D&A

Deep Session 1차시

ORIENTATION.

2022 / 03 / 10

D&A 학회장 권유진

CONTENTS.

01. ORIENTATION

세미나

ZOOM을 이용한 실시간 강의 (카메라 ON)
매주 목요일 06:00PM ~ 07:40PM ※ 시험 기간 2주일 제외
실시간 수업은 녹화해서 Youtube에 업로드
45분 이론 수업, 10분 휴식, 45분 실습 수업

스터디

매 주 한 명씩 번갈아 가며 세미나 내용 Review
과제 수행 후 발표 및 어려운 점 상의
활동사진, 활동내용, 참여인원을 포함한 스터디 보고서 작성

과제

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

01. ORIENTATION

청강기간

3월 10일 ~ 3월 31일 (3주간)
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

초청강연회

5월 16일(화) 06:00PM 예정
교수님 초청강연회 진행 예정

01. ORIENTATION

구글 드라이브

세션 세미나 자료 다운로드 & 매 주 과제 수행 후 업로드
https://drive.google.com/drive/folders/1HFYzZv3UxKWEJ9wA5vC6YQfafL5LS9TE?usp=sharing

카카오톡 채널(플러스 친구)

각종 질문, 건의 사항 문의
https://pf.kakao.com/_zTEGb

인스타그램, 페이스북

학회 및 세션 관련 공지, 매주 세미나 요약
인스타그램: https://www.instagram.com/kmu_dna/
페이스북: https://www.facebook.com/kookmin.bigdata.dna2013/

유튜브

실시간 강의 녹화 (매주 링크 제공)
https://www.youtube.com/channel/UCaypwX7F1SKXM0X7J6Uzs7A

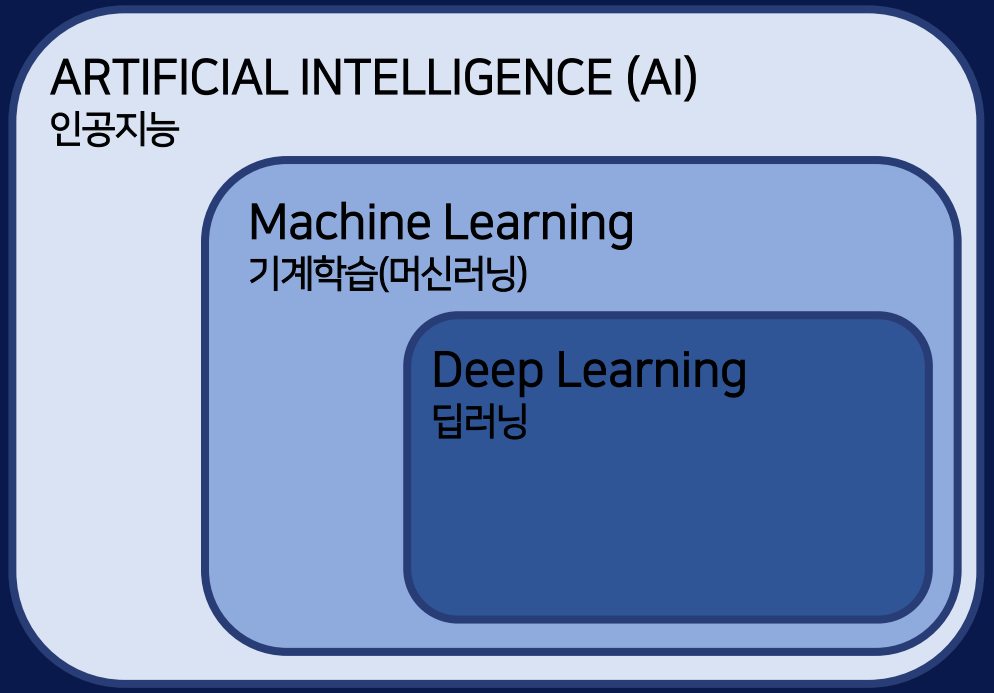
01. ORIENTATION

월1				운영진
김건우	김은욱	이성규	황태균	권유진
월2				
김진호	신예주	한병규		윤경서
화1				
고민성	이현지	이효빈	최여진	나요셉
화2				
이수현	이승준	이태범	주민지	김정하
수1				
김나은	문찬우	서동혁	한윤지	이수빈
수2				
김종윤	장성현	조문주	서하은	이예진
금1				
김진비	박선혜	이승학	최민석	이경욱

