개념정리/필사

WEEK 1주차

1장 | 머신 러닝과 딥러닝

1.1 인공지능, 머신 러닝과 딥러닝

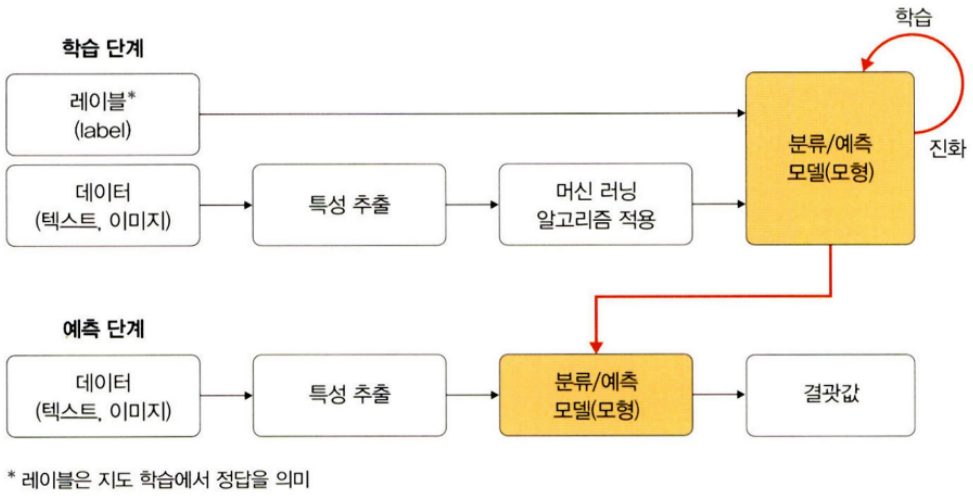
❗ **인공지능 > 머신 러닝 > 딥러닝**

구분	머신 러닝	딥러닝
동작 원리	입력 데이터에 알고리즘을 적용하여 예측을 수행 - 인간의 전처리로 데이터 특징 추출	정보를 전달하는 신경망을 사용하여 데이터 특징 및 관계 해석
재사용	입력 데이터를 분석하기 위해 다양한 알고리즘을 사용 / 동일한 유형의 데이터 분석을 위한 재사용은 불가	구현된 알고리즘은 동일한 유형의 데이터를 분석하는 데 재사용
데이터	일반적으로 수천 개의 데이터 필요	수백만 개 이상의 데이터 필요
훈련 시간	단시간	장시간
결과	일반적으로 점수 또는 분류 등 숫자 값	출력은 점수, 텍스트, 소리 등 어떤 것이든 가능

