

Chapter 0

PyTorch란?

학습목표

- PyTorch의 주요 특성과 기능 (동적 계산 그래프, 자동 미분, 텐서 연산 등)을 이해하고, PyTorch의 사용 사례를 이해할 수 있다.
- Python, Anaconda, Visual Studio Code, Google Colab 등을 설치하고 설정하여 PyTorch 개발 환경을 구축할 수 있다.
- GPU의 역할과 중요성을 이해하고, GPU를 사용하여 PyTorch 연산을 실행하는 방법을 학습합니다. 또한, GPU가 없는 경우의 오류 처리할 수 있다.
- 텐서 생성 및 조작, 모델 정의 및 학습, CPU와 GPU 간의 텐서 이동 등을 포함한 PyTorch의 기본 사용법을 이해할 수 있다.

1. PyTorch 개요

- 파이토치(PyTorch)는 Facebook(Meta AI)의 AI 연구팀에서 개발한 오픈소스 머신러닝 라이브러리, Python 기반의 딥러닝 프레임워크입니다.
- PyTorch는 2016년에 처음 발표되었고, Facebook AI Research(FAIR) 팀이 주도하여 개발하였습니다.
- Meta는 PyTorch를 사용하여 컴퓨터 비전, 자연어 처리(NLP), 추천 시스템 등 다양한 분야에서 연구하고 있습니다.
- Microsoft, Google, Tesla, Uber, NVIDIA 등 기업들이 널리 사용하고 있습니다.

활용 분야

- 컴퓨터 비전, 자연어 처리, 강화학습, 음성 인식, 생성 모델, 딥러닝 연구와 개발에 사용됩니다.
- 자율 주행, 수요 예측 등

PyTorch 특징

1. 사용 편의성, 유연성, 딥러닝 모델의 빠른 개발 속도가 특징인 오픈 소스 딥러닝 프레임워크입니다.
2. **파이토치는 동적 계산 그래프(dynamic computation graph)를 지원합니다.** 모델의 구조를 학습 중에 동적으로 변경할 수 있어, 디버깅이나 실험을 더 유연하게 수행할 수 있게 합니다. 이를 통해 연구자와 개발자가 모델의 구조를 학습 과정 중에도 변경할 수 있습니다. 이 기법은 모델을 실험하고 디버깅하는 데 매우 유용합니다.

3. **자동 미분(Autograd):** 파이토치는 자동 미분 기능을 제공하여, 역전파(backpropagation) 과정에서 기울기를 자동으로 계산합니다. 이 기능은 신경망 학습에서 필수적인 역할을 하며, 복잡한 미분을 수동으로 계산할 필요가 없게 만듭니다.

torch.autograd 패키지를 통해 계산 그래프를 동적으로 구성하고, 역전파(Backpropagation) 알고리즘을 통해 파라미터를 업데이트할 수 있습니다. 역전파(backpropagation) 과정에서 기울기를 자동으로 계산해 주는 기능으로, 신경망 학습에서 필수적입니다. 4. **텐서(Tensor):** 파이토치는 텐서 연산을 기본으로 하는 라이브러리로, NumPy와 유사합니다. 텐서는 데이터를 나타내는 다차원 배열로, CPU 및 GPU에서 연산을 효율적으로 처리할 수 있습니다. 텐서를 이용해 행렬 연산이나 벡터 연산 등을 수행할 수 있습니다. 5. **GPU 가속:** 파이토치는 CUDA를 지원하여, GPU에서 연산을 수행하여 딥러닝 모델 학습을 가속화할 수 있습니다. 이는 대규모 데이터셋과 복잡한 모델을 다룰 때 매우 유리합니다. torch.cuda 모듈을 사용하면 CUDA를 통한 GPU 연산을 손쉽게 할 수 있습니다. 6. **모델과 학습:** 파이토치는 딥러닝 모델을 쉽게 정의할 수 있게 도와주는 nn.Module 클래스를 제공합니다. 모델을 정의할 때 forward 메서드를 통해 순전파 과정을 정의합니다. 모델 학습을 위한 torch.optim 모듈을 제공하여, 경사하강법(Gradient Descent) 및 다양한 최적화 알고리즘(SGD, Adam 등)을 지원합니다. torch.nn 모듈은 신경망 레이어, 손실 함수, 최적화 방법 등을 포함하고 있어, 복잡한 딥러닝 모델을 손쉽게 구현할 수 있습니다.

