두산 Rokey Boot Camp

스터디 주간 활동 보고서

팀명	벌꿀오소리	**제출자 성명**	송주훈
참여 명단	**서준원, 정찬원, 임소정, 강인우, 정민섭, 송주훈**		
모임 일시	**2025년 2월 13일 21시 ~ 22시05분**		
장소	온라인	**출석 인원**	6/6
학습목표	파이 토치에 대한 각자 조사 및 사용법 익히기		

학습내용	정찬원 : 파이토치의 개요 및 특징, 구성요소 / 탠서 활용 방법 / 신경망의 구조등을 조사했다. 신경망 구축에 대한 코드예시도 작성했다.
	• 신경망 구축(torch.nn)
	`import torch.nn as nn #pytorch의 신경망 모듈`
	`import torch.nn.functional as F #활성함수와 같은 함수형 API`
	`#PyTorch에서 모든 신경망 클래스는 nn.Module을 상속해야 함.`
	`#nn.Module: 신경망 정의, 파라미터 추적, forward() method 정의`
	`class SimpleNN(nn.Module):`
	`  def __init__(self):`
	`  super(SimpleNN, self).__init__() #부모 클래스 nn.Module 호출`
	`  #첫번째 완전 연결층 정의`
	`  self.fc1 = nn.Linear(3, 3)  # 입력 차원 3 -> 출력 차원 3`
	`  self.fc2 = nn.Linear(3, 1)  # 입력 차원 3 -> 출력 차원 1`
	`  #신경망의 forward pass 정의`
	`  def forward(self, x):`
	`  #self.fc1(x): 첫번째 완전 연결층 통과한 결과`
	`  x = F.relu(self.fc1(x))  # ReLU활성화 함수 적용`
	`  x = self.fc2(x) #두 번째 완전 연결층 통과 결과.`
	`  return x`
	`model = SimpleNN()`
	`print(model)`		

RELU 함수는 음수 값을 0으로 만들고, 양수 값은 그대로 유지하는 함수.

- **모델 학습(Optimization)**

```
#손실 함수 정의
#예측값과 실제 값 간의 차이 제곱 후 평균 계산.
criterion = nn.MSELoss()  # 평균 제곱 오차 (MSE)

#옵티마이저 설정
#SGD (확률적 경사 하강법) 활용하여 모델 가중치 update.
#model.parameters(): 모델의 모든 학습 가능한 가중치 반환.
#lr=학습률
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

#학습 과정
#epoch: 데이터셋을 모델이 한번 완전 학습하는 과정
#각 에폭마다 입력값, 예측값, 손실계산, 역전파, 가중치 update 이루어짐.
for epoch in range(100):
#100번 반복
    optimizer.zero_grad()  # 기울기 초기화
    outputs = model(torch.rand(3))  # 모델 예측
    # 예측값 outputs과 실제값 torch.tensor([1.0]) 간 손실 계산
    loss = criterion(outputs, torch.tensor([1.0]))  #모델이 1.0 예측하도록 학습.
    loss.backward()  # 역전파 수행(기울기 계산)
    optimizer.step()  # 가중치 업데이트
```

강인우 : 파이토치 기초에 대한 코드를 작성하여 업로드하였다. 코랩을 사용한 gpu사용법에 대해 자세하게 설명해주었다.

In [1]:
```
import torch
```

```
torch.version
```

In [2]:
```
torch.cuda.is_available()
```

Out[2]: True

In [6]:
```
# gpu사용 여부
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

Out[6]: 'cuda'

In [ ]:
```
# gpu사용 여부 - 런타임 변경 후
import torch
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

Out[ ]: 'cuda'

In [3]:
```
# NVIDIA 드라이버
!nvidia-smi
```

```
Thu Feb 13 12:13:05 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   58C    P8             10W /   70W |       2MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

텐서

In [ ]:
```
import torch