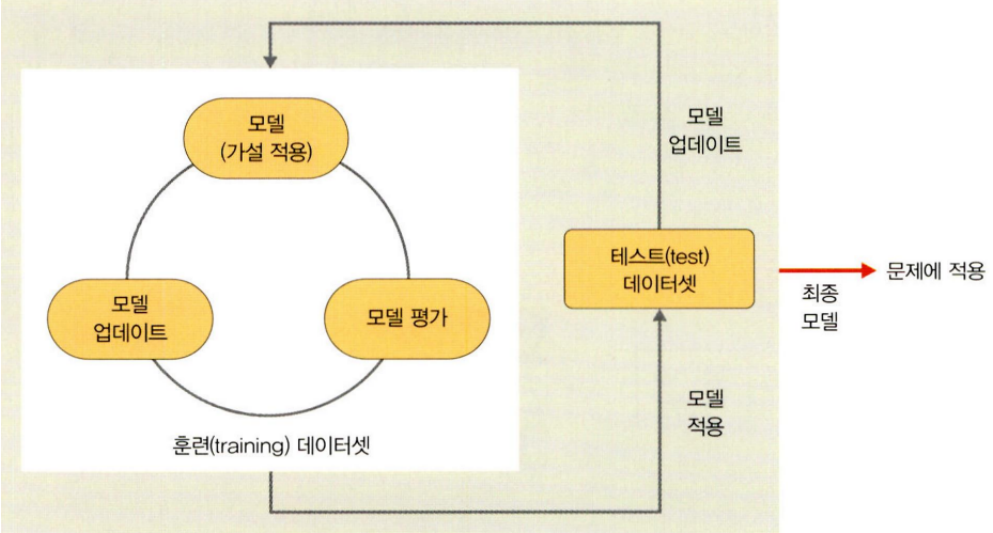
1.2 머신 러닝이란

머신 러닝 : 컴퓨터 스스로 대용량 데이터에서 지식이나 패턴을 찾아 학습하고 예측을 수행하는 것

1.2.1 머신 러닝 학습 과정



머신 러닝 학습 과정



머신 러닝의 문제 풀이 과정

1.2.2 머신 러닝 학습 알고리즘

- **지도 학습**

  정답이 무엇인지 컴퓨터에 알려 주고 학습시키는 방법

  - **분류(classification)** 알고리즘
    - KNN (K-Nearest Neighbor)
    - SVM (Support Vector Machine)
    - decision tree
    - logistic regression
  - **회귀(regression)** 알고리즘
    - linear regression

- **비지도 학습**

  정답을 알려 주지 않고 특징을 클러스터링(범주화)하여 예측하는 학습 방법

  - **군집(clustering)** 알고리즘
    - K-means clustering
    - DBSCAN : 밀도 기반 군집 분석
  - **차원 축소(dimensionally reduction)** 알고리즘
    - PCA (Principal Component Analysis) : 주성분 분석

- **강화 학습**

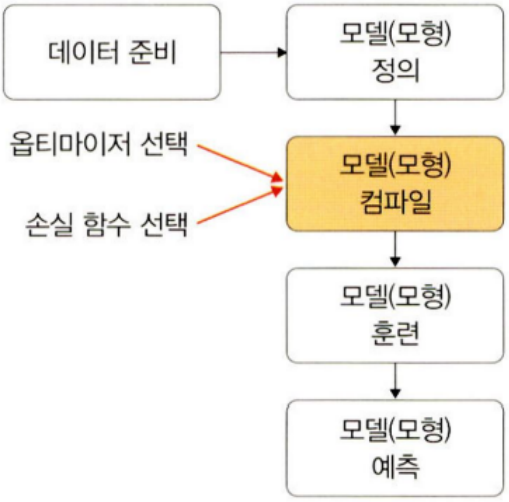
  행동에 대한 보상을 받으며 학습 진행: 보상이 커지는 행동을 자주 하도록 하고, 줄어드는 행동은 덜 하도록 하여 학습 진행

  - 알고리즘
    - MDP (Markov Decision Process) : 마르코프 결정 과정

1.3 딥러닝이란

딥러닝 : 인간의 신경망 원리를 모방한 심층 신경망 이론을 기반으로 고안된 머신 러닝 방법의 일종

1.3.1 딥러닝 학습 과정



딥러닝 모델의 학습 과정

① 데이터 준비

② 모델(모형) 정의

신경망 생성 단계

: 은닉층 개수 ↑ ... 성능 ↑ △ 과적합 발생 확률 ↑

③ 모델(모형) 컴파일

활성화 함수, 손실 함수, 옵티마이저 선택 단계

: 데이터 형태에 따라 다양한 옵션 / 과적합을 피할 수 있는 활성화 함수 및 옵티마이저 선택이 중요

④ 모델(모형) 훈련

한 번에 처리할 데이터양 지정 단계: 배치, 에포크 선택이 중요

파라미터와 하이퍼파라미터 최적 값 찾기

⑤ 모델(모형) 예측

검증 데이터셋을 생성한 모델(모형)에 적용하여 실제로 예측을 진행해 보는 단계

: 예측력이 낮다면 파라미터 튜닝 혹은 신경망 자체를 재설계

딥러닝 학습 과정 핵심 구성 요소

- 신경망

  ★ **심층 신경망**: 머신 러닝과의 차이점
- 역전파

  가중치 값 업데이트에 사용

  △ 파이토치 이용 → 딥러닝 알고리즘 구현이 간단, 편리

1.3.2 딥러닝 학습 알고리즘

- **지도 학습(supervised learning)**
  - **이미지 분류** 알고리즘

    이미지 또는 비디오 상의 객체를 식별하는 컴퓨터 비전 기술
    - **CNN** (Convolutional Neural Network) : 합성곱 신경망

