[Week 1]

chelseey · 5일 전

DL Euron

> ## 1. 머신러닝과 딥러닝

머신 러닝

컴퓨터가 데이터에서 패턴을 찾아 학습하고 예측 수행

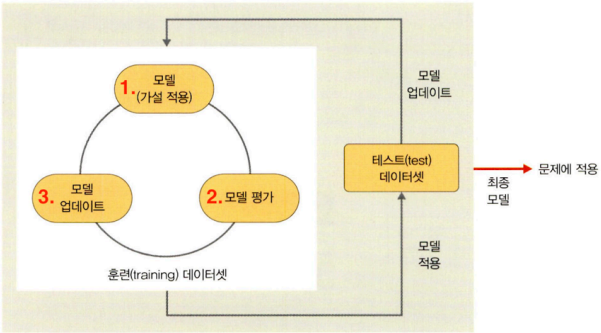
주요 구성 요소

• 데이터
학습 모델을 만드는데 사용
훈련(훈련+검증) 데이터셋 / 테스트 데이터셋으로 분류

검증 데이터셋
: 학습하는 과정에서 모델의 성능을 평가하기 위해 사용
모델의 과적합(overfitting)을 방지

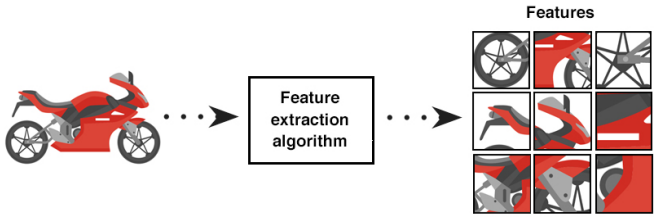
테스트 데이터셋
: 학습을 모두 마친 후, 최종적으로 모델의 성능을 객관적으로 평가하기 위해 사용

• 모델
학습 단계의 최종 결과물

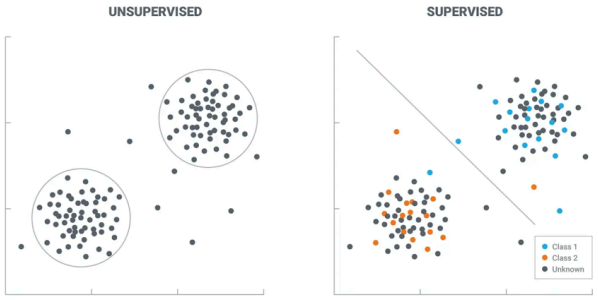


머신 러닝 학습 방법

특성(특징) 추출 : 데이터 → 벡터 변환



머신 러닝 알고리즘



지도 학습

: 정답을 알려주고 학습시키는 방법
데이터를 명확하게 분류 가능

비지도 학습

: 비슷한 데이터를 클러스터링하여 예측하는 학습 방법
데이터 간 거리를 기반으로 유사한 데이터끼리 범주화

강화 학습

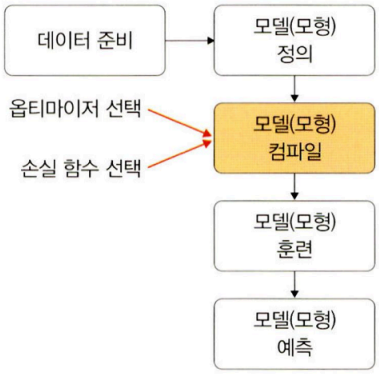
: 에이전트가 행동에 대한 보상을 받으며 학습

구분	유형	알고리즘
지도 학습	분류 (classification)	- k-최근접 이웃(k-Nearest Neighbor, KNN) - 서포트 벡터 머신(Support Vector Machine, SVM) - 결정 트리(decision tree) - 로지스틱 회귀(logistic regression)
지도 학습	회귀 (regression)	- 선형 회귀(linear regression)
비지도 학습	군집화 (clustering)	- k-평균 군집화(k-means clustering) - 밀도 기반 군집 분석(DBSCAN)
차원 축소	(dimensionality reduction)	- 주성분 분석(Principal Component Analysis, PCA)
강화 학습	(reinforcement learning)	- 마르코프 결정 과정(Markov Decision Process, MDP)

