



2024 D&A

ORIENTATION

Deep Session 1차시



CONTENTS.

01. ORIENTATION

- 수업방식
- 스터디
- 커리큘럼

02. DEEP LEARNING

- 딥러닝 배경
- 딥러닝 툴
- 딥러닝 데이터셋

03. TENSOR

- 텐서

ORIENTATION

수업 방식

세미나

매주 목요일 (18:00 ~ 20:00) ※ 4/18, 4/25는 중간고사 기간이므로 수업 X

장소 : 경영관 103호

강의는 매 주차 녹화를 통해 YouTube 업로드 예정

45분 이론 수업, 10분 휴식, 45분 실습 수업

스터디

매주 한 명씩 이전 주차 세미나 내용 Review

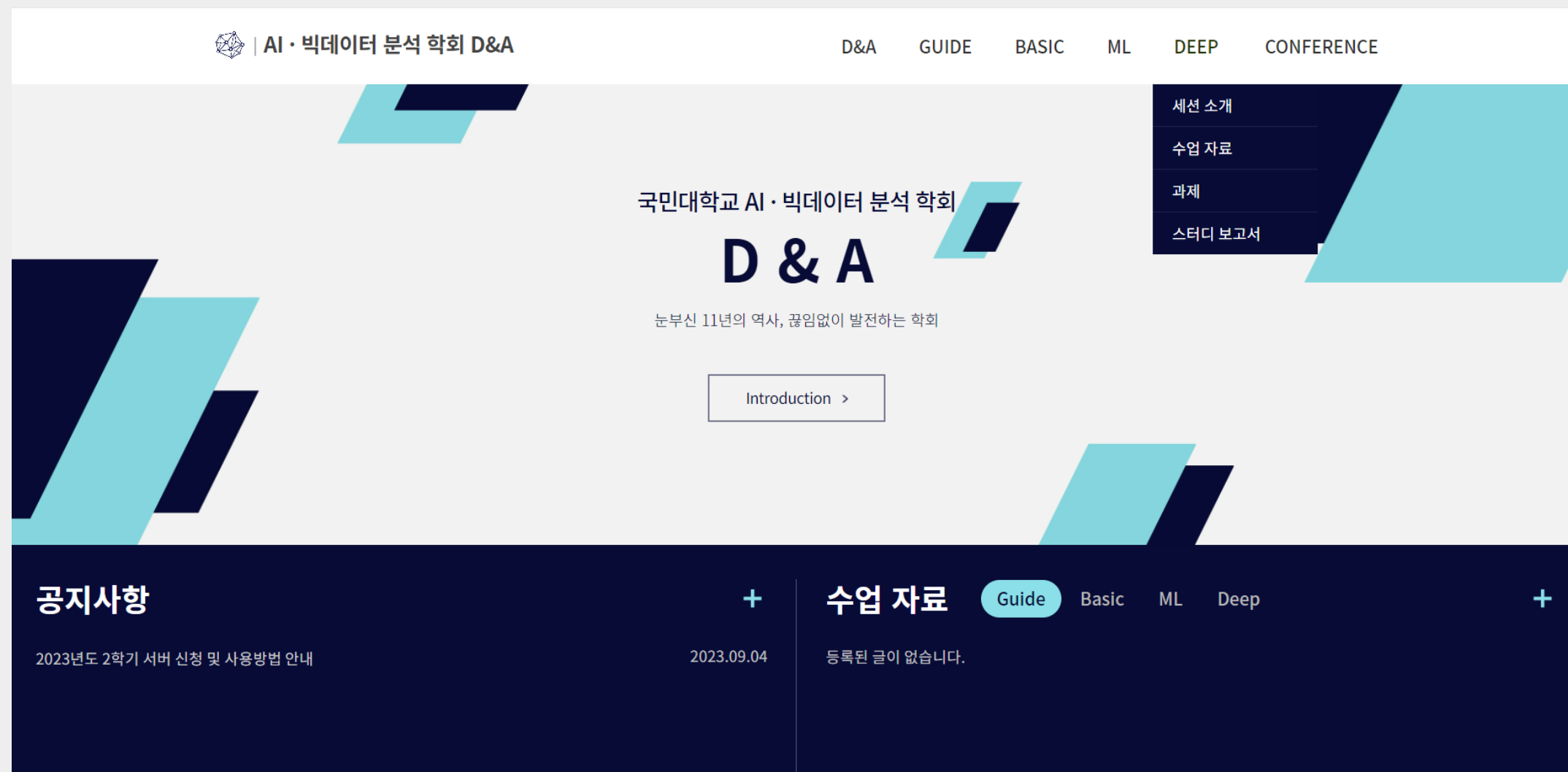
이전 주차 과제 수행 후 매주 한 명씩 발표 및 어려운 점 상의
활동사진, 활동내용, 참여인원을 포함한 스터디 보고서 작성

과제

세미나 시작 하루 전(매주 수요일 11:59PM) 까지 과제 완성본 홈페이지에 제출

ORIENTATION

수업 방식



과제 제출


‘Deep – 과제’ 게시판을 통해 과제 제출
과제 제출 시, 제목은 ‘[조 이름] N주차 성명’으로 제출
ex. [1조] 1주차 김서령

수업 자료

‘Deep – 수업 자료’ 게시판에 업로드 예정

ORIENTATION

수업 방식

 AI · 빅데이터 분석 학회 D&A

D&A BASIC ML DEEP CONFERENCE

D&A

HOME > D&A > Q&A

Q&A

전체

검색어를 입력해 주세요

Q

번호	제목	파일	작성자	등록일	조회수
등록된 글이 없습니다.					

학회 소개

운영진

학회 일정

수상 실적

공지사항

Q&A

등록

Q&A

‘D&A – Q&A’ 게시판을 통해 언제든지 질문 가능
스터디 시간 외에 궁금증이 생겼을 때 적극 활용!

ORIENTATION

수업 방식

청강기간

3월 14일 ~ 3월 28일

본인이 끝까지 학회 활동에 성실히 참여할 수 있는지 판단하는 기간(매주 세미나 스터디 참가, 과제 제출)

성실한 참여가 어려울 것 같다고 느껴지면 하차 가능
단, 청강 기간이 끝난 후 책임감을 갖고 참여해야 함

상품

청강기간 후 학회에 남아있는 학회원들께 굿즈 제공
과제 완성도, 참여도 등을 종합하여 우수자 선정

ORIENTATION

수업 방식

홈페이지

세션 세미나 자료 다운로드 & 매 주 과제 수행 후 업로드
<https://cms.kookmin.ac.kr/dna/index.do>

카카오톡

각종 질문, 건의 사항 문의
http://pf.kakao.com/_lfpJG

인스타그램

학회 및 세션 관련 공지, 매주 세미나 요약
https://www.instagram.com/kmu_dna/

유튜브

실시간 강의 녹화 (매주 링크 제공)
<https://www.youtube.com/@kmudna>

ORIENTATION

스터디

멘토		화					
이지민	신지후	송은아	김상욱	박세현	정진영	이지수	안병민
멘토		수					
조현식	김서령	김무연	권민지	김진하	황예은	진준용	이지안
멘토		목					
이준혁	손아현	백경린	신유인	편서연	박은우	박원우	
멘토		금					
김예향	김차미	유승후	송승원	김규리	김윤재	이재원	