# 텐서 생성
t1 = torch.tensor([1,2,3]).int()
print(t1)
print(type(t1),t1.dtype, t1.shape, sep='₩n')
```

```
tensor([1, 2, 3], dtype=torch.int32)
<class 'torch.Tensor'>
torch.int32
torch.Size([3])
```

In [4]:
```
# 행렬 연산
import torch

# 2x2 행렬(텐서) 생성
A = torch.tensor([[1, 2], [3, 4]])
B = torch.tensor([[5, 6], [7, 8]])

# 행렬 덧셈
C = A + B
print("행렬 A + B:₩n", C)
```

```
행렬 A + B:
 tensor([[ 6,  8],
        [10, 12]])
```

gpu로 연산해보기

In [7]:
```
import time

# 큰 크기의 Tensor 생성
size = (100, 100)

# CPU 연산
tensor_cpu = torch.rand(size)
start = time.time()
result_cpu = tensor_cpu @ tensor_cpu  # 행렬 곱 연산
end = time.time()
print(f"CPU 연산 소요시간: {end - start:.4f} 초")

# GPU 연산
tensor_gpu = tensor_cpu.to(device)  # GPU로 이동
start = time.time()
result_gpu = tensor_gpu @ tensor_gpu  # GPU에서 행렬 곱 연산
torch.cuda.synchronize()  # GPU 연산 동기화
end = time.time()
print(f"GPU 연산 소요시간: {end - start:.4f} 초")
```

