AI활용 표현과 문제해결

[MNIST 실습]

허세훈

충남대학교
지능소프트웨어연구실

2021.03.22

지난 주 Review

Anaconda 가상환경 & Jupyter Notebook

Anaconda 가상환경을 Jupyter Notebook에서 이용하는 방법에 대해서 공부

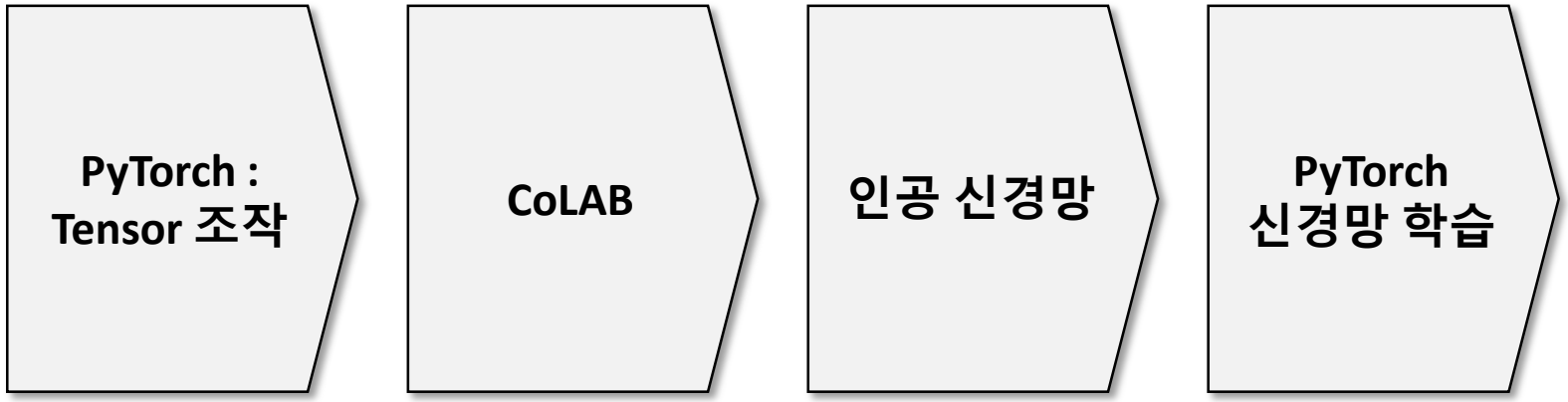
Pandas

Series, DataFrame에 대해 공부
Pandas에서 제공하는 여러 함수들 사용

PyTorch

PyTorch에서 제공하는 Tensor에 대해서 공부

이번 주 실습 내용



- 차원 변경 함수
- 타입 캐스팅
- tensor 병합 함수

- 소개
- 환경 구축

- 단층 퍼셉트론
- 다층 퍼셉트론 (MLP)

- 기본적인 신경망 학습 절차
- Dataset, DataLoader
- MNIST with. MLP

PyTorch : 텐서 조작 | 차원 변경 함수

(1) 뷰(view) : 원소의 수를 유지하면서 텐서의 크기 변경

```
tensor = torch.FloatTensor([[[0, 1, 2],
                             [3, 4, 5]],
                            [[6, 7, 8],
                             [9, 10, 11]]])
tensor
```

```
tensor([[[ 0.,  1.,  2.],
         [ 3.,  4.,  5.]],

        [[ 6.,  7.,  8.],
         [ 9., 10., 11.]]])
```

```
tensor.shape
```

```
torch.Size([2, 2, 3])
```

```
viewed_tensor = tensor.view([-1, 3])
viewed_tensor.shape
```

```
torch.Size([4, 3])
```

(?, 3)의 크기로 변경
→ view([-1, 3])는 (?, 3)으로 변경하라는 의미
→ 3차원 텐서를 2차원 텐서로 변환하는 효과

```
viewed_tensor2 = tensor.view([-1, 1, 3])
viewed_tensor2.shape
```

```
torch.Size([4, 1, 3])
```

(?, 1, 3)의 크기로 변경

PyTorch : 텐서 조작 | 차원 변경 함수

(2) 스퀴즈(squeeze) : 원소의 개수가 한 개인 차원을 제거

```
tensor = torch.FloatTensor([[0], [1], [2]])
tensor
```

```
tensor([[0.],
        [1.],
        [2.]])
```

```
tensor.shape
```

```
torch.Size([3, 1])
```

```
squeezed_tensor = tensor.squeeze()
squeezed_tensor.shape
```

```
torch.Size([3])
```

(3) 언스퀴즈(unsqueeze) : 특정 위치에 원소의 개수가 한 개인 차원을 추가

```
tensor = torch.FloatTensor([0, 1, 2])
tensor.shape
```

```
torch.Size([3])
```

```
unsqueezed_tensor = tensor.unsqueeze(0)
unsqueezed_tensor.shape
```

```
torch.Size([1, 3])
```

0번째에
차원 하나 추가

```
unsqueezed_tensor
```

```
tensor([[0., 1., 2.]])
```

PyTorch : 텐서 조작 | 타입 캐스팅

(4) 타입 캐스팅 : 다른 타입의 Tensor로 캐스팅

참고 : https://pytorch.org/docs/stable/tensors.html

64-bit integer (signed)	`torch.int64` or `torch.long`	`torch.LongTensor`	`torch.cuda.LongTensor`
32-bit floating point	`torch.float32` or `torch.float`	`torch.FloatTensor`	`torch.cuda.FloatTensor`

```
long_tensor = torch.LongTensor([1, 2, 3, 4])
long_tensor
```

```
tensor([1, 2, 3, 4])
```

```
long_tensor.type()
```

```
'torch.LongTensor'
```

```
float_tensor = long_tensor.float()
float_tensor
```

```
tensor([1., 2., 3., 4.])
```

```
float_tensor.type()
```

```
'torch.FloatTensor'
```

PyTorch : 텐서 조작 | tensor 병합 함수

(5) cat 함수(concatenate) : 두 개의 텐서를 연결

```
x = torch.Tensor([[1, 2], [3, 4]])
y = torch.Tensor([[5, 6], [7, 8]])
```

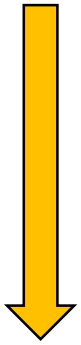
```
concat_tensor1 = torch.cat([x, y], dim=0)
concat_tensor1
```

```
tensor([[1., 2.],
        [3., 4.],
        [5., 6.],
        [7., 8.]])
```

첫번째 차원에 연결
(dim=0)

```
concat_tensor1.shape
```

```
torch.Size([4, 2])
```



```
concat_tensor2 = torch.cat([x, y], dim=1)
concat_tensor2
```

```
tensor([[1., 2., 5., 6.],
        [3., 4., 7., 8.]])
```

두번째 차원에 연결
(dim=1)

```
concat_tensor2.shape
```

```
torch.Size([2, 4])
```

PyTorch : 텐서 조작 | tensor 병합 함수

(6) 스택킹(stacking) : 텐서 쌓기

```
x = torch.Tensor([1, 4])
y = torch.Tensor([2, 5])
z = torch.Tensor([3, 6])
```