      컴퓨터 비전에 가장 많이 사용됨 / 목적에 따라 이미지 분류, 인식, 분할로 분류 가능

        - AlexNet
        - ResNet
    - **시계열 데이터 분석** 알고리즘
        - **RNN** (Recurrent Neural Network) : 순환 신경망

          역전파 과정에서 기울기 소멸 문제
        - **LSTM** (Long Short-Term Memory)

          RNN 문제점 개선하고자 gate 3개 추가한 것: 망각 게이트 / 입력 게이트 / 출력 게이트

          시계열 문제에서 가장 활발히 사용
- **비지도 학습(unsupervised learning)**
    - **워드 임베딩** 알고리즘

      자연어 변환: 단어를 벡터로 표현

      번역, 음성 인식 등의 서비스에서 사용
        - Word2Vec
        - GloVe
    - **군집(clustering)** 알고리즘
        - GMM (Gaussian Mixture Model)
        - SOM (Self-Organizing Map) : 자기 조직화 지도
    - **차원 축소** 알고리즘
        - AutoEncoder
        - PCA
- **전이 학습(transfer learning)**

  pre-trained model(사전 학습 모델)을 가지고 우리가 원하는 학습에 미세 조정 기법을 이용하여 학습시키는 방법
    - **전이 학습** 알고리즘 (사전 학습 모델)
        - BERT
        - VGG
        - MobileNetV2
- **강화 학습(reinforcement learning)**
    - 알고리즘
        - MDP : 마르코프 결정 과정

2장 | 실습 환경 설정과 파이토치 기초

2.1 파이토치 개요

파이썬 기반의 **과학 연산 패키지**로, 다음 두 집단을 대상으로 함

- **넘파이를 대체하면서 GPU를 이용한 연산이 필요한 경우**
- **최대한의 유연성과 속도를 제공하는 딥러닝 연구 플랫폼이 필요한 경우**

2.1.1 파이토치 특징 및 장점

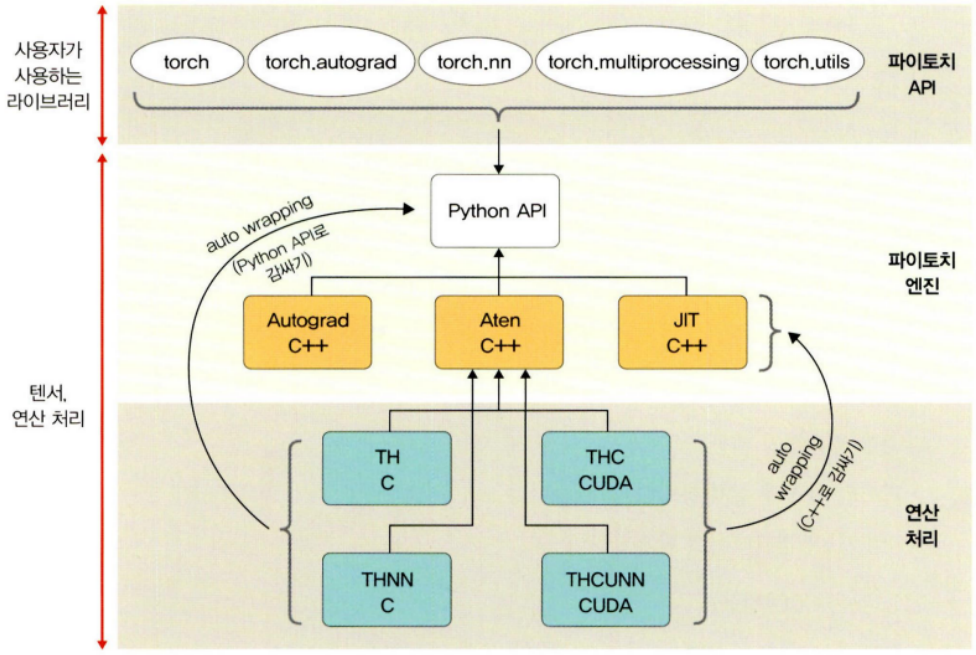
> ❗ **GPU에서 텐서 조작 및 동적 신경망 구축이 가능한 프레임워크**