01. ORIENTATION

차시	날짜	내용	발표자
1	03/10	딥러닝 툴 설치, 클래스, 텐서	권유진
2	03/17	MLP, 인공신경망, Back Propagation	나요셉
3	03/24	딥러닝 관련 학습 기법들	이수빈
4	03/31	CNN 기초	이예진
5	04/07	CNN 심화1(LeNet, AlexNet, VGG)	윤경서
6	04/14	CNN 심화2(GoogLeNet, ResNet, 전이학습)	나요셉
7	05/05	RNN 기초(Tokenization, Representation)	권유진
8	05/12	RNN 심화(RNN, LSTM, GRU, Seq2Seq)	김정하
9	05/19	Attention, Transformer, BERT	이경욱
10	05/26	Other Topics(AE, GAN, 강화학습)	김정하

02. Deep Learning



인공지능: 인간의 학습능력, 추론능력, 지각능력, 그 외에 인공적으로 구현한 컴퓨터 프로그램 또는 이를 포함한 컴퓨터 시스템

머신러닝: 경험을 통해 자동으로 개선하는 컴퓨터 알고리즘의 연구

딥러닝: 여러 비선형 변환기법의 조합을 통해 높은 수준의 추상화를 시도하는 기계 학습 알고리즘의 집합

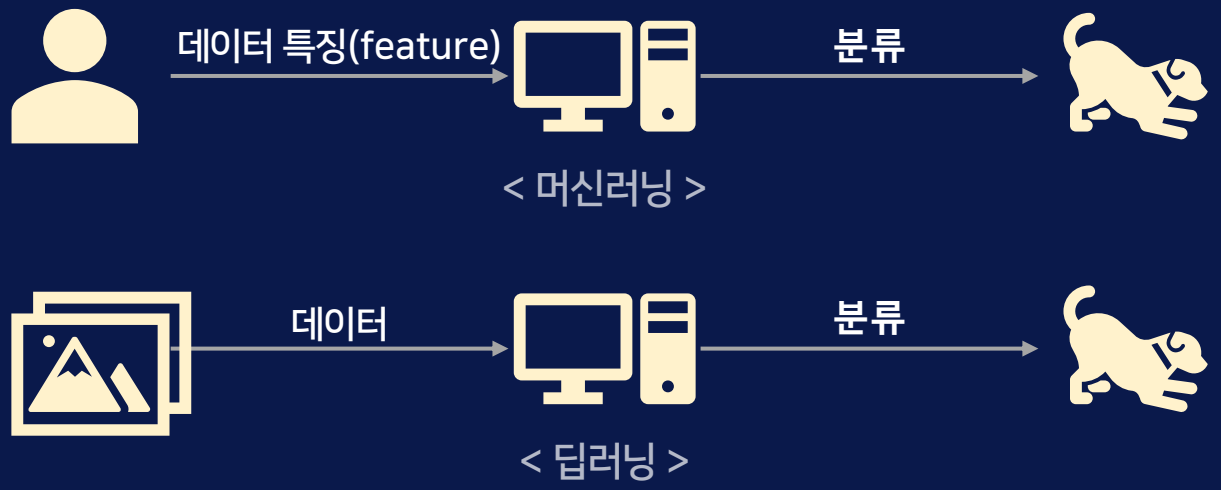
02. Deep Learning

전통적인 머신러닝 vs 딥러닝

전통적인 머신러닝은 사람이 직접 특징(feature)을 추출하여 정형 데이터 형태로 바꾸고 모델링

딥러닝은 사람의 관여 없이 기계가 알아서 특징(feature)을 추출하여 모델링

> 정형 데이터 뿐만 아니라 비정형 데이터까지 사람이 분류하는 것보다 정확하게 예측



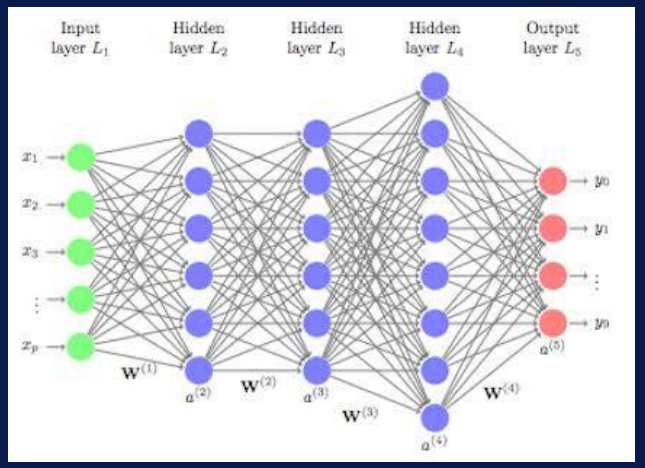
02. Deep Learning

딥러닝 정의

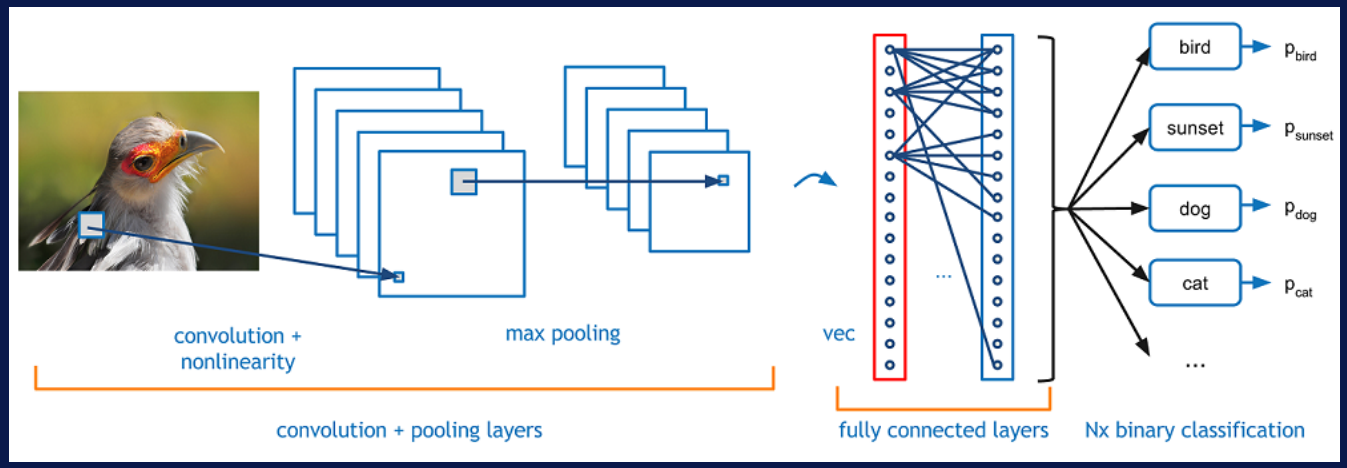
2개 이상의 Hidden Layer을 지닌 다층 신경망

딥러닝 기본 구조

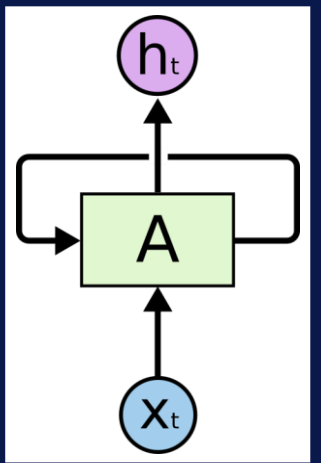
여러 개의 레이어를 가지고 있는 MLP(Multi-Layer Perceptron)
이미지 관련 분야에서 많이 사용되는 CNN(Convolutional Neural Network)
텍스트와 같은 시계열 분야에서 많이 사용되는 RNN(Recurrent Neural Network)



