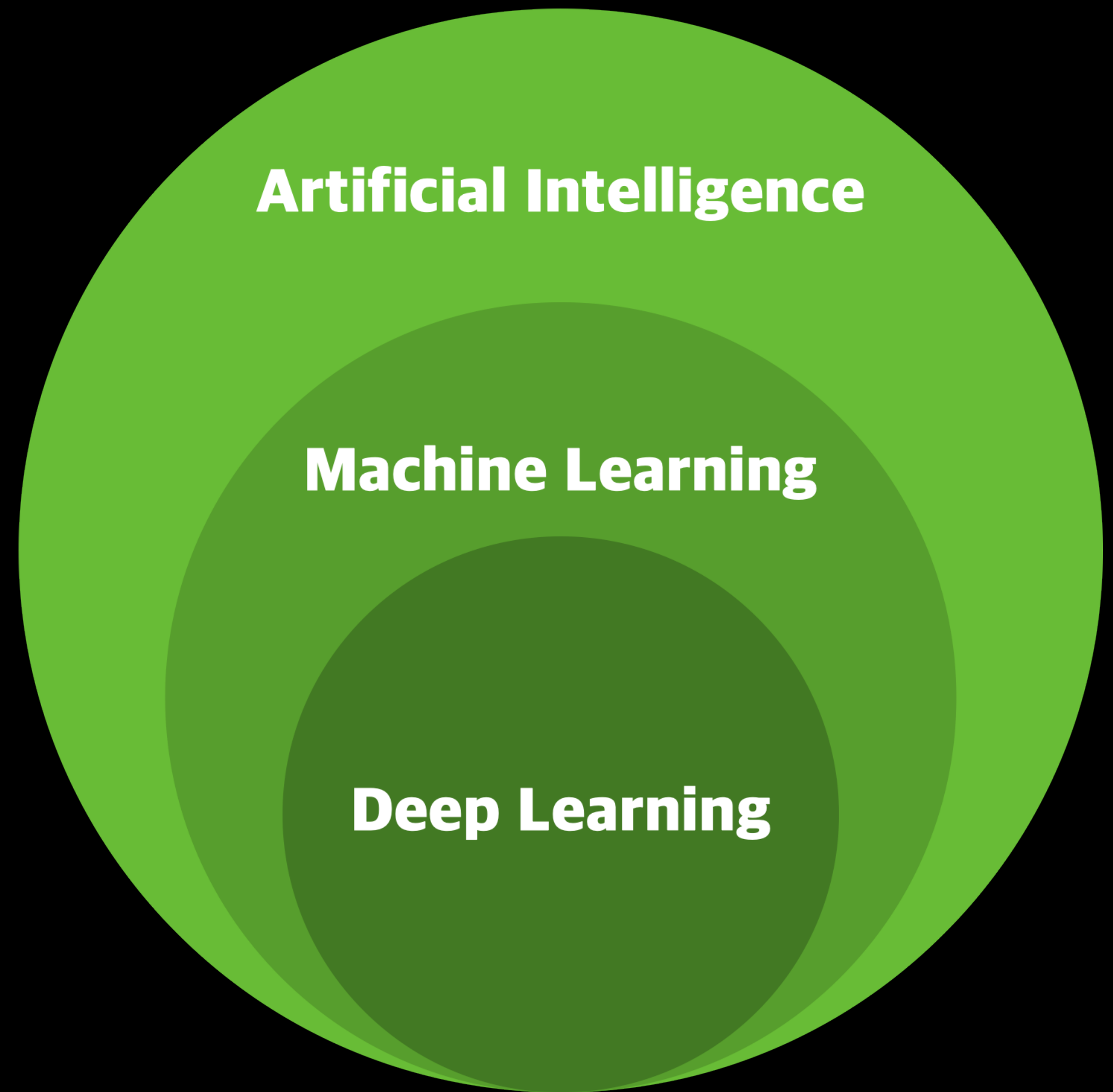
```
import tensorflow  
from tensorflow import keras
```


