# 1장 기초 수학과 미적분

## 오일러의 수와 자연로그

### 1) 오일러의 수

- 오일러의 수 e는 미분값이 자기 자신인 수
- 기울기 계산 시 효율적이며, 인공신경망에서 시그모이드 함수나 소프트맥스 함수에 사용 됨
- 오일러의 수 e를 밑으로 사용하는 로그 $ln()$를 자연로그 라고 함
- 이자를 복리로 계산할 때, 월 -> 일 -> 분 -> 시 -> 무한히 작은 순간을 n에 적용
- $(1+\frac{1}{n})^n$ 에서 n이 무한대로 커지면 약 2.781828에 수렴

### 2) 자연로그

- 오일러의 수 e를 밑으로 사용하는 로그: $ln()$
- $ln(10)$은 e를 거듭제곱해서 10이 되는 수

## 미분(Derivative)

- x의 아주 작은 변화량에 대한 y의 변화량, 기울기
- 합성함수를 미분할 때 연쇄법칙을 적용할 수 있으며,
- 연쇄법칙을 이용해 신경망에서 오차역전파를 수행할 때 각 노드의 도함수를 곱하여 손실함수에 대한 미분값을 구할 수 있다.
  $\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$
- 편도함수란?
  - $f(x,y) = 2x^3 + 3y^3$ 일 때, x와 y변수는 각각 고유한 도함수(기울기=gradient) $\frac{df}{dx}$와 $\frac{df}{dy}$를 갖음

## 적분

# 2장 확률

## 확률 이해

- 확률(probability)은 아직 일어나지 않은 사건에 대한 예측이고 가능도(likelihood)는 이미 발생한 사건의 빈도입니다. 머신러닝에서는 확률(미래)을 예측하기 위해 데이터 형태의 가능도(과거)를 사용합니다.
  - 확률: Sum(모든 상호 배타적인 결과의 확률) = 1
  - 가능도: Sum(모든 상호 배타적인 결과의 가능도) 이 1 아닐 수 있음

## 오즈(Odds)

- 어떤 사건이 일어날 확률을 일어나지 않은 확률과 비교하여 표현합니다. 오즈가 2.0 이면? 일어날 확률이 그렇지 않을 확률보다 두배 더 큽니다.
  $P(X) = \frac{O(X)}{1+O(X)} \qquad O(X) = \frac{P(X)}{1-P(X)}$
- 백분율보다 더 직관적으로 설명 가능

# 확률의 Indepenency

- 사건 A가 발생할 때, 사건 B가 일어날 확률에 영향을 받지 않으면, 두 사건은 서로 독립적입니다.
- 이 때, A와 B가 동시에 일어날 확률은 각 사건이 일어날 확률의 곱으로 표현합니다.
  $P(A\,and\,B) = P(A) \times P(B)$
- 반대로 사건 A가 일어날 확률이 사건 B가 일어날 확률로부터 영향을 받을 때 두 사건은 독립적이지 않으며, 이 때 A와 B가 동시에 일어날 확률은 B가 일어났다는 가정하에 A가 일어날 확률에 B가 일어날 확률의 곱으로 표현합니다.
  $P(A\,and\,B) = P(A|B) \times P(B)$
- 이를 확률의 결합법칙이라고 합니다.

# 확률의 덧셈법칙

- 두 사건 중 하나라도 일어날 확률은 각각의 확률을 더한 뒤 동시에 일어날 확률을 빼줘야 합니다.
  - $P(A\,or\,B) = P(A) + P(B) - P(A|B) \times P(B)$

# 조건부 확률(Conditional Probability)

- B가 발생했다는 조건 하에서 A가 발생할 확률을 의미
- $P(AGivenB)$ 또는 $P(A|B)$
  - $\frac{P(A|B)=P(A\,and\,B)}{P(B)}$
- P(암|커피)
  - 커피가 암과 연관이 있는지 연구하려면, 커피를 마시는 사람이 암에 걸릴 확률(조건부 확률)을 구해야 함

# 베이즈 정리(Bayesian Rule)

- 사후 확률을 구하기 위한 규칙으로 사후 확률을 라이클리후드와 사전 확률의 곱으로 표현한 것
- "과거의 믿음(Prior)"을 "새로운 증거(Likelihood)"로 갱신하여 "최종 판단(Posterior)"을 내리는데 유용
  - $P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} = \frac{P(A\ AND\ B)}{P(B)}$
  - 사후확률$(Posteior) = \frac{\text{우도}(Likelihood) \times \text{사전확률}(Prior)}{\text{증거}}$
  - 우도: 관측된 데이터

# Maximum Likelihood Estimation (MLE, 최대우도추정)

- 관측된 결과를 가장 잘 설명하는 확률 찾기
- 관측된 데이터가 가장 일어날 법한(가능성 높은) 모델 파라미터 값 찾기
- 이항분포, 정규분포 등 특정 확률 분포를 가정하여 확률 값을 최대화 하는 파라미터를 찾는 것
  - 이항분포의 파라미터: 성공 확률 p
  - 정규분포의 파라미터: 평균과 분산
- 예시
  - 동전을 10번 던져서 앞면 7번, 뒷면 3번 나왔을 때, 이를 가장 잘 설명하는 앞면이 나올 확률($\theta$)을 MLE로 추정하면?
  - 한번 시행할 때의 확률
    - $p(x|\theta) = \theta^x(1-\theta)^{1-x},\ x \in 0, 1$

        - x는 0 또는 1의 값을 갖음
        - x=0 이면, $p(x|\theta) = 1 - \theta$
        - x=1 이면, $p(x|\theta) = \theta$
        - 이렇게 기재하는 이유는 x가 0이든 1이든 상관없이 한 줄로 스마트하게 표현할 수 있기 때문
    - 전체 데이터의 가능도
        - $L(\theta) = \prod_{i=1}^{n} p(x_i|\theta) = \theta^7(1-\theta)^3$
    - 계산 편의를 위한 Log 화
        - $log\, L(\theta) = 7\, log\, \theta + 3\, log\, (1-\theta)$
    - 미분을 이용해 최대값 구하기
        - $log\, L(\theta)$의 미분 결과가 0일 때의 $\theta$ 구하기
        - 기울기가 0(수평)일 때, 확률이 가장 높다고 봄
        - 자연로그 x의 미분값은 1/x
        - 자연로그 (1-x)의 x에 대한 미분값은 -1/(1-x)
        - $\frac{d}{d\theta} log\, L(\theta) = \frac{7}{\theta} + \frac{3}{1-\theta}$
        - $\frac{7}{\theta} - \frac{3}{1-\theta} = 0$
        - $\theta = \frac{7}{10}$

## 이항 분포

- 성공과 실패 두 가지 결과만 있는 독립적인 시행에 대한 확률 분포
    - $X \sim Binomial(n, p)$
    - 성공 확률 p
    - 실패 확률 1-p
    - 독립적인 시행 횟수 n
    - 다른 말로 "베르누이 실행"이라고도 함
- 이 때의 확률은? (k는 성공 횟수)
    - $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$
- 동전을 10번 던져 앞면이 7이 나왔을 때, MLE 찾기
    - $L(p) = P(X = 7 \mid n = 10, p) = \binom{10}{7} p^7 (1-p)^3$
    - 여기서, L(p)를 최대화 하는 p 값 찾기

## 베타 분포

# 3장 기술 통계와 추론 통계

- 기술 통계
    - 데이터를 요약
    - 평균 mean, 중앙값 median, 모드 mode, 차트, 종 곡선을 계산
        - 중앙값: 정렬된 값 집합 중 가장 가운데 값, 이상치로 평균을 신뢰할 수 없을 때 유용한 대안

        - 모드: 가장 자주 발생하는 값
- 추론 통계
    - 표본을 기반으로 모집단에 대한 속성을 발견
    - 일종의 유추

# 3-1 기술 통계

## 모집단

- 연구하고자 하는 특정 관심 그룹
- 분산
    - 편차의 제곱의 평균
    - $\sigma^2 = \frac{\sum(x_i-\mu)^2}{N}$
- 표준편차
    - 분산에서 제곱을 제거거
    - $\sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{N}}$

## 표본

- 모집단의 하위 집합
- 무작위, 편향되지 않음
- 모집단에 대한 속성을 추론하기 위한 목적
- 분산
    - $s^2 = \frac{\sum(x_i-\bar{x})^2}{n-1}$
- 표준편차
    - $s = \sqrt{\frac{\sum(x_i-\bar{x})^2}{n-1}}$
- n이 아닌 n-1로 나누는 이유는?
    - 표본의 편향을 줄이고 표본에 기반한 모집단의 분산을 과소평가하지 않기 위해
    - 분모에서 하나 작은 값을 사용함으로써 분산을 증가시키고 표본의 불확실성을 더 많이 포착

## 정규 분포(Normal Distribution), 가우스 분포(Gaussian distribution)

- $N(m, \sigma^2)$
- 평균 근처가 가장 질량이 크고, 대칭 형태를 띤 종 모양 분포
- 자연과 일상생활에서 일어나는 많은 현상이 이 분포를 따르고 있으며, 중심 극한 정리 덕분에 정규 분포가 아닌 문제도 표본의 크기가 충분히 크면 정규 분포를 따른다고 볼 수 있습니다.
- 인공신경망의 초기 가중치 값이 정규 분포를 따른다고 가정하며 이는 각 가중치를 무작위로 설정하여 학습을 가능하게 하는데 유용합니다.
- 이 분포의 퍼짐 정도 = 표준 편자