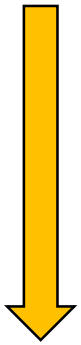
```
stacked_tensor1 = torch.stack([x, y, z])
stacked_tensor1
```

```
tensor([[1., 4.],
        [2., 5.],
        [3., 6.]])
```

첫번째 차원에 쌓기
(dim=0)

```
stacked_tensor1.shape
```

```
torch.Size([3, 2])
```



```
stacked_tensor2 = torch.stack([x, y, z], dim=1)
stacked_tensor2
```

```
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

두번째 차원에 쌓기
(dim=1)

```
stacked_tensor2.shape
```

```
torch.Size([2, 3])
```

PyTorch : 텐서 조작 | 실습 파일 다운로드 링크

https://drive.google.com/drive/folders/1Mpdi1iFOw8PjeADKNloZZOrKHRGKNBmb?usp=sharing

.ipynb 파일을 다운로드 받아서 Jupyter Notebook으로 열어주시면 됩니다.

CoLAB | 소개

클라우드 기반의 무료 Jupyter Notebook

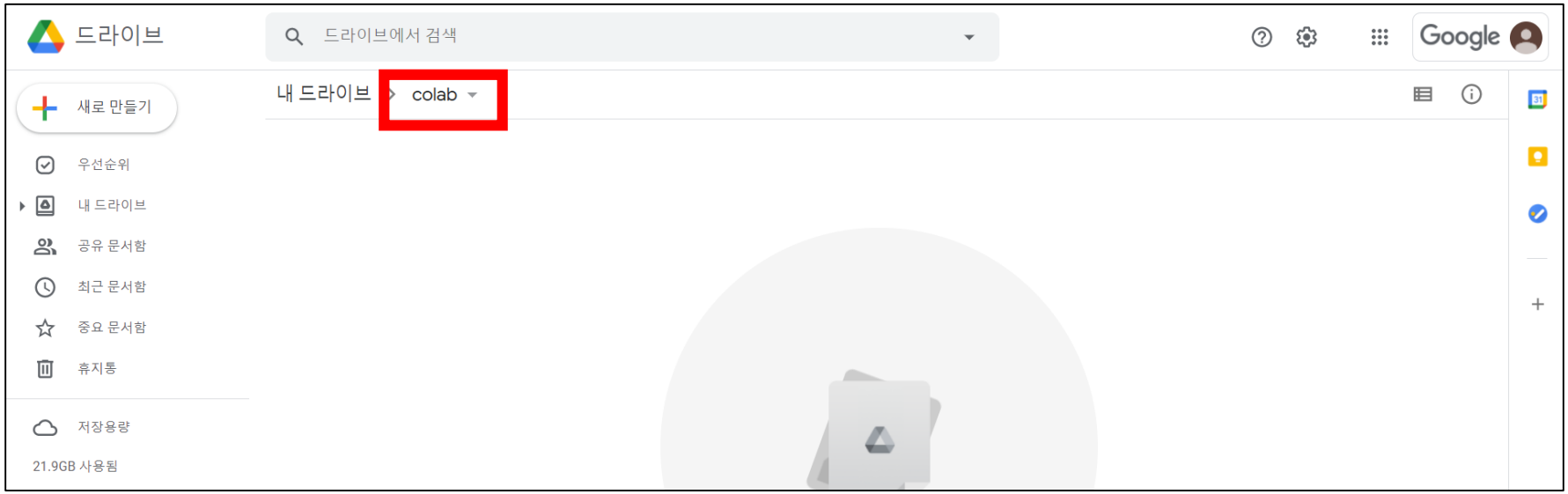
구글 드라이브와 Jupyter Notebook을 사용해봤다면 간단히 사용 가능

GPU 지원

대부분의 학생들은 노트북에 GPU가 없음
→ CoLAB을 이용하여 GPU 연산 기능을 수행

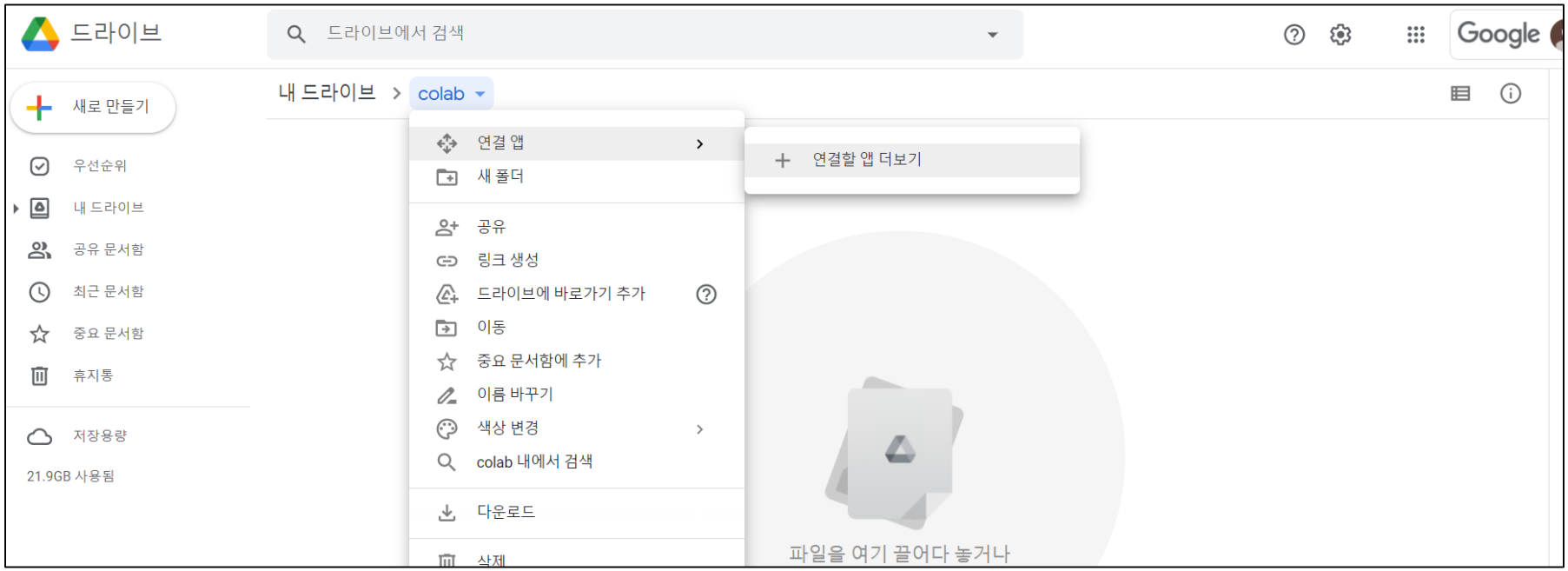
CoLAB | 환경 구축

(1) 구글 드라이브에서 “colab”이라는 이름의 폴더 생성