기초 이론



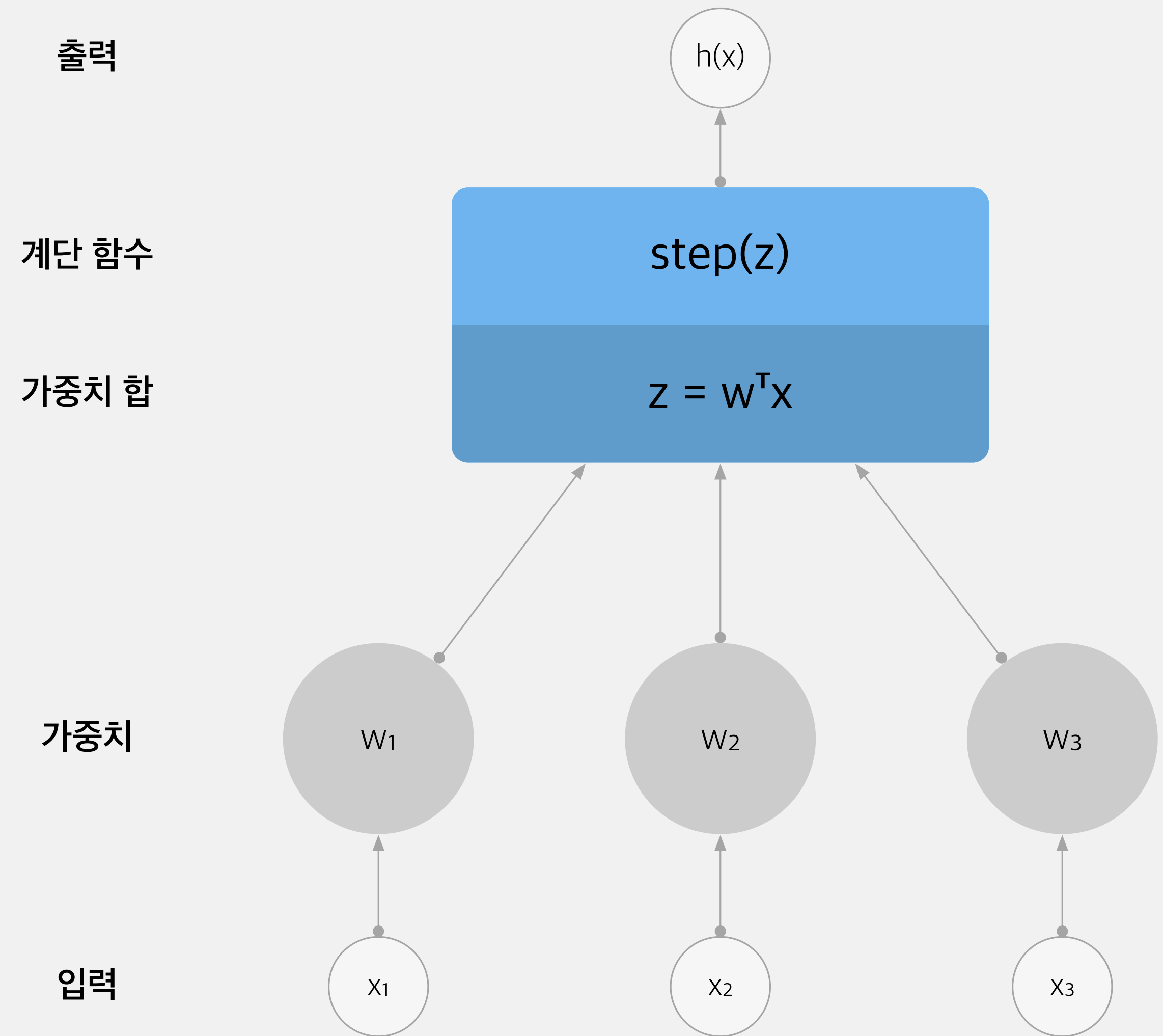
딥러닝

- AI, ML, Deep Learning의 관계
- 딥러닝은 새로운 분야가 아닌,
머신러닝의 한 종류



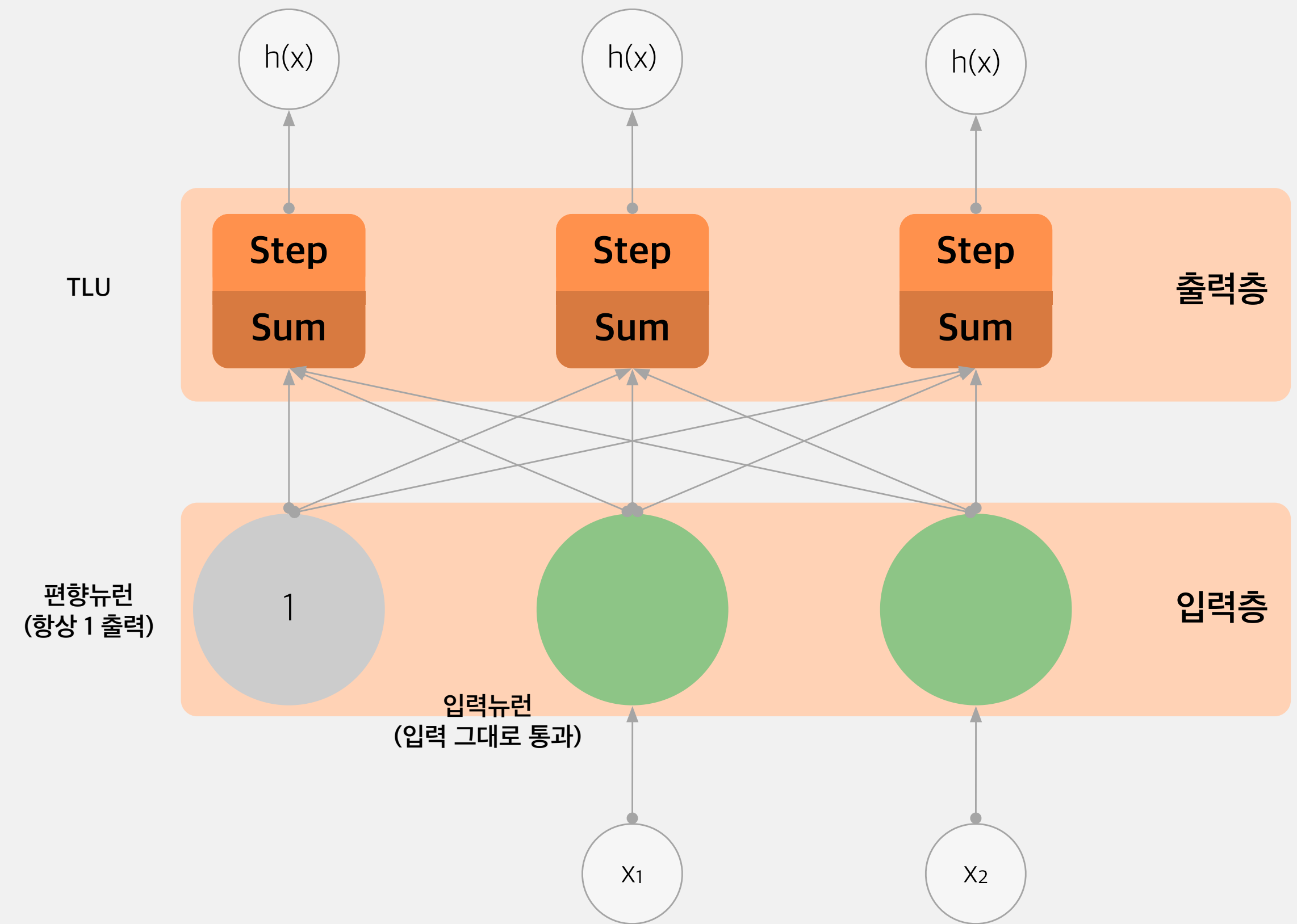
퍼셉트론

- 가장 간단한 인공신경망 구조
- 입력 x **가중치**의 합
- 이후, 계단함수 통과
 - 0보다 작으면, 0
 - 0보다 같거나 크면, 1
- $z = w_1x_1 + w_2x_2 + \dots + w_nx_n = \mathbf{x}^T \mathbf{w}$
- $h_w(\mathbf{x}) = \text{step}(z)$