## 표준 정규 분포(Standard Normal Distribution)

- 평균이 0이고 표준 편차가 1이 되도록 정규 분포의 크기를 정규화한 분포입니다. 정규화를 통해 평균과 분산이 다른 정규 분포들을 쉽게 비교할 수 있습니다.
- 모든 x 값을 표준 편차, 즉 z 점수로 표현
  - $z = \frac{x-\mu}{\sigma}$

# 3-2. 추론 통계

## 중심 극한 정리(Central Limit Theorem, CLT)

- 모집단에서 충분히 많은 표본을 추출하면 해당 모집단이 정규 분포를 따르지 않더라도 표본의 평균이 정규 분포를 따른다.
- 단, 표본은 독립적이고 같은 분포에서 무작위로 추출되어야 합니다.(IID: Independently Identically Distributed) 된 Random Sample
- 특히, 모집단이 어떤 분포를 따르더라도 표본 평균의 분포는 무조건 정규 분포를 따르게 할 수 있다는 것이 가장 핵심이라고 생각합니다.
- 이 정규분포는 평균은 모집단의 평균과 같아지고, 분산은 모집단의 분산을 표본의 크기로 나눈 값과 같습니다.
  - 어떤 정규 분포? $N(\mu, \frac{\sigma^2}{n})$
  - $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$
    - 표본이 무한히 많아질수록 표본 평균의 표준편차는 모집단의 표준편차와 유사해짐

## 신뢰구간

- 표본으로 모집단을 추정할 때, 얼마의 신뢰도를 갖을 것인지를 측정
- 예를 들어 표본의 평균이 모집단의 평균의 특정 구간에 속한다고 얼마나 확신하는지?
- 예시
  - 표본 평균이 64.408이고 표준 편차가 2.05인 골든 리트리버 31마리의 표본을 기준으로 모집단 평균이 63.686에서 65.1296 사이에 있다고 95% 확신한다.
- $\pm 1.95996$ 은 표준 정규 분포의 중심에서 확률의 95%에해당하는 임계 z값

## p 값이란?

## 가설 검정

## t 분포: 소규모 표본 처리

# 4장 선형대수학

## 벡터란?

- 벡터는 데이터를 시각적으로 표현한 것으로 방향과 길이를 가지고 있습니다.
- 벡터간의 덧셈과, 스칼라(Scalar)곱을 이용해 벡터를 늘이거나 줄일 수 있습니다.
  - 덧셈의 결합법직: $\overrightarrow{v}$이동 후 $\overrightarrow{w}$ 이동 하든지, 그 반대로 하든지 상관 없음
- 스케일링

- 스칼라(Scalar)라는 하나의 값을 곱해 벡터를 늘이거나 줄임
  - 스칼라는 방향은 없고 크기만 있는 수(일반적인 실수나 복소수)
- 스케일을 조정해도 여전히 같은 선상에 존재함 = 선형종속

# Vector Space란?

- 8개의 공리를 만족하며, 덧셈과 스칼라곱 연산에 의해 항상 같은 공간 안에 머무는 벡터의 집합입니다.
- 8개의 공리는 덧셈과 스칼라곱에 대한 법칙으로 교환 법칙, 결합 법칙, 분배법칙, 항등원 존재 등이 있습니다.
- 대표적으로 linear subspace가 있으며, 벡터들의 linear combination으로 표현되는 space 입니다.
  - 덧셈 관련
    (1) 교환 법칙: u+v=v+u
    (2) 결합 법칙: (u+v)+w=u+(v+w)
    (3) 항등원 존재: 0 벡터가 존재하여 $v$+0=$v$
    (4) 역원 존재: v+(−v)=0
  - 스칼라곱 관련
    (5) 분배법칙 1: a(v+w)=av+aw
    (6) 분배법칙 2: (a+b)v=av+bv
    (7) 결합법칙: a(bv)=(ab)v
    (8) 항등원: 1·$v$=$v$

# 내적(Dot Product)이란

- 두 벡터가 얼마나 닮았는지를 표현하는 개념으로, 하나의 스칼라 값을 반환합니다.
  - $\begin{bmatrix}1\\3\end{bmatrix} \cdot \begin{bmatrix}4\\2\end{bmatrix}$ = 1 x 4 + 3 x 2 = 10
- 대수적으로는 각 성분을 곱하고 그 결과를 모두 더한 값이며,
- 기하학적으로는 하나의 벡터를 다른 벡터에 projection 하여 각 길이를 서로 곱한 것으로 방향이 유사할 수록 내적의 크기가 증가하며, 두 벡터가 서로 수직이면 내적이 0이 되고 서로 관계가 없음을 의미합니다.
  - a·b=∥a∥ ∥b∥cos(θ)
    - ∥a∥, ∥b∥: 벡터의 크기(길이)
- 인공지능에서는 데이터 간의 유사성을 구할 때 많이 이용됩니다. 예를 들어, Word2Vec에서 임베딩 모델에서 단어를 벡터로 표현한 뒤, 두 단어 벡터의 내적을 통해 의미의 유사성을 판별합니다.

# 스팬과 선형 종속

- 다른 두 개의 벡터에 덧셈과 스칼라곱을 적용하여 펼쳐지는 공간입니다.
  - 두개의 벡터가 서로 다른 방향을 향하고 있어, 선형 독립(Linearly Independent)이면 모든 영역을 스팬하고,
  - 같은 방향이나 동일 선상에 존재하면 선형 종속(Linearly Dependent)이기 때문에 제한된 영역만 스팬할 수 있습니다.
  - (하나의 벡터를 스케일링하여 다른 벡터를 만들 수 있을 때 선형 종속)

# 선형 변환(Linear Transformation)

- 선형성을 유지하면서 벡터 공간의 점들을 이동하는 것입니다.

- 선형성이란 벡터의 덧셈과 스칼라곱 모두를 만족한다는 것이다.
- 선형 변환은 크게 스케일, 회전, 반전(Transpose), 전단 등이 있다.
    - 스케일: 늘어나거나 줄어듬
    - 회전: 공간을 돌림
    - 반전: 공간을 뒤집어 $\hat{i}$과 $\hat{j}$의 위치를 바꿈(Transpose)
    - 전단: 특정 방향의 직선과의 거리에 비례하여 각 포인트를 이동

# 기저 벡터(Basis Vector)

- 어떤 벡터 공간을 완전히 표현할 수 있게 해주는 기초적인 벡터로, 길이가 1이고 서로 수직이거나 독립적인 방향을 갖는다.
    - 모든 벡터를 만들거나 변화하기 위한 구성 요소
    - $\hat{i} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \hat{j} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, 기저벡터 $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

# 행렬 벡터의 곱셈

- 행렬 벡터의 곱셈은 벡터를 선형 변환하여 새로운 벡터를 만드는 것으로 Ax = b 에서 행렬 A가 변환을, 벡터 x는 입력을 의미하며, Ax는 변화 후의 새로운 벡터를 나타낸다.
- 행렬 A의 각 행벡터가 입력 벡터 x와 내적하여 새로운 원소를 만드는 것이다.
- 즉, 행렬의 곱셈은 벡터 공간에 여러 개의 변환을 적용하는 것
    - 순서: 가장 안쪽부터 바깥쪽으로 변환을 적용
    - 행렬A * 행렬B * 행렬C = (행렬A * 행렬B) * 행렬C = 행렬A * (행렬B * 행렬C)

# Matrix의 4가지 Fundamental Vector Spaces

- $A \in \mathbb{R}^{m \times n}$이고 $y = Ax$ 일 때
    - n: 입력 벡터의 차원 (열의 수)
    - m: 출력 벡터의 차원 (행의 수)
    - A: 입력 벡터 x를 받아 출력 벡터 y로 보내는 선형 변환
    - x: 입력공간 $\mathbb{R}^n$의 벡터
    - y: 출력공간 $\mathbb{R}^m$의 벡터

1. 열공간 (Column Space) = $C(A)$ or $range(A)$
    - 행렬 A의 행백터들이 span하는 공간으로 A와 입력 벡터 x의 선형 결합이 만들어내는 출력 공간을 의미한다.

$$A = \begin{bmatrix} | & | & | \\ a_1 & a_2 & a_3 \\ | & | & | \end{bmatrix} \in \mathbb{R}^{m \times 3}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$Ax = x_1 a_1 + x_2 a_2 + x_3 a_3$$

    - 예시:클래스 확률 벡터, 회귀 값, 문장 임베딩 등

2. 행공간 (Row Space) = $C(A^T)$
    - 행공간은 A의 **행벡터들(row vectors)**이 span하는 공간으로 입력 벡터 x는 행공간 내에서 작동한다.
        - 각 행벡터는 입력 벡터 x와의 내적을 통해 출력값의 한 요소를 만듦
    - A의 행들을 열처럼 생각해서 만든 $A^T$의 열공간
    - 예시: 이미지 벡터 (784차원), 텍스트 임베딩 벡터 등
3. 영공간 (Null Space) = $N(A)$
    - Ax=0(출력이 0이 되는)을 만족하는 모든 벡터 x의 집합으로 입력 공간인 행공간과 직교하는 형태를 갖는다.
        - A의 행공간과 입력 벡터가 만날 수 없어, 출력이 없음
    - 어떤 입력 x가 영공간에 속한다는 것은 행렬 A를 통해 아무 정보도 전달하지 못한다는 것을 의미한다.
4. 좌영공간 (Left Null Space) = $N(A^T)$
    - $A^Ty=0$ 을 만족하는 모든 출력 벡터 y의 집합으로 출력공간인 열공간과 직교하는 형태를 갖는다.
    - 행렬 A의 열공간에 출력 벡터가 존재하지 않기 때문에 선형변환으로 출력 벡터를 구할 수 없고, 입력 벡터 x의 해를 찾을 수 없다.