ORIENTATION

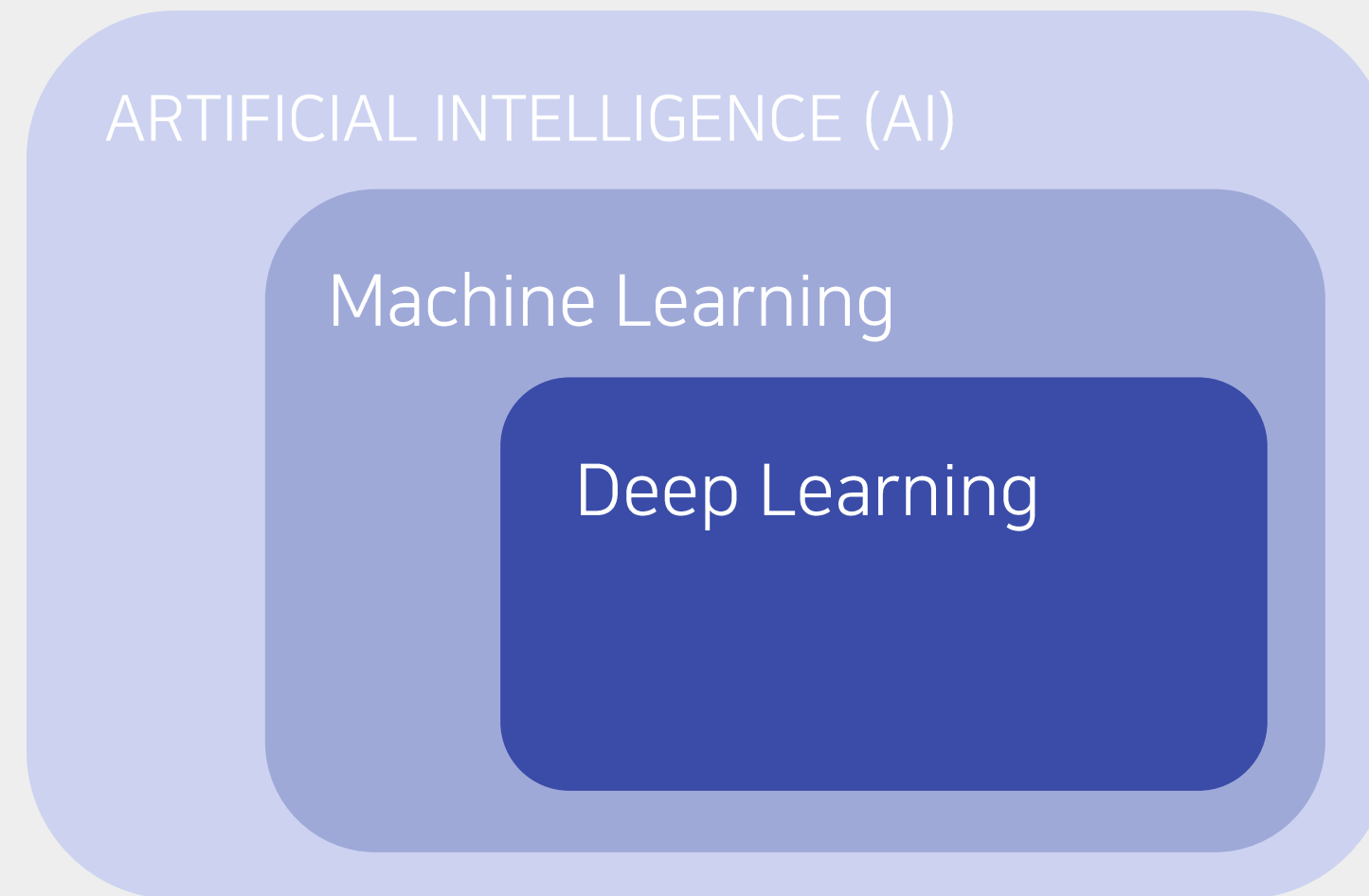
커리큘럼

차시	날짜	내용	발표자
1	03/14	OT, 딥러닝 기초, 텐서	김서령
2	03/21	머신러닝 리뷰, 경사하강법, 인공신경망, 오차역전파법	이준혁
3	03/28	딥러닝 관련 학습 기법들	손아현
4	04/04	CNN 기초	이지민
5	04/11	CNN 심화 I (LeNet, AlexNet, VGG)	김차미
6	05/02	CNN 심화 II (GoogLeNet, ResNet)	신지후
7	05/09	RNN 기초 (Tokenization, Embedding)	김예향
8	05/16	RNN 심화 (RNN, LSTM, GRU, seq2seq)	조현식
9	05/23	Attention, Transformer	김서령

DEEP LEARNING

딥러닝 배경

딥러닝



인공지능 : 인간의 학습 능력, 추론 능력, 지각 능력, 그 외에 **인공적으로 구현한 컴퓨터 프로그램** 또는 **이를 포함한 컴퓨터 시스템**