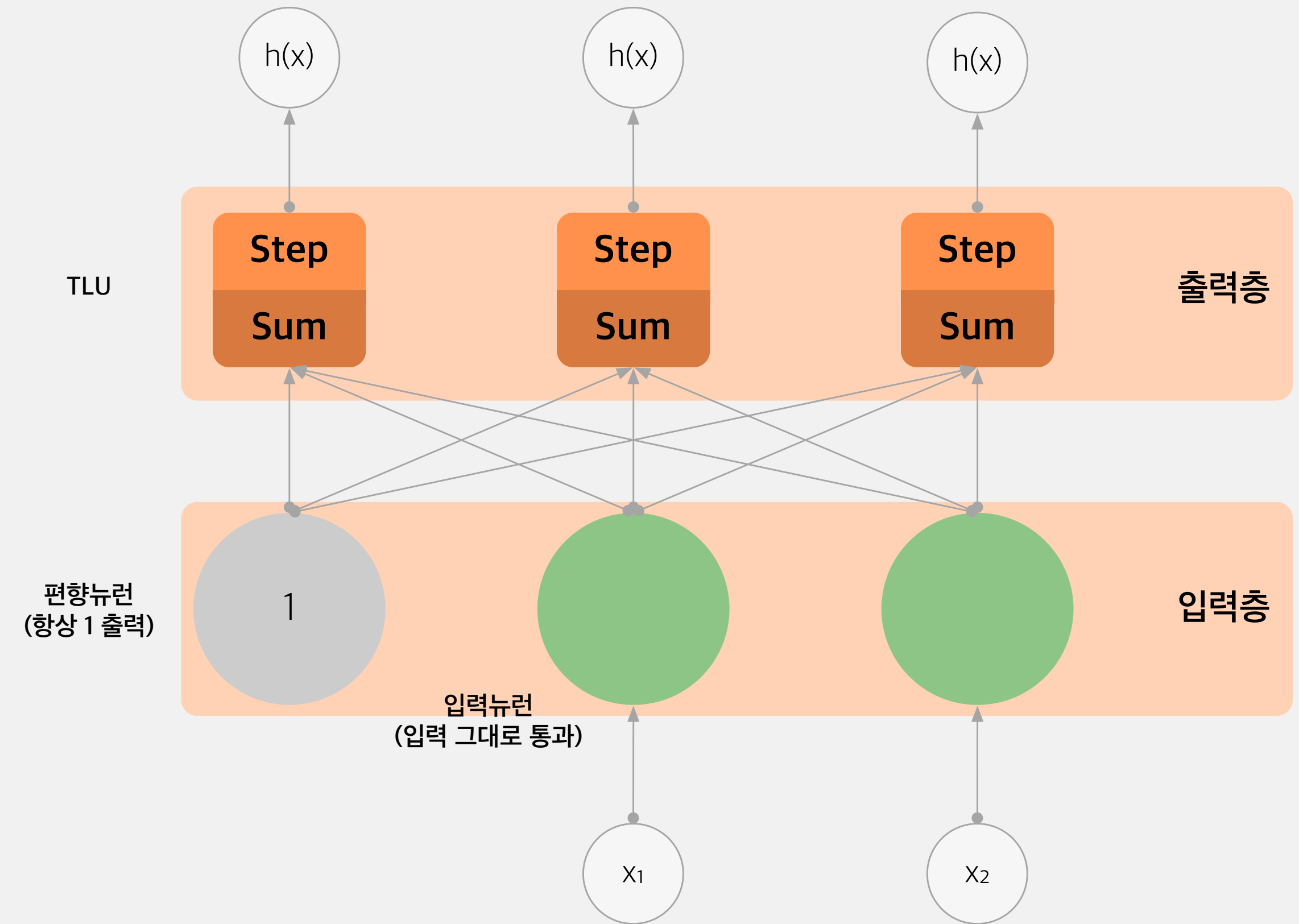
퍼셉트론

- 입력 뉴런 2개 (X)
- 편향 뉴런 1개 (b)
- 출력 뉴런 3개 (h)
- $h_{W,b}(X) = \phi(XW + b)$



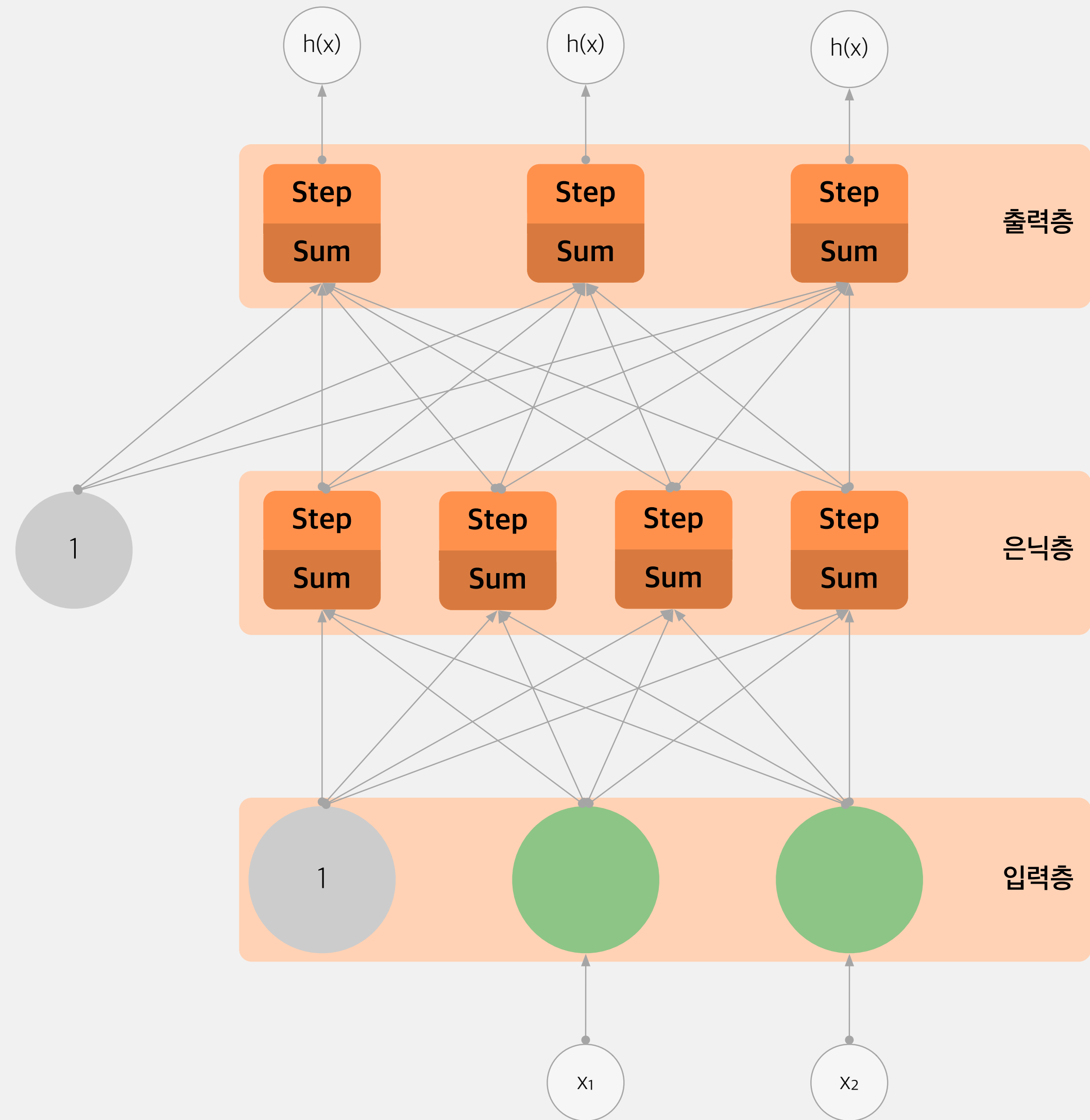
퍼셉트론

- $w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$
 - $w_{i,j}$ - i번째 입력 뉴런, j번째 출력 뉴런 사이의 가중치
 - x_i - 현재 훈련 샘플의 i번째 뉴런의 입력값
 - y_j - 현재 훈련 샘플의 j번째 뉴런의 타깃값
 - \hat{y}_j - 현재 훈련 샘플의 j번째 뉴런의 출력값
 - η - 학습률



다층 퍼셉트론

- 입력 층 하나 - 하위 층(lower layer)
- 은닉 층 하나 이상
- 출력 층 하나 - 상위 층(upper layer)
- 여러 개의 은닉층 - 심층 신경망 (Deep Neural Network, DNN)



경사 하강법

- 한 번에 (예를 들어, 32개 샘플이 포함된) 하나의 미니배치씩 진행하여, 전체 훈련 세트를 처리
 - 이 과정을 여러 번 반복. 각 반복을 **에포크(epoch)**라 한다
- 처음 은닉 층에서, 뉴런의 출력을 계산하고, 다음 층으로 전달
- 마지막 출력층의 출력을 계산할 때까지 계속
 - 이것이 **정방향 계산(forward pass)**
 - 역방향 계산을 위해 중간 계산값을 모두 저장 (네트워크가 트레이닝에 많은 메모리와 시간이 걸리는 이유)
- 알고리즘이 네트워크의 **출력 오차(loss)**를 측정

경사 하강법

- 각 출력 연결이 이 오차에 기여하는 정도를 계산
 - 연쇄 법칙(chain rule)을 적용
- 이전 층의 연결 가중치가 이 오차에 기여하는 정도를 측정. 입력층에 도달할 때까지.
 - 계속적인 연쇄 법칙(chain rule)의 적용
 - 이것에서 역전파라는 이름이 유래
- 오차 그래디언트를 거꾸로 전파함으로써 효율적으로 네트워크에 있는 모든 연결 가중치에 대한 오차 그래디언트 측정
- 경사 하강법을 수행하여 방금 계산한 오차 그래디언트를 사용해 네트워크에 있는 모든 연결 가중치를 수정

경사 하강법

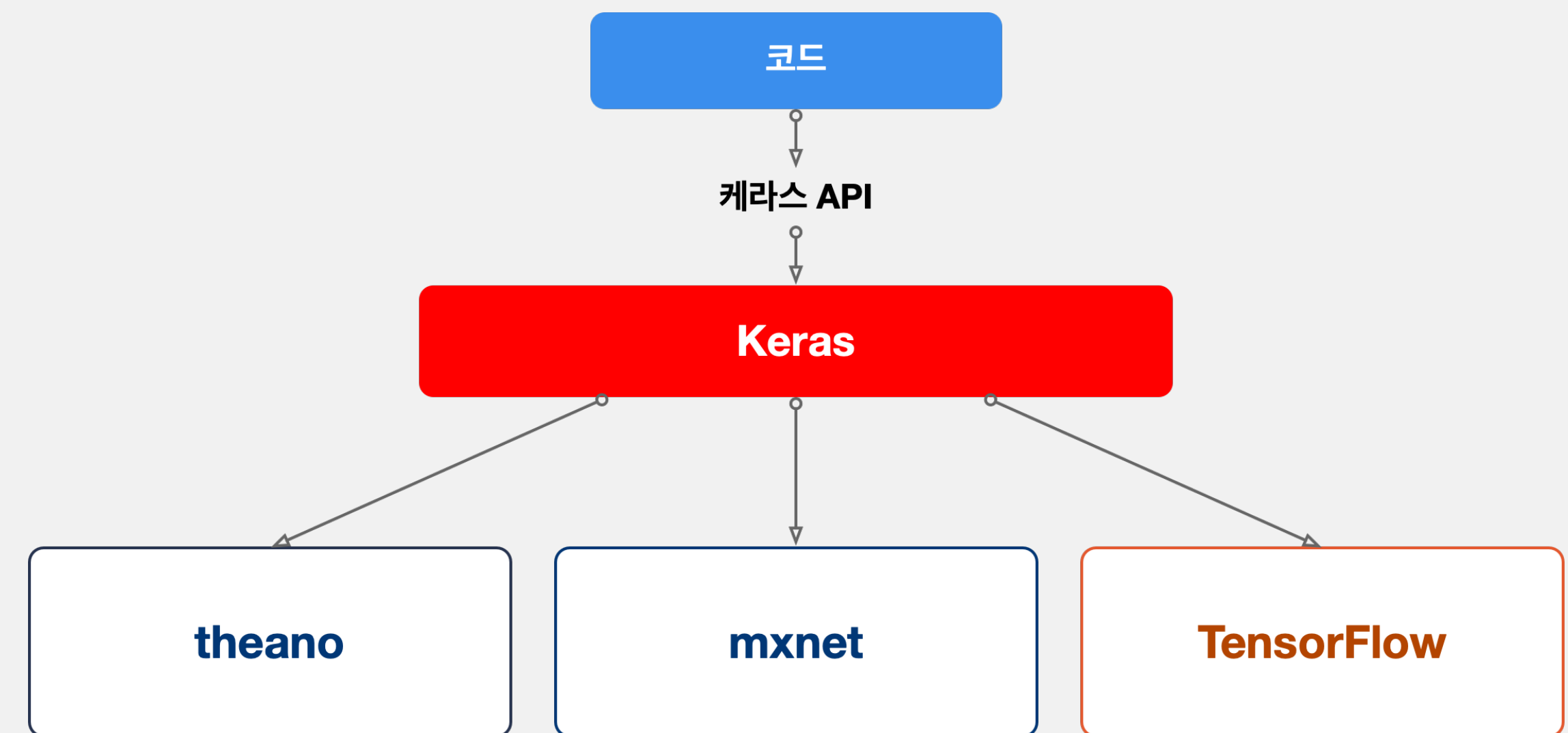
요약

- 각 훈련 샘플에 대해 역전파 알고리즘이 먼저 예측을 만들고 (**정방향 계산**)
- 역방향으로 각 층을 거치면서 각 연결이 오차에 기여한 정도를 측정 (**역방향 계산**)
- 이 오차가 감소하도록 가중치를 조정 (**경사 하강법**)