```
CPU 연산 소요시간: 0.0049 초
GPU 연산 소요시간: 0.1378 초
```

서준원 : 파이토치가 어디서 쓰이는지(활용분야)에 대한 조사를 진행하였고 협동로봇에서 파이토치가 어디서 쓰이는지에 대한 조사, 관련 기사를 조사하였다.

파이토치(PyTorch)는 Facebook 의 AI Research(FAIR)에서 개발한 오픈 소스 딥러닝 프레임워크로, 주로 파이썬(Python)을 기반으로 합니다. 주요 특징은 다음과 같습니다:

1. **동적 계산 그래프 (Dynamic Computation Graph):**
   파이토치는 실행 시점에 계산 그래프를 생성하는 동적 방식(dynamic computation graph)을 사용합니다. 이로 인해 코드 작성과 디버깅이 직관적이며, 모델의 구조를 유연하게 변경할 수 있어 실험적인 연구에 적합합니다.
   (참고: Paszke et al., 2019)
2. **텐서 연산 및 GPU 가속:**
   파이토치는 NumPy 배열과 유사한 텐서를 사용하지만, GPU 를 통한 병렬 연산 지원으로 대규모 데이터와 복잡한 모델의 연산 속도를 획기적으로 향상시킬 수 있습니다.
3. **자동 미분 (Automatic Differentiation):**
   내장된 autograd 시스템은 모델 학습 시 필요한 자동 미분 기능을 제공하여, 역전파(backpropagation)를 손쉽게 구현할 수 있게 합니다.
4. **풍부한 생태계:**
   파이토치는 torchvision, torchtext, torchaudio 와 같은 다양한 서브 라이브러리를 통해 컴퓨터 비전, 자연어 처리, 음성 처리 등 여러 분야의 딥러닝 응용 프로그램을 지원합니다.
5. **유연성과 확장성:**
   연구자와 개발자들이 실험적인 모델을 빠르게 프로토타이핑하고, 생산 환경에 적용할 수 있도록 유연한 인터페이스와 다양한 확장 기능을 제공합니다.

이러한 특징들 덕분에 파이토치는 최신 딥러닝 연구는 물론, 산업 현장에서의 다양한 응용 분야에서 널리 사용되고 있습니다.

1. 파이토치의 활용 분야

(1) 컴퓨터 비전 (Computer Vision)

- **용도:** 이미지 분류, 물체 검출, 영상 분할, 포즈 추정 등
- **설명:** CNN(Convolutional Neural Network) 기반 모델을 PyTorch 로 구현하여, 복잡한 이미지 데이터를 처리하고 특징을 추출하는 데 사용됩니다.
- **참고:**
  - Girshick, R. (2015). *Fast R-CNN*. In Proceedings of the IEEE International Conference on Computer Vision (ICCV). Link
  - He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. In CVPR. Link

(2) 자연어 처리 (Natural Language Processing, NLP)

- **용도:** 텍스트 분류, 기계 번역, 질의응답, 언어 모델링 등
- **설명:** Transformer 와 같은 최신 모델들이 PyTorch 를 통해 구현되어, 대규모 텍스트 데이터의 처리 및 이해에 기여하고 있습니다.
- **참고:**
  - Vaswani, A., et al. (2017). *Attention is All You Need*. In Advances in Neural Information Processing Systems. Link

(3) 강화학습 (Reinforcement Learning)

- **용도:** 자율주행, 로봇 제어, 게임 플레이 등
- **설명:** 에이전트가 환경과 상호 작용하며 최적의 정책을 학습하는 강화학습 알고리즘(예: DQN, Policy Gradient 등)이 PyTorch 를 통해 구현됩니다.
- **참고:**
  - Mnih, V., et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529-533. Link
  - Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). *End-to-End Training of Deep Visuomotor Policies*. JMLR, 17(39), 1-40. Link

(4) 생성 모델 (Generative Models)

- **용도:** 이미지 생성, 스타일 변환, 데이터 증강 등

2. 두산로보틱스 협동로봇에서 파이토치의 역할

두산로보틱스에서는 협동로봇(코봇)을 통해 인간과 함께 작업할 수 있는 안전하고 효율적인 로봇 시스템을 개발하고 있습니다. 이러한 시스템에서는 다양한 센서 데이터를 실시간으로 처리하고, 복잡한 의사결정을 내려야 하는데, PyTorch 는 다음과 같은 측면에서 큰 역할을 할 수 있습니다.

(1) 비전 기반 인식 및 감지

- **설명:**
    - **환경 인식:** 협동로봇이 작업 공간 내의 물체나 사람, 장애물을 인식하기 위해 카메라와 같은 비전 센서에서 수집한 이미지를 처리합니다.
    - **기능:** PyTorch 를 활용한 CNN 기반 모델을 통해 물체 검출, 분할, 포즈 추정 등의 작업을 수행할 수 있으며, 이를 통해 로봇이 주변 환경을 정확히 파악하고 적절한 행동을 결정할 수 있습니다.
- **참고:** 위의 컴퓨터 비전 관련 참고문헌 (Girshick, 2015; He et al., 2016)

(2) 강화학습을 통한 제어 알고리즘 개발

- **설명:**
    - **자율 제어:** 협동로봇이 예측 불가능한 환경이나 동적인 상황에서도 안정적으로 동작하기 위해 강화학습 기반의 제어 알고리즘을 사용할 수 있습니다.
    - **기능:** PyTorch 는 딥 강화학습 알고리즘 구현에 적합하여, 로봇이 시뮬레이션 환경 및 실제 작업 환경에서 스스로 최적의 행동 정책을 학습할 수 있도록 지원합니다.
- **참고:** Mnih et al. (2015); Levine et al. (2016)

(3) 모션 플래닝 및 경로 최적화

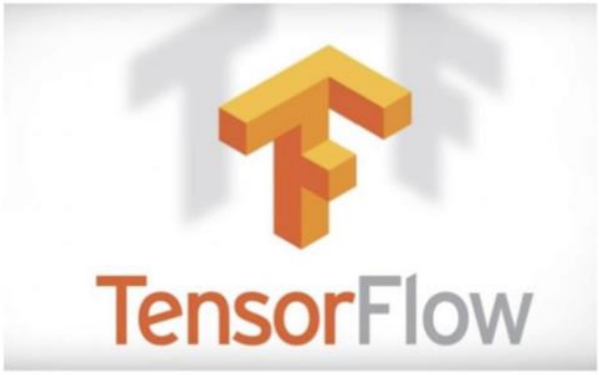
최신 뉴스 기사 1

2025/02/08 12:43

출처: https://zdnet.co.kr/view/?no=20250207171815

[딥시크가 촉발한 '오픈워싱' 논란, 오픈소스의 미래는]

AI 발전 과정에서 오픈소스는 중요한 역할을 해왔다. 초기 AI 연구자들은 개방된 모델과 데이터를 활용해 기술을 발전시키고 협업을 통해 새로운 혁신을 이끌어냈다.



버트, 텐서플로 등을 공개하며 오픈소스AI 생태계를 이끌어온 구글

대표적으로 구글과 메타 같은 기업들은 AI 연구에 필수적인 오픈소스 프레임워크와 모델을 공개하면서 생태계를 확장했다.

구글은 2015년 텐서플로를 공개해 AI 연구 및 개발의 표준을 만들었고, 2014년에는 쿠버네티스(Kubernetes)를 통해 클라우드 환경에서 AI 모델을 효율적으로 운영할 수 있도록 했다.

메타 역시 AI 오픈소스를 적극적으로 활용한 기업으로 꼽힌다. 2016년 공개한 파이토치(PyTorch)는 현재 가장 널리 사용되는 AI 개발 프레임워크 중 하나다. 2022년에는 리눅스 재단 산하 파이토치 재단으로 이관해 커뮤니티 주도의 발전을 지원하고 있다.

PyTorch 는 현재 가장 널리 사용되는 AI 개발 프레임 워크 중 하나이다.

임소정 : 파이토치에 대한 기초적인 연습 코드를 작성하여 배포해였다. 이를 통해 텐서 연산을 연습할 수 있었다.

Import

In [3]:
```
import torch
torch.__version__
```

Out[3]: '2.5.1+cu124'

In [ ]:
```
torch.tensor([1,2,3,4,5])