머신러닝 : 경험을 통해 자동으로 개선하는 컴퓨터 알고리즘 연구

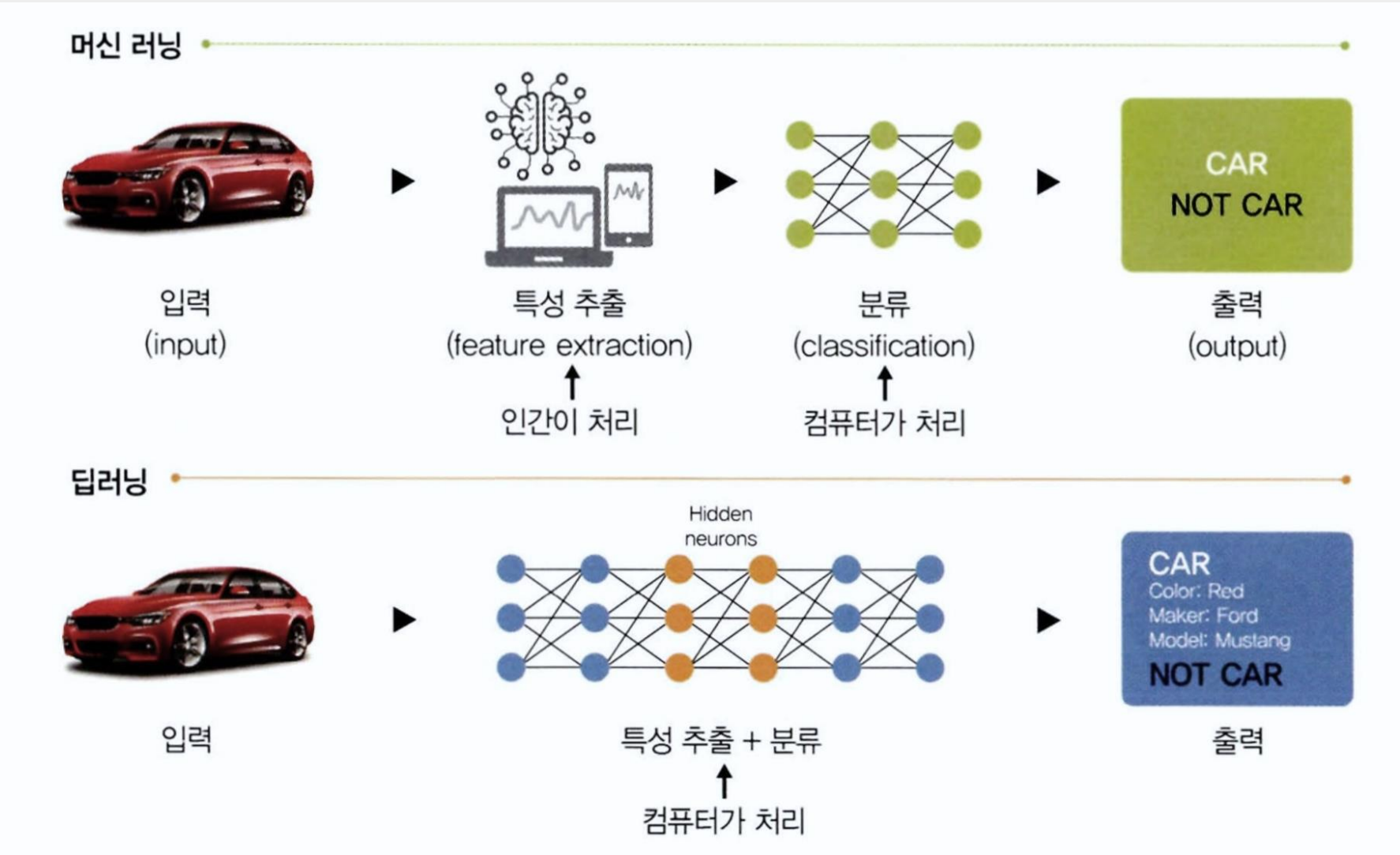
딥러닝 : 여러 비선형 변환 기법의 조합을 통해 **높은 수준의 추상화**를 시도하는 기계 학습 알고리즘의 집합

DEEP LEARNING

딥러닝 배경

머신러닝 vs 딥러닝

구분	머신러닝	딥러닝
특징 추출	사람이 직접 특징을 추출	사람이 특징 추출에 관여 X
동작 원리	입력 데이터에 알고리즘을 적용하여 분석 수행	정보를 전달하는 신경망을 사용하여 데이터 특징 및 관계를 해석
재사용	동일한 유형의 데이터 분석을 위한 재사용이 불편적이지 않음	동일한 유형의 데이터 분석을 위한 재사용에 적합함
데이터	정형 데이터	비정형 데이터
결과	일반적으로 점수 또는 분류 등 숫자값	점수, 텍스트, 소리 등 어떤 것이든 가능



DEEP LEARNING

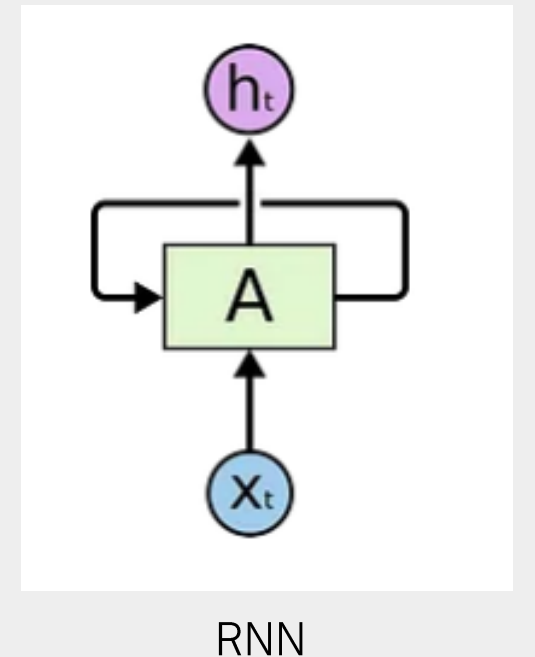
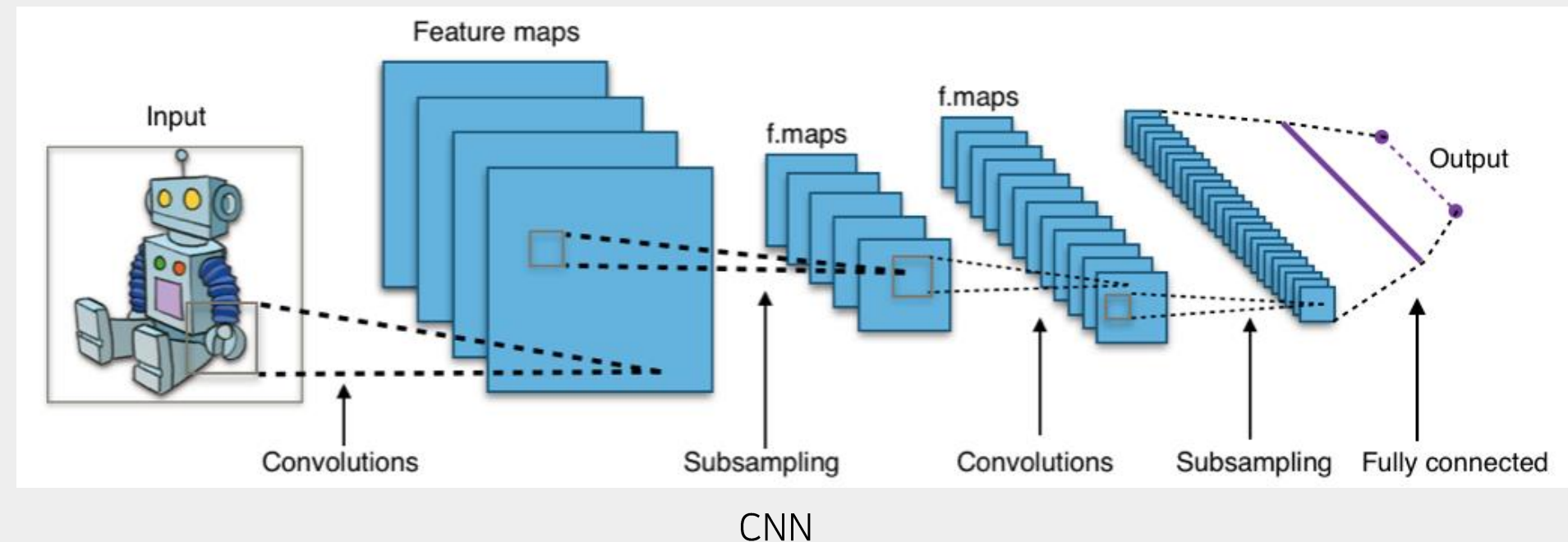
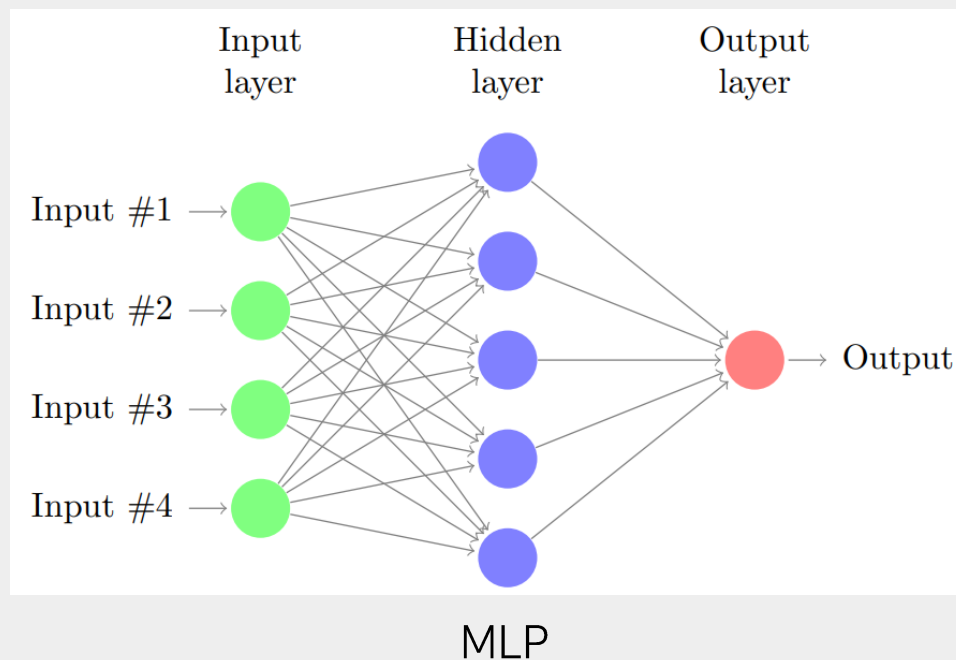
딥러닝 배경