케라스, 텐서플로

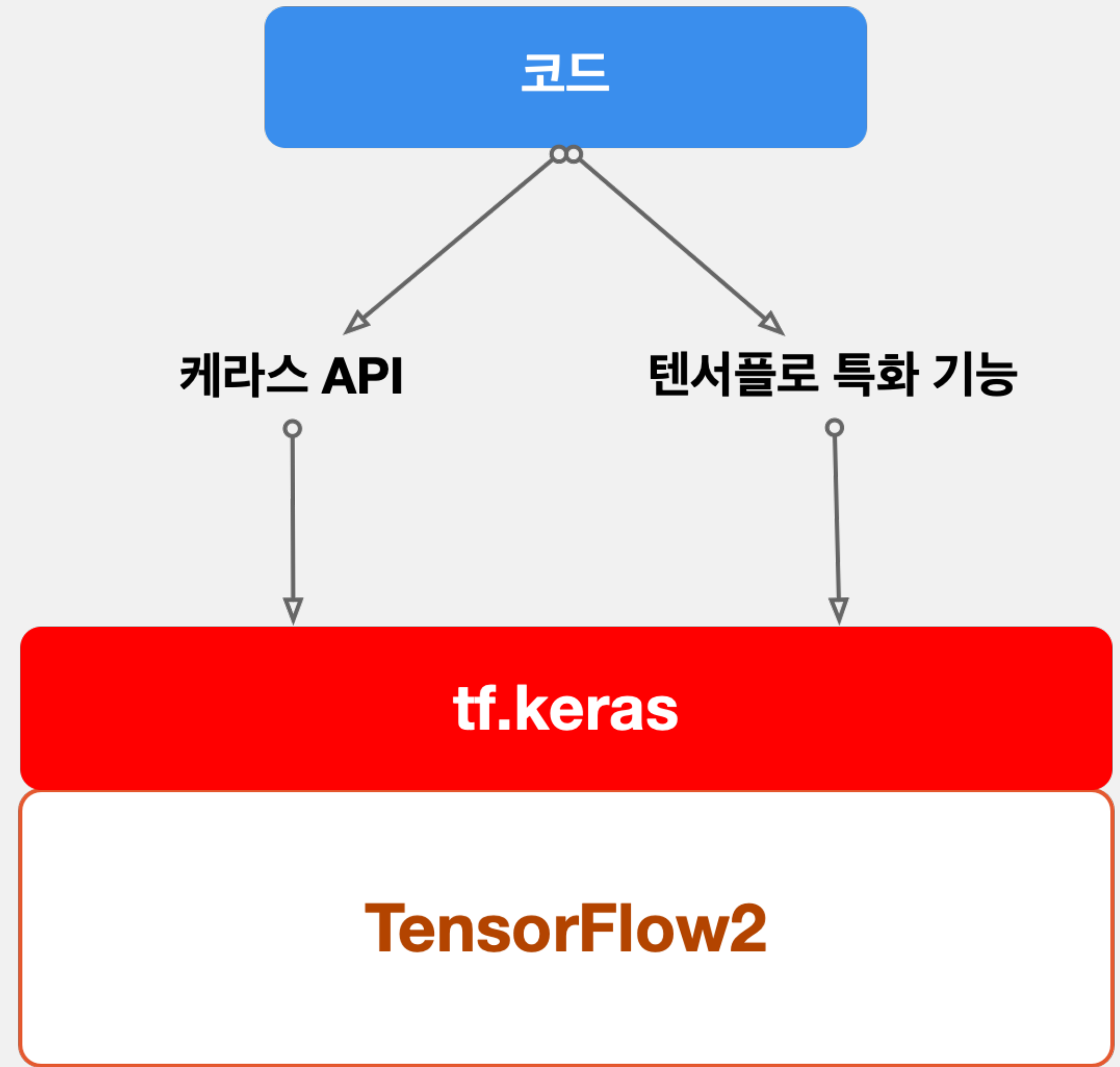
케라스, 텐서플로

- 딥러닝의 여러 라이브러리가 존재
 - MS(CNTK), theano, 텐서플로
- 케라스가 API 형식으로 묶어서 제공
- 참고로 Keras는 pytorch 백엔드는 제공하지 않음



케라스, 텐서플로

- 텐서플로2 부터는 케라스를 공식 API로 채택
- 대부분의 경우, 텐서플로 코드로 깊이 접근하지 않고, 케라스 만으로 딥러닝 가능

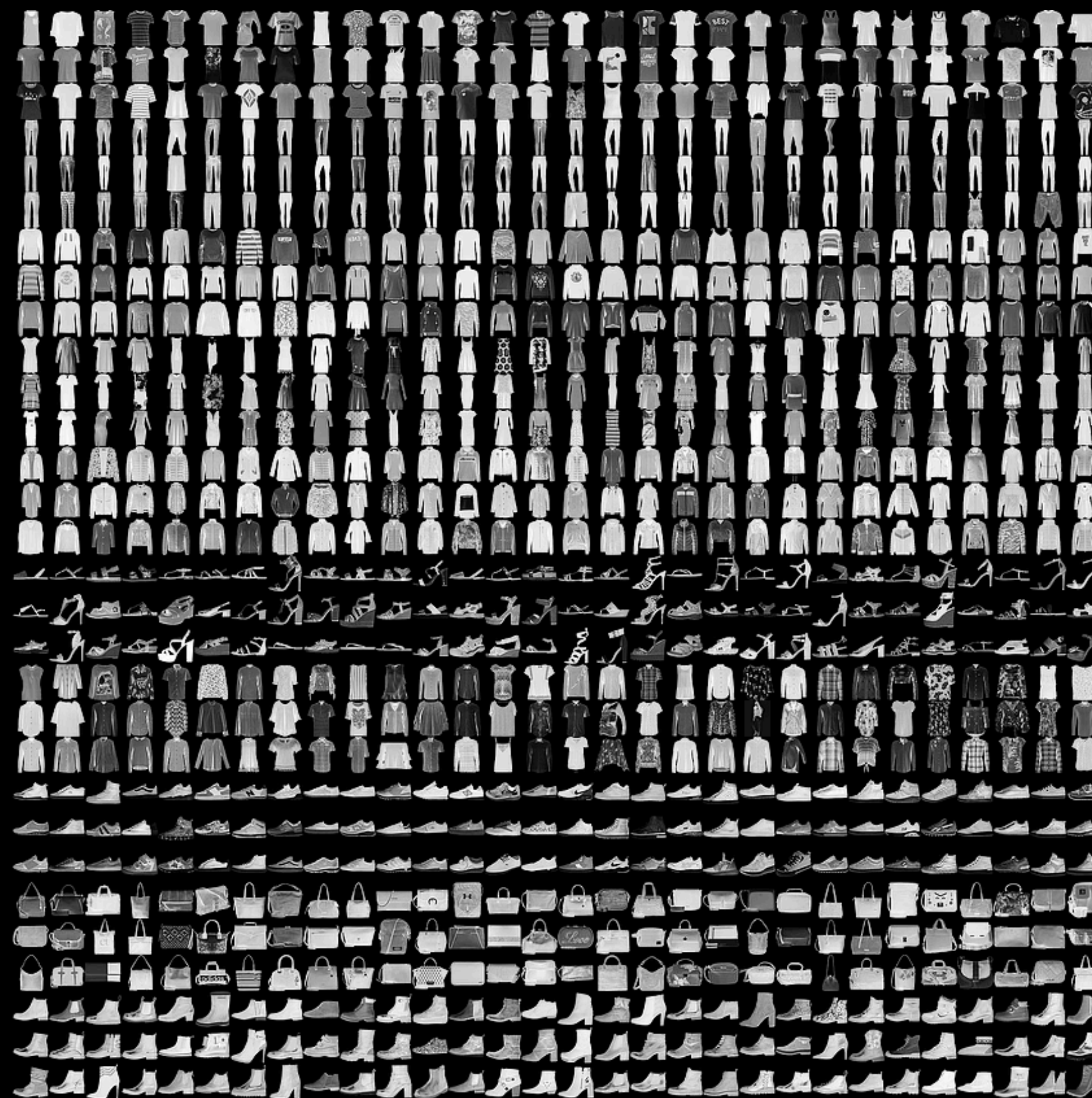


예제. 이미지 분류기



간단한 분류 네트워크

- Fashion MNIST를 활용한 이미지 분류
- <https://github.com/zalandoresearch/fashion-mnist>



Fashion MNIST

- 28x28 크기의 그레이스케일 이미지
 - 0~255

- 라벨은 단일 숫자

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

```
from tensorflow import keras

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
    fashion_mnist.load_data()

print("트레이닝 이미지 shape: {}, dtype: {}".format(train_images.shape,
    train_images.dtype))
print("트레이닝 라벨 shape: {}, dtype: {}".format(train_labels.shape,
    train_labels.dtype))

# 트레이닝 이미지 shape: (60000, 28, 28), dtype: uint8
# 트레이닝 라벨 shape: (60000,), dtype: uint8

print("테스트 이미지 shape: {}, dtype: {}".format(test_images.shape,
    test_images.dtype))
print("테스트 라벨 shape: {}, dtype: {}".format(test_labels.shape,
    test_labels.dtype))

# 테스트 이미지 shape: (10000, 28, 28), dtype: uint8
# 테스트 라벨 shape: (10000,), dtype: uint8
```

Fashion MNIST

- 트레이닝 데이터 60000개
- 테스트 데이터 10000개

```
from tensorflow import keras

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
    fashion_mnist.load_data()

print("트레이닝 이미지 shape: {}, dtype: {}".format(train_images.shape,
    train_images.dtype))
print("트레이닝 라벨 shape: {}, dtype: {}".format(train_labels.shape,
    train_labels.dtype))

# 트레이닝 이미지 shape: (60000, 28, 28), dtype: uint8
# 트레이닝 라벨 shape: (60000,), dtype: uint8

print("테스트 이미지 shape: {}, dtype: {}".format(test_images.shape,
    test_images.dtype))
print("테스트 라벨 shape: {}, dtype: {}".format(test_labels.shape,
    test_labels.dtype))

# 테스트 이미지 shape: (10000, 28, 28), dtype: uint8
# 테스트 라벨 shape: (10000,), dtype: uint8
```