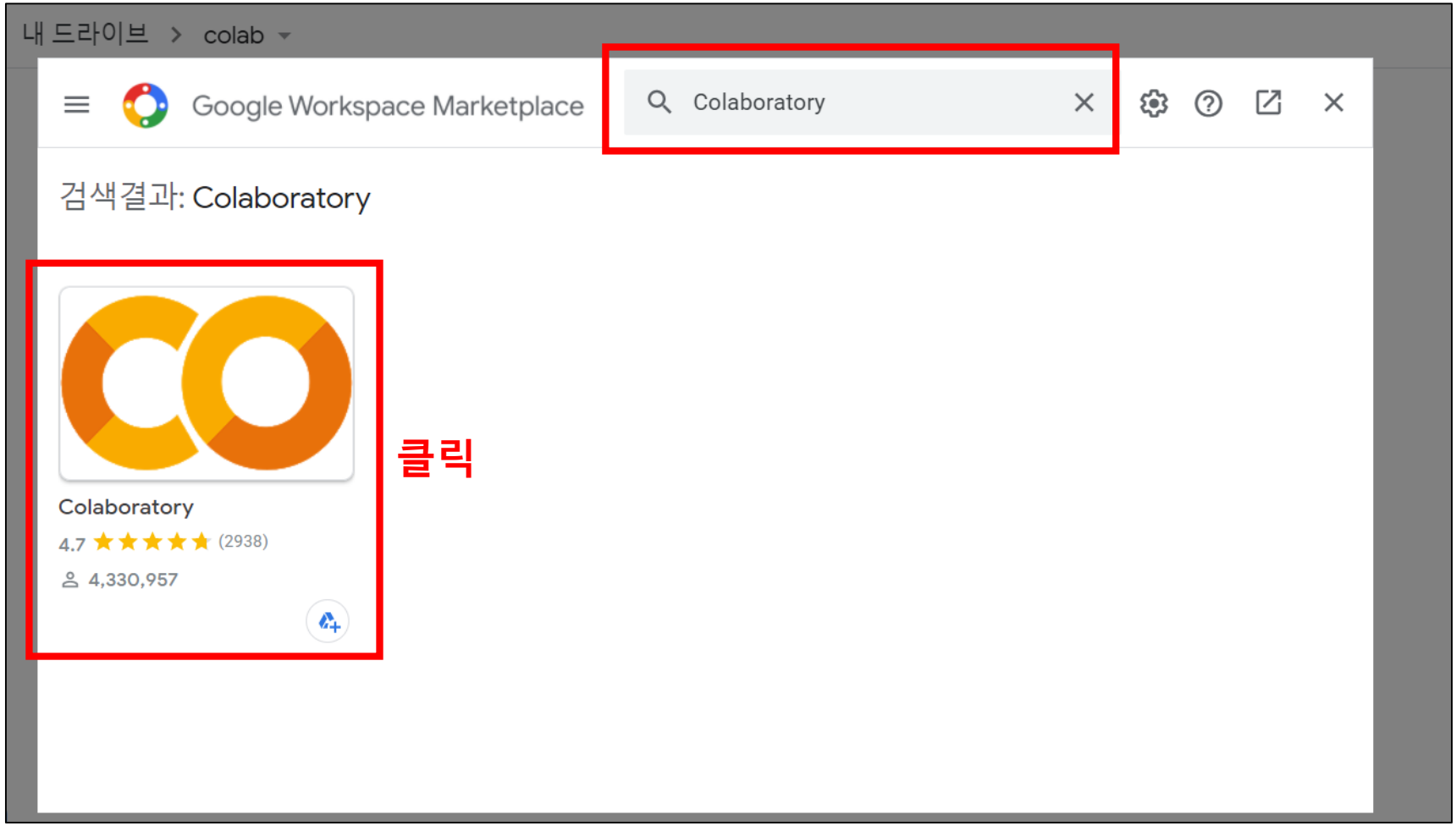
CoLAB | 환경 구축

(2) “연결할 앱” → “연결할 앱 더보기” 클릭



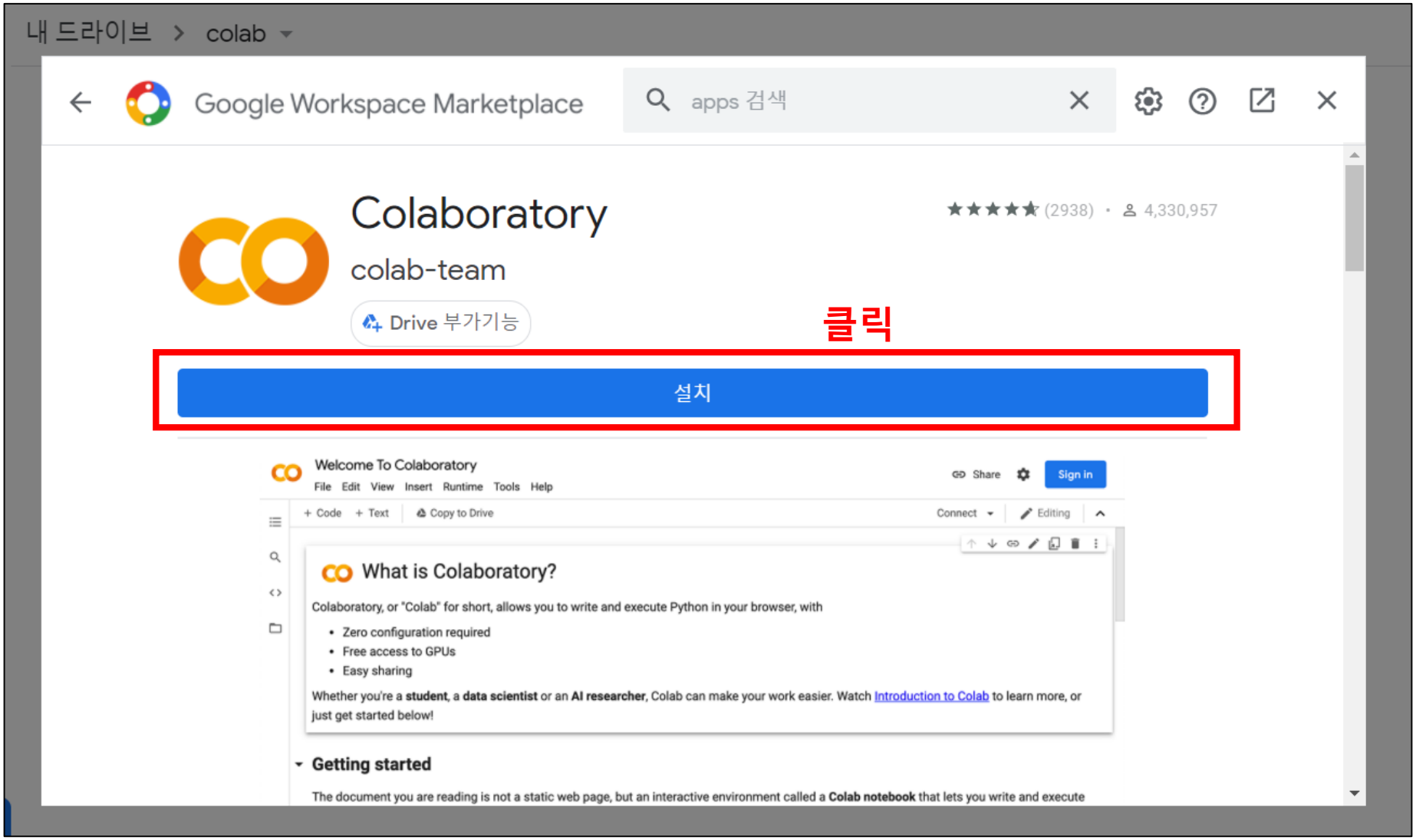
CoLAB | 환경 구축

(3) “Colaboratory”로 검색 후, 선택



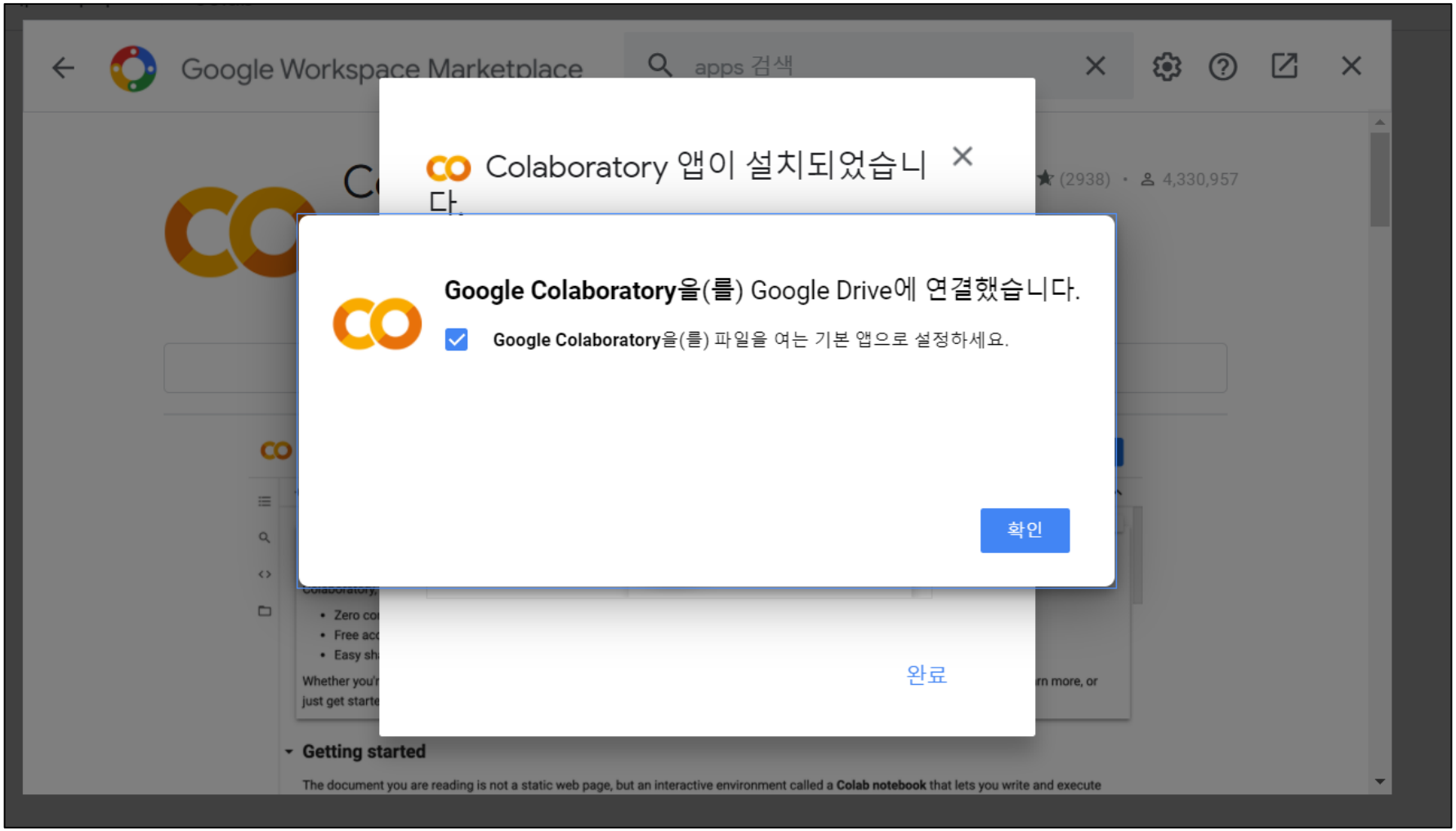
CoLAB | 환경 구축

(4) “설치” 버튼 클릭



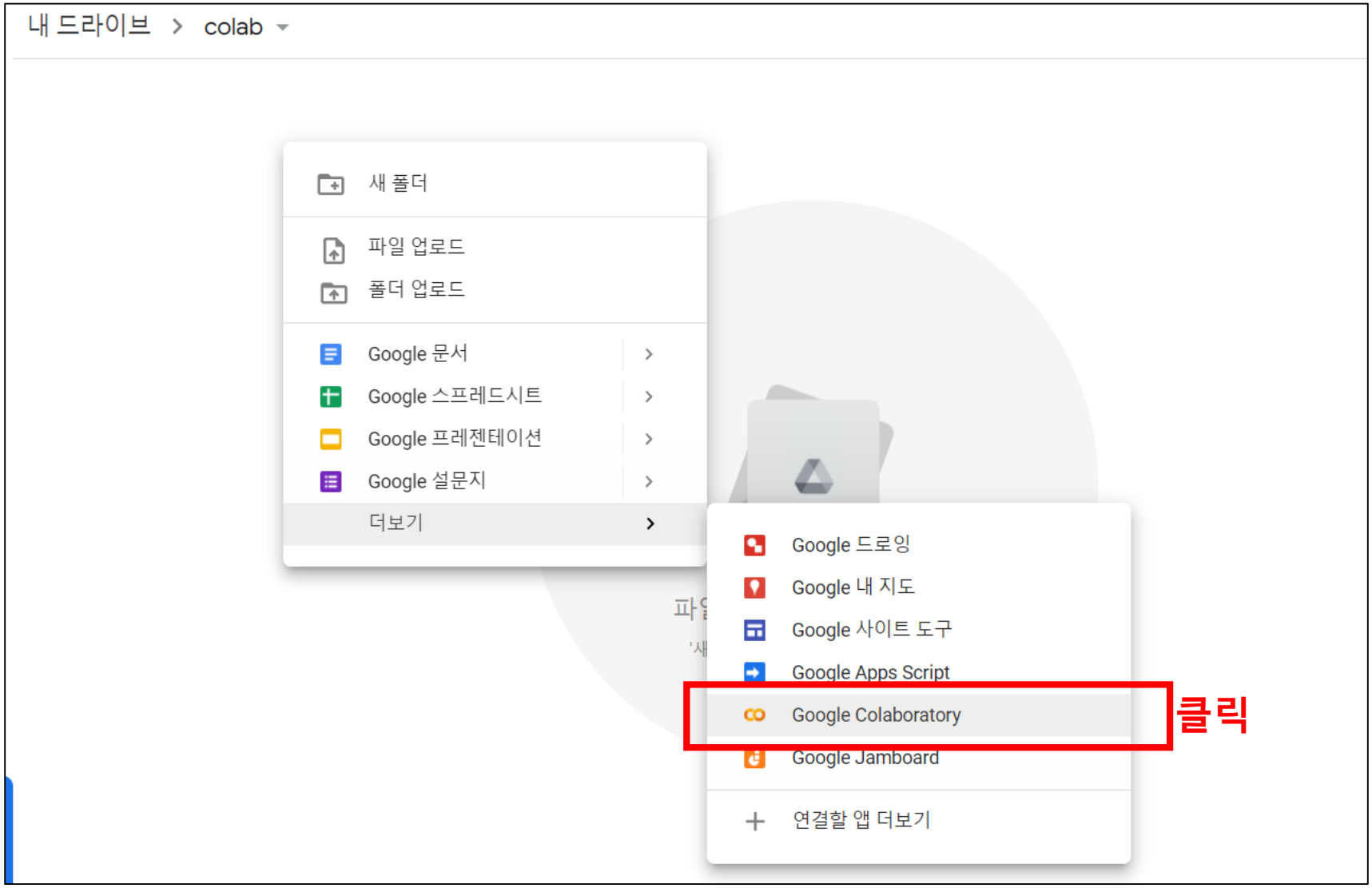
CoLAB | 환경 구축

(5) 설치 완료 : 모든 완료 버튼 및 확인 버튼을 클릭해주세요.



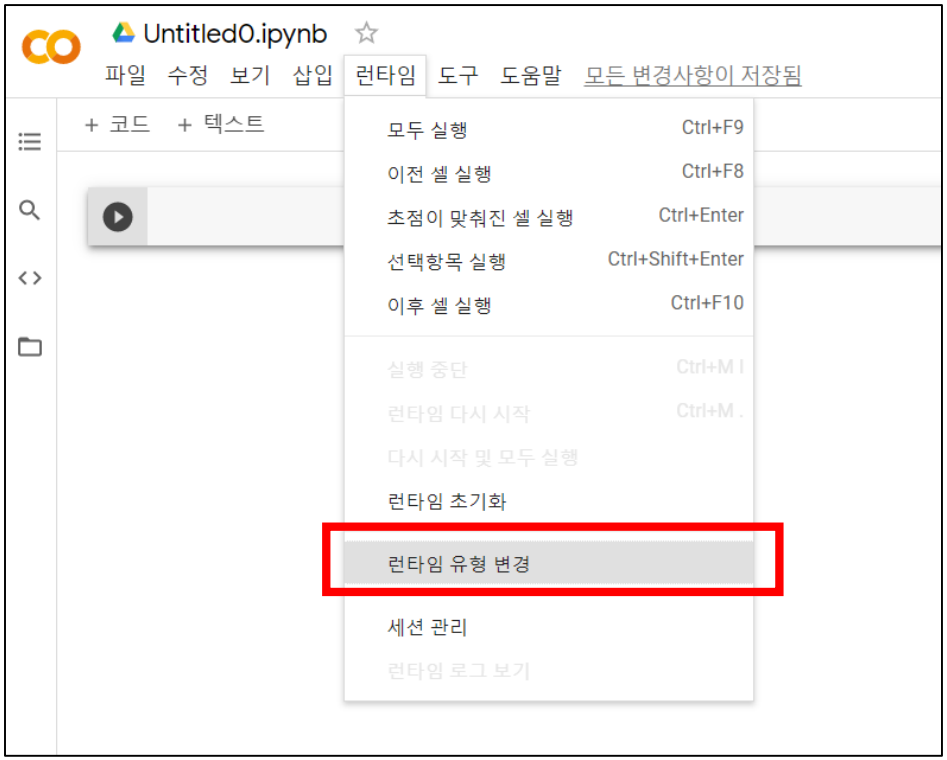
CoLAB | 환경 구축

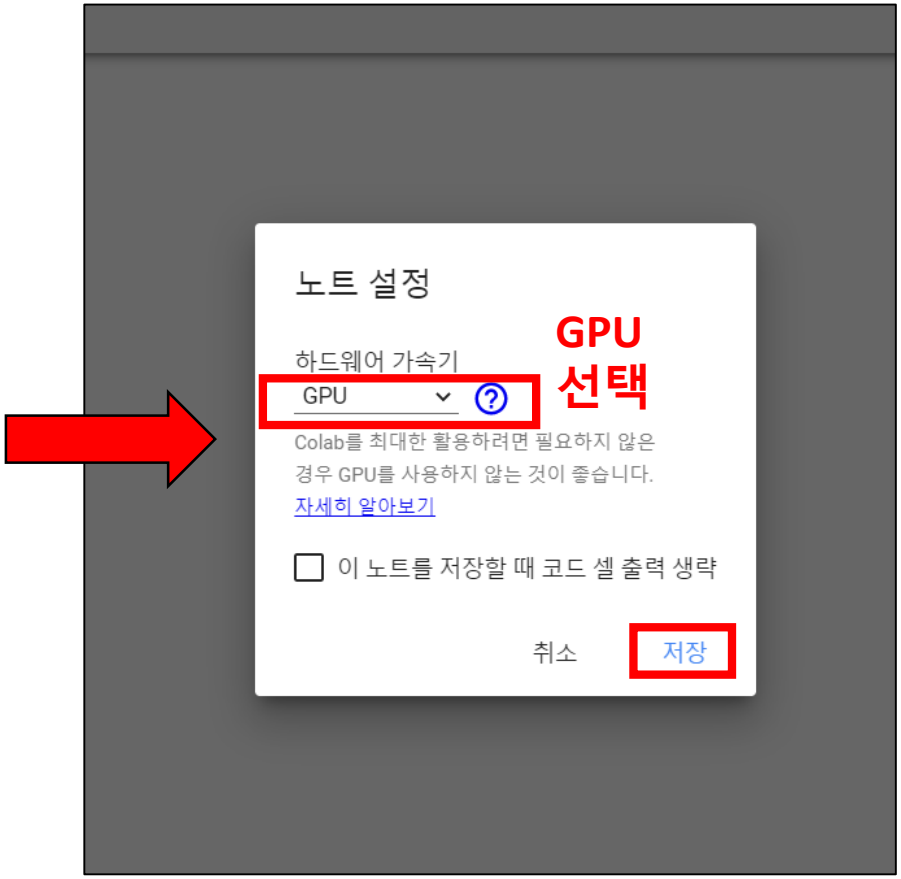
(6) 새로운 Colab 파일 생성