- GPU(Graphics Processing Unit) : 연산 속도를 빠르게 하는 역할
- 텐서(Tensor)
  - 파이토치의 데이터 형태
  - 단일 데이터 형식으로 된 자료들의 다차원 행렬
  - 간단한 명령어(변수.cuda())를 사용해서 GPU로 연산을 수행하게 할 수 있음
- 동적 신경망 : 훈련을 반복할 때마다 네트워크 변경이 가능한 신경망 (e.g. 학습 중 은닉층 추가/제거)

파이토치 장점

- 단순함(효율적인 계산)
  - 파이썬 환경과 쉽게 통합할 수 있음
  - 디버깅이 직관적이고 간결함
- 성능(낮은 CPU 활용)
  - 모델 훈련을 위한 CPU 사용률이 텐서플로와 비교하여 낮음
  - 학습 및 추론 속도가 빠르고 다루기 쉬움
- 직관적인 인터페이스 + 낮은 진입 장벽
  - 텐서플로처럼 잦은 API 변경이 없어 배우기 쉬움

2.1.2 파이토치의 아키텍쳐



파이토치의 아키텍쳐

- **파이토치 API**
  - 사용자가 이해하기 API 제공 → 텐서에 대한 처리, 신경망 구축 및 훈련
  - 실제 계산 수행 X, 파이토치 엔진으로 작업 전달
  - 사용자 편의성을 위해 제공되는 패키지
    - torch: GPU를 지원하는 텐서 패키지
    - torch.autograd: 자동 미분 패키지

        - torch.nn: 신경망 구축 및 훈련 패키지
        - torch.multiprocessing: 파이썬 멀티프로세싱 패키지
        - torch.utils: DataLoader 및 기타 유틸리티를 제공하는 패키지
- **파이토치 엔진**
    - 구성
        - AutoGrad C++: 가중치, 바이어스 업데이트 과정에서 필요한 미분 자동 계산
        - Aten C++: C++ 텐서 라이브러리 제공
        - JIT C++: 계산을 최적화하기 위한 JIT(Just In-Time) 컴파일러
        - Python API
- **연산 처리**
    - 제공 패키지는 CPU와 GPU(TH(토치), THC(토치 CUDA))를 이용하여 효율적인 데이터 구조, 다차원 텐서에 대한 연산 처리

2.2 파이토치 기초 문법

2.2.1 텐서 다루기

- 텐서 생성

```
import torch
print(torch.tensor([[1,2],[3,4]])) # 2차원 형태의 텐서 생성
print(torch.tensor([[1,2],[3,4]], device="cuda:0")) # GPU에 텐서 생성
print(torch.tensor([[1,2],[3,4]], dtype=torch.float64)) # dtype을 이용하여 텐서 생성
```

- 텐서를 ndarray로 변환

```
temp = torch.tensor([[1,2],[3,4]])
print(temp.numpy()) # 텐서를 ndarray로 변환

temp = torch.tensor([[1,2],[3,4]], device="cuda")
print(temp.to("cpu").numpy()) # GPU 상의 텐서를 CPU의 텐서로 변환한 후 ndarray로 변
```

- 텐서의 자료형
    - torch.FloatTensor: 32비트의 부동 소수점
    - torch.DoubleTensor: 64비트의 부동 소수점
    - torch.LongTensor: 64비트의 부호가 있는 정
- 텐서의 인덱스 조작

```
temp = torch.FloatTensor([1,2,3,4,5,6,7])
print(temp[0], temp[1], temp[-1]) # 인덱스로 접근
print('------------------------------')
print(temp[2:5], temp[4:-1]) # 슬라이스로 접근
```