실습 Session

1. Simple Classification.ipynb

Fashion MNIST와 함께하는 Dense, Softmax 레이어를 사용한 이미지 분류기

실습 Session

1. Simple Classification.ipynb

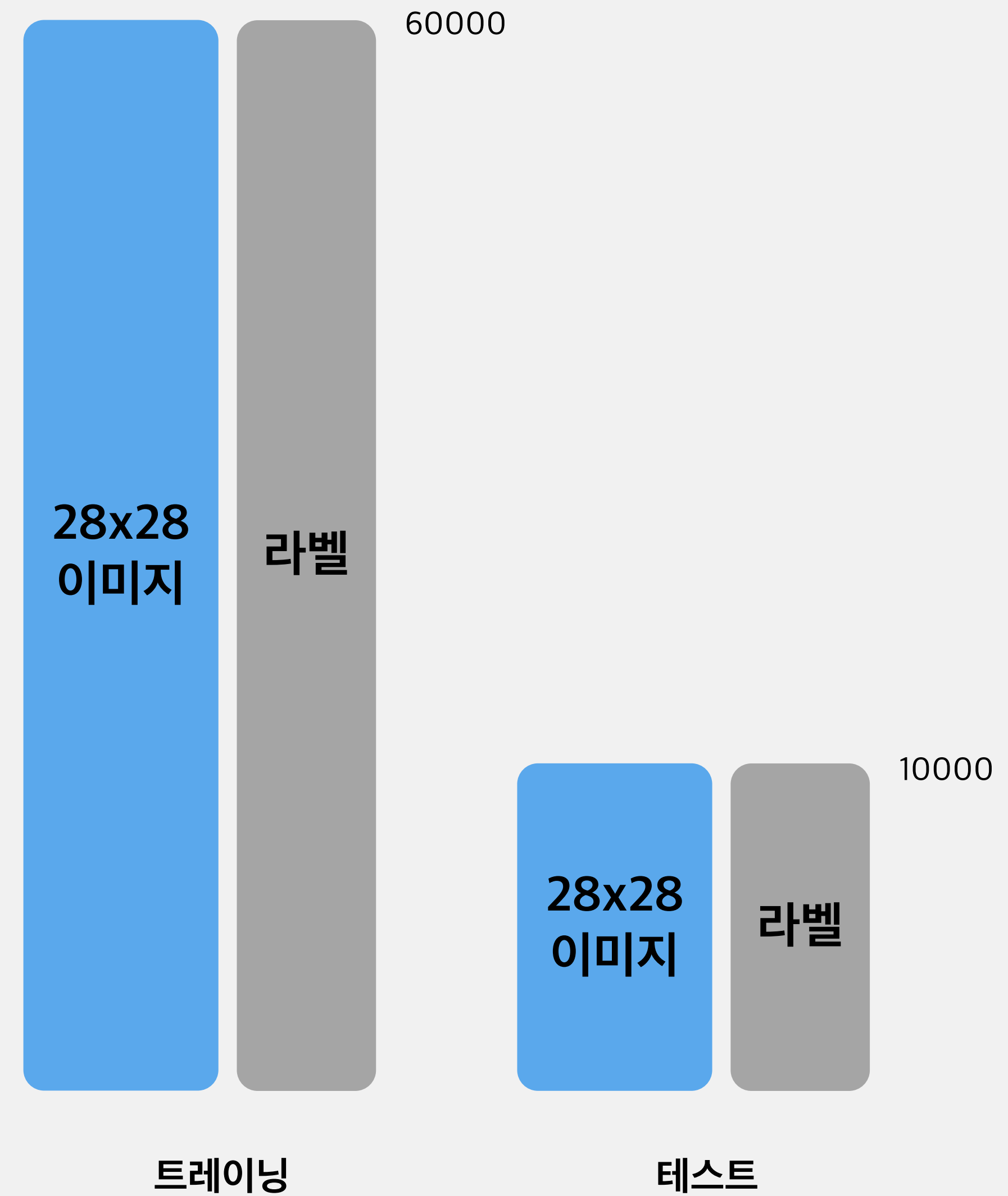
모델 저장 및 불러오기

예제를 통한 복습



데이터 분리

- 트레이닝 데이터
- 테스트 데이터
 - 사용자에게 비공개되는 경우도 있음
- 채점용으로 사용



데이터 분리

- 트레이닝 데이터

- 검증 데이터

- 순차로 데이터 분리

- 랜덤으로 데이터 분리

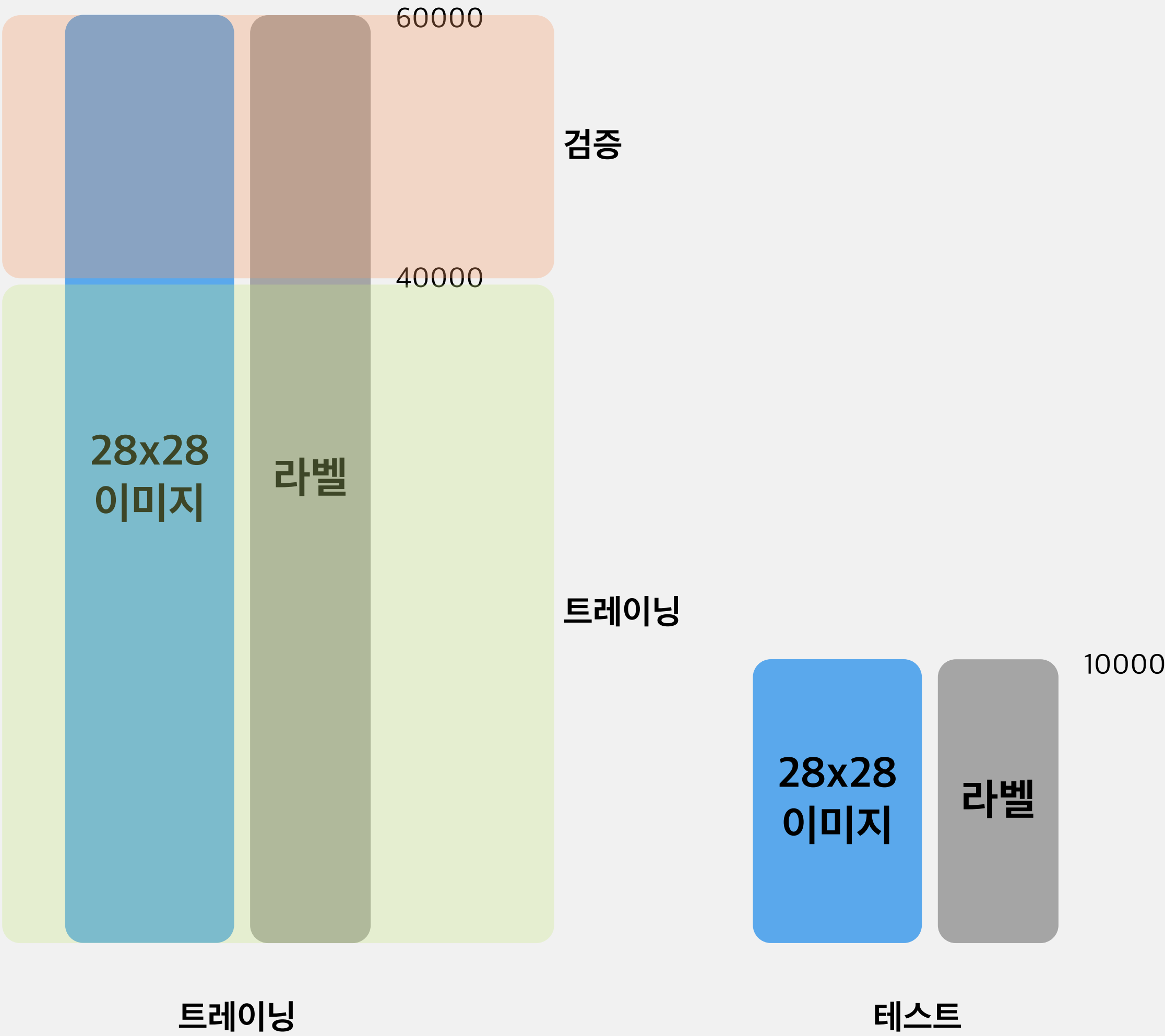
- 조건에 따라 분리

- 가령, 이전의 classification 같은 경우에는, 티셔츠, 바지, 코트 등이 검증 데이터와 트레이닝 데이터에 골고루 분포되어 있어야 한다.

- 트레이닝 데이터에 0~5번 까지가 많이 있고, 검증 데이터에 6~9 번 까지가 많다면, 트레이닝이 제대로 되지 않을 것이다.

