딥러닝 기초

1. 딥러닝

-인공지능 -> 머신러닝 -> 딥러닝
- 인공지능 : 강 인공지능(사람 같음), 약 인공지능(특정 영역에서 작업 수행)
- 머신러닝 : 규칙을 스스로 찾아 수정(학습/훈련)
    + 규칙 : 가중치(입력과 곱하는 수), 절편(더하는 수)
    ->잘 맞는 데이터 : 정형 데이터(데이터베이스, 레코드 파일, 엑셀 등)
    학습 방식 : 지도 학습 / 비지도 학습 / 강화 학습
    1. 지도 학습 : 훈련 데이터 활용(입력과 타깃으로 구성)
        - 입력 : 모델이 풀어야 할 문제, 타깃 : 정답
        -> 문제에 대한 답을 주는 방법으로 훈련 -> 학습을 통해 예측하는 프로그램
        ex) 날씨 예측, 스팸 메일 분류
    2. 비지도 학습 : 타깃이 없는 데이터 사용 -> 모델의 훈련 결과 평가가 어려움
    3. 강화 학습 : 주어진 환경으로부터 피드백을 받아 훈련
        - 에이전트 훈련 : 특정 환경에 최적화된 행동 수행 후 '보상'과 '현재 상태'를 받음
- 딥러닝 : 머신러닝 알고리즘 중 하나 -> 인공신경망을 다양하게 쌓은 것
    + 머신러닝 알고리즘 : 선형 회귀, 로지스틱 회귀, 인공신경망 etc...
    -> 딥러닝에 잘 맞는 데이터 : 비정형 데이터(이미지, 영상, 음성, 텍스트, 번역 등)
    + 경사 하강법 : 모델이 데이터를 잘 표현할 수 있도록 기울기를 사용하여 모델을
    조금씩 조정하는 최적화 알고리즘 -> 많은 양의 데이터를 사용하기 좋음

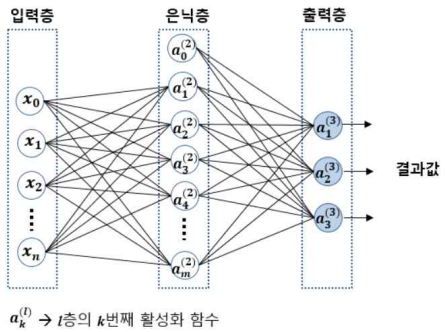
2. 비용함수와 손실함수

- 목적함수(Objective Function) -> 비용함수(Cost Function) -> 손실함수(Loss Function)
- 비용함수(Cost Function) : 전체 데이터세트에 대해 계산되는 손실
    -> 모델이 데이터를 얼마나 정확하게 표현하지 못하는가 표현
    -> 비용함수의 값이 최소화 일 때 모델의 성능 최적화
    -> 손실함수의 결과값들을 평균내어 계산
    ex) 평균 제곱 오차 : 여러 데이터 포인트에 대한 MSE의 평균
- 손실함수(Loss Function) : 단일 모델이 예측한 값과 실제값의 차이를 계산하는 함수
    -> 손실함수의 값이 낮을수록 모델의 예측이 실제 값에 가까워짐
    ex) 회귀 모델 - SE(Squared Error, 제곱 오차)
        SSE(Sum of Squared for Error, 오차 제곱합)
        MAE(Mean Absolute Error, 평균 절대 오차),
        MSE(Mean Squared Error, 평균 제곱 오차),
        RMSE(Root Mean Square Error, 평균 제곱근 오차)
    분류 모델 : Binary cross-entropy, Categorical cross-entropy

- 차이점
    -> 손실함수는 모델의 예측값과 실제값의 오차 계산
    -> 비용함수는 모델의 모든 데이터세트에 대해 오차 계산
    +비용함수는 모든 데이터세트에 대해 계산한 손실함수의 평균값으로 구할 수 있음