- 텐서의 연산

```
v = torch.tensor([1,2,3])
w = torch.tensor([3,4,6])
print(w-v) # 길이가 같은 벡터 간 뺄셈 연산
```

- 텐서의 차원 조작

```
temp = torch.tensor([[1,2],[3,4]]) # 2x2 행렬

print(temp.shape)
print('--------------------------------')
print(temp.view(4,1)) # 4x1로 변형
print('--------------------------------')
print(temp.view(-1)) # 1차원 벡터로 변형
print('--------------------------------')
print(temp.view(1,-1)) # 1x4로 변형
print('--------------------------------')
print(temp.view(-1,1)) # 4x1로 변형
```

2.2.3 모델 정의

파이토치에서 모델을 정의하기 위해서는 모듈을 상속한 클래스를 사용

→ 모델과 **모듈**의 **차이**

1. 계층(layer): 모듈 또는 모듈을 구성하는 하나의 계층으로 합성곱층(convolutional layer), 선형계층(linear layer) 등이 있음
2. 모듈(module): 한 개 이상의 계층이 모여서 구성된 것으로, 모듈이 모여 새로운 모듈을 만들 수도 있음
3. 모델(model): 최종적으로 원하는 네트워크로, 한 개의 모듈이 모델이 될 수도 있음

- **단순 신경망 정의하는 방법**

```
model = nn.Linear(in_features=1, out_features=1, bias=True)
```

- **nn.Module()을 상속하여 정의하는 방법**

  파이토치에서 nn.Module을 상속받는 모델은 기본적으로 **__init__()**과 **forward()** 함수를 포함함

  __init__()에서 모델에서 사용될 모듈(nn.Linear, nn.Conv2d). 활성화 함수 등을 정의

  forward() 함수에서 모델에서 실행되어야 하는 연산을 정의

- **Sequential 신경망 정의하는 방법**

  nn.Sequential을 사용하면 **__init__()에서 사용할 네트워크 모델들을 정의**해 줄 뿐만 아니라 **forward() 함수에서는 모델에서 실행되어야 할 계산을 더 가독성이 뛰어나게** 코드로 작성할 수 있음

  Sequential 객체는 그 안에 포함된 각 모듈을 순차적으로 실행해 줌

  nn.Sequential은 모델의 계층이 복잡할 수록 효과가 뛰어남

- **함수로 신경망 정의하는 방법**

  Sequential을 이용하는 것과 동일

  ⚠ 함수로 선언할 경우, 변수에 저장해 놓은 계층들을 재사용할 수 있는 장점

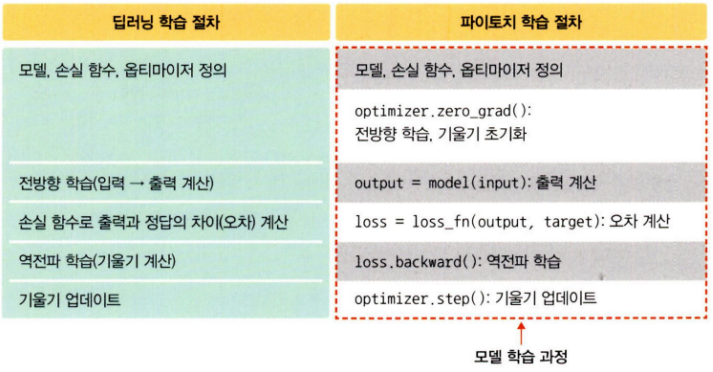
  참고: 복잡한 모델의 경우에는 함수를 이용하는 것보다 nn.Module()을 상속받아 사용하는 것이 편리

2.2.4 모델의 파라미터 정의

- **손실 함수**(loss function): 학습하는 동안 출력과 실제 값(정답) 사이의 오차 측정 / 즉, 오차를 통한 모델의 정확성 측정