## Let $A \in R^{m\times n}$. Prove that if the rank of $A$ is $r$, then the dimension of its null space is $n-r$.

- Rank-Nullity 정리에 따르면, 행렬 A의 열 개수 $n$은 랭크 $r$과 널 스페이스의 차원의 합입니다. 따라서 널 스페이스의 차원은 $n-r$입니다.
- Rank-Nullity 란?
    - rank($A$) + nullity($A$) = 열의 수
    - A가 어떤 선형 변환을 나타낼 때, 출력 벡터 공간에 영향을 미치는 방향들과 입력은 있지만 결과가 0이 되는 방향들의 합은 항상 전체 입력 공간(열의 수) $n$을 완전히 설명해야 함

## Orthogonal Matrix(직교 행렬) 이란?

- 어떤 행렬이 자기 자신을 전치한 행렬과 곱해서 항등 행렬이 될 때, 직교행렬이라고 한다.
- 직교 행렬의 열 벡터는 서로 직교해야 하며, 열 벡터의 길이 1로 정규화 되어 있어야 한다.

(추가 설명)

- 다음 조건을 만족
    - $Q^TQ=QQ^T=I$
    - 즉, 전치 행렬이 곧 역행렬
        - $Q^{-1}=Q^T$
- 특성
  1: 열 벡터끼리 서로 직교
  2: 각 열 벡터의 길이(크기)가 1(정규화된 단위 벡터)
    - $[[1,-1],[1,1]]$은 열 벡터가 서로 직교 하지만, 벡터(1,1)의 크기가 1이 아니라, $\sqrt{2}$이 때문에, 모든 원소를 $\sqrt{2}$로 나누어 정규화하면 직교 행렬이 됨
- 중요성

| 이유 | 설명 |
| --- | --- |
| ✅ 길이·각도 보존 | 원형 유지 |
| ✅ 역행렬 계산 쉬움 | $Q^{-1} = Q^T$ |
| ✅ 수치 안정성 | 머신러닝·수치해석에서 강력 |
| ✅ 해석 가능성 | PCA, SVD 등에서 주축 |
| ✅ 정보 손실 없음 | 선형 압축/변환에 유리 |

# 랭크(Rank)

- 행렬에서 컬럼 스페이스의 차원의 수로, 선형 독립적인 열벡터의 수를 의미합니다. 이는 선형 독립적인 행벡터의 수와 동일합니다.
- 랭크는 행렬이 담고있는 정보의 차원 수를 의미하기 때문에, 랭크가 높을수록 데이터가 더 "다양한 방향성"을 가진다고 볼 수 있습니다.
    - 데이터 행렬의 랭크
        - $X \in \mathbb{R}^{n \times d}$
        - n: 샘플 수(행), d: 특성 수(열)
        - rank(A)=d: 모든 특성이 독립적 → 완전한 차원 정보
        - rank(A)<d: 특성 간 중복 존재 → 차원 축소 가능 (PCA 적용 가능)
        - rank(A)<<d: 특성이 과도하게 중복됨 → 고차원 과적합 가능성↑

# 행렬식(Determinant)

- 행렬식은 벡터들로 형성된 영역의 면적이나 부피를 의미합니다.
- 또한 역행렬의 존재 여부를 나타내기도 하는데, $det(A) = 0$ 이면, 선형 종속의 발생으로 차원이 축소되고 Ax=b를 만족하는 x의 유일한 해가 존재하지 않음을 의미합니다.
    - x가 없거나: 열공간에 없음
    - 무수히 많음: 예를 들어 직선 위의 모든 점

(추가 설명)

- 행렬식은 그 변환이 단위 부피를 얼마나 키우거나 줄였는지를 나타냄. 행렬식이 0이면, 공간이 더 작은 차원으로 축소됨을 의미
- 선형 변환을 통해 공간이 찌그러질 경우 정보가 소실되고 이를 되돌릴 수 없음
- $det(A) \neq 0$ → 열벡터가 선형 독립 → full-rank → 역행렬 존재
- $det(A) = 0$ → 열벡터가 선형 종속 → rank-deficient → 역행렬 존재 안 함

🔹 **(2) 역행렬 공식 (2×2 행렬 예시)**

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \Rightarrow A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

여기서 $\boxed{ad - bc = \det(A)}$ 입니다.

- 만약 $\det(A) = 0$이면, 역행렬은 **0으로 나누는 꼴**이 되어 **정의되지 않습니다.**

# 항등 행렬(Identity Matrix)

- 곱해서 자기자신을 결과로 출력하게 하는 행렬로, 대각선의 값이 1이고 다른 값은 0인 정사각 행렬(가로 세로 길이 동일) 입니다.
- 항등 행렬을 만들 수 있다면 선형 변환을 취소하고 원래 기저 벡터를 찾을 수 있습니다.
- $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
-

# 역행렬(Inverse Matrix)

- 행렬의 변환을 취소하는 행렬($A^{-1}$)로
- $A^{-1}$과 $A$를 곱셈하면 항등 행렬이 됩니다.

# 역행렬을 구하는 원리는?

- 여러가지 방법이 있지만 가우스 조던 소거법으로 구할 수 있다.
  - 정사각행렬 $A$의 오른쪽에 항등 행렬 $I$를 붙여 확장 행렬 $[A \mid I]$을 만듦
    $\rightarrow$ 가우스-조던 소거법으로 $A$를 항등 행렬 $I$로 변경
    $\rightarrow$ 원래 $I$였던 부분은 $A^{-1}$로 변함
    $\rightarrow [I \mid A^{-1}]$ 가 만들어짐

# 역행렬을 구하는 시간복잡도는?

- 만약 가우스 조던 소거법을 이용한다면, 빅O 노테이션으로 $n^3$이 되는 것으로 알고 있습니다.

# 고유값(eigenvalue), 고유 벡터(eigenvector)

- 행렬 A와 열벡터 x를 곱하는 행위를 선형 변환이라고 할 수 있는데, 선형 변환 후 방향이 변하지 않는 벡터를 아이겐벡터라고 한다.
  - 이 때, 크기가 변한 정도를 고유값이라고 합니다.
- $Ax = \lambda x$
  - $\lambda$: 고유값
  - 벡터 $x$: 고유벡터
- 고유값을 구하는 방법?
  - $det(A - \lambda I) = 0$ 을 만족하는 $\lambda$의 값 찾기
- 고유벡터를 구하는 방법?
  - $(A - \lambda I)x = 0$ 을 만족하는 벡터 $x$ 찾기

    - 즉, $(A-\lambda I)x$의 Null Space(영공간) 찾기

(추가 설명)

- 고유값이 존재한다는 것은?
    - 어떤 정방 행렬 A가 작용할 때 **방향은 유지되면서 크기만 변하는 벡터(고유벡터)**가 존재한다는 의미
    - 즉, 특정한 방향을 보존하는 축 또는 성분이 존재한다는 증거
    - 선형 변환 A이 벡터 공간에서 특정 방향을 유지하면서 늘리거나 줄이는 축이 존재
    - 데이터를 구성하는 주된 방향성이 존재하며, 이를 기반으로 분석·해석할 수 있다는 뜻
- 왜 정방 행렬이어야 하는가?
    - **행렬식(Determinant)**은 정방행렬에서만 정의됨
        - 비정방행렬(예: 3×2)에는 행렬식이 존재하지 않음
    - 정방행렬만이 입력과 출력 차원이 같음
        - Av=λv 에서 좌변과 우변의 차원이 같아야 비교 가능
- 비정방 행렬도 고유값을 가질 수 있나?
    - 직접적인 고유값 정의는 어렵지만, SVD를 이용하여 고유값과 유사한 개념을 사용할 수 있음. 바로, 특이값(singular values)

# 고유값 분해(Eigendecomposition)

- 어떤 정방방행렬 A를 고유벡터를 열벡터로 갖는 행렬과 고유값으로 만든 대각행렬로 분해하는 것
- 이 때, 행렬 A는 $det(A-\lambda I)=0$를 구할 수 있는 정방행렬이어야 하며
- 고유벡터가 선형 독립이어야 함(그렇지 않으면 공간이 찌그러짐)
- 고유값 분해를 통해 **복잡한 선형 변환 $A$를 단순한 대각 행렬 Λ**로 변환해 계산을 쉽게할 수 있음
- $A=V\Lambda V^{-1}$
    - $V$: 고유벡터들을 열벡터로 갖는 $n$×$n$ 행렬
    - Λ: 고유값을 대각선에 가지는 대각행렬 (diagonal matrix)
    - $V^{-1}$: $V$의 역행렬

(추가 설명)

- 즉, 복잡한 선형 변환을 단순한 스케일(배수) 변환으로 표현
- A는 각 고유벡터 방향으로 단순히 고유값만큼 스케일링하는 작용
- 연산이 쉬워지며, PCA 등에 적용 가능

# PCA 란?

- Principal Component Analysis의 준말로, 우리 말로는 주성분 분석입니다.
- 고차원 데이터를 더 낮은 차원으로 줄이면서 핵심 정보(분산이 큰 방향)를 최대한 유지하는 차원 축소(dimensionality reduction) 기법
- 용량이 큰 데이터를 처리할 때, 차원을 축소하면 불필요한 노이즈를 제거하고 메모리 사이즈를 줄여 처리 속도를 향상하는 데 유용합니다.
- 방법은 데이터의 분포를 가장 잘 설명하는 축을 찾고, 그 축으로 데이터를 정사영 내리는 것

- 데이터의 분포를 가장 잘 설명하는 축은 공분산 행렬(코베리언스 메트릭스)의 고유벡터(아이겐벡터)
    - 고유값이 큰 방향이 가장 정보가 많은 주성분
- 공분산 행렬에 고유값 분해를 적용