- **GPU 사용 여부 체크하기**
- 텐서간의 연산을 수행할 때, 기본적으로 두 텐서가 같은 장치에 있어야 한다.
- 따라서 가능하면, 연산을 수행하는 텐서들을 모두 GPU에 올린 뒤에 연산을 수행한다.
- GPU 사용여부 확인 코드 torch.cuda.is_available()

Python 설치:

Python 3.x 다운로드 페이지에서 본인 운영체제에 맞는 Python 설치 파일을 다운로드하고 설치합니다.

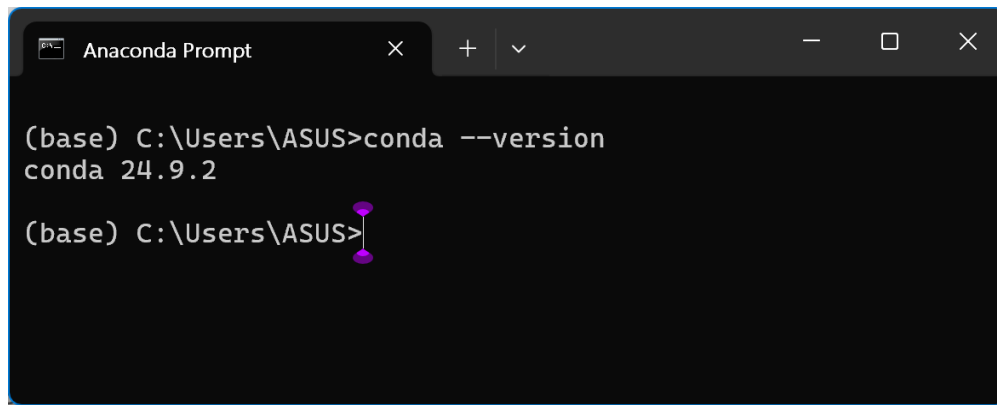
설치 후 python --version 명령으로 설치가 제대로 되었는지 확인합니다.

```
문제    출력    디버그 콘솔    터미널    포트    GITLENS    JUPYTER

PS C:\PyTorch> python --version
Python 3.10.6
PS C:\PyTorch> 
```

Anaconda 설치:

- Anaconda Individual Edition 다운로드 페이지에서 Anaconda를 다운로드하고 설치합니다.
- Anaconda는 필요한 패키지를 쉽게 설치하고 관리할 수 있습니다.
- 설치 후 conda --version 명령으로 설치가 제대로 되었는지 확인합니다.



```
Anaconda Prompt
(base) C:\Users\ASUS>conda --version
conda 24.9.2
(base) C:\Users\ASUS>
```

PyTorch 설치:

- Anaconda 프롬프트를 엽니다.

1. GPU가 없는 경우 :

conda install pytorch cpuonly -c pytorch

2. CUDA를 지원하는 GPU가 있는 경우 :

conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch

```
Anaconda Prompt x + v - □ x

(base) C:\Users\ASUS>conda install pytorch cpuonly -c pytorch
Channels:
- pytorch
- conda-forge
- defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\ProgramData\anaconda3

added / updated specs:
- cpuonly
- pytorch

The following NEW packages will be INSTALLED:

cpuonly          pytorch/noarch::cpuonly-2.0-0
libuv            conda-forge/win-64::libuv-1.49.2-h2466b09_0
pytorch          pytorch/win-64::pytorch-2.5.1-py3.12_cpu_0
pytorch-mutex   pytorch/noarch::pytorch-mutex-1.0-cpu

The following packages will be UPDATED:

ca-certificates  2024.8.30-h56e8100_0 --> 2024.12.14-h56e8100_0
certifi          2024.8.30-pyhd8ed1ab_0 --> 2024.12.14-pyhd8ed1ab_0
conda            24.9.2-py312h2e8e312_0 --> 24.11.1-py312h2e8e312_0
openssl          3.3.2-h2466b09_0 --> 3.4.0-h2466b09_0

Proceed ([y]/n)? y

Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: failed

EnvironmentNotWritableError: The current user does not have write permissions to the target env.
  environment location: C:\ProgramData\anaconda3

(base) C:\Users\ASUS>
```

```
Anaconda Prompt
(base) C:\Users\ASUS>conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
Channels:
- pytorch
- conda-forge
- defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\ProgramData\anaconda3

added / updated specs:
- cudatoolkit=11.3
- pytorch
- torchaudio
- torchvision

The following packages will be downloaded:

package | build | size | channel
-----|-----|-----|-----
cudatoolkit-11.3.1 | hf2f0253_13 | 610.8 MB | conda-forge
-----|-----|-----|-----
Total: 610.8 MB

The following NEW packages will be INSTALLED:

cudatoolkit      conda-forge/win-64::cudatoolkit-11.3.1-hf2f0253_13
libjpeg-turbo    conda-forge/win-64::libjpeg-turbo-2.1.4-hcfcfb64_0
libuv            conda-forge/win-64::libuv-1.49.2-h2466b09_0
libwebp          conda-forge/win-64::libwebp-1.3.2-hcfcfb64_1
pytorch          pytorch/win-64::pytorch-2.5.1-py3.12_cpu_0
pytorch-mutex    pytorch/noarch::pytorch-mutex-1.0-cpu
torchaudio       pytorch/win-64::torchaudio-2.5.1-py312_cpu
torchvision      pytorch/win-64::torchvision-0.20.1-py312_cpu

The following packages will be UPDATED:

ca-certificates  2024.8.30-h56e8100_0 --> 2024.12.14-h56e8100_0
certifi          2024.8.30-pyhd8ed1ab_0 --> 2024.12.14-pyhd8ed1ab_0
conda            24.9.2-py312h2e8e312_0 --> 24.11.1-py312h2e8e312_0
openssl          3.3.2-h2466b09_0 --> 3.4.0-h2466b09_0

Proceed ([y]/n)? y

Downloading and Extracting Packages:
Preparing transaction: done
Verifying transaction: failed

EnvironmentNotWritableError: The current user does not have write permissions to the target environm.
environment location: C:\ProgramData\anaconda3

(base) C:\Users\ASUS>
```