    - 종류
        - BCELoss: 이진 분류를 위해 사용
        - CrossEntropyLoss: 다중 클래스 분류를 위해 사용
        - MSELoss: 회귀 모델에서 사용
- **옵티마이저**(optimizer): 데이터와 손실 함수를 바탕으로 모델의 업데이트 방법을 결정
    - 주요 특성
        1. step() 메서드를 통해 전달받은 파라미터를 업데이트
        2. 파라미터별로 다른 기준(e.g. 학습률)을 적용시킬 수 있음
        3. torch.optim.Optimizer(params, defaults)는 모든 옵티마이저의 기본이 되는 클래스
        4. zero_grad() 메서드는 옵티마이저에서 사용된 파라미터들의 기울기를 0으로 만듦
        5. torch.optim.lr_scheduler는 에포크에 따라 학습률을 조절할 수 있음
    - 종류
        - optim.Adadelta, optim.Adagrad, optim.Adam, optim.SparseAdam, optim.Adamax
        - optim.ASGD, optim.LBFGS
        - optim.RMSProp, optim.Rprop, optim.SGD
- **학습률 스케줄러**(learning rate scheduler): 미리 지정한 횟수의 에포크를 지날 때마다 학습률 감소 시켜 줌 / 학습률 스케줄러를 이용하면 학습 초기에는 빠른 학습을 진행하다가 전역 최소점(global minimum) 근처에 다다르면 학습률을 줄여서 최적점을 찾아갈 수 있도록 함
    - 종류
        - optim.lr_scheduler.LambdaLR: 람다 함수의 이용 결과를 학습률로 설정
        - optim.lr_scheduler.StepLR: 특정 단계마다 학습률을 감마 비율만큼 감소
        - optim.lr_scheduler.MultiStepLR: StepLR과 비슷하지만 지정된 에포크에만 감마 비율로 감소
        - optim.lr_scheduler.ExponentialLR: 에포크마다 이전 학습률에 감마만큼 곱함
        - optim.lr_scheduler.CosineAnnealingLR: 학습률을 코사인 함수의 형태처럼 변화시킴
        - optim.lr_scheduler.ReduceLROnPlateau: 학습이 잘되고 있는지 아닌지에 따라 동적으로 학습률 변화시킴
- **지표**(metrics): 훈련과 테스트 단계를 모니터링함

2.2.5 모델 훈련



파이토치 학습 절차

2.2.6 모델 평가

1. 함수 이용

```
acc = torchmetrics.functional.accuracy(preds,target)
```

2. 모듈 이용

```
metric = torchmetrics.Accuracy()
...
acc = metric(preds,target) # 현재 배치에서 모델 평가
...
acc = metric.compute() # 모든 배치에서 모델 평가
```

3. sklearn의 metrics 모듈에서 제공하는 클래스 이용
    - confusion_matrix
    - accuracy_score
    - classification_report

2.4 파이토치 코드 맛보기

딥러닝 분류 모델의 성능 평가 지표

- 필요한 용어
    - True Positive: 모델이 '1' 이라고 예측했는데 실제 값도 '1'인 경우
    - True Negative: 모델이 '0' 이라고 예측했는데 실제 값도 '0'인 경우
    - False Positive: 모델이 '1' 이라고 예측했는데 실제 값은 '0' 인 경우로, Type I 오류라고도 함
    - False Negative: 모델이 '0' 이라고 예측했는데 실제 값은 '1' 인 경우로, Type II 오류라고도 함
- **정확도(accuracy)**: **전체 예측 건수에서 정답을 맞힌 건수의 비율**. 정답이 긍정이든 부정이든 상관 없음

$$\frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

- **재현율(recall)**: 실제로 **정답이 1이라고 할 때 모델도 1로 예측한 비율**. 따라서 처음부터 데이터가 1일 확률이 적을 때 사용하면 좋음

$$\frac{True\ Positive}{True\ Positive + False\ Negative}$$

- **정밀도(precision)**: **모델이 1이라고 예측한 것 중에서 실제로 정답이 1인 비율**

$$\frac{True\ Positive}{True\ Positive + False\ Positive}$$

- **F1-스코어(F1-score)**: **정밀도와 재현율의 조화 평균(harmonic mean)**. 일반적으로 **정밀도와 재현율은 트레이드오프(trade-off) 관계**. 이를 해결하기 위해 조화 평균을 이용.

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$