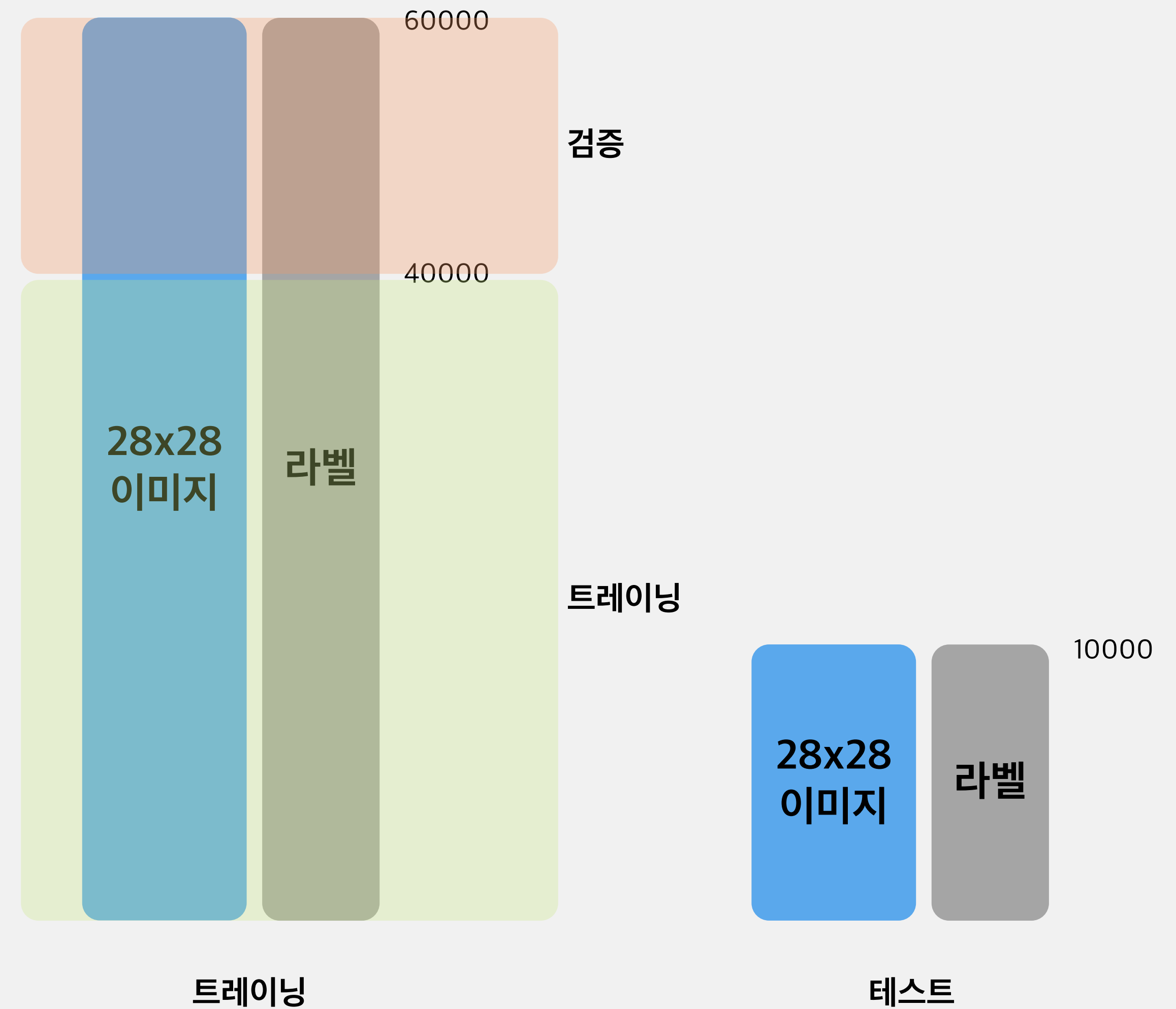
- 테스트 데이터

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



데이터 분리

- 왜 검증 데이터가 필요한가?
 - **과대적합** 방지
 - 검증 데이터로 훈련의 중지 지점을 지정한다.



네트워크 구조

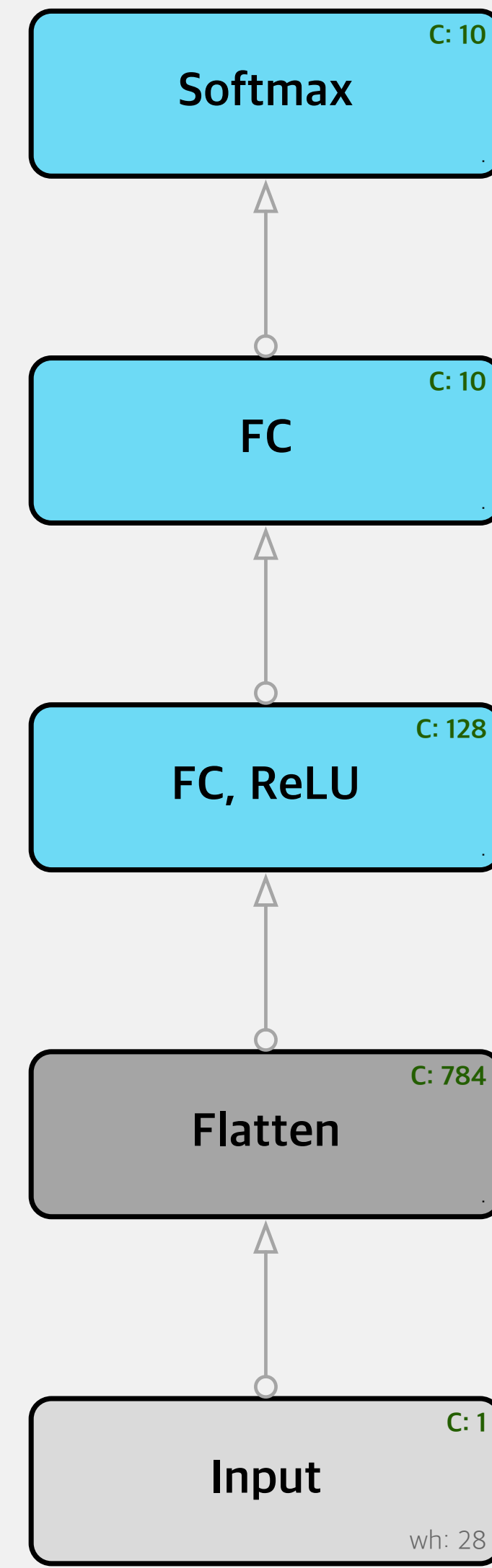
- 마지막 Softmax 후, 값은 10개

```
def fashion_classification_model():
    input = keras.layers.Input((28, 28))
    flatten = keras.layers.Flatten()(input)
    dense_1 = keras.layers.Dense(128, activation=keras.activations.relu)(flatten)
    dense_2 = keras.layers.Dense(10)(dense_1)

    return keras.models.Model(inputs=[input], outputs=[dense_2])

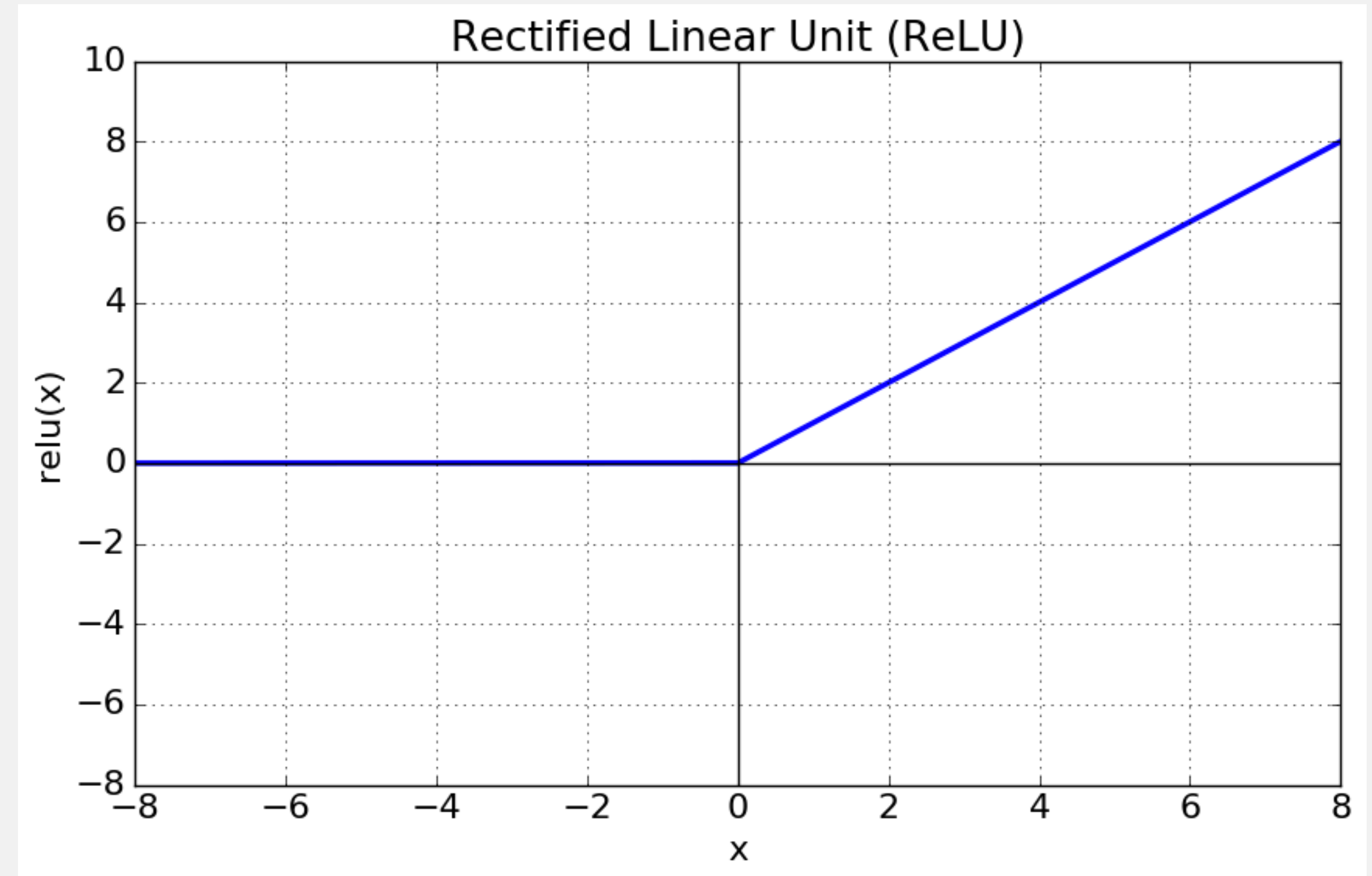
def fashion_classification_model_2(pre_trained_fashion_classification_model):
    pre_classification = pre_trained_fashion_classification_model(
        pre_trained_fashion_classification_model.inputs)
    softmax_1 = keras.layers.Softmax()(pre_classification)

    return keras.models.Model(
        inputs=pre_trained_fashion_classification_model.inputs,
        outputs=[softmax_1])
```