딥러닝 정의

2개 이상의 Hidden Layer를 지닌 다층 신경망

딥러닝 기본 구조

여러 개의 Layer를 가지고 있는 **MLP**(Multi-Layer Perceptron)
이미지 관련 분야에서 많이 사용되는 **CNN**(Convolution Neural Network)
텍스트와 같은 시계열 분야에서 많이 사용되는 **RNN**(Recurrent Neural Network)



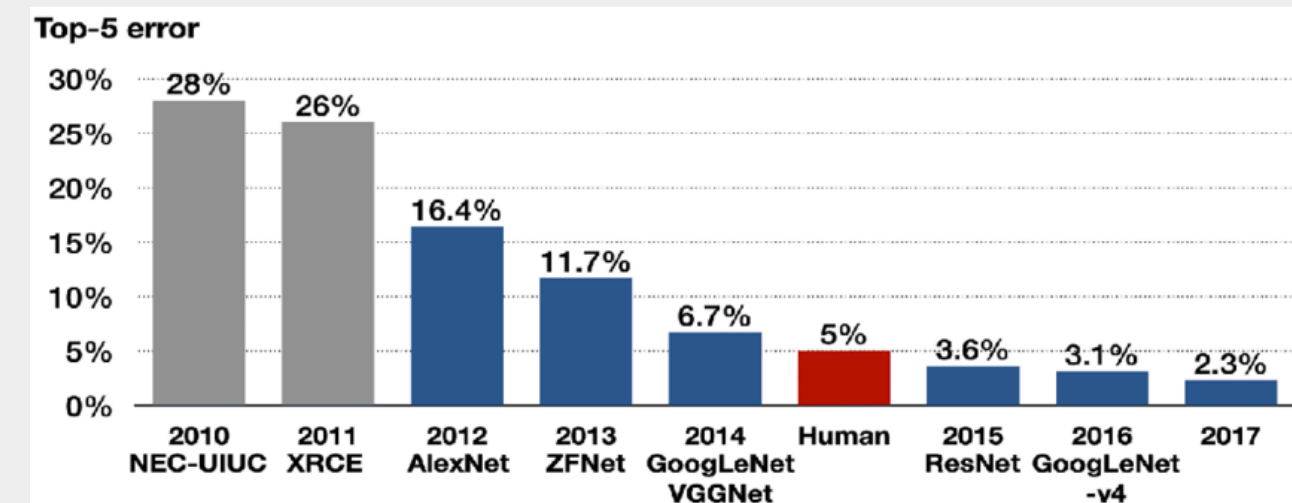
DEEP LEARNING

딥러닝 배경

딥러닝 응용 task

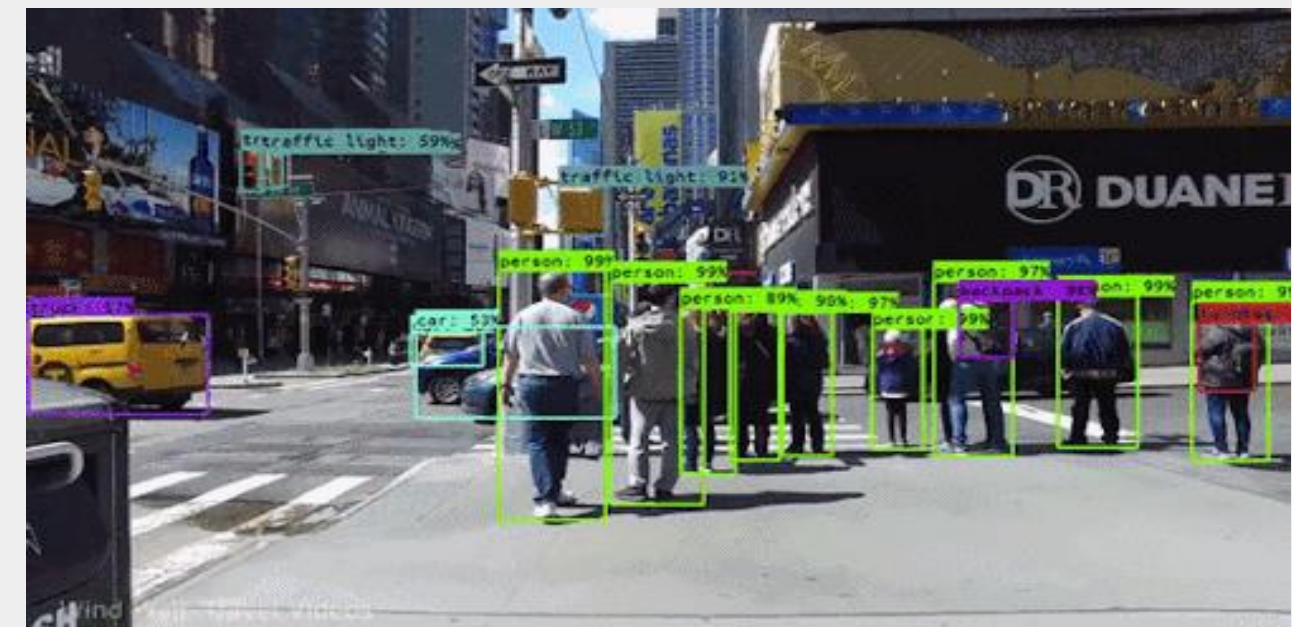
1. 이미지 분류

- 이미지가 주어졌을 때 그에 대한 라벨을 예측
- 2015년 이후로 이미지를 분류하는 ImageNet 대회에서 인간의 능력(95%)보다 더 뛰어난 성능을 보이는 딥러닝 모델이 발전(97%)