MLP



CNN



RNN

02. Deep Learning

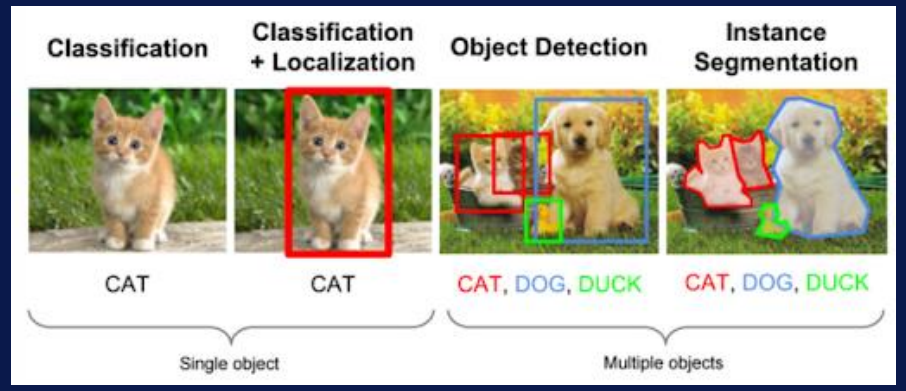
딥러닝 응용 task

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%)

객체 탐지

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



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

02. Deep Learning

딥러닝 응용 task

GAN

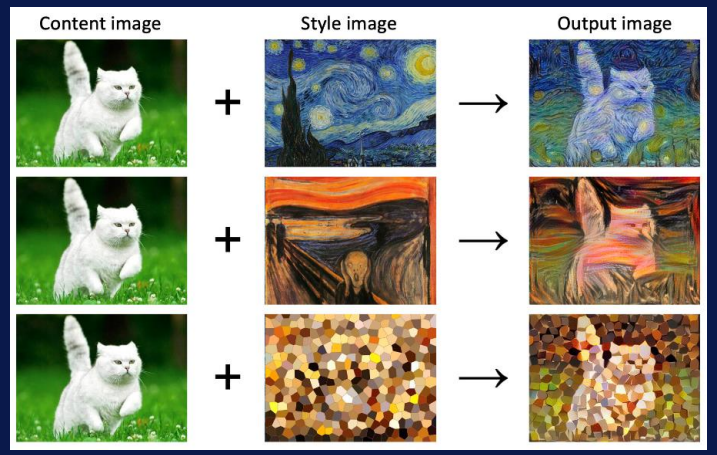
- 데이터를 예측하는 것을 넘어 데이터를 직접 생성하는 모델
- 인간의 눈으로 구분하지 못할 정도의 고품질의 데이터를 생성

Deepfake

- GAN을 활용하여 기존의 사진이나 영상을 원본이 되는 사진이나 영상에 겹쳐서 만들어내는 인간 이미지 합성 기술
- 포르노 동영상, 사기 등과 같이 범죄에 악용될 여지가 많음

Style Transfer

- 사진을 고흐풍으로 바꿔준다거나, 여름 풍경을 겨울 풍경으로 바꿔주는 등의 기술 포토샵을 활용할 필요 없이 GAN을 활용하여 사진의 스타일을 바꿔주는 기술



심층 강화학습

- 강화학습과 딥러닝을 결합한 심층 강화학습
- 현재 상태에서 어떤 행동을 취해야 먼 미래에 보상이 최대가 되는지 학습

02. Deep Learning

딥러닝 발전 계기

딥러닝의 단점

과적합, Gradient Vanishing

⇒ ① 학습 데이터 수 증가
⇒ ② 알고리즘의 발전

딥러닝의 연산

단순한 연산 수행 ex) 행렬곱
계층의 깊이가 깊어질수록 파라미터 수 급격히 증가

∴ 다른 알고리즘에 비해 학습 시간 오래 걸림!

⇒ ③ Graphics Processing Unit(GPU)를 신경망 연산에 사용해 학습 시간 단축

처리 장치	CPU	GPU
코어 수	10개 미만	수백~수천 개
계산	복잡한 계산 수행	단순한 계산 수행

02. Deep Learning

딥러닝 툴

CUDA(Compute Umified Device Architecture)

GPU에서 병렬 처리를 수행하는 알고리즘을 각종 프로그래밍 언어에 사용할 수 있도록 해주는 GPGPU(General-Purpose computing on Graphics Processing Units) 기술

