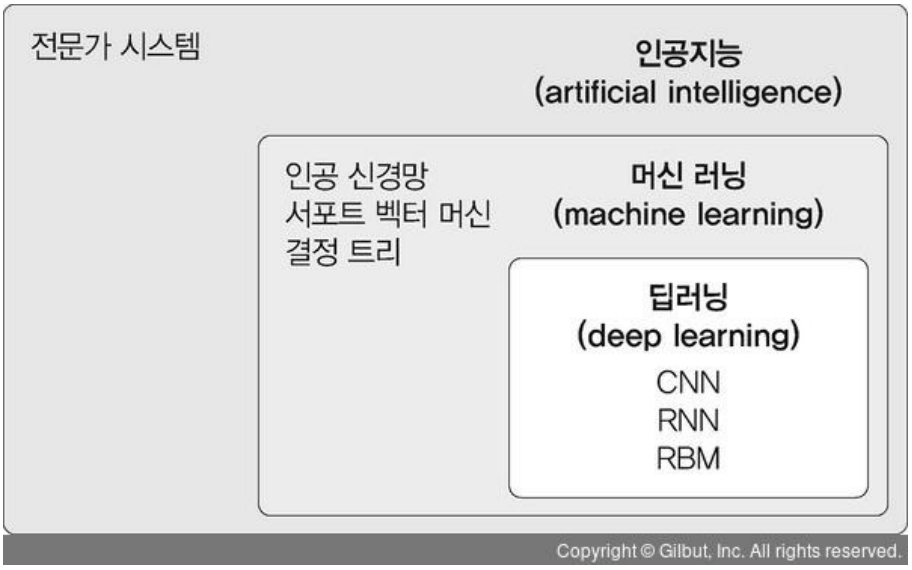
Week1_*예습과제*_김여은

[딥러닝 파이토치 교과서]를 읽고 정리한 글입니다. 🐱

📌 인공지능, 머신 러닝과 딥러닝

- **AI(인공지능)** : 인간의 지능을 모방하여 사람이 하는 일을 컴퓨터가 할 수 있게 하는 것
- **ML(머신러닝)**: 데이터 특징을 스스로 추출X, 데이터를 인간이 전처리 -> 학습 데이터를 컴퓨터가 인식할 수 있도록
    - 동일한 유형의 데이터 분석을 위한 재사용 불가
    - 수천 개의 데이터, 단시간
- **DL(딥러닝)** : 인간이 하는 작업을 생략. 대량의 데이터 적용 -> 컴퓨터가 스스로 분석
    - 동일한 유형의 데이터 분석에 재사용
    - 수백만 개의 데이터, 장시간
- 범위: AI > ML > DL



📌 머신 러닝이란

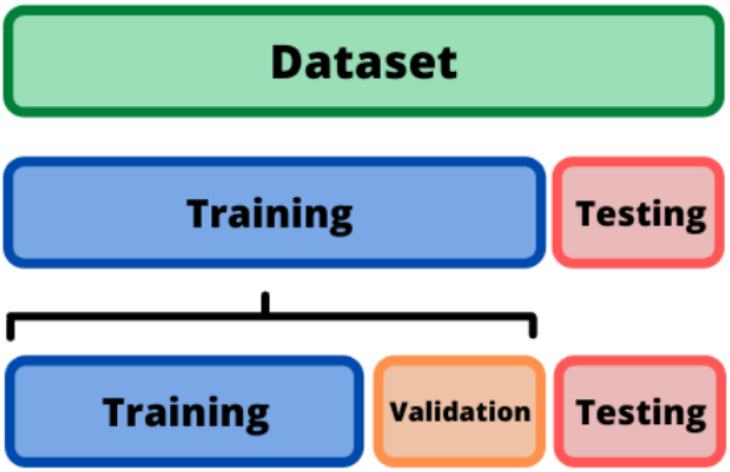
컴퓨터가 스스로 대용량 데이터에서 지식이나 패턴을 찾아 학습 후 예측

✔️ 학습 과정

- learning → prediction
- 특성 추출: 데이터별 특징을 찾아 그것을 토대로 벡터로 변환하는 작업. ML에서는 컴퓨터가 데이터에서 일정한 패턴이나 규칙을 찾아낼 수 있게 **데이터를 컴퓨터가 인지할 수 있는 데이터로 변환해 주어야 함**

✔️ 구성 요소

- 데이터: 실제 데이터의 특징이 잘 반영되고 편향되지 않아야 함
    - **훈련 데이터셋(훈련 데이터셋 + 검증 데이터셋) & 검증 데이터셋**
        - 검증 데이터셋?모델의 성능을 평가하기 위해서 사용.훈련 데이터셋에서 일부를 떼어내서 사용하기 때문에 학습 데이터셋의 양이 적을 때는 검증 데이터셋을 사용하는 것이 좋지 않음.