2. 객체 탐지

- 이미지 및 비디오 속에 포함돼 있는 물체에 대해 해당 물체가 어떤 물체인지를 분류하는 문제와 위치를 찾아내는 문제
- 자율주행 자동차, CCTV 등에 도입



DEEP LEARNING

딥러닝 배경

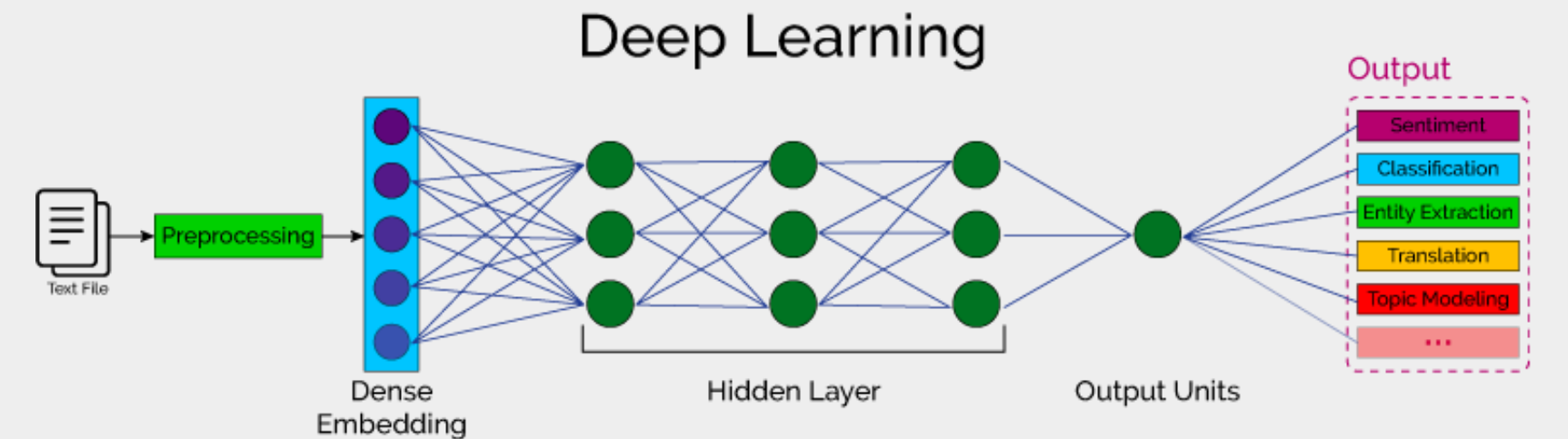
딥러닝 응용 task

3. 텍스트 분야

- 기계 번역, 문장 분류, 질의 응답 시스템, 개체명 인식 등
- RNN 모델의 한계
- 2017년 Transformer Model 연구의 시작으로 인간의 성능을 넘어서는 Language Model 개발

4. GAN

- 데이터를 예측하는 것을 넘어 데이터를 직접 생성하는 모델
- 인간의 눈으로 구분하지 못할 정도의 고품질의 데이터를 생성
ex) Deepfake, Style Transfer



GPU 사용 방법

CUDA(Computed Unified Device Architecture)

GPU에서 병렬 처리를 수행하는 알고리즘을 각종 프로그래밍 언어에 사용할 수 있도록 해주는 GPGPU 기술
파이토치, 텐서플로우 등 대다수의 딥러닝 프레임 워크에서 GPU를 사용하려면 CUDA를 설치해야 함!

※ GPGPU(General-Purpose computing on Graphics Processing Units)

CuDNN(nvidia CUDA Deep Neural Network Library)

딥러닝 모델을 위한 GPU 가속화 라이브러리의 기초 요소와 같은 일반적인 루틴을 빠르게 이행할 수 있도록 해주는 라이브러리
파이토치, 텐서플로우 모두 지원하며 반드시 CUDA와 함께 설치!



GPU 사용 방법

CUDA 사용법 in Pytorch

torch module을 GPU를 이용해 계산할 수 있는지 파악하고 device를 할당하는 코드

torch.cuda.is_available() : GPU를 이용해 계산할 수 있는지 파악

torch.device('cuda') : device에 GPU를 할당

torch.device('cuda' if torch.cuda.is_available() else 'cpu') : GPU가 사용 가능하면 cuda, 그렇지 않으면 CPU를 device에 할당

```
In [1]: import torch

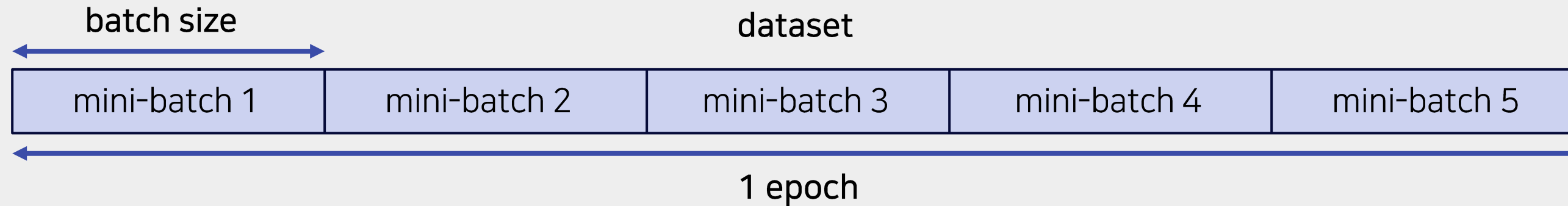
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

print(f'사용하는 디바이스: {device}')

tensor = torch.tensor([1, 2, 3]).to(device)
```

사용하는 디바이스: cuda

딥러닝 학습의 전체적인 과정



BATCH SIZE : 파라미터를 업데이트할 때 계산되는 데이터의 개수

MINI-BATCH : 데이터를 batch size 씩 나눈 1개의 부분

ITERATION : 한 개의 mini-batch를 이용해 학습하는 횟수

EPOCH : 전체 데이터를 이용해 학습을 진행한 횟수

batch size = 32
epoch = 10
(사용자 정의 hyperparameter)

(전체 데이터가 640개일 경우)



32개 데이터로 1개의 mini-batch 구성
1 epoch당 20회의 iteration
총 200번 반복해서 학습 진행

Dataset

ImageNet



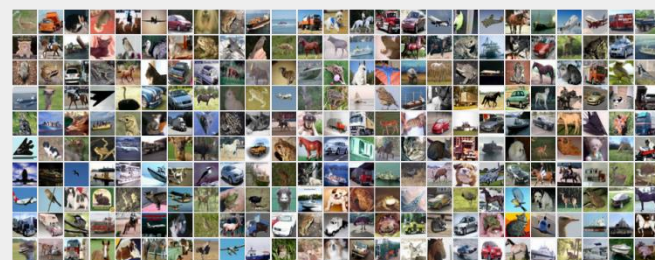
- 세상의 모든 객체를 인식하고 overfitting 문제를 극복하고자 제안
- 22만개의 카테고리 와 1400만 장의 이미지가 모두 주석으로 표기
- ILSVRC라는 이미지 인식 경진대회로 컴퓨터 비전 분야가 매우 큰 발전