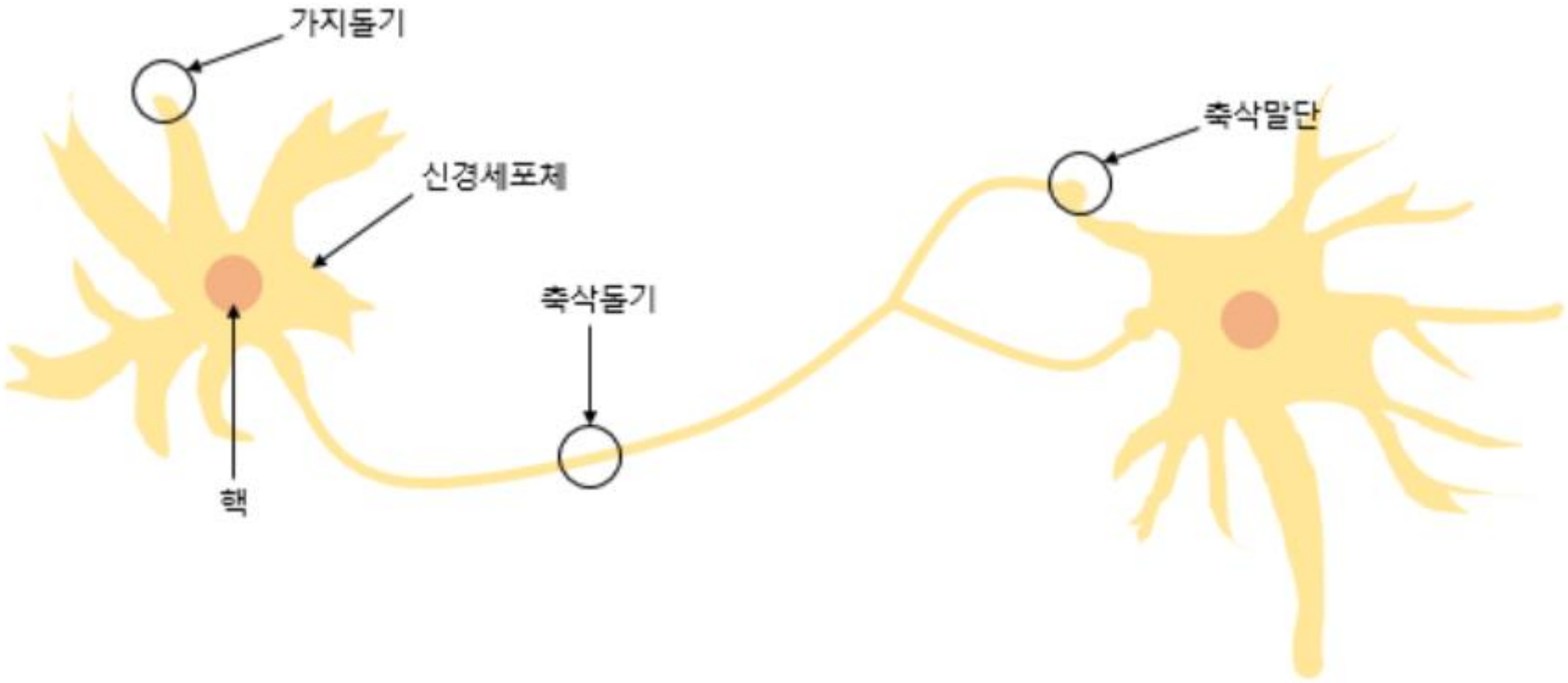
CoLAB | 환경 구축

(7) GPU 환경으로 변경





인공 신경망 | 퍼셉트론

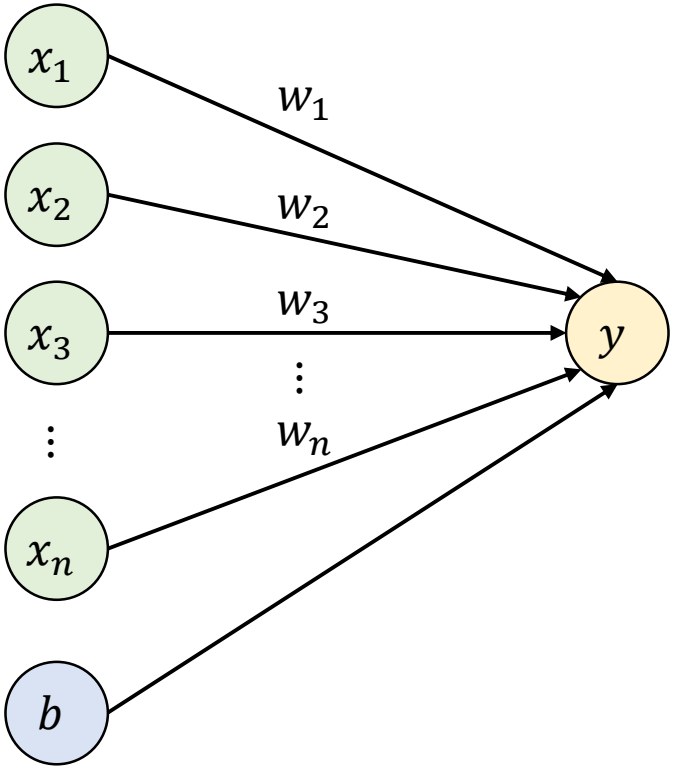


퍼셉트론은 인간의 신경 세포인 뉴런의 구조를 모방

→ 여러 개의 입력으로부터 하나의 결과를 내놓는 구조
→ 역치 이상의 자극이 있을 경우 활성화

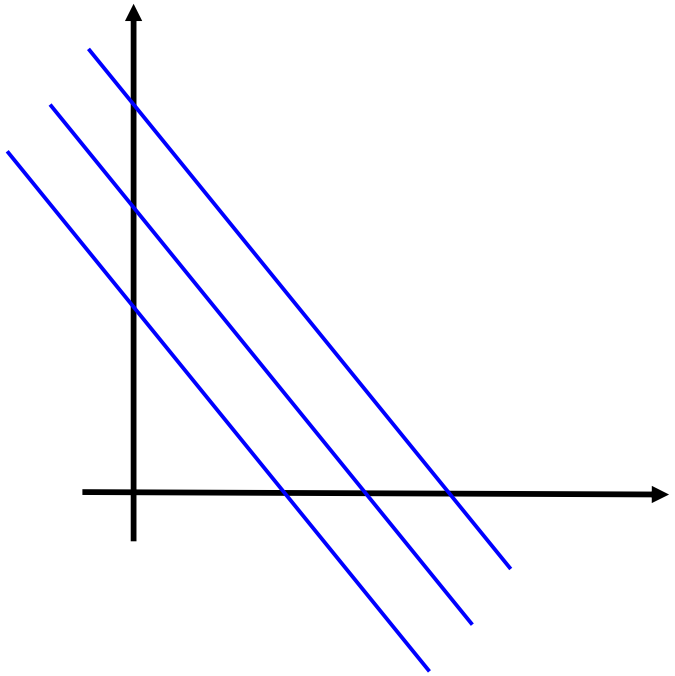
참고 : https://wikidocs.net/60680

인공 신경망 | 단층 퍼셉트론(Single Layer Perceptron)



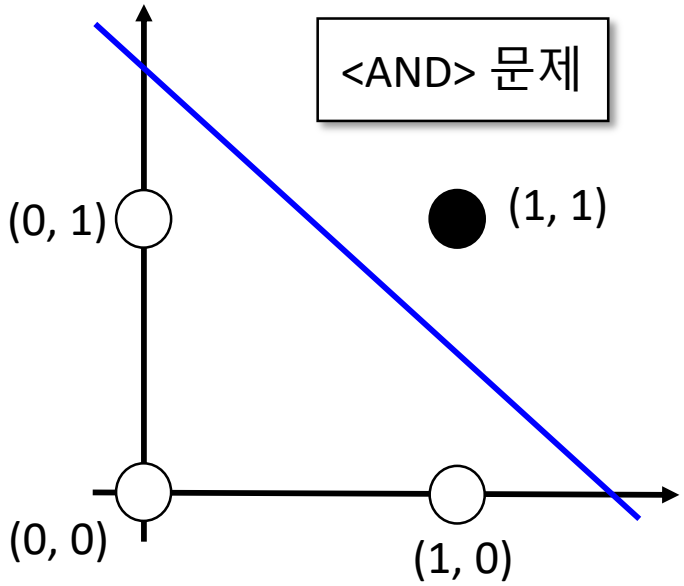
$y = Wx + b$의 구조

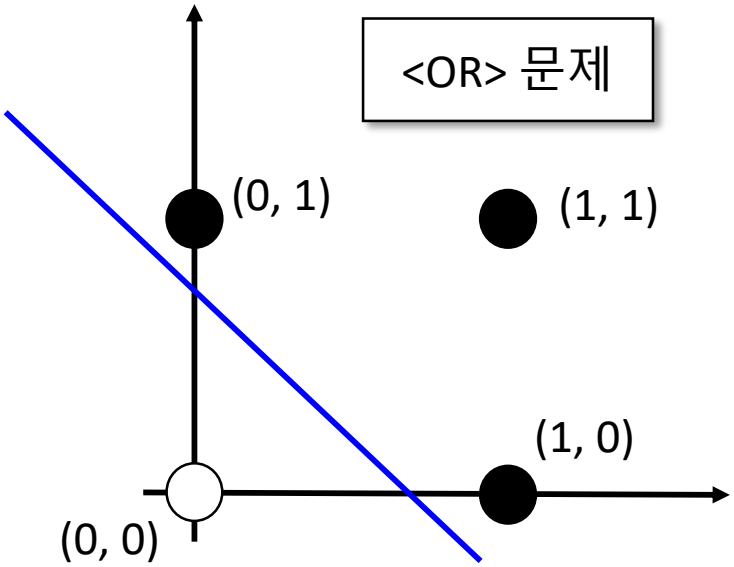
선을 찾는 문제와 동일