데이터셋이 이런 식으로 잘게 쪼개진다.

- 모델: 학습의 결과물, 가설.
  - 학습 절차
    - 1) 모델 선택
    - 2) 모델 학습.평가
    - 3) 모델 업데이트

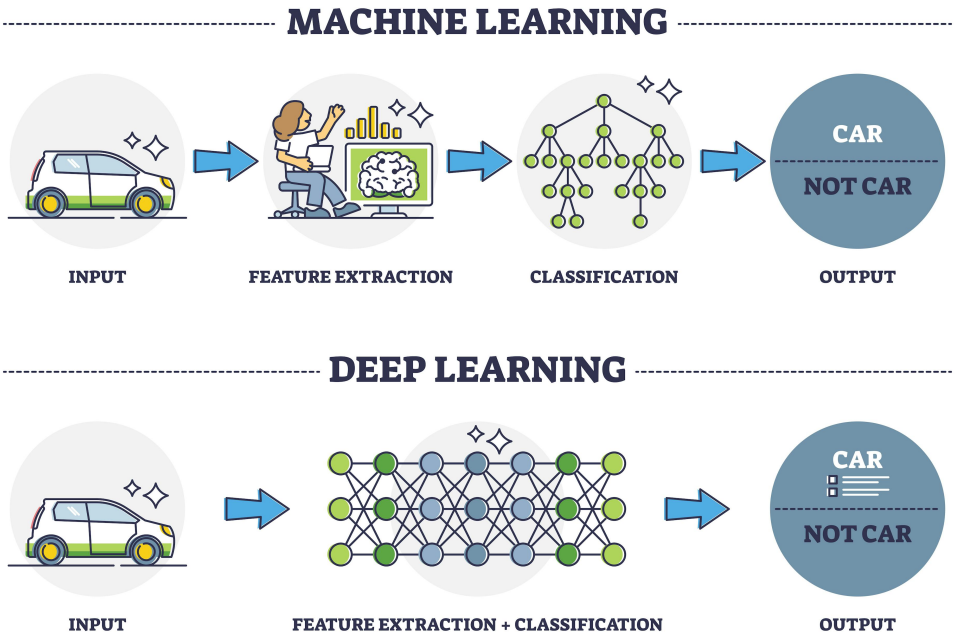
✔️ ML 학습 알고리즘

1. **Supervised Learning**(지도 학습): 정답이 무엇인지 알려주고 학습
   - Classification(분류), Regression(회귀)
2. **Unsupervised Learning**(비지도 학습): 정답 알려주지 않음. 특징 비슷한 데이터를 클러스터링하여 예측하는 학습
   - Clustering(군집), Dimensionality Reduction(차원 축소)
3. **Reinforcement Learning**(강화 학습): 자신의 행동에 대한 보상을 받으며 학습. 보상이 커지는 행동은 자주, 줄어드는 행동은 덜하기.

📌 딥러닝이란

인간의 신경망 원리를 모방한 심층 신경망 이론을 기반으로 고안된 ML 방법

- 인간의 뉴런: 뉴런의 병렬 연산 -> 복잡한 음성. 영상 인식 처리



ML과 DL의 차이점을 보여주는 이미지로, DL이 여러 신경망으로 구성되어 있음을 확인할 수 있다.