(추가 설명)

풀이 순서

1. 평균 정규화
    1. 데이터의 평균을 0이 되도록 이동
    2. 각 열을 행의 수로 나눈 다음, 각 열의 평균을 각 열의 원소에서 뺌
    3. 이후 각 열을 더하면 0이 됨
2. 정규화된 행렬의 공분산 행렬 계산
    1. $C = \frac{1}{n-1} X^{rT} X$
    2. n은 샘플의 개수
    3. 왜 공분산을 구하지?
        1. 데이터의 분산은 정보량을 의미
        2. 공분산은 두 변수의 선형적 관계를 수치로 나타냄
        3. 공분산 결과에 따른 해석
            1. 양수: 한 변수가 증가할 때 다른 변수도 증가하는 양의 상관 관계
            2. 음수: 한 변수가 증가할 때 다른 변수는 감소하는 음의 상관 관계
            3. 0 근처: 두 변수는 거의 무관하거나 선형적 관계가 없음
3. 공분산에 대한 고유값 분해
    1. 고유값의 크기가 큰 고유벡터 산출 = 주성분 방향
4. 정규화된 행렬를 주성분 방향으로 Projection
5. PCA 결과 해석: 2차원 데이터를 1개의 주성분으로 축소한 결과 원본 데이터의 약 79% 정보를 설명 가능
    1. $79 = \frac{\text{큰고유값}}{(\text{고유값들의합})}$

# SVD 란?

- Singular Value Decomposition, 특이값 분해
- 어떤 행렬이든 3개의 행렬(U, Σ, V^T)로 분해하는 기법입니다.
- 시그마는 A의 정보량(특이값)을 담고 있는 대각행렬이고, 유는 A의 열방향 구조를 나타내는 직교 행렬, 브이는 A의 행방향 구조를 나타내는 직교 행렬이다.
- 고유값 분해(Eigendecomposition)는 정방행렬에서만 가능하지만 SVD는 모든 실수 행렬에 대해 항상 존재하기 때문에 보편적으로 적용할 수 있다.
- 왜 분해하는가?
    - 데이터에서 핵심 패턴만 남기고 → 불필요한 차원이나 잡음 제거
- 모든 실수 행렬 $A \in \mathbb{R}^{m \times n}$ 에 대해 다음과 같은 분해가 존재
    - $A = U \sum V^T$
    - $U$: 좌측 직교 행렬(열: $A$의 열공간 기저), m x m

    - $\sum$: 특이값을 포함하는 대각 행렬, m x n
    - $V^T$: 우측 직교 행렬(열: $A^TA$의 고유벡터), n x n
- $x$가 $A$를 통과하여 $U\sum V^Tx$가 되었다는 의미는?
    - **기준 변경 $\rightarrow$ 크기 조정 $\rightarrow$ 다시 회전**
    - 1. $x$를 $V^T$로 회전하여 기준 축 변경
    - 2. $\sum$로 각 축을 스케일
    - 3. $U$로 다시 회전하여 출력 공간으로 이동

(추가 설명)

- 특성1. $A^TA$와 $AA^T$는 무조건 symmetric
    - Why? $(A^TA)^T = A^TA$ 이고 $(AA^T)^T = AA^T$ 이기 때문
    - What is symmetric? $A = A^T$
    - So? symmetric 이면 뭔가 고유값 분해($V\Lambda V^T$)처럼 할 수 있다는 뜻
- 특성2. $A^TA$와 $AA^T$는 무조건 정방행렬(square)
- <중요> non-zero singular value의 수 = rank(A)

(풀이)

- $A = U\sum V^T$, $A$는 $m \times n$
- $U$를 구하려면?
    - $AA^T = U\sum V^T(U\sum V^T)^T$
    - $AA^T = U\sum V^TV\sum^T U^T$
    - $AA^T = U(\sum\sum^T)U^T$
- $V$를 구하려면?
    - $A^TA = V(\sum^T\sum)V^T$
- $\sum\sum^T$는 m x m 행렬
- $\sum^T\sum$는 n x n 행렬
- $\sum$에서 $\sqrt{\sigma^2}$의 결과인 $\sigma$는 항상 양수로 가정

# 5장 선형 회귀(Linear Regression)

## 개념

- 독립 변수(x)와 종속 변수(y)간의 상관관계를 분석하여, 어떤 독립 변수에 대한 종속 변수를 예측하는 통계적 기법
- 예측값과 실제값 사이의 오차를 최소화하는 직선 또는 평면(고차원일 경우)을 찾는 것이 핵심
- 데이터를 가장 잘 대변하는 최적의 선을 찾은 과정
- 주로 예측 알고리즘에서서 활용

## 수식

- 단순 선형회귀(독립변수 1개)
    - $y = wx + b$

    - y: 예측값, x: 입력값(독립변수), w: 기울기 (slope, 계수), b: 절편 (bias, y절편)
- 다중 선형회귀(독립변수 여러개)
    - $y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \cdots + w_nx_n$
    - 계산 시 행렬을 이용

## 최적화 과정

- 주어진 데이터에 대해 오차(잔차)의 제곱합이 최소가 되도록 w와 b를 찾는 것
- $Loss(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

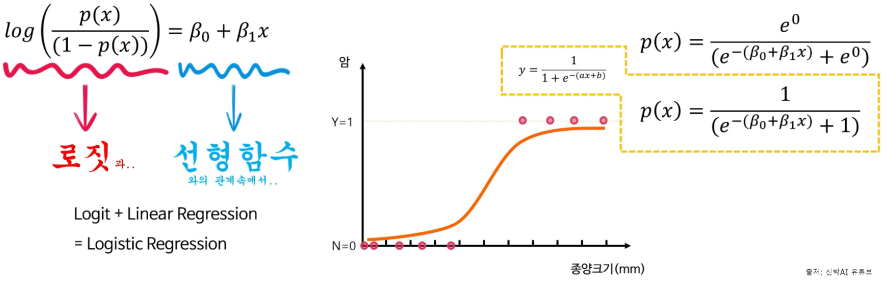
# 6장 로지스틱 회귀

## 개념

- 이진 분류를 위한 방법
    - 입력값에 대해 0 또는 1 (혹은 두 클래스 중 하나)에 속할 확률을 예측
- 로짓(logit)을 입력 변수의 선형 함수 표현
- 로짓이란 오즈(승산도)에 로그를 취한 것
- 로짓을 시그모이드 함수에 통과시킨 결과가 0.5 보다 크면 1로, 그렇지 않으면 0으로 분류

## 수식

- $\sigma(z) = \frac{1}{1+e^{-z}} = q(\text{확률})$
    - $z = w^Tx + b$
    - $\sigma(z) \in (0, 1)$: 항상 0과 1사이의 실수값을 출력
- 선형함수의 결과 = 로짓(오즈에 로그를 취함) 이라는 가정이 핵심
    - P(x)는 확률이기 때문에 0과 1 사이
    - P(x)를 로짓으로 표현하면 x에 대한 선형 함수가 나타남




## 최적화 과정

- Loss 함수 정의: Binary Cross Entropy
    - 시그모이드 함수를 통과한 결과인 q를 최대한 키우자!(아무리 키워도 확률이라 어차피 0과 1 사이)
    - 즉, 강아지 사진(y=1)일 확률과 고양이 사진(y=0)일 확률을 동시에 가장 크게하는 함수를 하나 정의하면?
        - $P(q_i) = q_i^{y_i}(1-q_i)^{(1-y_i)}$
    - 여러 개의 사진을 동시에 테스트 하면?

        - $\prod_i^n P(q_i) = \prod_i^n q_i^{y_i}(1-q_i)^{(1-y_i)}$
    - log를 취하면?
        - $\sum_i^n log\, q_i^{y_i}(1-q_i)^{(1-y_i)}$
        - 이것을 Maximize 해야 함
    - Loss로 정의하면?
        - $-\sum_i^n log\, q_i^{y_i}(1-q_i)^{(1-y_i)}$
        - 이것을 Minimize 해야 함
- 이 Loss 함수를 w와 b로 미분한 기울기에 learning rate를 곱하여, w와 b를 조정하면서 Loss를 최소화 하는 것이 목표

# 7장 신경망

# OOP 객체 지향 프로그래밍이란?

- 데이터나 메서드 등의 객체를 중심으로, 여러 객체가 서로 상호작용하도록 프로그램을을 구성하는 방식이다. 객체 지향 프로그램은 코드의 재사용성, 유지보수성, 확장성이 높다.
- 상속, 추상화 같은 기법이 있다.

# C와 파이썬의 차이

- C는 기계어에 가까운 저수준의 언어로 개발자가 메모리를 직접 관리하고 변수 선언 시 데이터 형식을 직접 명시해야 하는 등 문법이 복잡하지만 실행 속도가 빠르며,
- 파이썬은 인간 언어에 가까운 고수준의 언어로 자동으로 메모리를 할당하고, 동적으로 데이터 타입이 선언되는 등 문법인 간결하고 쉬우나, C 보다는 속도가 느리다.

# 리스트와 딕셔너리의 차이

- 리스트는 특정 순서로 데이터를 나열한 자료 구조이고 인덱스로 접근하며
- 딕셔너리는 Key와 Value의 쌍으로 데이터가 나열된 자료구조이고 키로 접근한다.

# 배열과 리스트의 차이

- 배열과 리스트는 자료를 순서대로 나열한다는 공통점이 있지만, 배열은 동일한 타입의 자료만 담을 수 있고, 리스트는 서로 다른 타입의 자료도 담을 수 있다. 리스트 보다는 배열의 계산이 더 빠르다.