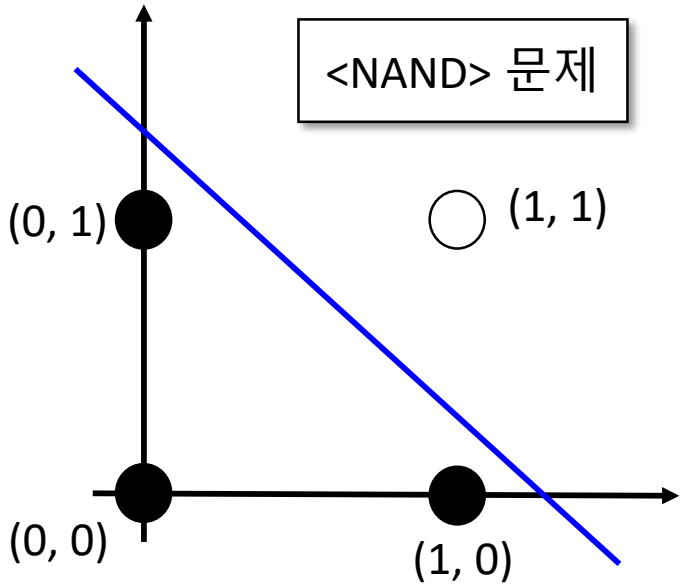


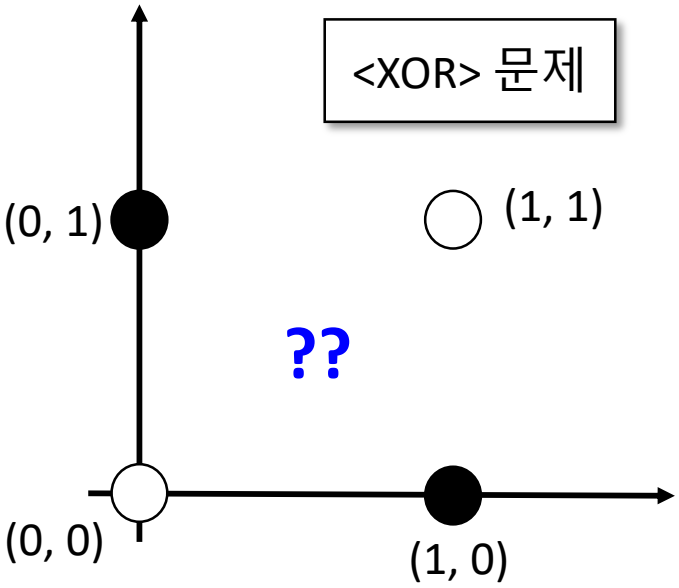
- $[x_1, x_2, x_3, \cdots, x_n]$은 입력 값
- $[w_1, w_2, w_3, \cdots, w_n]$은 가중치
- b는 편향값
- y는 출력 값

인공 신경망 | 단층 퍼셉트론의 문제점

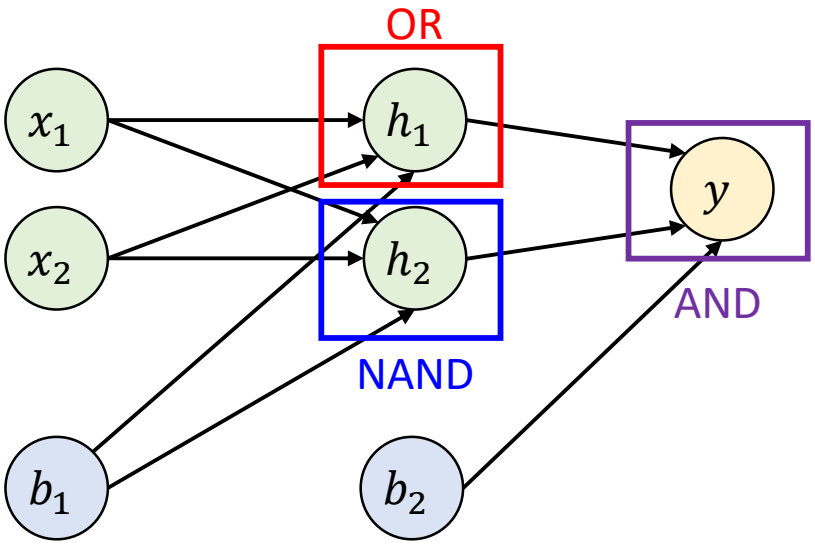








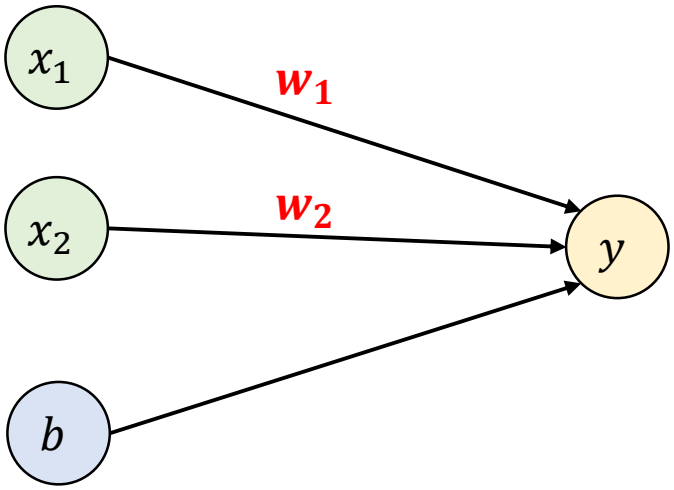
인공 신경망 | 다층 퍼셉트론(Multi Layer Perceptron)



신경망을 여러 층으로 쌓아 올림
→ 그 정도가 깊으면 Deep Learning

x_1	x_2	$OR(x_1, x_2)$	$NAND(x_1, x_2)$	$AND(h_1, h_2)$	y
0	0	0	1	0	0
1	0	1	1	1	1
0	1	1	1	1	1
1	1	1	0	0	0

인공 신경망 | 가중치 최적화



W 값은 어떻게?

Back Propagation이라고 하는 미분 기울기를 활용하는 알고리즘을 통해 기울기를 갱신

(이는 PyTorch와 같은 프레임워크를 통해 자동으로 할 수 있음)

PyTorch 신경망 학습 | 기본적인 신경망 학습 절차

데이터 준비 ..... 입출력 데이터를 담는 Tensor를 생성하는 과정

신경망 모델 구축 ..... 신경망 학습을 위한 모델을 구현하는 과정

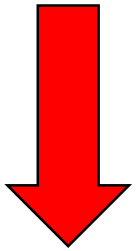
오차 함수 준비 ..... 모델이 얼마나 틀렸는지를 측정하기 위한 함수를 구현하는 과정

옵티마이저 준비 (신경망 최적화) ..... 초기에 설정된 가중치에서 최적의 가중치로 조정하는 옵티마이저를 구현하는 과정

PyTorch 신경망 학습 | Dataset