✔ 학습 과정

1. 데이터 준비
2. 모델 정의

- 신경망 생성
  - 은닉층 많음 -> 성능 up & overfitting 확률 up

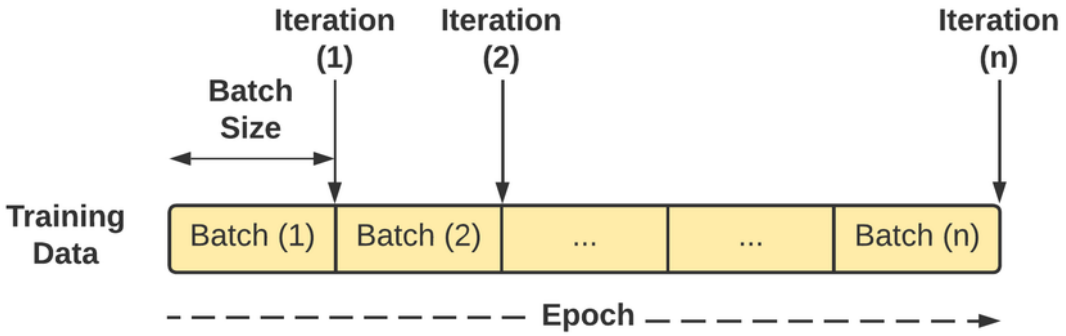
3. 모델 컴파일

- 활성화 함수, 손실 함수, 옵티마이저 선택
  - **활성화 함수**: 입력 신호 >= 일정 기준 -> 출력 신호 변환
  - **손실 함수**: 모델 출력값과 사용자가 원하는 출력 값(레이블)의 차이를 구하는 함수
  - **옵티마이저**: 손실 함수 기반으로 네트워크 업데이트 방법 결정
- ex) 연속형 훈련 데이터셋 -> MSE(Mean Squared Error) / Binary Classification → Cross Entropy

4. 모델 훈련

- 한 번에 처리할 데이터양 지정

- 데이터양 많아짐 -> 학습 속도 down & 메모리 부족 확률 up
- 적절한 Batch, Epoch 선택
  - **Batch**(배치): 전체 훈련 데이터셋에서 일정한 묶음으로 나누어 처리
  - **Epoch**(에폭): 훈련 횟수
  - ex) 훈련 데이터셋 1000개, batch=20 → 20개마다 모델 가중치 업데이트(총 50번)if epoch=10 → 50번의 업데이트를 10번 반복

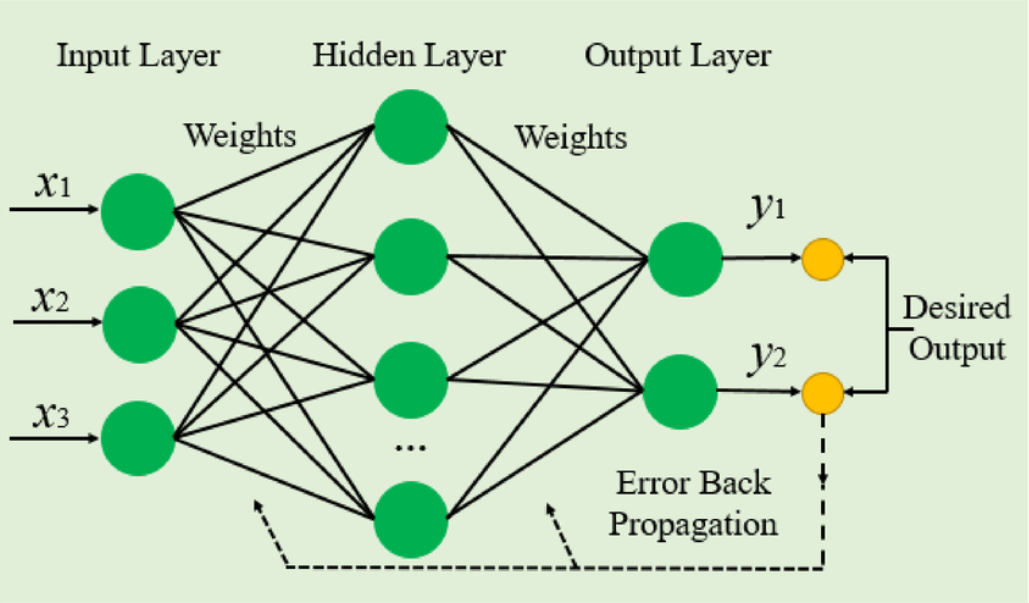


- **파라미터**(모델 내부에서 결정되는 변수), **하이퍼파라미터**(최적화해야 하는 변수)에 대한 최적의 값을 찾아야함

5. 예측

✔️ 구성 요소

- **Neural Network**(신경망): 딥러닝의 경우 DNN(Deep Neural Network)을 사용하여 데이터셋의 어떤 특성들이 중요한지 스스로에게 가르쳐 줄 수 있는 기능이 있음
- **Backpropagation**(역전파): 가중치 값 업데이트/ 오차를 각 가중치로 미분한 것이 성능에 영향을 미침



점선이 Back Propagation을 의미한다.