PyTorch 설치 확인:

- Python 인터프리터를 열고 다음 코드를 실행하여 PyTorch가 제대로 설치되었는지 확인합니다.

```
In [15]: import torch
print(torch.__version__)
print(torch.cuda.is_available())
```

2.5.1+cu118

True

※ 기존에 설치된 파이썬, 아나콘다의 버전이 상이하거나 패키지가 충돌되어 에러가 발생할 때
참고하세요.

conda install conda=24.9.2

conda create -n pytorch python=3.12.7

conda activate pytorch

```
In [16]: import torch

# GPU가 사용 가능한지 확인
if torch.cuda.is_available():
    print("GPU 사용 가능")
    print("사용 가능한 GPU 개수:", torch.cuda.device_count())
    print("현재 사용 중인 GPU:", torch.cuda.current_device())
    print("GPU 이름:", torch.cuda.get_device_name(torch.cuda.current_device()))
else:
    print("GPU 사용 불가능")
```

GPU 사용 가능

사용 가능한 GPU 개수: 1

현재 사용 중인 GPU: 0

GPU 이름: NVIDIA GeForce RTX 2060

```
In [17]: # GPU가 없다면 에러
import torch

sample = [[2, 4], [6, 8]]

x = torch.tensor(sample) # torch.tensor(sample)을 호출하여 텐서를 생성합니다.
print(x.is_cuda)

# GPU로 옮기기
x = x.cuda()
print(x.is_cuda) # print(x.is_cuda)는 False를 출력합니다.

x = x.cpu() # CPU로 옮기기
print(x.is_cuda)
```

False

True

False

- 결괏값 해설

- 결괏값 1 : 처음 생성된 텐서 x는 GPU가 아닌 CPU에 할당되어 있기 때문에 x.is_cuda는 False를 출력합니다.
- 결괏값 2 : 텐서를 GPU로 옮기면 x.is_cuda는 True를 출력합니다.
- 결괏값 3 : 다시 CPU로 옮기면 x.is_cuda는 False를 출력합니다.

※ 따라서 GPU가 없는 경우 다음과 같은 코드 블록을 통해 x.cuda() 호출 전에 GPU가 사용 가능한지 확인하고, 그렇지 않으면 경고 메시지를 출력하는 방법을 사용할 수 있습니다.

```
In [18]: import torch

sample = [[2, 4], [6, 8]]

x = torch.tensor(sample)

print(x.is_cuda) # Output: False

if torch.cuda.is_available():
    x = x.cuda()
    print(x.is_cuda) # Output: True

    x = x.cpu()
    print(x.is_cuda) # Output: False
else:
    print("CUDA를 사용할 수 없습니다. GPU가 없는 시스템입니다.")
```

False
True
False

```
In [19]: # device 확인
import torch

tensor = torch.rand(3, 4)
print(f"Device: {tensor.device}")
```