MNIST



- 0부터 9까지의 숫자들로 이루어진 손글씨 데이터
- 60,000개의 Training 이미지와 10,000개의 Test 이미지 (28X28픽셀)
- Yann Lecun 교수가 고안한 것으로 784 피처의 1-D numpy 배열

CIFAR10



- 10개의 서로 다른 클래스에 각각 6,000개의 32X32 컬러 이미지
- 비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 트럭

torchvision.datasets

Pytorch에서 연구용으로 자주 이용하는 데이터
모든 데이터셋은 torch.utils.data.Dataset의 하위 클래스
[torch.utils.data.DataLoader](#)를 통해 데이터 불러오기 가능

```
In [4]: from torchvision import datasets

train_dataset = datasets.MNIST(root = '../data/MNIST', # 데이터가 저장될 장소 지정
                               train = True, # 학습용 데이터 여부
                               download = True, # root에 없을 경우 다운로드
                               transform = transforms.ToTensor()) # 데이터 전처리

test_dataset = dataset.MNIST(root = '../data/MNIST',
                              train = False,
                              download = True,
                              transform = transforms.ToTensor())

train_loader = torch.utils.data.DataLoader(dataset = train_dataset, # 할당할 데이터 셋
                                           batch_size = BATCH_SIZE, # batch size 지정
                                           shuffle = False) # 데이터의 순서를 섞고자 할 때

test_loader = torch.utils.data.DataLoader(dataset = test_dataset,
                                           batch_size = BATCH_SIZE,
                                           shuffle = False)
```

TENSOR

텐서

텐서 (Tensor)

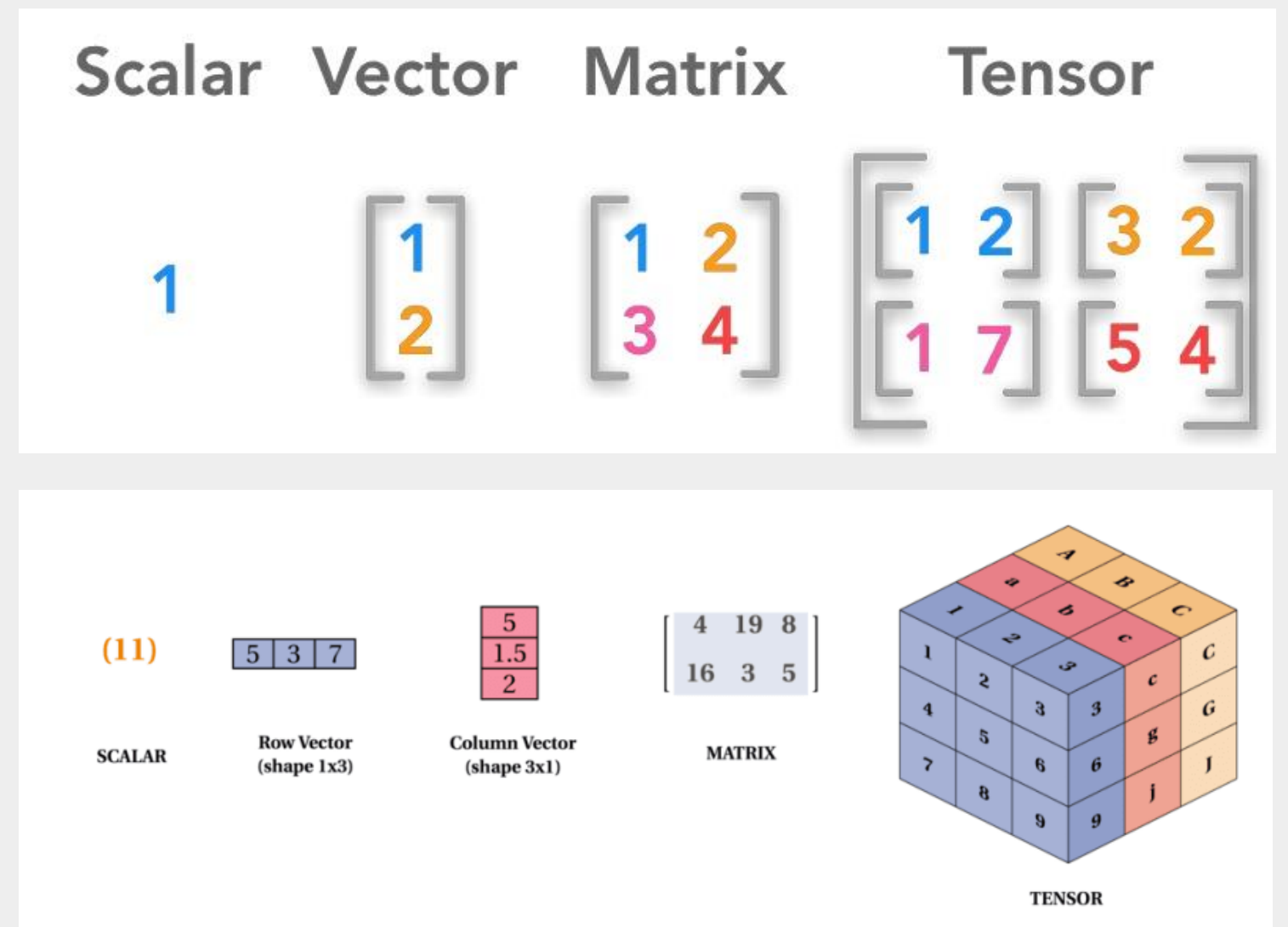
데이터를 표현하는 단위
NumPy의 배열과 비슷한 다차원 배열

스칼라(Scalar) : 하나의 값을 표현할 때 1개의 수치로 표현 (상수)

벡터(Vector) : 하나의 값을 표현할 때 2개 이상의 수치로 표현 (1차원)

행렬(Matrix) : 2개 이상의 벡터 값을 통합해 구성된 값 (2차원)

텐서(Tensor) : 3차원 이상의 배열



TENSOR

텐서

텐서의 종류

torch.tensor : tensor를 생성하는 함수

자료형	CPU 텐서	GPU 텐서
32비트 부동소수점	torch.FloatTensor	torch.cuda.FloatTensor
64비트 부동소수점	torch.DoubleTensor	torch.cuda.DoubleTensor
16비트 부동소수점	torch.HalfTensor	torch.cuda.HalfTensor
8비트 정수 (부호 없음)	torch.ByteTensor	torch.cuda.ByteTensor
8비트 정수 (부호 있음)	torch.CharTensor	torch.cuda.CharTensor
16비트 정수 (부호 있음)	torch.ShortTensor	torch.cuda.ShortTensor
32비트 정수 (부호 있음)	torch.IntTensor	torch.cuda.IntTensor
64비트 정수 (부호 있음)	torch.LongTensor	torch.cuda.LongTensor