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

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

In [ ]:
```
#list나 set, numpy array 등에도 적용 가능함
temp_list=[1,2,3,4,5]
torch.tensor(temp_list)
```

Out[ ]: tensor([1, 2, 3, 4, 5])

In [ ]:
```
temp_set=(1,2,3,4,5)
torch.tensor(temp_set)
```

Out[ ]: tensor([1, 2, 3, 4, 5])

In [ ]:
```
import numpy as np

temp_np_array = np.array(temp_list)
temp_np_array
```

Out[ ]: array([1, 2, 3, 4, 5])

Tensor initializaiton and data type

Member functions: convert data into tensor object

- torch.tensor()

Parameter

- data
- dtype : type of data
- device : cpu/gpu
- requires_grad : False (default), indicates the use of gradient

Not initialized tensor

n [ ]:
```
x=torch.empty(3,3)
print(x)
print(x.data)
print(x.dtype)
print(x.device)
print(x.requires_grad)   #x에 requires_grad속성이 빠져 있으므로 False
```

```
tensor([[0.0000e+00, 0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [2.2605e+15, 4.3760e-41, 2.2605e+15]])
tensor([[0.0000e+00, 0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [2.2605e+15, 4.3760e-41, 2.2605e+15]])
torch.float32
cpu
False
```

Randomly initialized tensor

n [ ]:
```
x=torch.rand(3,3)
print(x)
```

```
tensor([[0.8024, 0.8870, 0.4981],
        [0.5270, 0.0628, 0.3081],
        [0.2844, 0.7853, 0.9921]])
```

Zero initialized tensor with data type (dtype=*long*)

n [ ]:
```
x=torch.zeros(3,3, dtype=torch.long)
print(x)
```

```
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

CUDA Tensors (Compute Unified Device Architecture)

- `.to` manage a tensor using cpu or gpu
- GPU에서 병렬 연산 수행 가능하도록 지원하는 기술
- GPU는 한 번에 여러 연산 병렬 수행 가능하므로 빠름

```
x=torch.randn(1)
print(x)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
y = torch.ones_like(x, device=device)
print(y)

x=x.to(device)
print(x)
print(x.device)
```

```
tensor([0.3248])
tensor([1.], device='cuda:0')
tensor([0.3248], device='cuda:0')
cuda:0
```

Multi-dimensional tensor

0D Tensor(Scalar)