Device: cpu

```
In [20]: import torch

# 샘플 데이터
sample = [[2, 4], [6, 8]]

# GPU가 사용 가능한지 확인
if torch.cuda.is_available():
    device = torch.device("cuda") # GPU를 사용
else:
    device = torch.device("cpu") # CPU를 사용

# 텐서를 생성하고, GPU로 옮기기
x = torch.tensor(sample).to(device)
print(f"Torch 텐서가 GPU에 있는지 여부: {x.is_cuda}") # True여야 GPU에서 연산 중

# CUDA 장치 이름 출력 (확인용)
if torch.cuda.is_available():
```

```

print(f"사용 중인 GPU: {torch.cuda.get_device_name(0)}")

# 텐서를 다시 CPU로 옮기기
x = x.cpu()
print(f"Torch 텐서가 GPU에 있는지 여부: {x.is_cuda}")
# False여야 CPU에서 연산 중

```

Torch 텐서가 GPU에 있는지 여부: True
 사용 중인 GPU: NVIDIA GeForce RTX 2060
 Torch 텐서가 GPU에 있는지 여부: False

```

In [13]: import torch

# 샘플 데이터
tensor = torch.rand(3, 4)

# GPU가 사용 가능한지 확인
if torch.cuda.is_available():
    device = torch.device("cuda") # GPU를 사용
    print(f"사용 가능한 GPU: {torch.cuda.get_device_name(0)}")
else:
    device = torch.device("cpu") # CPU를 사용

# 텐서를 GPU로 옮기기
tensor = tensor.to(device)

# 디바이스 출력
print(f"Device: {tensor.device}") # tensor가 GPU에 있는지 확인

# 만약 GPU가 아닌 'cuda:0' 장치에 텐서를 명시적으로 옮기고 싶다면 아래처럼 작성
tensor = tensor.to('cuda:0')

# 다시 디바이스 출력
print(f"Device (명시적 이동 후): {tensor.device}")

```

사용 가능한 GPU: NVIDIA GeForce RTX 2060
 Device: cuda:0
 Device (명시적 이동 후): cuda:0

- 사용 가능한 GPU: NVIDIA GeForce RTX 2060:
- 시스템에서 사용할 수 있는 GPU가 성공적으로 인식되었습니다. Device: cuda:0:
- 텐서가 GPU로 옮겨졌고, cuda:0 장치(첫 번째 GPU)에서 연산이 이루어지고 있음을 확인할 수 있습니다. Device (명시적 이동 후): cuda:0:
- 추가로 명시적으로 cuda:0 장치로 이동되었음을 확인하는 내용입니다. 동일하게 GPU에서 연산이 이루어지고 있음을 확인할 수 있습니다.
- GPU 장치 지정:
 - cuda:0은 첫 번째 GPU 장치를 의미합니다.
 - 만약 여러 개의 GPU가 있는 시스템이라면, cuda:1, cuda:2 등으로 다른 GPU로 텐서를 이동할 수 있습니다.
- 일관된 디바이스 사용:
 - 일반적으로 모델과 데이터를 동일한 디바이스에 두어야 효율적인 연산이 가능합니다. 예를 들어, 모델이 GPU에 있다면 입력 데이터 또한 GPU로 이동시켜야 합니다.


```
In [21]: import torch
import torch.nn as nn

# 간단한 예제 모델
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        # SimpleModel이라는 간단한 선형 모델을 정의합니다.
        self.linear = nn.Linear(4, 2)

    def forward(self, x):
        return self.linear(x)

# 데이터 생성
sample_data = torch.rand(3, 4)
# 샘플 데이터를 생성하고, 텐서와 모델을 device (GPU 또는 CPU)로 옮깁니다.

# GPU가 사용 가능한지 확인
if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f"사용 가능한 GPU: {torch.cuda.get_device_name(0)}")
else:
    device = torch.device("cpu")
    print("GPU를 사용할 수 없습니다. CPU에서 연산합니다.")

# 데이터와 모델을 디바이스로 옮기기
sample_data = sample_data.to(device)
model = SimpleModel().to(device)
# 모델과 데이터의 디바이스를 출력하여 올바르게 이동되었는지 확인합니다.

# 디바이스 출력
print(f>Data Device: {sample_data.device}")
print(f>Model Device: {next(model.parameters()).device}")

# 모델 연산 실행
output = model(sample_data)
print(f>Output Device: {output.device}")
# 모델 연산을 실행하고, 결과 텐서의 디바이스를 출력하여 연산이 올바른 디바이스에서 수행되었는지 확인합니다.
```