딥러닝

심층 신경망 이론을 기반으로 고안된 머신러닝

딥러닝 학습 과정



• 모델 정의 : 신경망을 생성
은닉층 수가 많을수록 성능 증가, but 과적합 가능성 증가

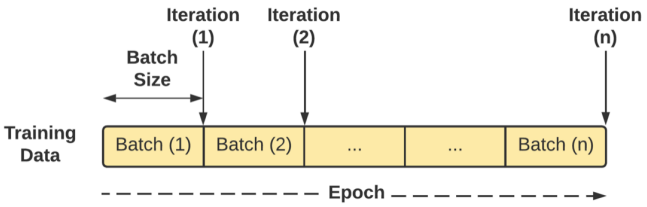
- 은닉층 : 입력층과 출력층 사이에 존재하는 층. 입력 데이터를 여러 단계에 걸쳐 변환, 추출하여 유용한 특징을 학습.

• 모델 컴파일 : 활성화 함수, 손실 함수, 옵티마이저 선택

- 활성화 함수 : 신경망에서 입력 값을 어떻게 변환해서 출력할지 결정. 비선형 변환을 적용함으로써 신경망이 복잡한 패턴을 학습하도록 함.
- 손실 함수 : 모델의 출력 값과 사용자가 원하는 출력 값의 차이.
- 옵티마이저 : 손실 함수를 줄이기 위해 모델 파라미터를 업데이트하는 알고리즘.

• 모델 훈련 : 한 번에 처리할 데이터양을 지정. batch와 epoch 선택

- batch : 학습에서 한 번에 처리하는 데이터 묶음
- epoch : 전체 훈련 데이터를 1번 모두 학습시키는 과정 = 1에포크

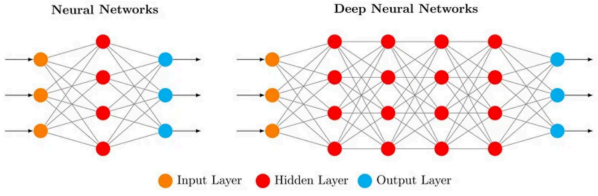


• 모델 예측 : 검증 데이터셋으로 실제 예측을 진행

딥러닝 구성 요소

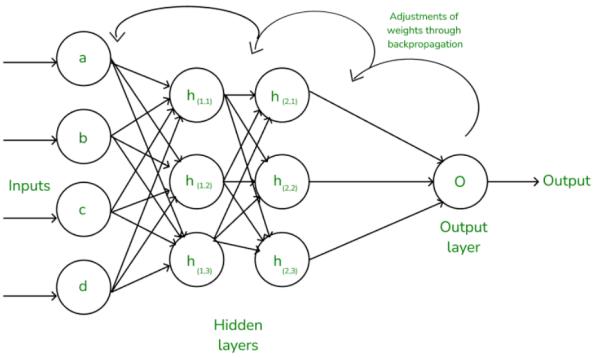
: 심층 신경망, 역전파

• 심층 신경망 (deep neural network)



은닉층(Hidden Layer)이 여러 개 있는 신경망

• 역전파 (Backpropagation)



출력층에서 계산된 오차(손실)를 입력층 방향으로 거슬러 올라가며 각 가중치(weight)와 편향(bias)이 얼마나 오차에 기여했는지를 계산해주는 알고리즘.
계산 결과를 바탕으로 가중치와 편향을 업데이트, 모델이 오차를 줄이는 방향으로 학습을 진행.
→ pytorch가 자동으로 처리해 줌

딥러닝 학습 알고리즘

: 지도 학습, 비지도 학습, 전이 학습

• **지도 학습** : 입력(sequence)과 그에 해당하는 정답(레이블)이 주어진 경우
CNN(합성곱 신경망)은 대표적 지도학습 알고리즘
CNN의 응용

- 분류 : 이미지를 하나의 라벨로 분류
- 인식(탐지) : 이미지 속 객체의 종류와 위치를 찾아내기
- 분할 : 각 픽셀마다 어느 객체에 속하는지 구분

RNN, LSTM은 주로 지도학습 형태로 사용됨

RNN(순환 신경망, Recurrent Neural Network)

- 시계열 데이터나 연속적인 순서 정보가 중요한 문제에서 주로 사용
- 이전 시점의 출력을 다음 시점의 입력으로 재사용, 데이터의 순서적 특성을 학습
- 순서가 긴 데이터를 처리할 때, 역전파를 거듭하면서 **기울기가 점점 작아져 학습이 어려워지는 문제가 발생**