# Linked List 란?

- 데이터를 저장할 때, 각 노드가 자신의 값과 다음 노드의 주소를 함께 저장하는 자료구조

# Linked List와 배열의 차이?

| 항목 | 배열 (Array) | 연결 리스트 (Linked List) |
|---|---|---|
| **메모리 구조** | 연속된 메모리 공간에 저장 | 노드들이 포인터로 연결된 구조 |
| **메모리 할당** | 정적(크기 고정) 또는 동적 (ex. Python list) | 동적 (필요할 때마다 노드 생성) |
| **접근 속도** | 임의 접근 `O(1)` → 빠름 | 순차 접근 `O(n)` → 느림 |
| **삽입/삭제 (중간/앞)** | 느림 `O(n)` (이동 필요) | 빠름 `O(1)` ~ `O(n)` (포인터 변경) |
| **삽입/삭제 (끝)** | 빠름 (Python list 기준 `append`) | 단일 연결리스트는 느림 (`O(n)`), 이중 연결리스트 또는 tail 포인터 있으면 `O(1)` |
| **메모리 사용** | 요소만 저장 (공간 효율적) | 데이터 + 포인터 필요 (오버헤드 있음) |
| **캐시 친화성** | 좋음 (연속 메모리, CPU 캐시에 잘 맞음) | 나쁨 (비연속 메모리, 캐시 효율 낮음) |
| **정렬/검색** | 이진 탐색 가능 `O(log n)` | 이진 탐색 불가 (선형 탐색만 가능) |

| 항목 | 배열 (Array) | 연결 리스트 (Linked List) |
| --- | --- | --- |
| **사용 예** | 이미지 배열, 행렬 계산 등 연속 데이터 | 큐, 스택, 트리, 그래프 등 동적 구조 |

| 상황 | 추천 구조 |
| --- | --- |
| 빠른 인덱스 접근이 필요할 때 | 배열 |
| 삽입/삭제가 빈번할 때 | 연결 리스트 |
| 메모리 크기를 예측 가능할 때 | 배열 |
| 유동적으로 크기가 변할 때 | 연결 리스트 |
| 캐시 성능이 중요한 연산일 때 | 배열 |

# 포인터 란?

- 변수나 데이터가 저장된 메모리의 주소를 저장하는 변수, 즉 값이 저장된 위치(주소)를 저장
- 값 자체가 아닌, 값이 어디 있는지를 가리킴

# 시간복잡도 $O$ 란?

- 입력 크기 n이 커질 때, 알고리즘이 수행하는 연산 수의 증가율을 나타내는 척도
- 빅오 표기법(Big-O)으로 표현하며, 최악의 경우에 대한 연산량을 제공

| 시간복잡도 | 설명 | 예시 |
| --- | --- | --- |
| **O(1)** | 상수 시간, 입력 크기와 무관 | 배열 접근 arr[5] |
| **O(log n)** | 로그 시간, 입력이 반씩 줄어듦 | 이진 탐색 |
| **O(n)** | 선형 시간 | 단일 for문 |
| **O(n log n)** | 로그 성분 포함된 선형 | 병합정렬, 퀵정렬 평균 |
| **O(n²)** | 이중 반복문 | 버블 정렬, 삽입 정렬 |
| **O(2ⁿ)** | 지수 시간 | 피보나치 재귀 |
| **O(n!)** | 팩토리얼 시간 | 순열 생성 |

# 멀티쓰레드(Multi-thread)와 멀티프로세스(Multi-process)의 차이

✅ 1. 멀티스레드(Multi-thread)

📌 정의:

하나의 프로세스 내에서 여러 개의 **스레드(Thread)**가 생성되어 작업을 병렬로 수행하는 방식.

```
모든 스레드는 메모리(코드, 데이터, 힙)를 공유함. 효율적인 자원 사용과 빠른 통신 가능
```

💡 예:

워드 프로세서에서 자동 저장, 타이핑, 맞춤법 검사를 동시에 실행

✔ 장점:
스레드 간 데이터 공유가 쉬움
자원 소모가 적음 (프로세스보다 메모리 부담 적음)
컨텍스트 스위칭 비용이 작음
IO-bound 작업에 유리
- 입출력(I/O, Input/Output) 때문에 작업 속도가 제한되는 경우(디스크에서 파일 읽기/쓰기, 웹 요청 보내고 응답 기다리기)

❗ 단점:
하나의 스레드에서 오류가 발생하면 전체 프로세스가 죽을 수 있음
**동기화 문제(데이터 충돌)**가 발생하기 쉬움 → 락(lock) 관리 필요

✅ 2. 멀티프로세스(Multi-process)

📌 정의:
여러 개의 **독립된 프로세스(Process)**가 각각 자신만의 메모리 공간을 사용하며 실행
각 프로세스는 완전히 별개로 동작함 → 진정한 병렬 처리 가능

💡 예:
크롬 브라우저의 각 탭은 별도 프로세스로 실행됨 (하나 죽어도 나머지는 영향 없음)

✔ 장점:
하나의 프로세스가 죽어도 전체 시스템에는 영향 없음
병렬 처리가 안전함 (메모리 공유 안 해서 충돌 적음)
CPU-bound 작업에 매우 효과적
- CPU 계산이 집중되는 작업에서 특히 좋은 성능(수학 연산 (큰 수의 소인수분해), 대규모 행렬 연산, 선형대수 계산, 머신러닝 모델 학습, 이미지 처리, 영상 인코딩)

❗ 단점:
프로세스끼리 독립되기에 프로세스 사이에서 공유할 자원이 있다면 복잡하고 느린, 프로세스 간 통신 메커니즘(IPC)을 활용(IPC, Inter-Process Communication)

자원 소비가 많음 (각자 메모리, 스택 등 독립적으로 할당)

# 스택메모리와 힙메모리의 차이

- 작고 빠른 스택에는 주소만 저장하고, 크고 유동적인 데이터는 힙에 따로 저장하면,프로그램이 더 빠르고, 메모리 관리가 쉬움

| 항목 | 스택 메모리 | 힙 메모리 |
|---|---|---|
| **주요 용도** | 함수 호출 시 지역 변수 저장 | 동적 메모리 할당, 참조형 객체 저장 |
| **할당 방식** | 컴파일 타임 또는 함수 호출 시 자동 할당 | 실행 중(runtime) 동적 할당 |
| **해제 방식** | 함수 종료 시 자동 해제 (LIFO) | 개발자 또는 가비지 컬렉터에 의해 해제 |
| **데이터 크기** | 작고 고정된 크기 (정적 데이터) | 크기 제한이 없고 유연 (가변 데이터) |
| **접근 속도** | 빠름 (메모리 바로 접근) | 느림 (포인터로 간접 접근) |

| 항목 | 스택 메모리 | 힙 메모리 |
| --- | --- | --- |
| **저장되는 데이터** | 지역 변수, 함수 매개변수, 참조 주소 | 리스트, 딕셔너리, 클래스 인스턴스 등 실제 참조형 데이터 |
| **주소 저장 여부** | 실제 값 또는 참조(주소) 저장 | 참조된 값 자체가 저장됨 |
| **메모리 구조** | 선입후출(Last In, First Out) | 자유롭게 할당 (Free List 기반) |
| **대표 오류** | 스택 오버플로우(Stack Overflow) | 메모리 누수(Memory Leak) |
| **언어 처리 예** | Python에서 지역 변수 등 자동 처리 | Python의 객체, 리스트, 클래스 등 모두 힙에 저장됨 |

# 리스트와 튜플의 차이

- 리스트와 튜플 모두 순서대로 나열된 자료구조
- 리스트는 변경할 수 있으나, 튜플은 변경 불가
- 단, 튜플이 가변적인 리스트를 변수로 가지고 있으면, 그 리스트는 변경 가능

# 클래스와 인스턴스의 차이

- 클래스는 일종의 설계도로 속성과 메서드로 구성
- 인스턴스는 클래스를 호출하여 실체화 한 것

# 변수의 SCOPE

- 크게 전역 변수(global)와 지역 변수(local)로 구분
- 전역 변수는 특정 클래스 전체에서 사용할 수 있고
- 지역 변수는 해당 메서드 에서만 사용할 수 있음

# 딕셔너리와 해시테이블의 차이

- 딕셔너리는 해시테이블을 기반으로 파이썬에서 고도화한 자료구조
- Key와 Value의 쌍으로 이루어져 있고
- 더 나은 점: 메모리 최적화된 구현, 충돌 해결 방식 최적화 등

# for문과 while문의 차이

- 둘 다 반복문 이지만,
- for 문은 정해진 회수를 반복하고
- while 문은 특정 조건 하에서 계속 반복

# 재귀함수란?

- 자기자신을 호출하는 함수로 특정 조건 하에서 종료됨
- 예를 들어, heap 자료구조에서 삽입이나 삭제가 발생할 때, max나 min 등 조건을 만족할 때까지 heapify()라는 순서 정렬 함수를 구현하여 계속 호출

# 예외 처리란 무엇이며, 왜 필요한가?

- 프로그램의 비정상적 종료를 방지하며
- 예외가 발생할 때 사용자에게 의미있는 메시지를 전달함
- 보통 try~except 구분으로 작성

# deep copy와 copy의 차이

- copy()는 얕은 복사로, 객체는 새로 만들지만 내부 참조 객체는 공유함
- deepcopy()는 객체와 내부 참조까지 전부 새로 복사, 완전 독립적
- 얕은 복사에서, 내부 객체를 변경하면 원본도 같이 바뀔 수 있음
  import copy

```
# 원본 리스트
original = [[1, 2], [3, 4]]

# 얕은 복사
shallow = copy.copy(original)

# 깊은 복사
deep = copy.deepcopy(original)

# 내부 객체 수정
shallow[0][0] = 100

print(original)  # [[100, 2], [3, 4]] → 원본도 같이 바뀜
print(deep)      # [[1, 2], [3, 4]] → deepcopy는 영향 없음
```