3. MLP(Multi-Layer Perceptron, 다층 퍼셉트론)

- 퍼셉트론 : 이진 분류 문제에서 최적의 가중치를 학습하는 알고리즘 발표
    -> 샘플을 이진 분류하기 위해 계단함수(step function) 사용하고 통과한 값을 다시 가중치와 절편을 학습하는데 사용
    + 계단함수 : y = 1(z > 0) / -1 (1 : 양성 클래스, -1 : 음성 클래스)
- 아달린(적응형 선형 뉴런) : 퍼셉트론을 개선 -> 선형함수 출력 이후 역방향 계산 진행됨
- 로지스틱 회귀(logistic regression) : 아달린에서 더 발전
    -> 활성화 함수 : 선형함수를 통과시켜 얻은 z를 임계 함수에 보내기 전에 변형
    -> 임계함수 : 활성화 함수의 출력값을 사용 (임계함수를 사용하여 예측 수행)
- 로지스틱 손실함수 : 크로스 엔트로피 손실함수를 이진 분류 버전으로 만든 것
    -> 로지스틱 손실함수의 미분하면 로지스틱 손실 함수의 값을 최소로 하는 가중치와 절편을 찾을 수 있음
    -> 은닉층이 없는 신경망
- 신경망 : 입력층, 은닉층, 출력층으로 이루어짐
    -> 입력 신호를 받아 은닉층에서 연산하고 출력
- 퍼셉트론 VS 신경망
    -> 퍼셉트론은 활성화 함수(계단함수) 사용, 신경망은 활성 함수(시그모이드, 렐루) 사용
- 단층 퍼셉트론 : 비선형적으로 분리되는 데이터에 대해 학습 불가능
    -> AND연산 가능/ XOR연산 불가능
- 다층 퍼셉트론 : 단층 퍼셉트론의 한계를 극복하기 위해 은닉층 추가
    - 비선형적으로 분리되는 데이터에 대해서도 학습 가능
    - 심층 신경망 : 출력층 사이 여러개의 은닉층이 있음
    - 은닉층과 출력층에 존재하는 활성 함수와 가중치가 여러개
    + 딥러닝 : 심층 신경망을 학습하기 위해 고안된 알고리즘
    - 순전파 : 입력층에서 전달되는 값은 은닉층의 모든 노드로 전달 후 은닉층의 모든 노드의 출력값은 출력층의 모든 노드로 전달



4. 합성곱(Convolution)

- 두 함수에 적용하여 새로운 함수를 만드는 수학 연산자

$$x(t)*h(t) = \int x(\tau)h(t-\tau)d\tau$$

- 활용 : 딥러닝(행렬의 합성곱), 샘플링(신호의 필터링), 라플라스변환, 푸리에변환
- 합성곱 연산 : 이미지처리에서 필터(=커널) 연산에 해당함(더하기(합성)와 곱만 사용)
  + 패딩 : 원본 배열 양 끝에 빈 원소를 추가 / 스트라이드 : 미끄러지는 배열 간격 조절
- 풀 패딩 : 원본 배열 원소의 연산 참여도를 동일하게 만듦
  -> 제로 패딩 : 원본 배열의 양 끝에 가상의 원소를 추가
- 세임 패딩 : 출력 배열의 길이를 원본 배열의 길이와 동일하게 만듦

5. 과적합

- 알고리즘이 학습 데이터에 과하게 적합한 상태이거나 정확하게 일치할 때 발생
  -> 새로운 데이터를 효과적으로 일반화 X -> 의도한 분류나 예측 작업을 수행 불가
  =학습 데이터가 아닌 다른 데이터에서 정확한 예측을 생성하거나 결론 도출 불가
- 과적합 판단 : 학습 데이터의 오류율이 낮고 테스트 데이터의 오류율이 높음
- 방지 방법
  1. 조기 종료
  - 모델이 모델 내의 노이즈를 학습하기 전에 학습 종료(학습 프로세스 종료가 빨라 과적합과 과소적합을 일으킬 수 있어 과적합과 과소적합 사이 최적점을 찾아야함)
  2. 더 많은 데이터로 학습
  - 학습 세트를 확장하면 입력 변수와 출력 변수간의 관계를 더 많이 분석하여 모델의 정확도를 높일 수 있음(정제되고 관련성 높은 데이터가 모델에 주입될 때 효과적)
  3. 데이터 증대
  - 훈련 데이터 세트의 크기를 인위적으로 늘림
    ex) 회전, 변환, 크기 조정, 뒤집기 등
  4. 특징 선택
  - 학습 데이터 내에서 중요한 특징을 식별 후 관련 없거나 중복되는 특징 제거
  5. 정규화
  - 어떤 특징을 제거해야 할지 모르는 경우 사용 -> 노이즈를 식별하고 줄이는게 목표
    ex) 라쏘 정규화, 릿지 회귀, 드롭아웃 etc...
  6. 앙상블 방법
  - 노이즈가 있는 데이터 세트 내의 분산을 줄이는 데 사용
  -> 개별 모델을 조합하여 최적의 모델로 일반화
    ex) 배깅(Bagging), 부스팅(Boosting), 보팅(voting) etc...