```
x=torch.tensor(0)
print(x.ndim)
print(x.shape)
print(x)
```

```
0
torch.Size([])
tensor(0)
```

송주훈: 기초적인 텐서 연산 코드 및 신경망 예제 코드를 작성하였다.

텐서(Tensors)

- 데이터 표현을 위한 기본 구조로 텐서(tensor)를 사용
- 텐서는 데이터를 담기위한 컨테이너(container)로서 일반적으로 수치형 데이터를 저장
- 넘파이(NumPy)의 ndarray와 유사
- GPU를 사용한 연산 가속 가능

```
[ ] import torch

    x = torch.empty(4,2)
    print(x)    # 값은 초기화되지 않은 값이 들어감

    x = torch.rand(4,2)
    print(x)    # 랜덤을 기준으로 초기화됨

    x = torch.zeros(4,2, dtype = torch.long)
    print(x)

    x = torch.tensor([3, 2.3])
    print(x)      # 사용자가 입력한 값으로 텐서 초기화

    x = x.new_ones(2,4, dtype = torch.double)
    print(x)    # 2 x 4 크기 double타입, 1로 채워진 텐서

    x = torch.rand_like(x, dtype=torch.float)
    print(x)    # x와 같은 크기 | , float타입, 무작위로 채워진 텐서

    print(x.size())
```

CUDA Tensors

- `.to` 메소드를 사용하여 텐서를 어떠한 장치(cpu, gpu)로도 옮길 수 있음

+ 코드 + 텍스트

```
x = torch.randn(1)
print(x)
print(x.item())
print(x.dtype)

device =  torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
y = torch.ones_like(x, device = device)
print(y)
x = x.to(device)
print(x)
z = x + y
print(z)
print(z.to('cpu', torch.double))
```

```
tensor([0.5168])
0.5168253779411316
torch.float32
cpu
tensor([1.])
tensor([0.5168])
tensor([1.5168])
tensor([1.5168], dtype=torch.float64)
```

간단한 pytorch neural network 연습

-'torch.nn.Moudle'을 통한 뉴럴 네트워크 생성

```
import torch.nn as nn
import torch.nn.functional as F

class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.fc1 = nn.Linear(10, 20)  # 입력 10개 → 출력 20개
        self.fc2 = nn.Linear(20, 1)   # 입력 20개 → 출력 1개

    def forward(self, x):
        x = F.relu(self.fc1(x))  # 첫 번째 층을 거쳐 ReLU 활성화 함수 적용
        x = self.fc2(x)  # 두 번째 층을 거침 -> 출력되는 결과를 [5,1]에 맞추기 위한 작업
        return x

# 모델 생성
model = SimpleModel()

# 더미 데이터 생성 (배치 크기 5, 입력 크기 10)
x = torch.randn(5, 10)  # 5개의 샘플, 각 샘플은 10차원 벡터
print("x 출력")
print(x)

# 모델에 입력을 넣고 출력 확인
output = model(x)

print("입력 데이터 크기:", x.shape)      # torch.Size([5, 10])
print("출력 데이터 크기:", output.shape)  # torch.Size([5, 1])
print("출력 값:₩n", output)
```

정민섭 : 파이토치의 주요 특징 및 코드, 워크플로우등을 정리해서 연습하였다.

스터디_4주차

1. PyTorch 개요

PyTorch는 Facebook(현 Meta)이 개발한 오픈 소스 머신러닝 프레임워크로, 주로 딥러닝 연구 및 애플리케이션 개발에 사용됩니다. PyTorch는 동적 연산 그래프를 제공하며, Pythonic한 코드 스타일로 직관적인 사용이 가능합니다. 특히 연구자들에게 인기가 많으며, TensorFlow와 함께 가장 널리 사용되는 딥러닝 프레임워크 중 하나입니다.

2. PyTorch의 주요 특징

(1) 동적 연산 그래프 (Dynamic Computation Graph)

- TensorFlow와 달리 동적으로 계산 그래프를 생성하여 실행 중 네트워크 구조를 변경할 수 있습니다.
- 모델의 설계 및 디버깅이 용이하여 연구 및 프로토타이핑에 적합합니다.

▶ 동적 연산 그래프란?

(2) GPU 가속 지원

- CUDA를 활용한 GPU 연산 가속을 기본적으로 지원하여 학습 속도를 크게 향상시킬 수 있습니다.
- GPU 연산을 지원하는 Tensor 연산을 사용하여 손쉽게 가속화가 가능합니다.