# 모든 클래스는 초기화 함수로 무엇을 사용하는가?

- 모든 클래스는 객체 생성 시 `__init__()` 메서드를 초기화 함수로 사용
- 객체가 생성될 때 **자동으로 호출되며**, 별도로 호출하지 않아도 됨
- 이 함수 안에서 정의된 속성값들은 객체 생성 시 자동으로 설정됨

# 상속, 추상, 인터페이스란?

- 객체 지향 프로그래밍(OOP)의 핵심 개념
- 구조적 설계, 코드 재사용, 확장성, 다형성을 지원하기 위해 사용
- 상속: 자식 클래스는 부모 클래스의 속성이나 메소드을 그대로 상속받아 사용하거나, 오버라이딩하여 재정의. 주로 super() 함수로 호출
- 추상: 추상 클래스는 하나 이상의 추상 메서드(정의만 있고 구현은 없는 메서드)를 가진 클래스로 직접 객체를 생성할 수 없고, 상속받은 자식 클래스에서 추상 메서드를 반드시 구현해야 함
- 인터페이스: 인터페이스 클래스는 모든 메서드가 추상 메서드로만 구성된 일종의 추상 클래스로 **속성 없이, 오직 메서드 정의(명세)**만으로 구성됨
  - 다형성 이란? 같은 이름의 메서드가 다른 객체에서 다르게 동작

# numpy의 axis 연산이란?

- axis는 다차원 배열에서 연산이 적용되는 축
- 특정 행렬에 대해 axis를 기준으로 연산을 할 수 있다.
- 예를 들어 행렬 A가 $m \times n \times k$ 이고, axis=0 이라면, axis=0을 축으로 수행된다.
- 즉, $m$의 각 원소에 대해 $n \times k$ 끼리 연산이 이루어진다.

| `axis` 값 | 의미 | 연산 방향 |
|---|---|---|
| `axis=0` | **행을 따라 연산 (열 기준으로 집계)** | 세로 방향 (↓) |
| `axis=1` | **열을 따라 연산 (행 기준으로 집계)** | 가로 방향 (→) |
| `axis=2` | 3차원 이상의 배열에서 **3번째 축을 따라 연산** | 깊이 방향 |

# numpy의 sum, max, armax 란?

- np.sum() 은 합계
- np.max() 는 최대값
- np.argmax() 는 최대값의 인덱스

# 람다 함수란?

- 익명 함수를 만드는 방법으로, 한 줄로 간단한 함수를 정의할 때 사용
- "lambda 인자: 반환값_표현식" 형태로 표현되며, return 문은 사용하지 않지만, 콜론 뒤 표현식의 결과가 자동으로 반환
- 인공신경망을 학습할 때, 가중치 W에 대한 손실함수의 미분을 구하기 위해  "lambda W: 손실함수" 와 같이 사용한다.

# FCNN 신경망 이란?

- 완전 연결 신경망, Fully-Connected Neural Network
- 인공신경망의 기본적인 형태로, 각 층의 모든 노드가 다음 층의 모든 노드와 연결된 구조
- 입력층, 하나 이상의 은닉층, 출력층으로 구성되며, 각 연결마다 가중치가 존재

# 손실함수 란?

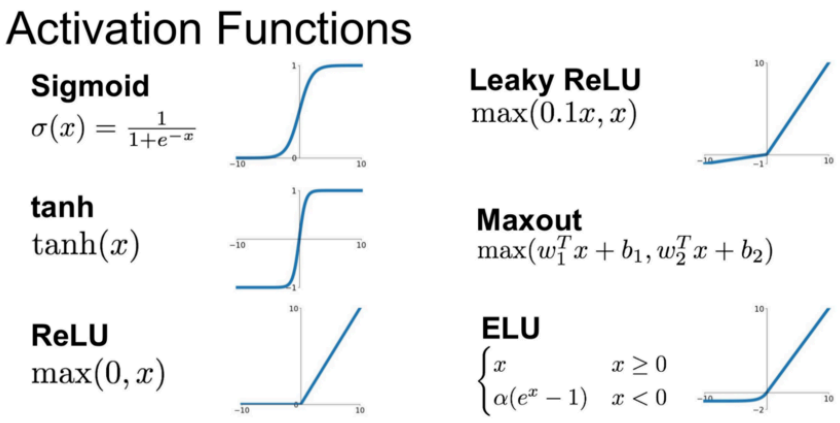
- 모델이 예측한 값과 실제 정답 사이의 차이를 수치화 하는 함수
- 회귀 문제에서는 평균제곱오차(Mean Squared Error),
- 분류 문제에서는 교차 엔트로피 손실(Cross-Entropy Loss)을 주로 사용
- 이 값을 최소화하는 방향으로 가중치를 업데이트

# 경사하강법 이란?

- 신경망이 손실 함수를 최소화하기 위해 사용하는 대표적인 최적화 알고리즘
- 손실 함수의 기울기, 즉 **그레이디언트(gradient)**를 계산하여, 손실함수를 최소화하는 방향으로 가중치를 조금씩 조정한다.
- 이 때, 학습률(learning rate)은 한 번에 얼만큼 가중치를 조정할 것인지를 결정한다.

# 활성화 함수란?

- 비선형 함수로 인공신경망에서 각 뉴런의 출력값을 받아 다음 뉴런에 전달하는 역할을 하는데, 비선형성을 통해 신경망의 표현력을 높인다.
    - 비선형성이 없으면 결국 하나의 선형 함수와 동일한 효과
- 대표적으로 ReLU(Rectified Linear Unit), Sigmoid, 하이퍼볼릭Tanh 등이 있다.




# 정규화 기법이란?

- Dropout, L2 Regularization 등은 과적합을 방지하고 모델을 일반화하는데 사용

# 초기 가중치 설정

- He 초기화, Xavier 초기화와 같은 방법은 학습 초기에 적절한 분산을 제공하여 안정적인 학습을 돕는다.

# 하이퍼파라미터 란?

- 모델이 학습하기 전에 사용자가 직접 설정해야 하는 값들로
    - 학습률(learning rate), 에폭 수(epoch), 배치 크기(batch size), 은닉층의 개수와 뉴런 수, 활성화 함수, 정규화 계수 등
- 과적합 방지와 모델 성능 향상에 중요하기 때문에 최적의 조합을 찾아 초기화 해야 함

# CNN 이란?

- 합성곱 신경망, Convolutional Neural Network
- 필터(커널)를 통해 국소적인 영역에서 특징을 추출하여 패턴을 학습
- 합성곱 연산 → 활성화 함수 → 풀링(특징맵의 크기를 줄이고, 정보를 일반화 하는 과정)
- 이미지, 영상, 음성 데이터처럼 공간적 구조가 있는 데이터 분석에 유리

# RNN 이란?

- 순환 신경망, Recurrent Neural Network
- 시계열 데이터나 순차적 데이터를 처리하기 위한 신경망 구조로, 이전 시점의 출력을 현재 시점의 입력으로 다시 활용함으로써 시간의 흐름에 따른 정보를 학습할 수 있는 모델
- hidden state(은닉 상태)가가 이전까지 입력된 정보를 요약해 담고 있는 일종의 "메모리" 역할
- 시간이 지남에 따라 과거 정보가 희미해지는 기울기 소실 문제 발생 가능
    - 역전파 시, 손실 함수의 기울기를 시점을 따라 계속 곱하면서 계산하는데 0보다 크고 1보다 작은 수를 계속 곱하면 0이 됨

# LSTM 이란?

- 장단기 메모리, Long short-term memory
- 장기 의존성(Long-term dependency) 문제를 해결하기 위해 고안된 RNN의 확장 버전전
- 셀 상태(Cell State)와 세 가지 게이트(입력, 망각, 출력 게이트)로 구성되어 있으며,
    - 셀 상태는 망각 게이트(Forget Gate)와 입력 게이트(Input Gate)의 조합으로 결정되며 과거 정보 중 중요한 것만 남기고 최신 정보를 반영한 상태를 유지
    - RNN과 달리 셀 상태에서는 곱셈만 하지 않고, 덧셈을 통해 정보를 업데이트하기 때문에 기울기 소실(gradient vanishing) 문제를 해결

# Word2Vec 이란?

- 단어를 고정된 크기의 실수 벡터로 변환하는 임베딩(embedding) 기법
- 단어의 의미가 유사할수록 가까운 벡터로 표현함으로써, 의미적 유사성을 수치적으로 표현
- 단어 간의 의미적 연산 가능: $king - man + woman \approx queen$

# Seq2Seq 란?

- Encoder-Context Vector-Decoder로 구성되어, 어순과 길이가 다른 다양한 언어를 번역하는데 유용한 모델
    - Encoder: 입력 시퀀스를 한 단어씩 읽어 전체 의미를 요약

    - Context Vector: 인코더가 만든 요약된 의미의 담고 있음
    - Decoder: 출력 시퀀스를 한 단어씩 생성
- 인코더와 디코더가 서로 다른 가중치를 사용하기 때문에 입력 개수와 출력 개수가 같을 필요가 없다.
- Context Vector에 모든 정보가 압축되기 때문에 입력 문장이 길수록 중요한 정보가 손실되는 한계가 있다.