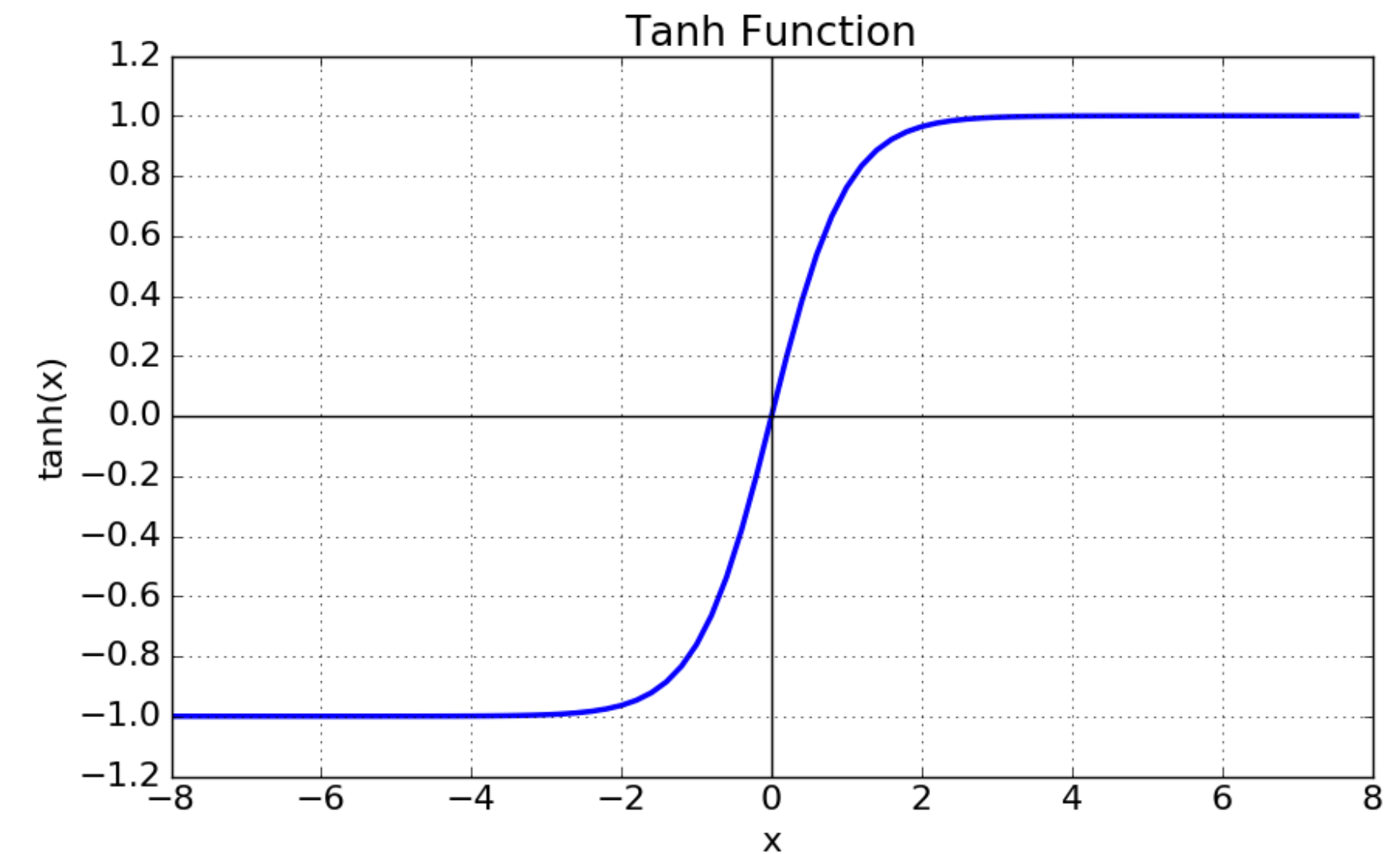
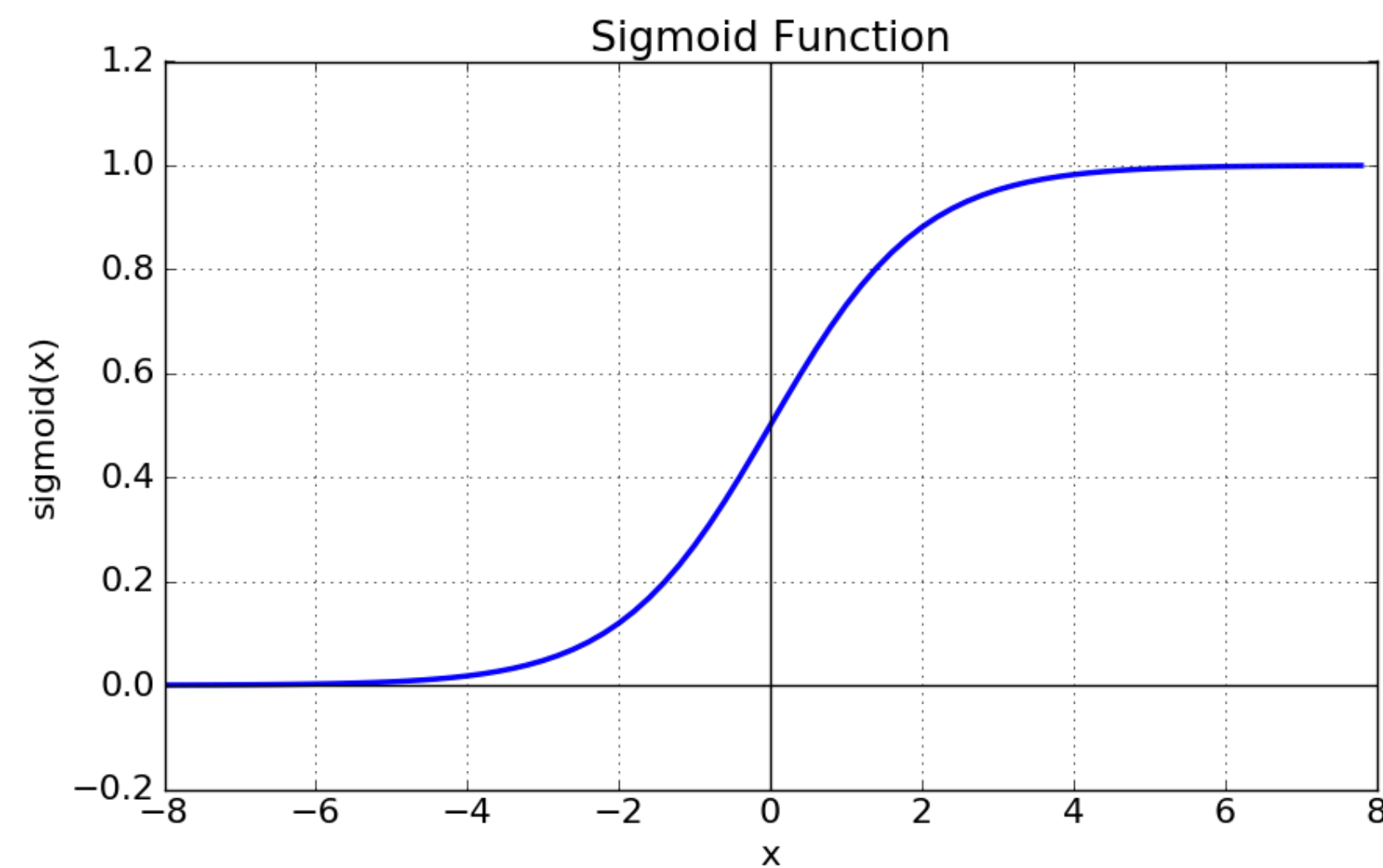
네트워크 구조

- ReLU
- 활성화 함수
 - 어떤 처리 이후, 값을 제한시킨다.
 - 선형 함수에 비선형적인 요소를 도입.