TENSOR

텐서

텐서 메소드

Pytorch에서 tensor를 다루는 방법이 numpy와 유사

1	2	3
4	5	6
7	8	9
10	11	12

tensor 생성 및 형상 확인

```
t = torch.tensor([[1, 2, 3], [4, 5, 6],  
                  [7, 8, 9], [10, 11, 12]])
```

```
print(t.dim()) # t의 차원 수 반환  
print(t.shape) # tensor의 형상(shape) 반환  
print(t.size()) # .shape와 동일
```

```
2  
torch.Size([4, 3])  
torch.Size([4, 3])
```

tensor 인덱싱 & 슬라이싱

```
print(t[0, 1]) # 인덱싱  
print(t[:2, 1:]) # 슬라이싱
```

```
tensor(2)  
tensor([[2, 3],  
        [5, 6]])
```

TENSOR

텐서

텐서 사칙연산

	$+, -, *, /$	내장 메소드
덧셈	<code>torch.tensor + torch.tensor</code>	<code>torch.add(tensor1, tensor2)</code>
뺄셈	<code>torch.tensor - torch.tensor</code>	<code>torch.sub(tensor1, tensor2)</code>
곱셈	<code>torch.tensor * torch.tensor</code>	<code>torch.mul(tensor1, tensor2)</code>
나눗셈	<code>torch.tensor / torch.tensor</code>	<code>torch.div(tensor1, tensor2)</code>

사칙연산은 각 요소 별 (element-wise)로 계산, 행렬 곱 연산은 `torch.matmul(tensor1, tensor2)`로 계산

Broadcasting

서로 다른 크기의 텐서를 연산할 때 자동적으로 사이즈를 맞춰줌

```
# scalar(1, ) + vector(1, 2)
torch.tensor([3]) + torch.tensor([[1, 2]])

tensor([[4, 5]])
```

TENSOR

텐서

평균, 합계

1	2	3
4	5	6
7	8	9
10	11	12

최대, 최소

Mean

```
t = torch.FloatTensor([[1, 2, 3], [4, 5, 6],  
                        [7, 8, 9], [10, 11, 12]])  
  
print(t.mean())           # 전체 원소에 대한 평균  
print(t.mean(dim=0))      # 열에 대한 평균  
print(t.mean(dim=-1))     # -1(1)차원에 대한 평균  
  
tensor(6.5000)  
tensor([5.5000, 6.5000, 7.5000])  
tensor([ 2.,  5.,  8., 11.])
```

Max, Argmax

```
print(t.max())           # 전체 원소에 대한 최댓값  
print(t.max(dim=0)[0])   # 열을 기준으로 최댓값 구함  
print(t.max(dim=0)[1])   # 열을 기준으로 최댓값의 위치를 구함  
print(t.argmax())        # 전체 원소 중 최댓값의 위치를 구함  
  
tensor(12.)  
tensor([10., 11., 12.])  
tensor([3, 3, 3])  
tensor(11)
```

Sum

```
print(t.sum())           # 전체 원소에 대한 합계  
print(t.sum(dim=0))      # 열에 대한 합계  
print(t.sum(dim=-1))     # -1(1)차원에 대한 합계  
  
tensor(78.)  
tensor([22., 26., 30.])  
tensor([ 6., 15., 24., 33.])
```

Min, Argmin

```
print(t.min())           # 전체 원소에 대한 최솟값  
print(t.min(dim=0)[0])   # 열을 기준으로 최솟값 구함  
print(t.min(dim=0)[1])   # 열을 기준으로 최솟값의 위치를 구함  
print(t.argmin())        # 전체 원소 중 최솟값의 위치를 구함  
  
tensor(1.)  
tensor([1., 2., 3.])  
tensor([0, 0, 0])  
tensor(0)
```

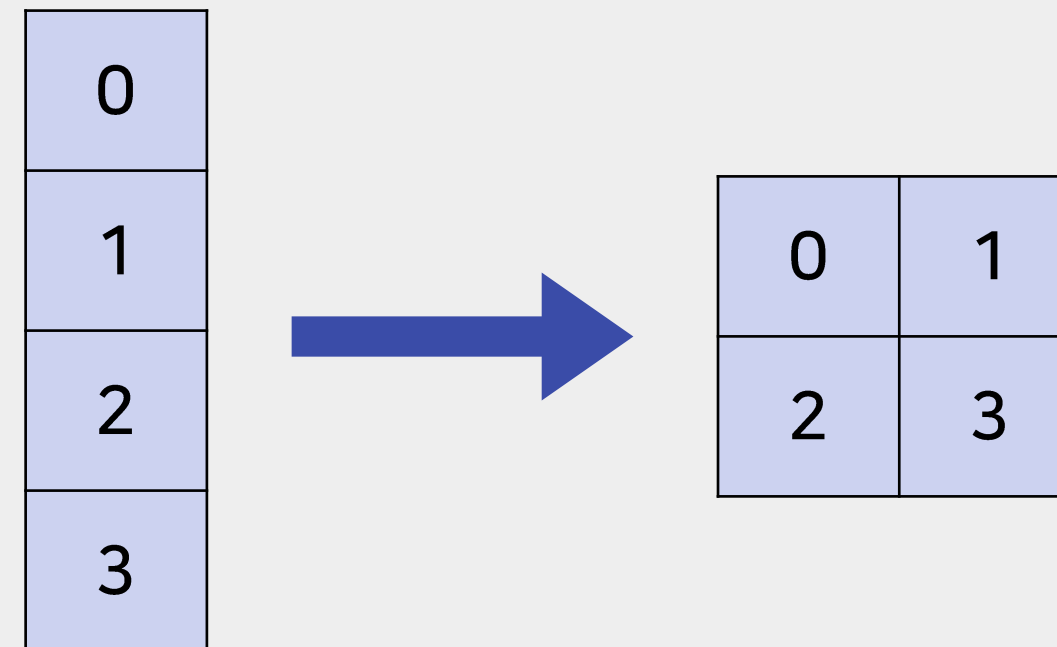

TENSOR

텐서

View (Reshape)

Shape를 바꿔줌

```
t = torch.FloatTensor([[0],[1],[2],[3]])  
t.view(-1, 2) # -1을 인자로 줄 시, 나머지에 맞춰 shape 변환  
tensor([[0., 1.],  
        [2., 3.]])
```



Squeeze, Unsqueeze

```
t = t.squeeze()  
print(t)  
tensor([0., 1., 2., 3.])
```

기존에 (4,1) 이었던 t에서 1을 제거하여 벡터로 변환

```
t.unsqueeze(1)  
tensor([[0.],  
        [1.],  
        [2.],  
        [3.]])
```

벡터였던 t를 다시 (4,1)의 텐서로 변환

TENSOR

텐서

Concatenate

두 개 이상의 텐서를 특정 축(axis)에 따라 결합

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

x		y	
1	2	5	6
3	4	7	8

dim = 0 방향으로 늘어남

```
torch.cat([x, y], dim=0)  
  
tensor([[1, 2],  
        [3, 4],  
        [5, 6],  
        [7, 8]])
```