# Attention 이란?

- 인코더와 디코더로 구성되어 있으며, 디코더가 출력 단어를 생성할 때 입력 문장 전체를 참고한다.
- 이 때, 출력 시점에서 예측해야 할 단어와 연관이 큰 입력 단어에 더 집중(attention)하는 기법
    - Seq2Seq의 Context Vector에 모든 정보가 요약되어 발생하는 문제를 해결
- 수식: $Attention(Q, K, V) = AttentionValue$

| 이름 | 역할 | 비유 (도서관 예시) | 구성 |
|---|---|---|---|
| **Query (Q)** | **찾고 싶은 질문** | "요리책 어디 있어요?" | 디코더의 현재 상태 |
| **Key (K)** | **모든 입력의 주소/태그** | 각 책의 제목/분류표 | 인코더의 각 단어에서 추출 |
| **Value (V)** | **실제 내용** | 책 안의 본문 내용 | 인코더의 각 단어에서 추출 |

- 작동 순서
    1. Query와 Key 간 유사도 계산 (어떤 입력이 중요한지 판단)
        - 내적하여 Score 생성: $Score_i = Q * K_i$
    2. Score를 Softmax로 정규화 (확률처럼 만듦)
        - $\alpha_i = Softmax(Score_i)$
    3. Value 벡터에 가중치 곱해서 최종 출력을 생성
        - $AttentionOutput = \sum a_i * V_i$
            - 이것이 일종의 Context Vector, 하지만 Seq2Seq와 달리 디코딩 시점마다 동적으로 생성되며 필요한 정보만 집중해서 가지고 있음
- Self-Attention 이란?
    - 하나의 문장 내에서 각 단어가 다른 단어들과의 연관성을 계산하여, 의미적으로 중요한 단어에 더 집중하는 메커니즘
    - 순차적인 계산 없이 병렬로 처리되며, 문맥 이해 능력을 극대화하기 위해 각 단어가 문장 전체를 참고
    - Attention은 "입력과 출력이 다른 두 시퀀스 사이"의 연관성을 계산하지만, Self-Attention은 "하나의 시퀀스 내"에서 각 요소끼리의 연관성도 계산

# Transformer 란?

- 자연어 처리 중, 번역에 특화된 모델
- Attention is all you need 라는 논문에서 제안됨
- Attention 메커니즘을 아주 적극적으로 활용
- 이전까지는 RNN 기반에서 Attention을 사용(seq2seq 모델)하여 문장의 길이가 길어지면 정보가 소실(gradient vanishing)되는 한계 존재
- 이를 극복하기 위해 Self-Attention을 적용

    - 입력 문장과 출력 문장 각각 내부적으로 단어끼리 Attention을 수행하여 hidden state에 정보를 더 잘 담아냄

(추가 설명)

- 순차적인 계산 없이 입력 전체를 동시에 처리할 수 있는 인코더-디코더 구조를 갖는 대표적인 자연어 처리 모델
- Self-Attention 통해 문맥을 효과적으로 이해하고 표현
- 인코더와 디코더 두 부분으로 구성
    - 인코더: 입력 문장에 Self-Attention을 적용하여 문맥 정보가 담긴 벡터로 재구성
    - 디코더: 지금까지 생성된 단어들을 참고하여 다음 단어를 예측
- 기존 RNN은 "나는 → 오늘 → 아침에..." 순서대로 처리, Transformer는 모든 단어를 한 번에 보고, "마셨다"가 "커피"와 가장 관련 있다고 Self-Attention을 통해 학습

# 지도학습과 비지도학습의 차이

- 지도학습은 입력 데이터와 함께 정답(label)이 주어져 모델이 이를 학습하는 방식이고, 비지도학습은 정답 없이 데이터 자체의 구조나 패턴을 학습하는 방식

| 항목 | 지도학습 (Supervised) | 비지도학습 (Unsupervised) |
|---|---|---|
| **입력 데이터** | 입력 + 정답(label) 존재 | 정답(label) 없음 |
| **목표** | 정답을 맞히도록 학습 | 데이터의 구조/패턴 발견 |
| **대표 과제** | 분류(Classification), 회귀(Regression) | 군집화(Clustering), 차원 축소(Dimensionality Reduction) |
| **예시** | 이메일 → 스팸 or 정상 | 고객 구매 패턴 군집화 |
| **필요한 데이터** | 라벨링된 데이터 필요 | 비라벨링된 데이터로 가능 |

# 과적합이란 무엇이며 어떻게 방지할 수 있나요?

과적합은 모델이 학습 데이터에 너무 맞춰져서 새로운 데이터에 대해 일반화가 잘 안 되는 현상입니다. 즉, 훈련 정확도는 높지만 테스트 성능은 낮습니다.이를 방지하기 위해 다음과 같은 방법이 있습니다.

- 더 많은 데이터를 수집
- Dropout: 학습 중 일부 뉴런을 임의로 비활성화 -> 특정 노드 의존도 감소
- Early Stopping: **검증 손실(Validation Loss)**이 더 이상 줄지 않으면 학습 조기 종료
- L1 정규화, L2 정규화: 가중치가 큰 만큼 손실함수에 더해서, 가중치를 업데이트할 때, 더 빨리 감소하게 함. 즉, 큰 가중치를 점점 작게 만듦

# 최적화 방법 정리

최적화란?

- **최적화(Optimization)**란,
  모델의 성능을 높이기 위해 **손실 함수(Loss function)**를 최소화하도록
  **가중치(Weight)와 편향(Bias)**을 조정하는 과정

방법은?

- 경사하강법: 손실 함수의 **기울기(gradient)**를 따라 가장 빠르게 감소하는 방향으로 가중치를 조금씩 이동
  - $w \leftarrow w - \eta \cdot \nabla_w L(w)$
- Momentum:
  - 이전 업데이트 방향을 고려해 관성 효과를 부여하여, 진동을 줄이고 더 빠르게 수렴
  - $v_t = \gamma v_{t-1} + \eta \nabla_w L(w), \quad w \leftarrow w - v_t$
  - 경사가 자꾸 바뀌는 방향(진동 구간)에서 매우 유리
- Adam: 모멘텀 + RMSprop을 결합한 방식으로, 각 파라미터마다 학습률을 조정하고, **속도(1차 모멘트)**와 **변화량 크기(2차 모멘트)**를 모두 고려
  - 방법
    - $$1\text{차모멘텀}:$$ $$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
    - $$2\text{차모멘텀}:$$ $$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
    - $$\text{편향보정}:$$ $$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
    - $$\text{가중치업데이트}:$$ $$w \leftarrow w - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$
  - 특성
    - 실무에서 가장 널리 사용됨
    - 초기 수렴 속도 빠르고 안정적
    - 각 가중치마다 학습률을 자동 조절

| 항목 | 경사하강법 (GD) | 모멘텀 (Momentum) | 아담 (Adam) |
| --- | --- | --- | --- |
| 🔧 원리 | 기울기만 사용 | 기울기 + 과거 방향의 누적 (속도 개념) | 기울기 + 과거 + 학습률 자동 조정 |
| 🌀 진동 억제 | ❌ 없음 | ✅ 관성 효과로 진동 줄임 | ✅ 매우 안정적 |
| ⏱️ 수렴 속도 | 느릴 수 있음 | 더 빠름 | 가장 빠르고 안정적 |
| ⚙️ 계산 복잡도 | 가장 간단 | 중간 | 가장 복잡 (하지만 실무에서는 기본값처럼 사용) |
| 🧠 실무 사용 | 거의 안 씀 (이론적 중요) | 일부 사용됨 | 거의 모든 최신 모델의 기본 옵티마이저 |

# "모델이 학습한다"는 말의 의미는?

모델이 학습한다는 것은 주어진 데이터를 기반으로 가중치와 편향 같은 매개변수를 조정하여, 손실 함수를 최소화하는 방향으로 최적화해 나가는 과정을 의미합니다. 즉, 입력 데이터와 정답 간의 관계를 수학적으로 추정하는 함수를 찾아내는 과정입니다. 이를 통해 모델은 새로운 데이터에 대해서도 적절한 예측을 할 수 있게 됩니다.

# 강화학습이란?

강화학습은 에이전트가 환경과 상호작용하면서 보상을 최대화하는 방향으로 학습하는 방법입니다. 정답(label)이 주어지지 않고, 시행착오(trial-and-error)를 통해 경험을 쌓으며 학습합니다.

- 예시
  - 알파고: 바둑에서 승리 보상을 최대화하기 위해 강화학습 사용
  - 로봇 제어: 로봇 팔이 물건을 정확히 잡는 법을 반복 학습

# Gaussian Processes(가우시안 프로세스, GP) 란?

함수 f(x) 를 명시적으로 정의하지 않고도, 주어진 입력값들에 대해 출력이 어떤 확률 분포를 가질지를 추정

- 예를 들어, 3개의 데이터 포인트가 주어졌을 때, GP는 이를 통과할 가능성이 높은 모든 함수의 분포를 고려
- 파라미터 수가 고정되어 있지 않고 데이터에 따라 늘어나며
- 예측값만 제공하지 않고 예측값의 신뢰도(신뢰 구간)도 같이 제공

# 1. Heap

- Priority Queue를 빠르게 구현하게 위해 만든 트리기반 자료 구조
- Complete Binary Tree 형태를 취함
-

## 1-1. Priority Queue(우선순위 큐)란?

- 일반적인 큐: FIFO 선입선출
- Priority Queue: 데이터에 우선순위 부여 → 우선순위 높은 것이 먼저 나감
  - 삽입순서 무관
  - 우선순위는 주로 정수로 표현
  - 예시1: 응급실에서는 심각한 환자부터 치료
  - 예시2: 운영체제 스케쥴러는 우선순위 높은 자료부터 처리