Keras

1. **환경설정** (개발 환경 세팅)

- Anaconda 설치 후 프로젝트 디렉토리 만들기 (가상환경 구축)
  -> 텐서플로 설치 -> 케라스 설치

2. **구성요소**

- Sequential 모델 : 층으로 스택을 쌓아 만들었고 가장 단순(한 종류의 입력값과 출력값만 사용가능 / 데이터 변환은 지정된 층의 순서대로 적용)
  -> layer 처음 호출될 때 fit 만들어짐 = 가중치를 만듦 /
  build() 함수 호출로 모델의 가중치를 설정
  -> 사용하기 쉽지만 단일 입, 출력만 가능하여 적용이 제한적
  + 가중치 : 확률적 경사 하강법에 의해 학습되는 하나 이상의 tensor
  + Dense : 뉴런 구성 요소 중 하나인 fully connected layer를 구현한 클래스
- 함수형 API : 그래프 같은 모델, 사용성과 유연성 적절
  -> 다중 입/출력 또는 비선형적 구조에서 사용
  -> Shape(입력의 크기), Input layer(입력층)을 모델 앞단에 정의
- Model 서브클래싱 : 모든 것을 밑바닥부터 만들 수 있는 저수준 방법
  -> 딥러닝 모델 전부 구현 가능
  -> 서브클래스의 객체를 만들고, 가중치를 만듦
  -> __init__() : 모델이 사용할 층 정의 / call() : 정방향 패스 정의

3. **특징**

- 딥러닝 패키지를 편리하게 사용하기 위해 만들어진 래퍼 패키지
- 인공신경망의 층을 직관적으로 설계할 수 있음
- 동일한 코드로 CPU, GPU 실행 가능
- 다중 입/출력 모델, 층의 공유, 모델 공유 등 여러 네트워크 구조를 만들 수 있음

데이터 전처리

1. 데이터 전처리

- 데이터를 분석과 처리에 적합한 형태로 가공하는 과정

2. 데이터 전처리 목적성

- 불필요한 데이터 제거하거나 처리하여 데이터의 질 향상
  ex) 잡음, 이상치, 부적합, 결측치, 중복 등
- 가공된 데이터 -> 분석 모델을 구축하고 결과 도출하는데 유용함

3. 데이터 전처리 절차 및 방법

-절차

1. 데이터 정리

=> 결측값(값이 존재하지 않고 비어있음), 잡음, 이상치(표준편차 3 이상)

- 이상값 관리 : 이상값을 식별한 후 제거하거나 통계적으로 추정된 값으로 대체
- 누락 데이터 채우기 : 누락이나 유효하지 않은 데이터를 식별 후 추정된 값으로 대체
- 평활화 : 잡음 필터링

2. 데이터 변환

- 정규화 및 재스케일링 : 스케일이 서로 다른 데이터세트를 균일한 스케일로 표준화
- 추세 제거 : 다항식 추세를 제거하여 데이터세트 내부 편차의 가시성 향상

3. 구조 관련 작업

- 결합 : 공통된 키 변수를 행을 기준으로 두 개의 테이블이나 타임테이블 결합
- 변수 쌓기 및 변수 해체 : 데이터를 통합 또는 재분포하여 분석 편의성 증진
- 그룹화 및 비닝 : 개별적인 데이터값을 특정한 구간이나 그룹으로 묶는 과정
- 피벗 테이블 계산 : 데이터세트를 하위 테이블로 나누어 집중된 정보 획득

4. 데이터 전처리 기초 문법

- info() : 데이터프레임을 구성하는 행/열의 크기, 컬럼(명)을 구성하는 값의 자료형을 출력
- head() : n개의 행을 가져와서 보여줌 -> 전체적인 데이터 형태를 살펴보기에 좋음