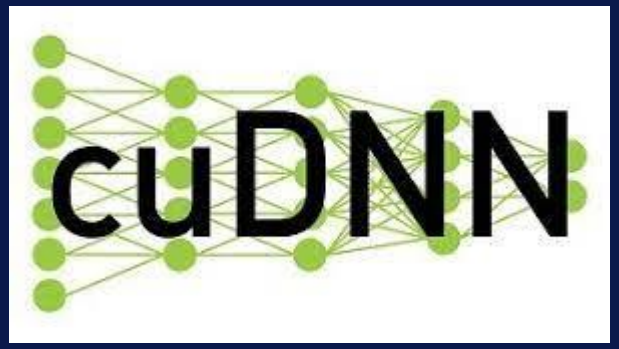
⇒ 파이썬에서 GPU를 인식할 수 있도록 하기 위해 사용

파이토치, 텐서플로우 등 대다수의 딥러닝 프레임 워크에서 GPU를 사용하려면 CUDA를 설치해야 함!

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!



02. Deep Learning

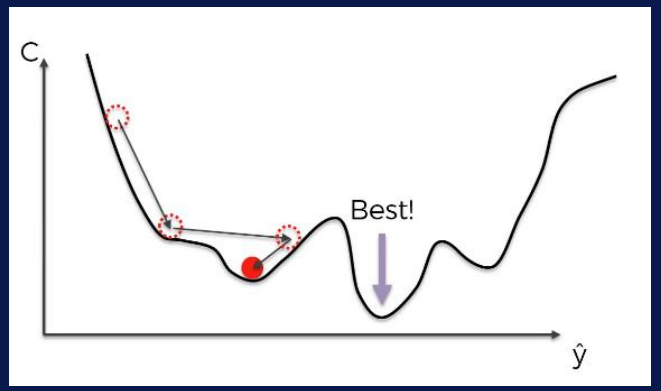
```python
import torch

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

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('cpu): device에 CPU를 할당

> device(type='cuda')
> device(type='cpu')



Batch Gradient Descent: 한 번의 학습에 모든 학습 데이터를 사용
→ 최적값으로 학습하기 좋지만 계산 시간이 오래 걸림

Stochastic Gradient Descent: 한 번의 학습에 한 개의 학습 데이터를 사용
→ sample의 편차로 인해 매 Update 시 계산되는 Gradient의 크기 편차가 심함

Mini-Batch Gradient Descent: 모든 학습 데이터를 미니배치로 쪼개 한 번의 학습에 한 개의 미니배치 사용

02. Deep Learning

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

32개의 데이터로

1개의 mini batch 구성

1 epoch당 20회의 iteration

총 200번 반복해서 학습 진행

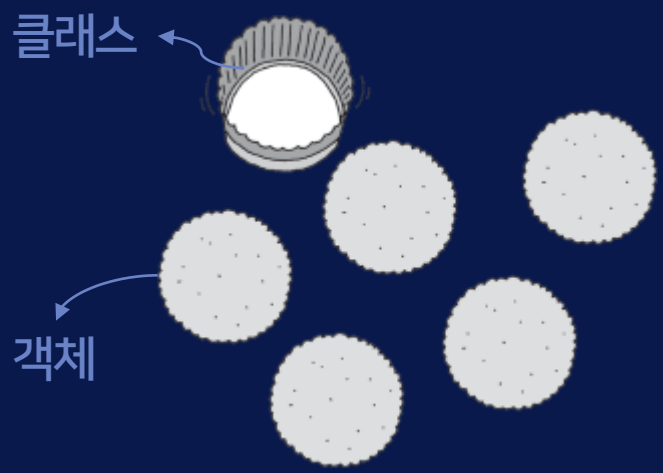
03. 클래스

클래스

똑같은 무엇인가를 만들어내는 설계 도면 (≒ 과자 틀)

객체

클래스로 만든 피조물 (≒ 과자)
객체마다 고유한 성격을 지님 >>> 어느 한 객체에 영향을 주어도 다른 객체에는 영향이 없음
(어느 과자 하나를 베어 먹어도 다른 과자에는 영향 없음)



인스턴스

클래스가 지시한대로 만든 값 (≒ 초코하임, 쿠크다스 등)
이 말은 특정 개체가 어떤 클래스의 객체인지 관계 위주로 설명할 때 사용

객체와 인스턴스의 차이점!!!
우측 예시에서 cookie1은 객체, cookie1은 Cookie의 인스턴스라고 표현