사용 가능한 GPU: NVIDIA GeForce RTX 2060

Data Device: cuda:0

Model Device: cuda:0

Output Device: cuda:0

2. 파이토치 설치

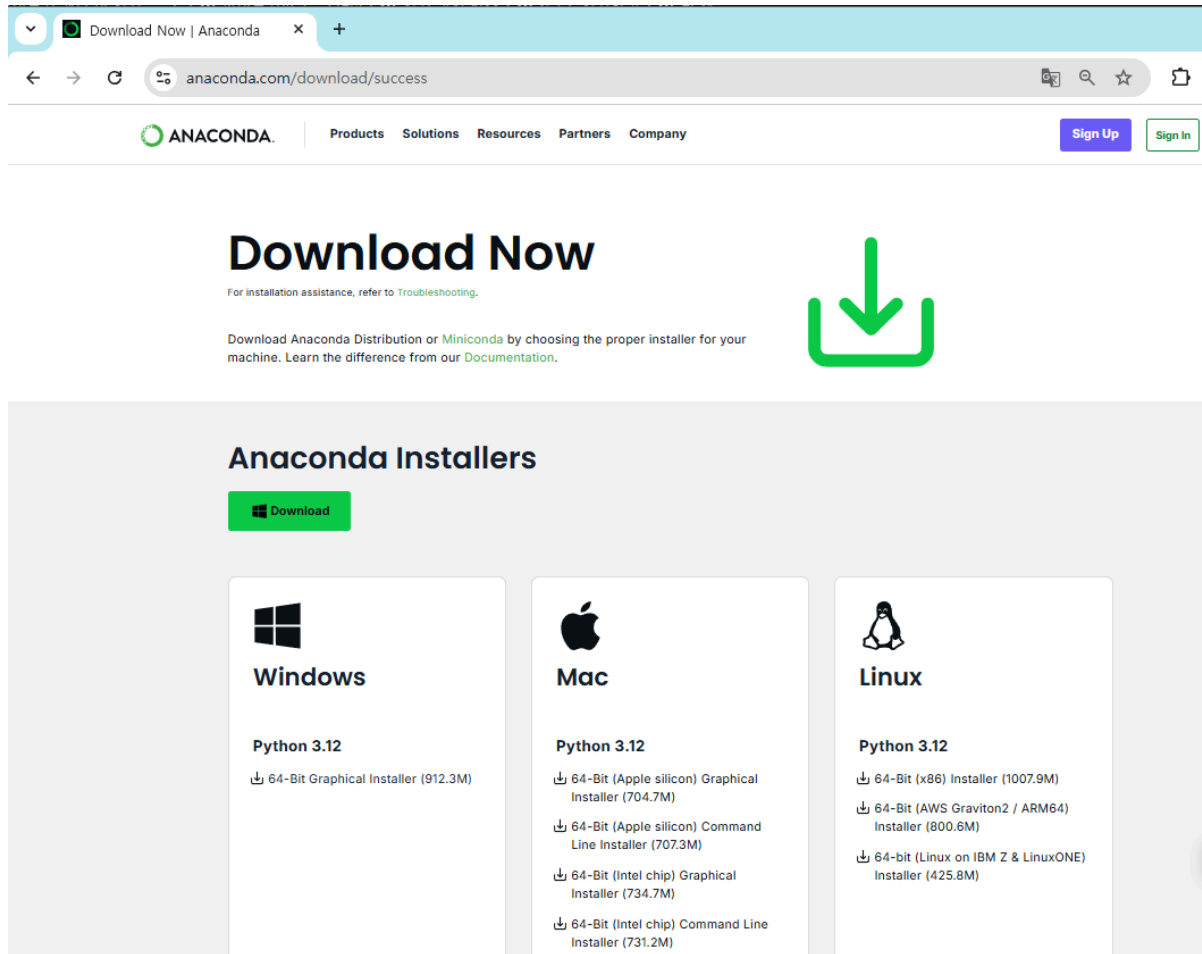
- 파이토치를 효과적으로 사용하기 위해서는 먼저 적절한 개발 환경을 설정해야 합니다.
- 환경 설정은 크게 파이썬(Python), 파이토치 그리고 필요한 의존성 패키지를 설치하는 과정으로 나눌 수 있습니다.
- 파이토치는 Python 3.x 버전에서 지원되므로, Python 3 이상이 설치되어 있어야 합니다. 파이썬은 공식 웹사이트(python.org)에서 다운로드할 수 있습니다. Python 배포판 Anaconda를 설치하면 실습에 필요한 패키지를 포함하고 있어서 편리합니다. 여기에서는 파이토치를 설치하기 전에 아나콘다를 설치합니다.
- PyTorch는 공식 웹사이트(pytorch.org)에서 제공하는 명령어를 통해 설치할 수 있습니다. GPU를 사용할 경우, CUDA(Computed Unified Device Architecture) 버전에 맞는 PyTorch를

설치해야 합니다.