✔️ DL 학습 알고리즘

1. **Supervised Learning**
    - 이미지 분류: **CNN**(Convolutional Neural Network)
    - 시계열 데이터 분석: **RNN**(Recurrent Neural Network)
        - RNN 단점: 역전파 과정에서 기울기 소멸 발생 -> LSTM으로 해결
            - **LSTM**: input gate, output gate, forget gate → 기울기 소멸 문제 해결
2. **Unsupervised Learning**
    - **Word Embedding**(워드 임베딩): 단어 -> 벡터
        - ex) Word2Vec, GloVe
    - Clustering(군집): 클러스터 내에서는 매우 비슷하게, 클러스터 밖과는 매우 다르게
        - ML에서 DL과 함께 클러스터링 처리 시 모델 성능을 높일 수 있음
3. **Transfer Learning**(전이 학습)
    - 사전 학습 모델(pre-trained model)을 가지고 원하는 학습에 미세 조정 기법을 이용하여 학습

    - **pre-trained model**: 풀고자 하는 문제와 비슷하면서 많은 데이터로 이미 학습이 되어있는 모델ex) VGG, Inception, MobileNet

📌 파이토치란

페이스북에서 만든 딥러닝 프레임워크로

- numpy를 대체하면서 GPU 연산이 필요한 경우
- 최대한의 유연성과 속도를 제공하는 딥러닝 연구 플랫폼이 필요한 경우를 대상으로 하는 파이썬 기반의 과학 연산 패키지

GPU에서 텐서 조작 및 동적 신경망 구축이 가능한 프레임워크

1) **GPU**

- 딥러닝에서 기울기 계산시 미분을 사용하는데, 이때 GPU를 이용해 빠른 계산
- CUDA, cuDNN API를 이용해 GPU를 연산에 사용

2) **Tensor**

- 파이토치의 데이터 형태
- 단일 데이터 형식으로 된 자료들의 다차원 행렬

3) **동적 신경망**

- 훈련을 반복할 때마다 네트워크 변경이 가능한 신경망
- Define by Run: 연산 그래프를 정의함과 동시에 값도 초기화

✔️ 파이토치 특징

- CPU 사용률이 tensorflow에 비해 낮음
- tensorflow처럼 잦은 API 변경이 없어 배우기 쉬움

✔️ 벡터, 행렬, 텐서

- **벡터**: 1차원 리스트(axis 0)

- **행렬**: 2차원 배열(axis 1)
- **텐서**: 3차원 이상의 배열(axis 2)

✔️ 연산 그래프

- 신경망은 연산 그래프를 이용해 계산 수행
  - 손실함수의 기울기가 가중치와 바이어스를 기반으로 계산됨
  - 경사 하강법을 사용한 가중치 업데이트
- 방향성 있음
- 노드: 변수, 엣지: 연산

✔️ 파이토치 아키텍처

1. 파이토치 API
   - 사용자 인터페이스
     - 실제 계산X
   - ex) torch, torch.autograd, torch.nn
2. 파이토치 엔진
   - C++로 래핑한 다음 python API 형태로 제공
     - Autograd C++: 가중치, 바이어스 업데이트 과정에서 필요한 미분을 자동 계산
     - Aten C++: C++ 텐서 라이브러리 제공
     - JIT C++: 계산 최적화하기 위한 JIT 컴파일러
3. 텐서의 실질적 계산을 위한 라이브러리
   - C 또는 CUDA 패키지
   - CPU와 GPU를 이용해 효율적인 데이터 구조, 다차원 텐서에 대한 연산 처리
   - 거의 모든 계산 수행

✔️ 텐서를 메모리에 저장하기

- 텐서는 항상 메모리에 저장할 때 1차원 배열 형태(**storage**)여야 함
- **offset**: 텐서에서 첫번째 요소가 스토리지에 저장된 인덱스
- **stride**: 다음 요소를 얻기 위해 건너뛰기가 필요한 요소 개수
- offset과 stride는 행렬/텐서를 구분하기 위해 사용

📌 파이토치 기초 문법

✔️ 텐서 생성. 변환

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

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

실행결과:

```
[[1 2]
[3 4]]
```

✔️ 텐서의 인덱스 조작

```
print(temp[2:5]) # temp에서 2, 3, 4번째 값 가져오기
```

✔️ 텐서의 연산. 차원 조작

- 텐서 간의 타입이 다르면 연산 불가능

```
v=torch.tensor([1,2,3])
w=torch.tensor([3,2,9])
print(w-v)
```