```
class Cookie:
  pass

cookie1 = Cookie()
cookie2 = Cookie()
```

03. 클래스

메소드

클래스 안에 구현된 함수
메소드의 첫 번째 매개변수(parameter)는 특별한 의미
>>> 첫 번째 매개변수에는 호출한 객체가 자동으로 전달

객체 변수

객체에 생성되는 객체만의 변수
다른 객체들에 영향을 받지 않음 – 서로 다른 메모리 주소 가짐.

생성자

객체가 생성되는 시점에 자동으로 호출되는 메소드
메소드 이름을 __init__이라고 설정하면 생성자로 인식

```
class Fourcal:
  def setdata(self, first, second):
    self.first = first
    self.second = second
  def add(self):
    result = self.first + self.second
    return result
```

객체변수 생성

메소드

```
class Fourcal:
  def __init__(self, first, second):
    self.first = first
    self.second = second
  def add(self):
    result = self.first + self.second
    return result
```

생성자

03. 클래스

클래스의 상속

어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것
기존 클래스를 변경하지 않고 기능을 추가하거나 기존 기능을 변경할 때 사용
>>>기존 클래스가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황에 사용

super()

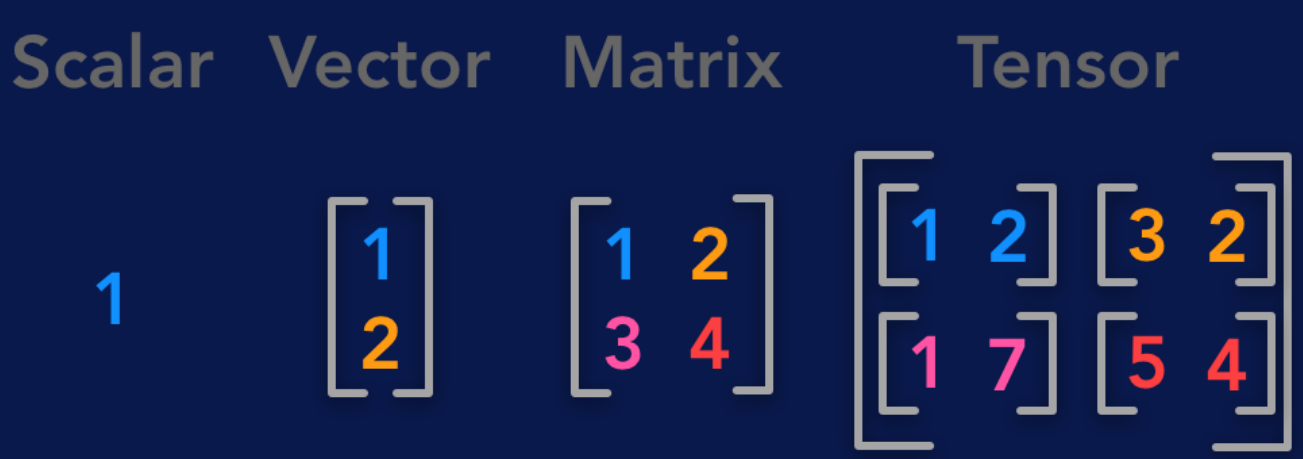
super()은 다른 클래스의 속성 및 메소드를 자동으로 불러와 해당 클래스에서도 사용 가능하게 해주는 함수
super().메소드명을 통해 부모 클래스의 메소드 사용 가능
__init__()메소드는 부모와 자식 클래스 모두 보유
⇒ super().__init__()을 입력하지 않으면 자식 클래스의 .__init__()에 의해 덮어쓰기 됨
super(파생클래스, self).__init__()로 현재 클래스가 어떤 클래스인지 명확하게 표시