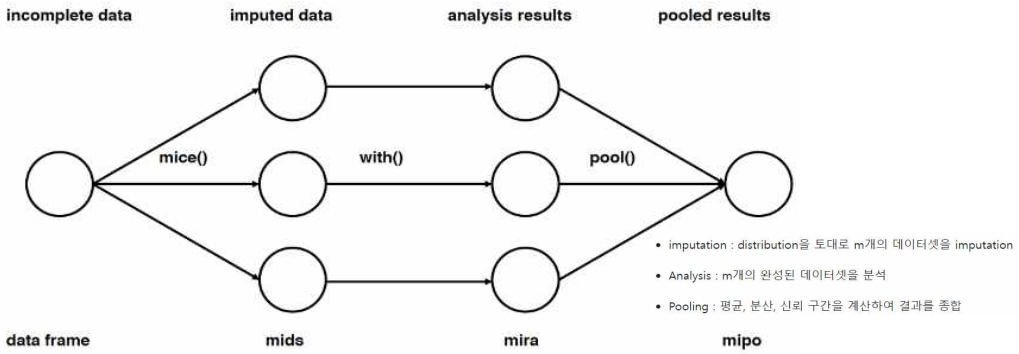
```
<class 'pandas.core.frame.DataFrame'>
Index: 6 entries, 1번 to 6번
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   이름      6 non-null     object
 1   지역      6 non-null     object
 2   나이      6 non-null     int64
 3   국어점수    6 non-null     int64
 4   수학점수    6 non-null     int64
 5   코딩      6 non-null     object
dtypes: int64(3), object(3)
memory usage: 336.0+ bytes
```

info()

	이름	지역	나이	국어점수	수학점수	코딩
1번	유재석	서울	19	86	86	Python
2번	박명수	부산	23	90	100	Java
3번	정준하	부산	20	80	66	
4번	노홍철	서울	25	65	70	Javascript
5번	정형돈	서울	18	50	40	PYTHON

head()

- csv(comma-separated values) 파일 불러오기 : 필드를 쉼표로 구분한 텍스트 데이터 및 텍스트 파일임
  -> 내장 함수인 read_csv로 파일 불러오기 -> pd.read_csv('경로/파일이름.csv')
- excel 파일 불러오기 : read_excel( ) 함수 활용 -> pd.read_excel('경로/파일이름.csv')
- 결측값 처리하기
  - 결측값 종류
  1. MCAR(Missing Completely At Random, 완전 무작위 결측) : 전산오류, 통신문제
  2. MAR(Missing At Random, 무작위 결측) : 특정 변수와 관련은 있으나 결과와는 상관 없음
  3. NMAR(Not Missing At Random, 비무작위 결측) : 다른 변수와 결측값 발생이 상관 있음
  - 결측값 처리 방법
  1. 결측치가 있는 행이나 열을 전체 제거(가진 데이터를 잃을 위험이 있음)
  2. 중앙값, 평균값으로 대체 -> median() or mean() 사용
     -> 절대 범주형 데이터엔 사용 불가 / 다른 feature간의 상관관계 고려 안함
  3. 최빈값, 0, 상수값으로 대체
     -> 다른 feature간의 상관관계 고려 안함 / 상수값에 따라 이상치가 될 수 있음
  4. K-NN 대체(K-Nearest Neighbor) : 가장 근접한 데이터를 k개 찾음
     -> 평균, 중앙값보다 정확하지만 이상치에 민감함
  5. MICE(Multivariate Imputation by Chained Equation)
     -> 누락된 데이터를 여러번 채움(연속형, 이진형, 범위형 패턴 처리 가능)



  6. 딥러닝을 이용한 Imputation : DNN을 이용하여 누락된 값 유추(자료형에 효과적)
- 행, 열 조작
  -> 열 생성해서 기존 데이터프레임에 추가하기 : df['열 이름 지정'] = dataframe
  -> df 전체의 열 이름 수정 : df.columns = ['name_1', 'name_2', 'name3']
  df.columns = list
  -> 특정 열 이름 수정 : df.rename(columns =
  {'old_1':'new_1','old_2':'new_2'}, inplace = True)
  -> 행 인덱스(이름) 변경 : df.index = list
  -> 특정 열 제거 : df.drop(['name1','name2'], axis = 'columns')
  -> 첫번째 행 제거 : df.drop(0, [i]) / 첫 번째 열 제거 : df.drop([i], 0)

- 데이터프레임 슬라이싱
    -> 열접근 : df.colname or df['colname']
    -> loc : df.loc[: , ['name', 'gender']]
    -> iloc : print(df.iloc[2, 4]) (인덱스(숫자)를 통해 슬라이싱)
    -> 조건부 슬라이싱 : print(df[df.gender == 'M']) (조건에 맞는 것만 출력)
- 데이터 병합
    -> pd.concat() : 데이터프레임 이어붙이기
    + axis = 0 : 행방향(위아래)으로 이어붙이기 / axis = 1 : 열방향(좌우)으로 이어붙이기
    outer : 합집합 / inner : 교집합
    -> pd.merge() : 데이터프레임을 존재하는 고유값을 기준으로 병합
    -> join() : 행 인덱스를 기준으로 결합