네트워크 구조

활성화 함수



- 참조 1 : <https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e>
- 참조 2 : <https://reniew.github.io/12/>

네트워크 구조

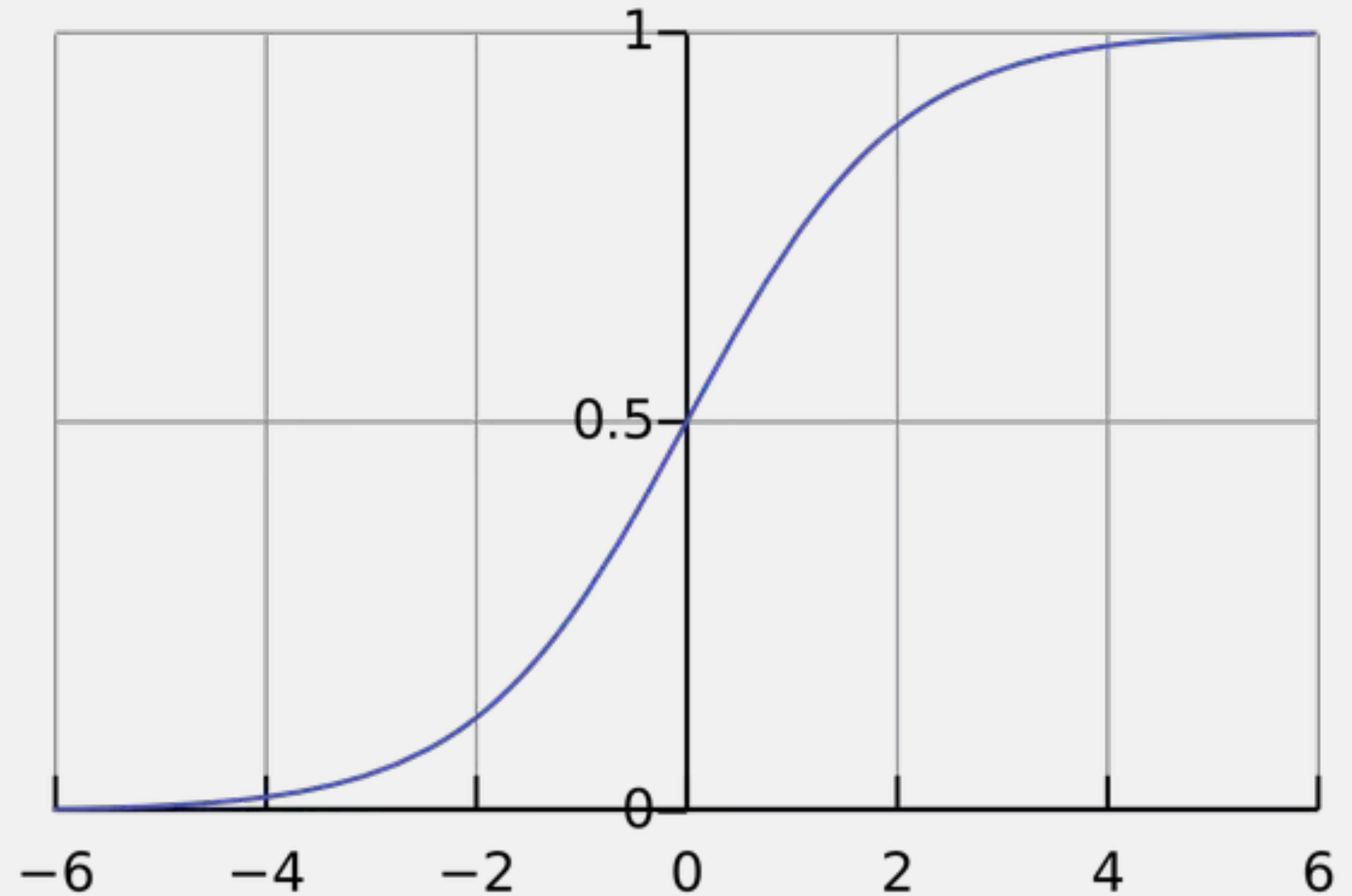
활성화 함수

- 선형 변환을 여러 개 연결해도 얻을 수 있는 것은 선형 변환뿐이다.
 - 비선형성을 추가하지 않으면, 층을 많이 쌓아도 하나의 층과 동일
 - 비선형 활성화 함수가 있는 충분히 큰 심층 신경망은 이론적으로 어떤 연속 함수도 근사할 수 있다.
- ReLU 함수
 - 잘 작동하고 계산속도가 빠르다.

네트워크 구조

- Softmax
 - 전체의 합계는 1
 - $-\infty \sim \infty$ 의 범위를 0~1의 범위로 줄인다.
 - 0 근처의 값은 영향이 크고, 절댓값이 커질수록 영향이 기하급수적으로 작아진다.
 - 다른 값을 보고 결정한다.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



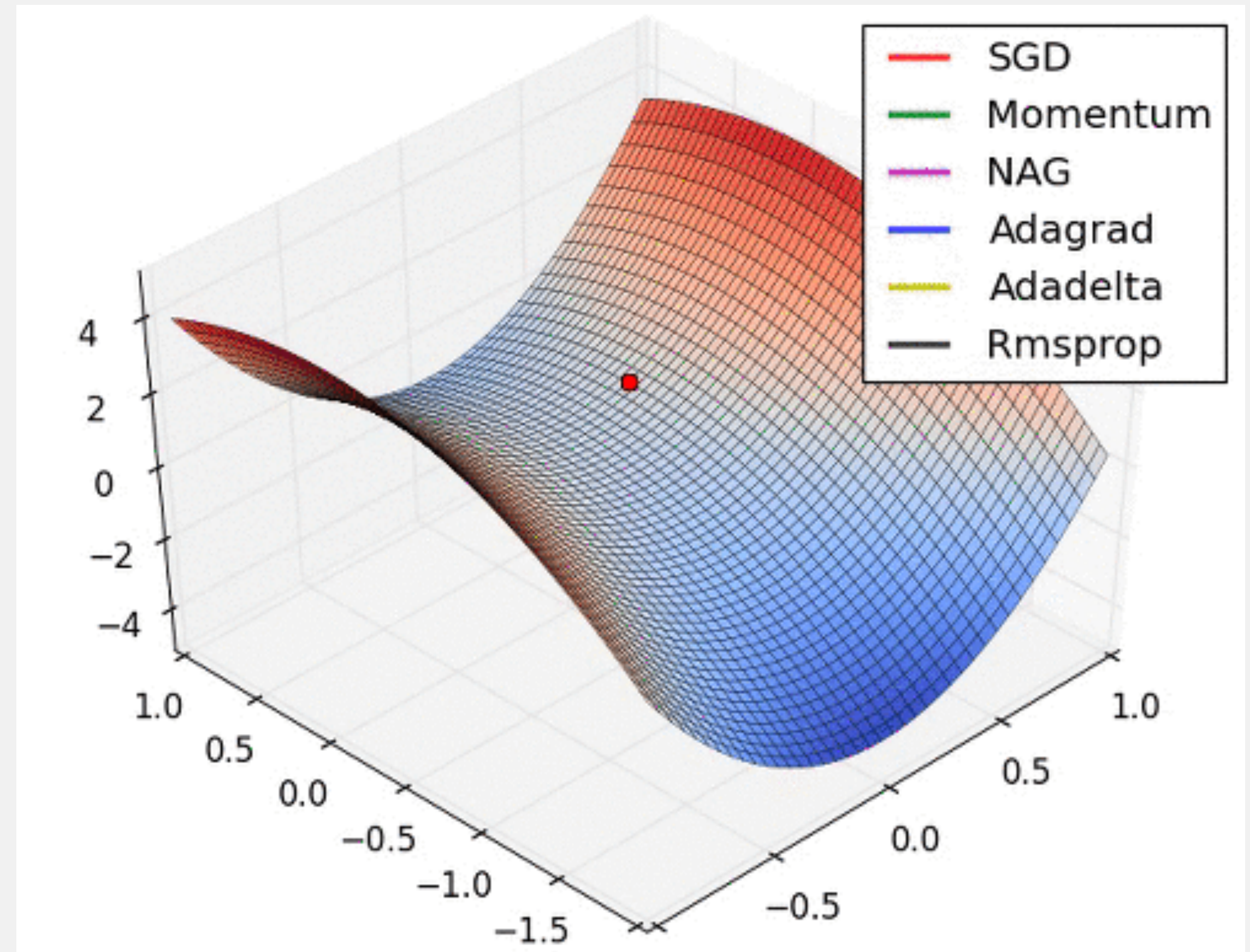
네트워크 구조

초기화

- Dense의 경우, 기본 초기화는 glorot_normal
- he_normal 등 다른 값들이 많이 존재한다.
- 예를 들어, 모든 가중치와 편향을 0으로 초기화하면, 층의 모든 뉴런이 완전히 같아진다.
 - 역전파도 뉴런을 동일하게 바꾸어 모든 뉴런이 똑같아진 채로 남는다.
 - 층에 뉴런이 수백 개 있더라도, 모델은 마치 뉴런이 하나인 것 처럼 작동할 것이다.
- 랜덤한 초기화로, 대칭성을 깨뜨려, 역전파가 전체 뉴런을 다양하게 훈련하도록 한다.

네트워크 구조

- 학습속도를 빠르게 하거나, 안정적으로 하는 전략
- 최상의 옵티마이저가 존재한다기보다는, 초기화와 네트워크 구조에 대해 적절히 동작하는 것을 찾는 하이퍼파라미터 튜닝을 해야 한다.



참조사항

- https://tykimos.github.io/2019/01/22/colab_getting_started/
- <https://teddylee777.github.io/colab/google-colab-%EB%9F%B0%ED%83%80%EC%9E%84-%EC%97%B0%EA%B2%B0%EB%81%8A%EA%B9%80%EB%B0%A9%EC%A7%80>
- <https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/classification.ipynb#scrollTo=DLdCchMdCaWQ>
- <https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e>
- <https://reniew.github.io/12/>
- <https://keras.io/ko/initializers/>
- <https://t1.daumcdn.net/cfile/tistory/996A04425AB85ED026>