```
class Parent: 부모 클래스
  def __init__(self, p1, p2):
    self.p1 = p1
    self.p2 = p2
  def act(self):
    pass
          상속
class Child(Parent): 자식 클래스
  def __init__(self, p1, p2, c1):
    super(Child, self).__init__(p1,p2)
    self.c1 = c1 속성 및 메소드 불러오기
  def parent_act(self):
    super().act()
    부모 클래스의 메소드 불러오기
```

04. 텐서

텐서(Tensor)

데이터를 표현하는 단위
Numpy의 배열과 비슷한 다차원 배열



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

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

04. 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.DoubleTensor	torch.cuda.DoubleTensor
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

04. 텐서

텐서 메소드

Pytorch에서 tensor을 다루는 방법이 numpy와 유사

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

```
t = torch.tensor([[1,2,3],[4,5,6],
                  [7,8,9],[10,11,12]])
t.dim() # t의 차원 수 반환 → 2
t.shape # tensor의 형상(shape) 반환 → (4,3)
t.size() # .shape와 동일 → (4,3)
```

```
t[0,1] # 인덱싱 → 2
t[0:2, 1:] # 슬라이싱 → [[2,3],[5,6]]
```

04. 텐서

텐서 사칙연산

	+, -, *, /	내장 메서드
덧셈	torch.tensor( ) + torch.tensor( )	torch.add(tensor1, tensor2)
뺄셈	torch.tensor( ) - torch.tensor( )	torch.sub(tensor1, tensor2)
곱셈	torch.tensor( ) * torch.tensor( )	torch.mul(tensor1, tensor2)
나눗셈	torch.tensor( ) / torch.tensor( )	torch.div(tensor1, tensor2)

사칙연산은 각 요소 별(element-wise)로 계산, 행렬 곱 연산은 **torch.matmul(tensor1, tensor2)**로 계산
>>> vector의 경우, **torch.dot(vector1, vector2)**로 벡터 내적 계산 가능(torch.matmul도 가능)
>>> matrix의 경우, **torch.mm(matrix1, matrix2)**로 행렬 곱 연산 가능(torch.matmul도 가능)

Broadcasting

서로 다른 크기의 텐서를 연산할 때 자동적으로 사이즈를 맞춰줌

ex) scalar(1,) + vector(1, 2)

[3] + [[1, 2]]

= [[3, 3]] + [[1, 2]] = [[4, 5]]

04. 텐서

평균, 합계

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

```
t = torch.FloatTensor([[1,2,3],[4,5,6],
                       [7,8,9],[10,11,12]])
t.mean() # 전체 요소의 평균
t.mean(dim=0) # 0차원을 소각시키는 기준으로 평균
t.mean(dim=-1) # -1(1)차원을 기준으로 평균
```

→ 6.5
→ [5.5,6.5,7.5]
→ [2,5,8,11]

```
t.sum() # 전체 요소의 합계
t.sum(dim=0) # 0차원을 소각시키는 기준으로 합계
t.sum(dim=-1) # -1(1)차원을 기준으로 합계
```

최대, 최소

```
t.max() # 최대값 반환
t.max(dim=0) # 0차원을 소각시키는 기준으로 최대값 구함
t.argmax() # 최대값의 index

t.min() # 최소값 반환
t.min(dim=0) # 0차원을 소각시키는 기준으로 최소값 구함
t.argmin() # 최소값의 index
```

→ 12
→ values = [10,11,12], indices = [3,3,3]
→ 11

→ 1
→ values = [1,2,3], indices = [0,0,0]
→ 0

04. 텐서

View(Reshape)

shape을 바꿔줌

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

→ [[0,1],[2,3]]

Squeeze, Unsqueeze

Squeeze: 쥐어 짜다 (1차원으로 바꿔줌)
Unsqueeze: squeeze를 반대로 수행

```
t = t.squeeze() # 1차원으로 만듦
t.unsqueeze(1) # dim=1, dimension 1에 1 입력해 view 수행 # (4,) -> (4,1)
```

→ [0,1,2,3]

→ [[0],[1],[2],[3]]

04. 텐서

Concatenate

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

torch.cat([x,y], dim=0) # dim 0이 늘어남
torch.cat([x,y], dim=1) # dim 1이 늘어남
```