1	2
3	4
5	6
7	8

dim = 1 방향으로 늘어남

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

1	2	5	6
3	4	7	8

TENSOR

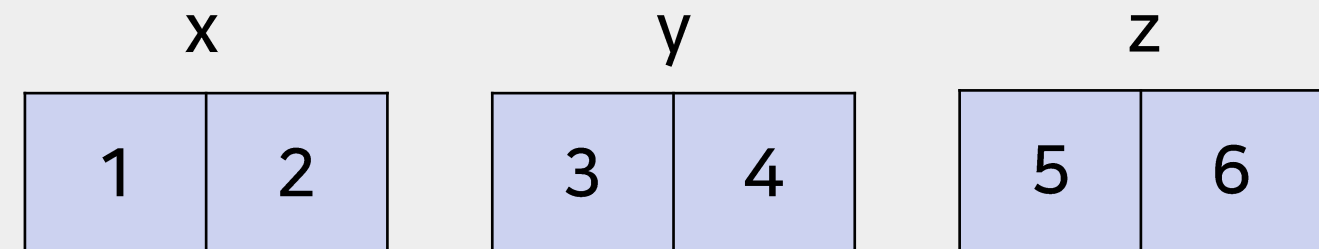
텐서

Stack

Concatenate를 좀 더 편하게 해줌

```
x = torch.tensor([1, 2])  
y = torch.tensor([3, 4])  
z = torch.tensor([5, 6])  
  
torch.stack([x, y, z])           # dim = 0, 0차원을 기준으로 stack  
torch.stack([x, y, z], dim=1)    # dim = 1, 1차원을 기준으로 stack  
  
tensor([[1, 3, 5],  
        [2, 4, 6]])
```

```
torch.stack([x, y, z])  
= torch.cat([x.unsqueeze(0), y.unsqueeze(0), z.unsqueeze(0), dim=0)
```



dim = 0

1	2
3	4
5	6

dim = 1

1	3	5
2	4	6

TENSOR

텐서

Type Casting

Type을 바꿔줌

```
torch.LongTensor([1, 2, 3, 4]).float()           # FloatTensor로 변환
torch.FloatTensor([1., 2., 3., 4.]).int()          # IntTensor로 변환
torch.ByteTensor([True, False, False, True]).long() # LongTensor로 변환
torch.ByteTensor([True, False, False, True]).float() # FloatTensor로 변환

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

In-place operation

변수에 할당하지 않아도 바로 저장

```
x.mul(2) # 실행 결과가 변수에 저장되지 않음
x.mul_(2) # _ 추가 시, 메모리에 새로 선언하지 않고 정답값에 바로 음

tensor([2, 4])
```

TENSOR

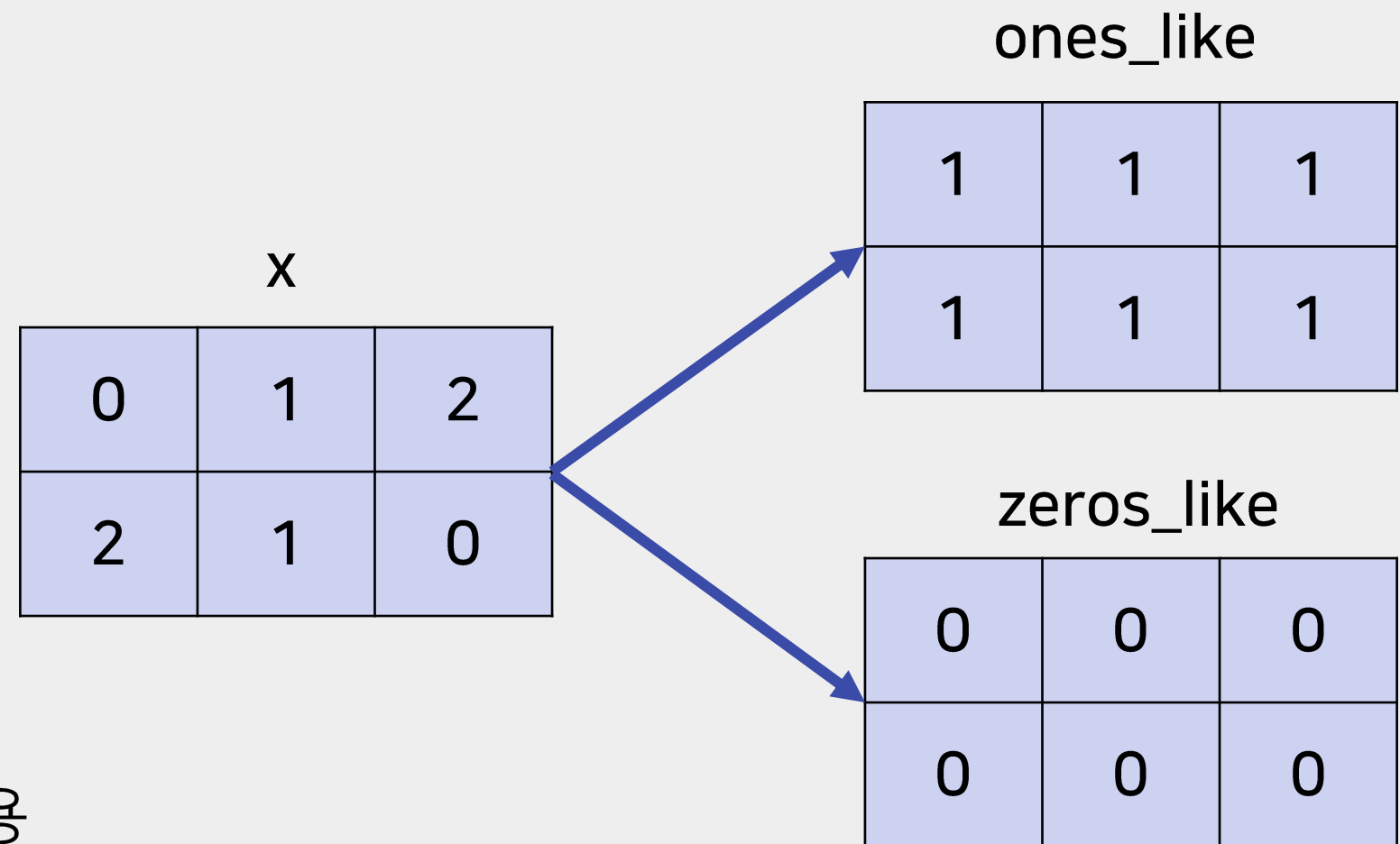
텐서

Ones & Zeros

입력값의 shape와 동일하게 0 or 1로 가득찬 tensor 생성

```
x = torch.tensor([[0, 1, 2],[2, 1, 0]])  
  
torch.ones_like(x)  
torch.zeros_like(x)  
  
tensor([[0, 0, 0],  
        [0, 0, 0]])
```

- device가 다른 텐서끼리는 서로 연산 불가
- device를 통일해서 수행해야 error가 나지 않기 때문에 사용



과제

1. 조 별로 조 이름과 조장, 발표 순서 정하기
2. 스터디 보고서 작성해서 홈페이지에 업로드하기



2024 D&A

THANK YOU

2024.03.14

Deep Session 1차시