실행결과:

`tensor([1,2,3])`

- view: 텐서의 차원 변경하는 대표적인 방법(numpy의 reshape과 유사)
- stack, cat: 텐서 결합
- t, transpose: 차원 교환

✔️ 데이터 준비

- 이미지 데이터: 분산된 파일에서 데이터 읽은 후 전처리하고 batch 단위로 분할하여 처리
- 텍스트 데이터: 임베딩 과정 거쳐 서로 다른 길이의 시퀀스를 batch 단위로 분할하여 처리

커스텀 데이터셋

- 데이터를 한 번에 메모리에 불러와서 훈련시키면 시간. 비용 측면에서 효율적이지 않음 -> 커스텀데이터셋(**CustomDataset**)으로 해결
    - **DataLoader**: 학습에 사용될 데이터를 전체 보관했다가 모델 학습 시 batch 크기만큼 데이터를 꺼내서 사용. 미리 잘라놓는게 아니라 반복자에 포함된 인덱스를 이용

✔️ 모델 정의

파이토치에서는 모델 정의 시 모듈을 상속한 클래스를 사용

- **계층**: 모듈 또는 모듈을 구성하는 한 개의 계층
- **모듈**: 한 개 이상의 계층이 모여서 구성된 것
- **모델**: 최종적으로 원하는 네트워크

1) 단순 신경망 정의

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

2) nn.Module()을 상속하여 정의

- __init__()과 forward() 함수 포함
  - __init__(): 모델에서 사용될 모듈, 활성화 함수 등을 정의
  - forward(): 모델에서 실행되어야 하는 연산 정의

3) Sequential 신경망을 정의

nn.Sequential 이용하면 __init__()에서 사용할 네트워크 모델 정의해줌 & forward()에서 모델에서 실행되어야 할 계산을 가독성 있게 작성 가능

- Sequential: 포함된 각 모듈을 순차적으로 실행
  - 모델의 계층이 복잡할수록 효과가 뛰어남

4) 함수로 신경망 정의

- 장점: 함수로 선언할 경우 변수에 저장해 놓은 계층들을 재사용할 수 있음
- 단점: 모델이 복잡해짐 -> 이 경우엔 nn.Module() 상속 받는 것 추천

✔️ 모델의 파라미터 정의

- **손실 함수**: 학습하는 동안 출력과 실제값 사이의 오차 측정
- **옵티마이저**: 데이터와 손실 함수를 바탕으로 모델의 업데이트 방법 결정
- **학습률 스케줄러**: 미리 지정한 횟수의 epoch을 지날 때마다 학습률을 decay(초기에는 빠른 학습. global minimum 근처에 다다르면 학습률 줄여서 최적점 찾기)
  - **global minimum**: 오차가 가장 작을 때의 값
  - **local minimum**(극소): global min 찾는 과정에서 만나는 hole
- **지표**(metrics): 훈련과 테스트 단계 모니터링

✔️ 모델 훈련

> 학습을 시키다 == y=wx+b라는 함수에서 적절한 w와 b를 찾다

1. 기울기 초기화 `optimizer.zero_grad()` 이용

- 기울기 값 계산을 위해 사용하는 `loss.backward()` → 새로운 기울기 값이 이전 기울기 값에 누적하여 계산됨
    - 누적되는 계산은 RNN에는 효과적이나 누적 계산이 필요없는 모델에 대해서는 불필요
    - 누적 계산 필요X일 때 입력 값을 모델에 적용하기 전 `optimizer.zero_grad()`로 초기화

✔️ 모델 평가

ex) torchmetrics.functional.accuracy, scikitlearn의 confusion matrix 등

✔️ 훈련 과정 모니터링

텐서보드: 학습 진행 과정에서 각 파라미터에 어떤 값들이 어떻게 변화하는지 모니터링

1. 텐서보드 set up

- model.train(): 훈련 데이터셋에 사용하여 모델 훈련이 진행될 것을 알림
- model.eval(): 검증, 테스트 데이터셋에 사용하여 모델 평가 시 모든 노드를 사용하겠다는 의미이 둘을 사용하여 모델의 정확도를 높일 수 있음
- 검증.테스트 과정에서 역전파를 사용하지 않으므로 torch.no_grad() 사용해 기울기 값 저장하지 않기

(파이토치가 모든 연산과 기울기 값을 저장하기 때문임)