1	2
3	4
5	6
7	8

1	2	5	6
3	4	7	8

1	2
3	4

x

5	6
7	8

y

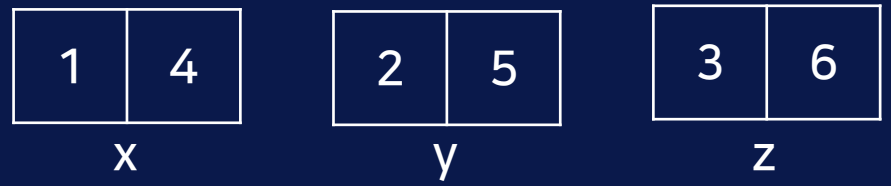
04. 텐서

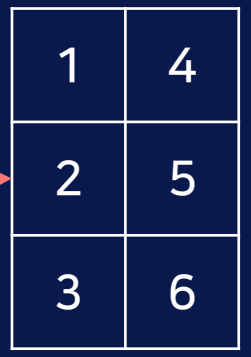
Stack

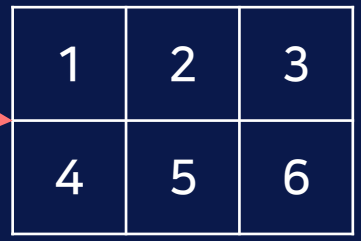
Concatenate을 조금 더 편리하게 해줌

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

torch.stack([x,y,z]) # dim=0, dimension 0(행)으로 쌓는다.
torch.stack([x,y,z], dim=1) # dim=1, dimension 1(열)으로 쌓는다.
```







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

torch.cat([x.unsqueeze(1), y.unsqueeze(1), z.unsqueeze(1)], dim=1)

04. 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
```

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 넣음
```

04. 텐서

Ones & Zeros

입력값의 shape과 동일하게 0/1로 가득찬 tensor 생성

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

torch.ones_like(x) # 똑같은 shape로 영행렬 만듦
        # device를 통일해서 수행해야 error 안나기 때문에 사용
torch.zeros_like(x)
```

0	1	2
2	1	0

x

1	1	1
1	1	1

0	0	0
0	0	0

05. DataSet

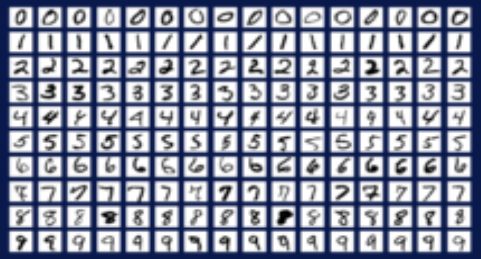
ImageNet

세상의 모든 객체를 인식하고 Overfitting 문제를 극복하고자 고안

22만개의 카테고리와 1400만 장의 이미지가 모두 주석으로 표시

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

그 외 데이터: https://pytorch.org/vision/0.8/datasets.html

05. DataSet

torchvision.datasets

pytorch에서 연구용으로 자주 이용하는 데이터
모든 데이터셋은 torch.utils.data.Dataset 의 하위 클래스
⇒ torch.utils.data.DataLoader을 통해 데이터 불러오기 가능
⇒ torch.multiprocessing을 사용해 여러 샘플을 병렬로 불러옴

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

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, # 할당할 데이터셋
                                           batch_size = BATCH_SIZE, # batch size 지정
                                           shuffle = True) # 데이터의 순서를 섞고자 할 때
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size = BATCH_SIZE,
                                          shuffle = False)
```

06. 과제

1. 조 별로 조이름과 조장, 스터디 시간, 발표 순서 정하기

2. 스터디 보고서 작성해서 구글 드라이브에 업로드하기

첨부자료 출처

02. Deep Learning

딥러닝_MLP 이미지 출처
: https://prathasaxena30.medium.com/multi-layer-perceptron-with-tensorflow-246b16d4e0dc
딥러닝_CNN 이미지 출처
:https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529
딥러닝_RNN 이미지 출처
:http://colah.github.io/posts/2015-08-Understanding-LSTMs/

03. 클래스

클래스_클래스 이미지 출처
: https://wikidocs.net/28

폰트

네이버 글꼴 모음 _ 나눔 스퀘어 사용
출처 : https://hangeul.naver.com/font

D&A

Deep Session 1차시 Orientation

Thank You.

2022 / 03 / 10

D&A 학회장 권유진