LSTM(Long Short-Term Memory)

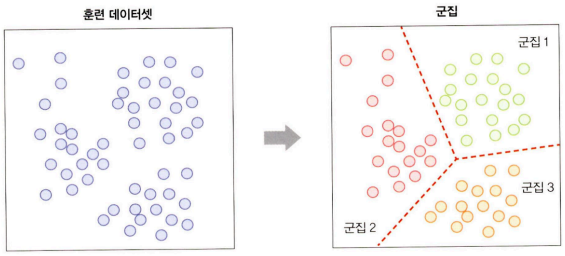
- RNN의 기울기 소실 문제를 완화하기 위해 3개의 gate를 추가
- 망각 게이트(Forget Gate): 이전 시점의 정보를 어느 정도 잊을지 결정
- 입력 게이트(Input Gate): 현재 시점의 새로운 정보를 얼마나 받아들일지 결정
- 출력 게이트(Output Gate): 다음 시점 혹은 출력으로 내보낼 정보를 결정

• **비지도 학습** : 훈련 데이터에 정답(레이블)이 없는 경우
워드 임베딩(word embedding)

- 사람이 사용하는 자연어를 벡터로 변환

- Word2Vec, GloVe

군집



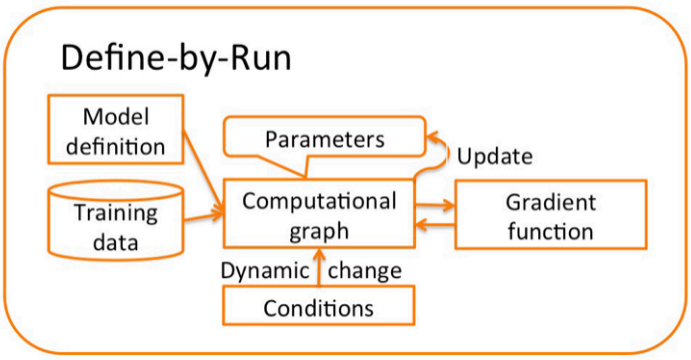
- 정답(레이블)이 없는 데이터에서, 비슷한 것끼리 묶어 클러스터를 만드는 기법
- 딥러닝과 함께 사용하면, 더 정교한 특징 추출을 통해 군집화 성능을 높일 수 있음

• **전이 학습** : 사전 학습을 모델을 다른 관련된 문제에 재사용하여 학습 효율과 성능을 높이는 기법

구분	유형	알고리즘
지도 학습	이미지 분류	- CNN - AlexNet - ResNet
지도 학습	시계열 데이터 분석	- RNN - LSTM
비지도 학습	군집 (clustering)	- 가우시안 혼합 모델(Gaussian Mixture Model, GMM) - 자기 조직화 지도(Self-Organizing Map, SOM)
비지도 학습	차원 축소	- 오토인코더(AutoEncoder) - 주성분 분석(PCA)
전이 학습	전이 학습	- BERT - MobileNetV2
강화 학습	(reinforcement learning)	- 마르코프 결정 과정(Markov Decision Process, MDP)

2. 실습 환경 설정과 파이토치 기초

파이토치 개요



파이토치 특징 및 장점

: GPU에서 텐서 조작 및 동적 신경망 구축이 가능한 프레임워크

GPU

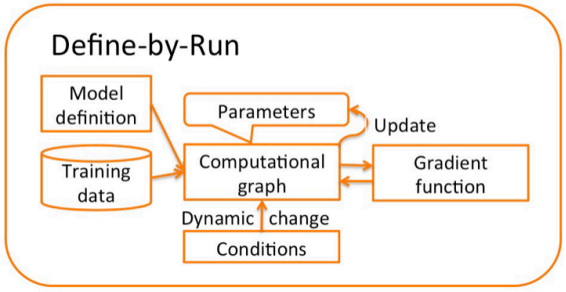
- 병렬 연산에 특화되어 딥러닝의 기울기 계산 같은 대규모 연산을 빠르게 처리
- CUDA, cuDNN API를 활용하여 이용

Tensor

- 숫자들이 여러 차원으로 배열된 구조로, 데이터를 저장하고 연산하는 기본 단위
  = 같은 크기의 행렬이 여러 개 묶여 있는 것
- x.cuda()는 텐서 x를 GPU 메모리로 복사해, GPU에서 연산이 수행되도록 함