## 1-2. Complete Binary Tree 란?

- 왼쪽부터 빈틈없이 채워진 이진 트리 = 모든 상위 레벨은 꽉 참 + 마지막 층만 왼쪽부터 참
- 노드 개수는 n개
- 루트에서 리프까지의 최장 경로 = 트리의 높이(h)
- 이진 트리는 각 레벨마다 노드 수가 2배씩 증가(따라서 n개의 노드를 모두 담기 위해 필요한 깊이는 $\log_2 n$ 레벨이면 충분)
- 총 노드 수의 합은 등비수열
  - $n = 1 + 2 + 4 + \cdots + 2^h = 2^{h+1} - 1$
- 높이를 구하는 공식
  - $log_2(n+1) = log_2 2^{h+1} = h + 1$

---

# 2. 왜 Complete Binary Tree 를 사용하는가?

- 삽입 또는 삭제할 때 트리모양을 정해진 규칙으로 유지해야 시간 복잡도 $O(logn)$를를 보장
- 시간 복잡도 $O(logn)$를 보장한다는 뜻은?
  - 항상 최악의 경우에도 $O(logn)$ 시간 안에 작업을 종료함

## 2-1. 왜 Complete Binary Tree에서는 $O(logn)$이 보장되는가?

- 힙은 완전 이진 트리이므로 트리의 높이(height)는 $\log_2 n$
- 트리의 노드 수가 n개일 때, 높이는 최악의 경우에도 $\log_2 n$에 가까우므로 연산이 루트부터 리프까지 진행돼도 log n 단계를 넘지 않음

---

| 연산 | 동작 설명 | 시간복잡도 |
|---|---|---|
| `insert()` | 새로운 값을 말단에 추가 후, 부모와 비교하며 위로 올라감(up-heap / bubble-up) | O(log n) |
| `pop()` | 루트(최댓값/최솟값)를 제거 후, 말단 노드를 루트에 올리고 아래로 내려감(down-heap / heapify) | O(log n) |
| `peek()` | 루트 노드의 값 조회 (최대/최소) | O(1) |

---

## 2-2. "log₂ n"과 "log n"이 빅오 표기(Big-O)에서는 왜 차이가 없나?

로그 함수는 서로 상수배로 변환 가능하기 때문
$log_2 n = \frac{log_{10} n}{log_{10} 2} \approx 3.32 \times log_{10} n$
→ 밑만 다를 뿐, 형태는 동일하고 상수배만 차이남

→ 빅오 표기에서는 상수를 무시하므로 다음은 모두 동일한 시간복잡도로 간주

→ 복잡도 분석에서는 어떤 로그든 O(log n)으로 통일해서 사용
$O(log_2 n) = O(log_{10} n) = O(ln n) = O(log n)$

# 3. Time Complexity(시간복잡도)

- 시간복잡도 "O" 란 : 입력 크기 n이 커질 때 알고리즘이 걸리는 시간(연산 수)이 어떻게 증가하는가?
- $O$: 빅오(더)
- $O(n)$: n 차수로 증가
- $O(n^2)$: $n^2$ 차수로 증가
- $O(log n)$: $log n$ 차수로 증가
- $O(2^n)$: $2^n$ 차수로 증가

## 예시

시간복잡도는 해당 함수에서 가장 영향이 큰 항만 남김
$O(3n^3 + 5n + 100) = O(n^3)$

# 1. Definition of Nearest Neighbor

Let's find cheese that meticulously well paired with my meat.
What makes one cheese better than another?
It turns out that all cheeses can be categorized by a few distinct characteristics.
The different characteristics of a cheese are known as features.
Cheese that share similar characteristics will be our definition of the nearest neighbor.

# 2. ML Algorithm of Nearest Neighbor

In the ML world, information about each cheese would be referred to as the data.
We are going to need a large amount of data about cheese. So we will index cheese.com.
This is considered factual information about cheese but also includes many opinionated discussions about cheese.
All this data together will be a wealth of information for making decisions.

A cheese expert would consider all the characteristics together to classify a given cheese.
A nearest neighbor algorithm does something similar but in a natural language processing (NLP) way.
It compares words (or phrases) from different cheeses against one another. Depending on how similar they are, a probability is returned.
It will be a probability that the two cheeses are a good fit as a number.

# 3. Measuring How Far between Vectors

If you've ever attempted text comparisons on a large dataset you will know that it's anything but performant.
To overcome this, the text is converted to a collection of numbers called vectors.
The act of converting text to numerics is known as tokenization.

Anything in the database that has vectors similar to those words is probably a neighbor - complementing cheeses.
Finding the nearest neighbor is the process of plotting all the vectors in all their dimensions and then comparing a context collection of vectors to them.
Using a simple coordinate system you can mathematically measure how far one point is from another (known as their distance).

# 4. Various Nearest Neighbor Algorithms

## 4-1. K-nearest neighbors (KNN)

Classify some piece of data against a large set of labeled data
Label: what each item in the data set is
Unsupervised data -> Labeling -> Supervised data
The "K" in KNN is a representation of bounds: meaning how many pictures of cheese are you willing to consider?  how many points in space you are willing to consider?
a prediction of how well the provided data fits the existing data label => a percentage and a classifier

## 4-2. Approximate Nearest Neighbor (ANN)

it works well on non-labeled (unsupervised) data
The return will be approximately what data is closely related to the input, but hallucinations are real so be careful.

## 4-3. Fixed radius nearest neighbor

Fixed radius is an extended approach to KNN. it is limited to a certain distance.
Limiting the number of points to consider is an easy way to speed up the overall calculation.
The context value is a vector and the radius(fixed value) is a measure of distance from that vector.

## 4-4. Partitioning with k-dimensional tree(k-d tree)

When data is tokenized (converted to vectors) the number of dimensions is chosen. based on how accurate you need a search to be
The more dimensions an embedding has the longer it’s going to take to compute a nearest neighbor. Need balance
k-d tree splits the single space into a number of spaces (called partitions)
How the spaces are sorted (so their shared context is not lost) is an implementation choice (median-finding sort).
the number of leaves that make up each space in the tree is balanced. This makes for uniform, predictable search performance.
the algorithm is given a sense of proximity.
it can choose to not search large portions of the tree because it knows those leaves are too far. That can really speed up a search.

# Jean-Ho Kim†, Eun-Hong Park†, Ha Young Kim*, "Optimizing Stock Price Prediction via Macro context and Filter-DQN Framework," Journal of The Korea Society of Computer and Information (한국컴퓨터정보학회논문지), vol. 30, no. 01, Jan. 2025.(KCI).

## 한계

첫째, 국제와 국내 지표를 통합적으로 충분히 활용하지 못함
- 한국의 KOSPI 지수는 미국, 유럽, 일본 등 주요 국제 시장의 움직임에 민감
- 국제 시장과의 복잡한 상호작용을 충분히 반영하지 못한 채 주식 시장 내재적인 변동성만을 설명하거나 상승과 하락의 분류로 접근

둘째, 다량의 변수 중 중요한 정보를 효과적으로 선별하지 못함
- 차원적이고 이질적인 변수들이 포함될 경우 불필요하거나 상관성이 낮은 변수로 인하여 모델의 복잡성이 증가하고 예측 성능이 저하되는 문제
- 고차원 데이터에서 변수 선택이 이루어지지 않을 경우 과적합 위험이 존재 하며, 모델의 복잡성 증가로 인하여 학습 속도가 저하될 가능성

## 개선

주요 수출 대상국의 시장 동향과 국제적인 거시경제 변수를 포함
- 국제 거시경제 지표와 주요 수출 대상국의 주식시장 지표를 포함해 다양한 변수로 구성되어 있는 데이터 셋에서 유의미한 변수를 선별

Filter-DQN 방법을 제안
- 다변량 금융 시계열 데이터에서 상호 정보량(Mutual Information, MI)[10]을 기반으로 가장 유의미한 변수의 하위 집합을 강화 학습(Reinforcement Learning) 방식으로 찾아내는 Filter-DQN을 제안

## 방법론

1. 변수 간의 상호 정보를 DQN(Deep Q-Network)[11]의 에이전트(Agent)에게 보상(Reward)으로써 부여하며 학습시키는 Filter-DQN 기법을 통해 다변량 데이터에서 유의미한 변수의 하위 집합을 선별
    1. Filter-DQN Framework: 기존 필터 기법에서 활용되었던 상호 정보 이론과 DQN을 결합한 프레임워크
2. 선별된 변수들을 입력으로 활용하여 Transformer 기반 시계열 예측 모델[12-14] 을 학습시키고, 이를 통해 KOSPI 종가 지수의 변동성을 정교하게 예측
3. 절차
    1. 훈련 데이터의 무작위 시점 시계열 정보를 상태로써 입력
    2. 에이전트는 어떤 변수를 선택 할지 행동을 결정(변수의 차원을 축소)
    3. 행동에 의해 선택된 변수와 목적 변수와의 상호 정보량을 기반으로 보상을 부여
    4. 반복되며 목적 변수와 높은 상호 정보를 가지게 되는 변수들에 대한 행동 가치를 에이전트가 학습
    5. 상위 k개의 변수를 필터링하고, 이를 통해 생성된 하위 변수 집합을 출력

6. Transformer 기반 예측 모델[12-14]에 입력되어 시간 흐름에 따른 변동성과 변수 간의 상관관계를 학습

# 기대 효과

계산 비용을 절감하고 예측 성능을 극대화

# 성능 평가

예측: 8.28%, 45.28% 향상
계산 비용: 2.6% 절감

# 나의 생각

코스피를 예측하기 위해 다양한 변수들을 고려(미국주가지수, 일본주가지수, 금가격, 원유가격등 국제적 지수 + 이동평균선 등의 기술적 지표)하였고
많은 변수 중 중요한 변수들을 유의미하게 선택하여 예측을 위한 입력으로 활용