설치 과정 실습

아나콘다 설치하기

- <https://www.anaconda.com/download>
- Skip registration 클릭
- 운영체제에 맞게 다운로드
- Windows 운영체제의 경우는 Anaconda3-2024.10-1-Windows-x86_64.exe가 다운로드 됨



파이토치 설치하기

- <https://pytorch.org/get-started/locally/>
- GPU를 사용하지 않을 경우 CPU 선택하고 그림 1의 Run this Command란의 설치 명령어를 Anaconda Prompt에서 실행
- CUDA 버전을 지원하는 PyTorch를 설치하려면, 그림 2의 Run this Command란의 설치 명령어를 Anaconda Prompt에서 실행

PyTorch Build	Stable (2.5.1)			Preview (Nightly)	
Your OS	Linux		Mac		Windows
Package	Conda	Pip		LibTorch	Source
Language	Python			C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	CUDA 12.4	ROCm 6.2	CPU
Run this Command:	conda install pytorch torchvision torchaudio cpuonly -c pytorch				

그림 1 파이토치 설치_CPU

PyTorch Build	Stable (2.5.1)			Preview (Nightly)	
Your OS	Linux		Mac		Windows
Package	Conda	Pip		LibTorch	Source
Language	Python			C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	CUDA 12.4	ROCm 6.2	CPU
Run this Command:	<code>conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia</code>				

그림 2 파이토치 설치_GPU

파이토치 버전 확인

```
In [22]: import torch
print(torch.__version__)
print(torch.cuda.is_available())
```

2.5.1+cu118
True

- **PyTorch 버전 (torch.__version__):**

출력값 2.5.1+cu118 는 현재 설치된 PyTorch의 버전 번호를 나타냅니다.

- 2.5.1 은 PyTorch의 주요 버전, 부 버전, 수정 버전 번호를 의미합니다. 2.5.1은 파이토치의 2.5.x 버전 시리즈 중 하나입니다.
- +cu118 은 이 버전이 CUDA 11.8을 사용하는 GPU 가속 기능이 포함되어 있음을 나타냅니다. CUDA는 NVIDIA에서 제공하는 병렬 컴퓨팅 아키텍처로, GPU의 처리 능력을 활용할 수 있게 해 주는 툴킷입니다.

- **CUDA 사용 가능 여부 (torch.cuda.is_available()):**

결과값 True 는 현재 PyTorch가 CUDA를 사용할 수 있는 환경에 구축되어 있음을 나타냅니다. 이는 다음을 의미합니다:

- 현재 시스템에 NVIDIA GPU가 설치되어 있으며,
- 해당 GPU에 맞는 CUDA 드라이버가 설치되어 활성화 되어 있습니다.

- PyTorch가 CUDA를 인식하고 사용할 준비가 되어 있다는 뜻입니다.
- PyTorch의 버전은 2.5.1 으로, CUDA 11.8과 호환되며 GPU를 이용한 연산이 가능하다는 것을 확인하였습니다.
- CUDA가 사용 가능하므로, 딥러닝 모델을 GPU를 통해 보다 빠르게 학습하고 추론할 수 있는 환경이 마련되어 있음을 알 수 있습니다. 이는 대규모 데이터셋과 복잡한 모델을 작업할 때 성능 향상에 큰 도움이 됩니다.

```
In [26]: import torch
#0~1사이 실수값으로 3행 3열 텐서 생성
x= torch.rand(3,3)
print(x)
```

```
tensor([[0.2231, 0.2245, 0.9489],
        [0.5041, 0.2989, 0.5751],
        [0.4086, 0.8923, 0.2272]])
```

Colab 설치하기

- 구글 클라우드 기반의 개발환경 Jupyter Notebook 호환
- 데이터 과학, 머신러닝, 딥러닝 개발
- 무료 GPU 사용(T4 GPU, TPU v2-8)
- 구글 드라이브와 연동되어 실시간으로 데이터와 코드를 저장하고 공유할 수 있음

설치 순서

- 구글 드라이브에 로그인 -> 더보기 -> 연결할 앱 더보기 -> Colaboratory 검색 -> 앱 설치
- 설치 후에는 신규 -> 더보기 -> Google Colaboratory
- 수정 -> 노트 설정 -> T4 GPU