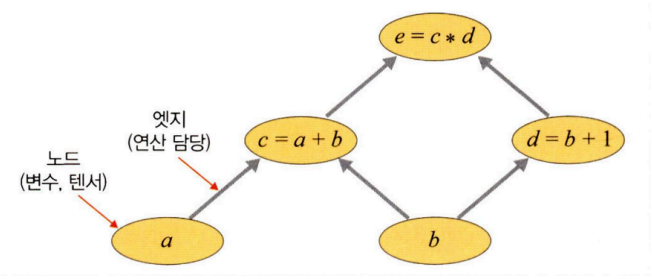
동적 신경망

- 훈련을 반복할 때마다 네트워크 변경이 가능한 신경망
  ex. 학습 중 은닉층 추가/제거

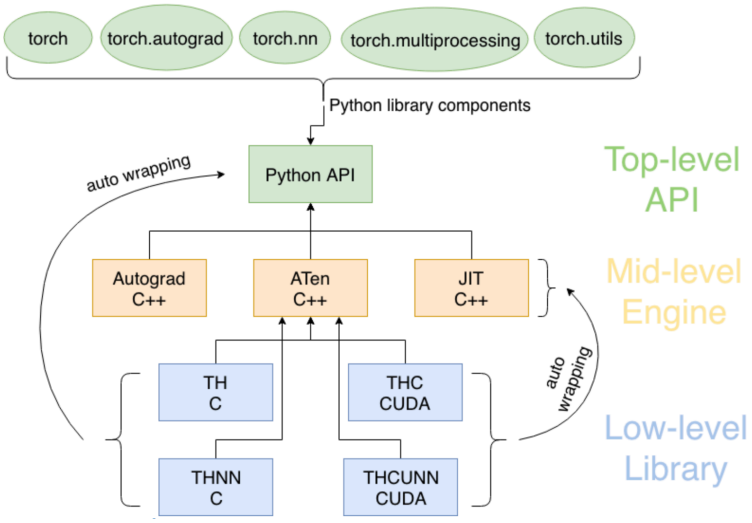


- Define by Run 방식 : 코드를 실행할 때마다 해당 연산이 즉시 그래프에 추가됨

• 연산 그래프
텐서(변수)와 연산이 화살표(방향성)로 연결된 형태



파이토치의 아키텍처



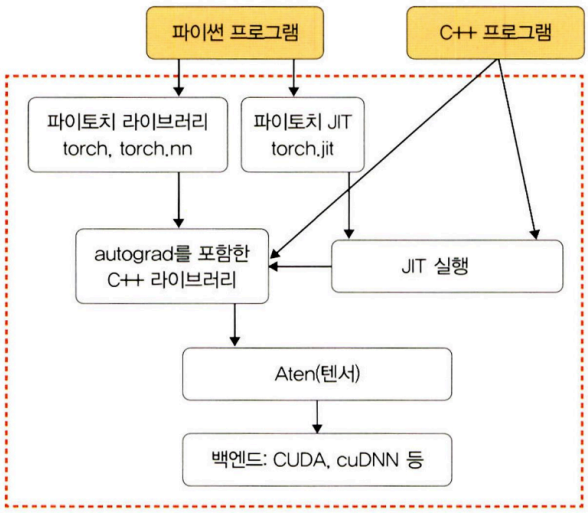
파이토치 API

사용자 인터페이스 제공, 실제 계산은 수행하지 않음.
파이토치 엔진으로 작업을 전달.

파이토치 API가 제공하는 패키지 :

- torch : GPU를 지원하는 텐서 패키지
  CPU뿐만 아니라 GPU에서 빠르게 연산할 수 있도록 구현됨
- torch.autograd : 자동 미분(Automatic Differentiation)
- torch.nn : 신경망 구축 및 훈련 패키지
- torch.multiprocessing : 멀티 프로세싱 패키지
  파이토치에서 사용하는 서로 다른 프로세스에서 텐서의 메모리 공유 가능
- torch.utils: 유틸리티 모음
  torch.utils.data.DataLoader, torch.utils.bottleneck, torch.utils.checkpoint

파이토치 엔진



파이토치 엔진 구성 요소:

• Autograd C++

- Backpropagation에 필요한 기능을 제공하는 모듈
- 연산 그래프를 추적하고, 기울기를 계산하는 자동 미분의 로직이 포함

• Aten C++

- Tensor 연산을 실제로 수행하는 백엔드 라이브러리

• JIT C++ (Just-In-Time 컴파일러)

- 계산을 최적화하기 위한 컴파일러

연산 처리

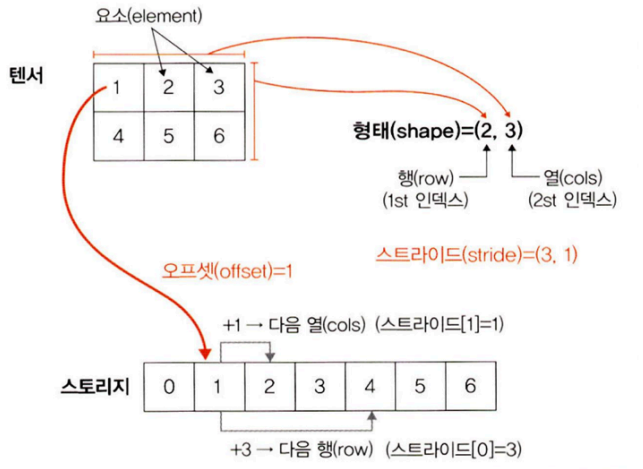
가장 아래 계층인 C, CUDA 패키지는 상위 API에서 할당된 거의 모든 계산 수행

Tensor가 메모리에 저장되는 과정

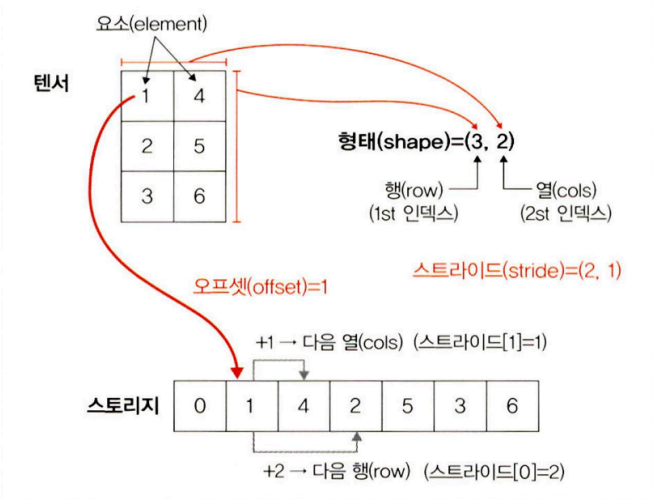
N차원 배열인 tensor의 각 요소가 1차원인 컴퓨터 메모리에 어떻게 배치되는지 정의

- 형태(Shape) : 텐서가 갖는 각 차원의 크기
  ex. (3,2)는 2차원 텐서. 첫 번째 차원은 3, 두 번째 차원은 2
- 오프셋(offset) : 텐서의 첫 번째 요소가 스토리지(1차원 메모리 배열)에서 시작되는 인덱스
- 스트라이드(Stride) : 각 차원을 따라 인접 요소로 이동하기 위해 건너뛰는 메모리 내 요소 수

예시



stride=(3,1) :
행(첫 번째 차원)을 1 증가시킬 때 스토리지 인덱스를 3 건너뛰고,
열(두 번째 차원)을 1 증가시킬 때 스토리지 인덱스를 1 건너뜀.



stride=(2,1) :
행(첫 번째 차원)을 1 증가시킬 때 스토리지 인덱스를 2 건너뛰고,
열(두 번째 차원)을 1 증가시킬 때 스토리지 인덱스를 1 건너뜀.

오프셋과 스트라이드가 필요한 이유

오프셋과 스트라이드는 실제 데이터 값을 복사·이동하는 데 쓰이는 것이 아니라, 1차원 메모리 배열에 저장된 데이터의 shape을 해석하기 위해 사용

실습 환경 설정

파이토치 코드 맛보기

라이브러리

- matplotlib : 2D, 3D 형태의 플롯을 그릴 때 사용하는 패키지
- seaborn : 시각화 차트 제공
- scikit-learn : 머신 러닝 알고리즘을 적용할 수 있는 함수 제공

chelseey
cat