```
import torch
x = torch.tensor([1.0, 2.0, 3.0], device='cuda')
print(x)
```

4. PyTorch Workflow (일반적인 워크플로우)

PyTorch를 이용한 머신러닝/딥러닝 모델 개발은 일반적으로 다음과 같은 단계로 이루어집니다:

(1) 데이터 로딩 및 전처리

- `torch.utils.data.Dataset` 및 `torch.utils.data.DataLoader`를 사용하여 데이터 로드 및 배치 처리 수행
- 데이터 증강 및 정규화를 적용

```
from torch.utils.data import DataLoader, TensorDataset
import torch

x_train = torch.randn(100, 10)
y_train = torch.randint(0, 2, (100,))
dataset = TensorDataset(x_train, y_train)
dataloader = DataLoader(dataset, batch_size=16, shuffle=True)
```

(2) 모델 정의

- `torch.nn.Module`을 사용하여 신경망 구조 정의

```
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = nn.Linear(10, 5)
        self.fc2 = nn.Linear(5, 1)
        self.relu = nn.ReLU()

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

(3) 손실 함수 및 옵티마이저 설정

- 손실 함수 (`torch.nn.functional` 또는 `torch.nn`) 및 옵티마이저 (`torch.optim`) 설정

```
model = MyModel()
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

(4) 모델 학습

- 미니배치 단위로 순전파 및 역전파 수행하여 가중치 업데이트

```
for epoch in range(10):
    for x_batch, y_batch in dataloader:
        optimizer.zero_grad()
        y_pred = model(x_batch)
        loss = criterion(y_pred.squeeze(), y_batch.float())
        loss.backward()
        optimizer.step()
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

▶ 미니배치란?

(5) 모델 평가 및 저장

- 검증 데이터로 성능 평가 후 모델 저장

```
torch.save(model.state_dict(), 'model.pth')
```

	## 5. PyTorch의 주요 활용 사례 ### (1) 이미지 처리 - CNN을 이용한 이미지 분류 (ResNet, VGG, EfficientNet 등) - `torchvision`을 이용한 데이터 증강 및 데이터셋 로딩 - GAN을 활용한 이미지 생성 및 스타일 변환 ### (2) 자연어 처리 - LSTM, Transformer, BERT 등을 활용한 언어 모델 개발 - `torchtext`와 함께 사용하여 텍스트 데이터 처리 및 모델 학습 가능 ### (3) 강화 학습 - OpenAI Gym과 함께 강화 학습 알고리즘 (DQN, PPO 등) 구현 가능 - `torch.distributions`를 활용한 확률 모델 적용 가능 ### (4) 의료 및 과학 데이터 분석 - 의료 영상 분석, 유전체 데이터 분석 등 다양한 과학 연구 분야에서 활용 - MRI 및 CT 영상 분석을 위한 PyTorch 기반 모델 활용
활동평가	각자 pytorch의 기본을 연습할 수 있는 시간을 가졌다. 이를 통해 기본적인 텐서 사용법 및 워크플로우를 이해하고 신경망 구축을 연습해볼 수 있었다.

과제	파이토치 프로그래밍 연습
향후 계획	2월 20일 컴퓨터 비전 스터디 예정, 각자 분야를 나눠서 조사 및 공부를 진행하고 발표하는 형식으로 진행
첨부 자료