```python
import torch
from torch.utils.data import Dataset


class CustomDataset(Dataset):
  def __init__(self):
    # 데이터의 전처리를 해주는 함수
    pass

  def __len__(self):
    # 데이터셋의 길이. 즉, 총 샘플의 수를 반환하는 함수
    pass

  def __getitem(self, idx):
    # 데이터셋에서 특정 1개의 샘플을 가져오는 함수
    pass
```

딥러닝 알고리즘에 사용할
데이터를 준비하는 과정 -- **복잡함**



PyTorch는 복잡한 과정을
추상화하기 위해 Dataset 클래스를
제공

PyTorch 신경망 학습 | Dataset 예제

```
class CustomDataset(Dataset):
  def __init__(self):
    self.x_data = [[73, 80, 75],
                   [93, 88, 93],
                   [89, 91, 90],
                   [96, 98, 100],
                   [73, 66, 70]]
    self.y_data = [[152], [185], [180], [196], [142]]

  # 총 데이터의 개수를 리턴
  def __len__(self):
    return len(self.x_data)

  # 인덱스를 입력받아 그에 맵핑되는 입출력 데이터를 파이토치의 Tensor 형태로 리턴
  def __getitem__(self, idx):
    x = torch.FloatTensor(self.x_data[idx])
    y = torch.FloatTensor(self.y_data[idx])
    return x, y
```

```
dataset = CustomDataset()
dataset[0]
```

```
(tensor([73., 80., 75.]), tensor([152.]))
```

PyTorch 신경망 학습 | DataLoader

데이터를 1개씩 처리할 경우, 상당한 시간 필요

→

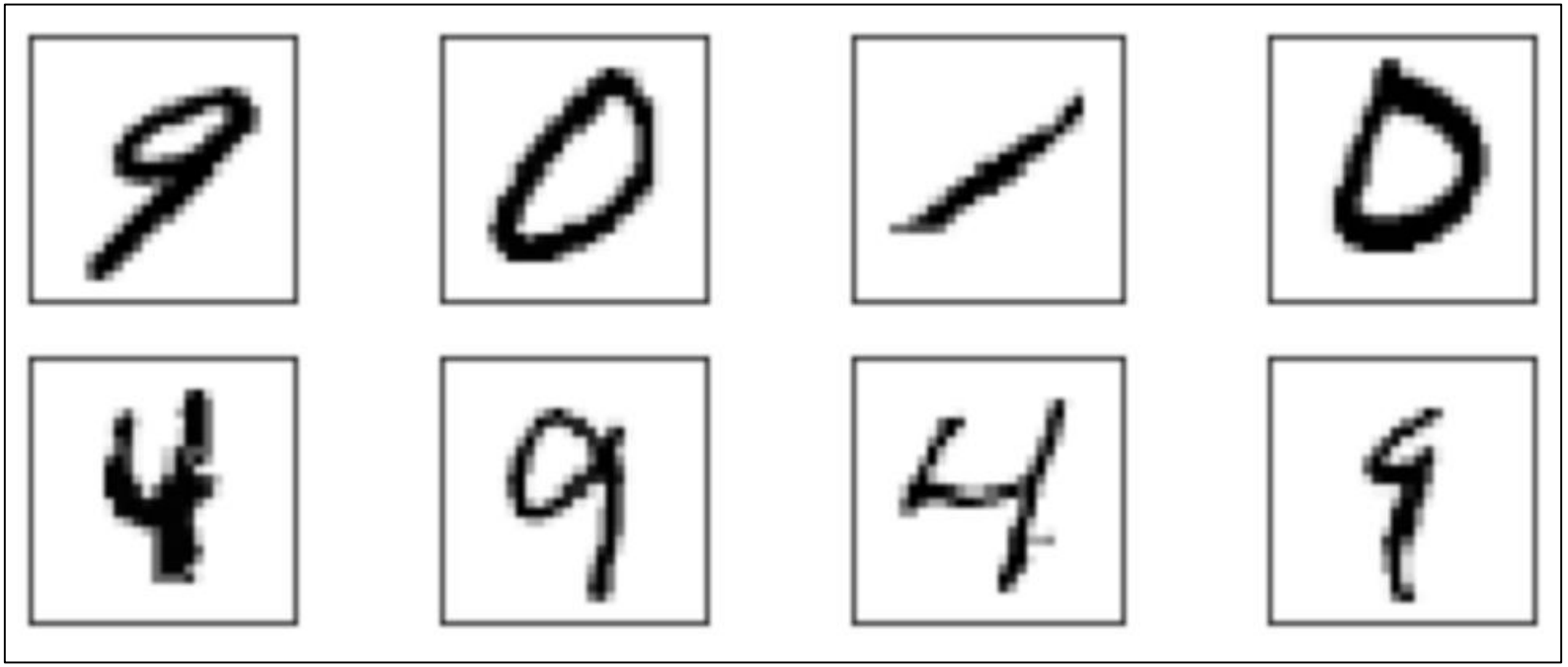
GPU를 활용한 병렬 배치 처리

PyTorch는 DataLoader를 통해 배치 크기 만큼 데이터를 전달할 수 있도록 함

```
from torch.utils.data import DataLoader

dataloader = DataLoader(dataset, batch_size=4)

for x, y in dataloader:
  print(x.shape)
  print(y.shape)

torch.Size([4, 3])
torch.Size([4, 1])
torch.Size([1, 3])
torch.Size([1, 1])
```

→ 4개의 데이터씩 전달됨

PyTorch 신경망 학습 | MNIST 소개



숫자가 적힌 이미지 → 어떤 숫자인지?

PyTorch 신경망 학습 | MNIST 예제 코드

https://colab.research.google.com/drive/191m012WgPM4xZIYEd4QvPWKcQj-vMQmm?usp=sharing