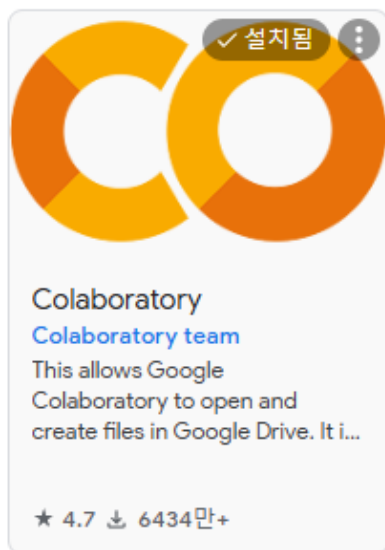


그림 3 코랩 설치

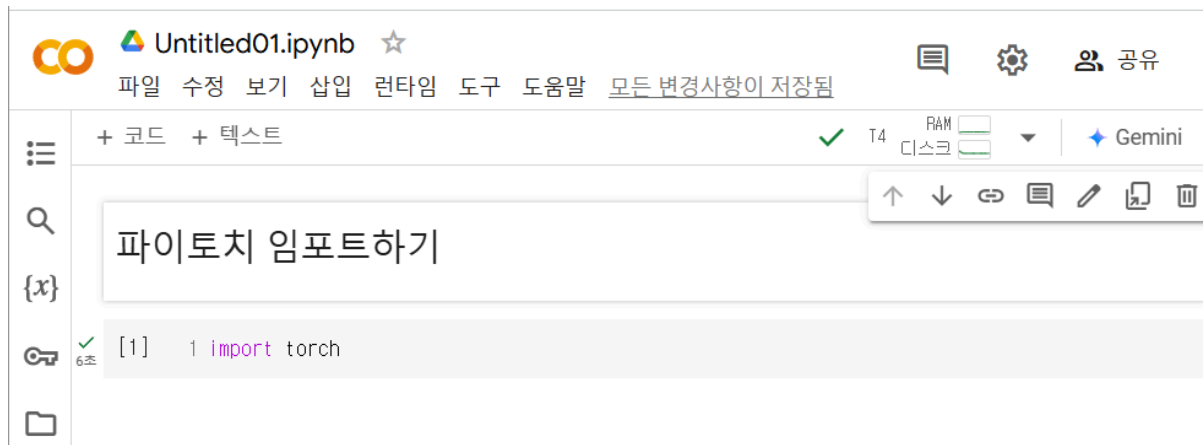


그림 4 코랩 연결

Visual Studio Code 설치

- Visual Studio Code는 파이썬을 포함한 다양한 프로그래밍 언어를 지원하며, 무료입니다.
- Visual Studio Code 다운로드 경로
- <https://code.visualstudio.com/download>
- 운영체제에 맞는 파일 다운로드하기
- 설치 후 파이썬 관련 확장을 추가하기

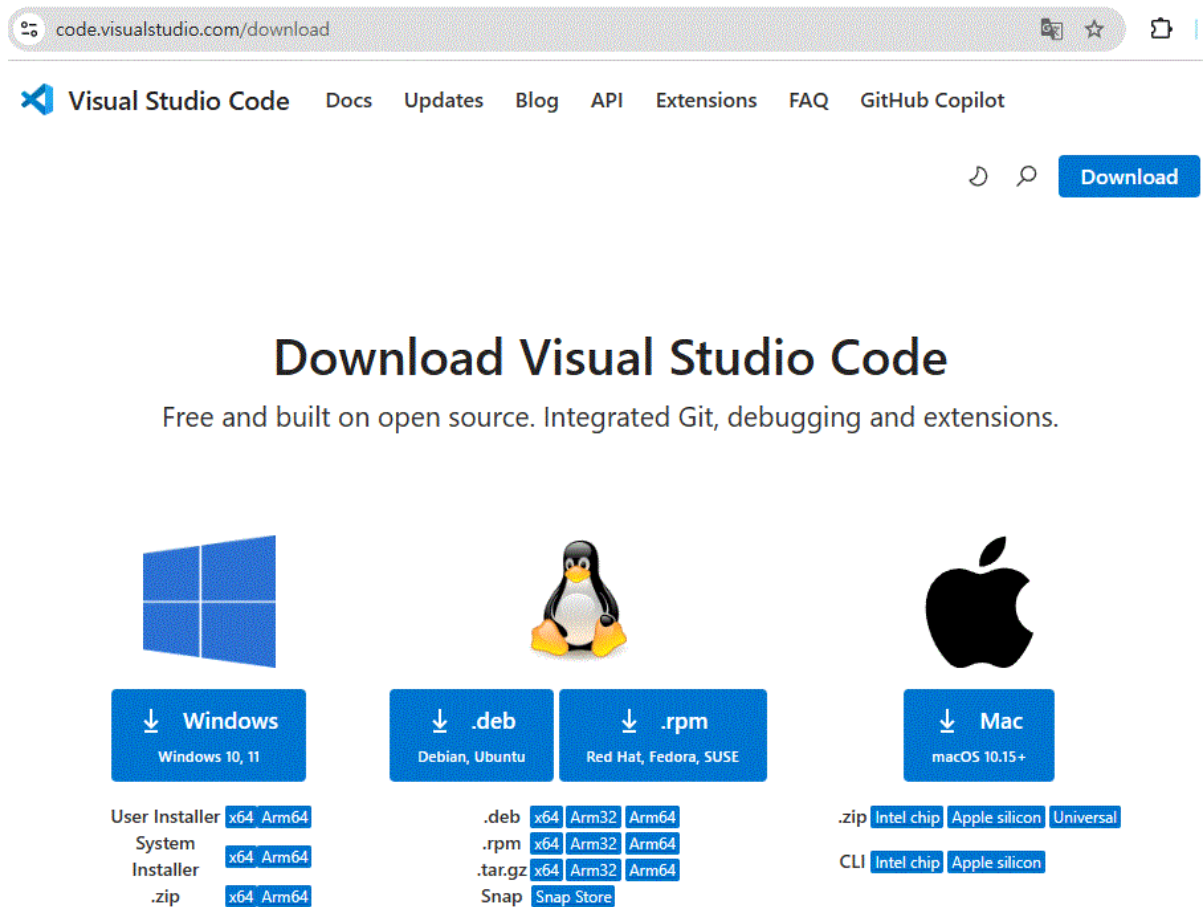


그림. VS Code

1. Python 환경 확인

- 현재 사용 중인 Python 환경이 올바른지 확인해야 합니다. VSCode에서 사용하는 Python 환경이 다른 가상 환경일 수 있기 때문에, 정확한 환경을 선택해야 합니다.
- VSCode에서 Python 환경 선택하기
- VSCode 왼쪽 하단에 있는 Python 버전(혹은 환경)을 클릭합니다.

설치된 Python 환경 목록이 나타납니다. 여기에서 원하는 환경을 선택하세요. (예: 가상 환경 또는 Anaconda 환경) 1.2. 터미널에서 Python 환경 확인

- VSCode에서 터미널을 열고, 아래 명령어를 실행하여 Python 환경을 확인합니다. (터미널-새터미널)

2. pip가 설치되어 있는지 확인

- VSCode 터미널에서 pip이 설치되어 있는지 확인하려면 아래 명령어를 실행하세요.
- `pip --version`

In [27]: `%pip --version`

```
pip 24.2 from c:\Users\ASUS\anaconda3\Lib\site-packages\pip (python 3.12)Note: you may need to restart the kernel to use updated packages.
```

3. 환경변수 설정

- Python을 설치할 때 환경 변수를 자동으로 설정하는 옵션이 있지만, 때때로 이 설정이 제대로 되지 않을 수 있습니다. Python과 pip의 실행 파일을 어디서든 사용할 수 있도록 하기 위해서는 환경 변수를 수동으로 설정해야 할 수 있습니다.
- Python 설치 확인:

C:\Users<사용자명>\AppData\Local\Programs\Python\Python<버전>

- Python 3.10를 설치했다면 C:\Users<사용자명>\AppData\Local\Programs\Python\Python310일 수 있습니다.
- 시스템 환경 변수 열기: Windows 10 이상: Windows 키를 눌러 "환경 변수"를 검색하고 "시스템 환경 변수 편집"을 클릭합니다.

또는, 제어판 -> 시스템 -> 고급 시스템 설정 -> 환경 변수 클릭.

- Path에 Python 경로 추가:

"시스템 변수"에서 Path를 선택하고 편집을 클릭합니다.

- 새로 만들기(N) 추가하고, 아래 경로들을 추가합니다:

Python 실행 파일 경로: 예를 들어 C:\Users<사용자명>

>\AppData\Local\Programs\Python\Python310 Scripts 폴더 경로: 예를 들어 C:\Users<사용자명>\AppData\Local\Programs\Python\Python310\Scripts

- 환경 변수를 설정한 후, 새로운 명령 프롬프트나 터미널을 열고 아래 명령어를 실행하여 python과 pip 명령어가 제대로 작동하는지 확인하세요.

- python --version
- pip --version

```
In [28]: import torch
import torch.nn as nn
import torch.optim as optim

# 간단한 신경망 모델 정의
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# 모델 초기화
model = SimpleNN().cuda() # 모델을 GPU로 이동

# 임의의 입력 데이터 (배치 크기 4)
input_data = torch.randn(4, 2).cuda() # 입력 데이터도 GPU로 이동

# 출력
output = model(input_data)
print(output)
```

```
tensor([[ 0.1847],
        [ 0.2187],
        [ 0.0339],
        [-0.1464]], device='cuda:0', grad_fn=<AddmmBackward0>)
```