심층학습 - 2

생성자	재 재환 김
태그	머신러닝

선형대수(Linear Algebra)

- **선형대수의 중요성**: 선형대수는 수학의 한 분야로, 과학과 공학 전반에서 널리 활용됩니다.
    - 특히 컴퓨터 과학에서는 알고리즘의 이해와 적용을 위해 선형대수 지식이 필수적입니다.
    - 실용적인 알고리즘을 이해하고 사용하는 데 있어 선형대수의 중요성이 두드러집니다.
- **선형대수의 기본 개념**: 심층 학습 알고리즘에서 선형대수의 개념이 중요하므로, 학습 전에 선형대수의 주요 개념을 복습하는 것이 좋습니다.
    - 선형대수에 이미 익숙한 독자는 이 장을 건너뛰어도 됩니다.
- **추천 서적**: 선형대수의 세부 개념을 더 깊이 공부하고자 하는 독자에게는 "The Matrix Cookbook (2006)"과 같은 참고 서적을 추천합니다.
- **전문적인 학습**: 더욱 전문적인 학습을 원하는 경우, "Shilov (1977)"과 같은 전문 서적을 권장합니다. 이 장에서는 다루지 않지만, 이러한 서적에는 필수적인 기초 개념들이 다수 포함되어 있습니다.

2.1 스칼라, 벡터, 행렬, 텐서

- **스칼라 (Scalar)**:
    - 선형대수에서 다루는 가장 기본적인 대상입니다.
    - 하나의 숫자로 이루어진 대상을 의미하며, 일반적으로 실수 집합 ᴅ에 속하는 값으로 나타냅니다.
    - 스칼라 변수는 단일 실수 값으로만 표현됩니다.
    - 예를 들어, 실수 $s \in \mathbb{R}$로 표현할 수 있습니다.

- **벡터 (Vector)**:

- 여러 개의 수를 특정한 순서로 나열한 집합입니다.
- 벡터는 선형대수에서 중요한 개념이며, n-차원의 공간을 나타내는 데 사용됩니다.
- 벡터의 각 성분은 특정한 위치를 지칭하며, 그 위치는 벡터의 차원에 따라 다릅니다.
- 일반적으로 벡터는 기호로 $\mathbf{v}$로 표현되며, 주로 n-차원 벡터 공간 $\mathbb{R}^n$에서 다룹니다.
- 벡터는 수직 또는 수평으로 나열되며, 각각의 성분들은 행렬의 열(column)로 표현됩니다.
- 예시: 3차원 벡터 z는 다음과 같이 나타낼 수 있습니다:

$$z = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$$

- **행렬 (Matrix)**:
  - 다수의 벡터들을 행(row)과 열(column)로 배열한 2차원 배열입니다.
  - 행렬의 각 성분은 특정 위치에 있는 스칼라 값으로, 이 성분들은 행렬의 크기(차원)로 구분됩니다.
  - 예를 들어, 2×2 행렬 A는 다음과 같이 표현될 수 있습니다:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

  - 행렬은 다차원 데이터를 표현하는 중요한 도구이며, 이를 통해 다양한 선형 변환을 표현합니다.
- **텐서 (Tensor)**:
  - 텐서는 벡터와 행렬의 일반화된 형태로, 다차원 배열을 의미합니다.
  - 예를 들어, 3차원 이상의 데이터를 다루는 경우 텐서로 표현할 수 있습니다.
  - 텐서는 물리학, 컴퓨터 과학 등 다양한 분야에서 활용됩니다.

2.1 행렬의 전치 (Transpose)

- **전치 행렬**:
  - 행렬의 전치(transpose)는 행렬의 행과 열을 바꾸는 연산입니다.
  - 주어진 행렬 A의 전치 행렬 A^T은 행렬 A의 i행과 j열에 있는 성분을 A^T에서는 j행과 i열에 위치하도록 합니다.

- 즉, $A_{ij} = A^T_{ji}$로 나타냅니다.
- 예시로 그림 2.1에서 행렬A를 전치한 A^T의 변환 과정을 도식적으로 보여주고 있습니다.

행렬 곱셈 (Matrix Multiplication)

- **행렬 곱셈**:
    - 행렬 곱셈은 두 행렬을 곱하는 연산으로, 주어진 두 행렬 A와 B가 있을 때, 결과 행렬 C는 A와 B의 곱으로 정의됩니다.
    - 행렬 곱셈의 성분을 나타내는 공식은 다음과 같습니다:Cij=k∑AikBkj

$$C_{ij} = \sum_k A_{ik}B_{kj}$$

    - 이 공식은 A의 각 행과 B의 각 열을 곱한 값을 더해 결과 행렬의 성분으로 계산하는 방식입니다.
    - **주요 성질**:
        - 행렬 곱셈은 일반적으로 교환 법칙을 만족하지 않습니다. 즉, $AB \neq BA$입니다.
        - 그러나 분배법칙과 결합법칙은 만족합니다:

$$A(B+C) = AB + AC$$

$$A(BC) = (AB)C$$

- **행렬 곱셈의 유형**:
- 행렬 곱셈에는 여러 방식이 있습니다. 성분별 곱셈은 요소 곱(Element-wise product) 또는 아다마르 곱(Hadamard product)이라고 합니다. 이는 행렬의 대응하는 성분들을 단순히 곱하는 방식입니다.
- 벡터의 내적(dot product)은 또 다른 형태의 곱셈입니다. 이는 두 벡터의 대응하는 성분들을 곱한 후 모두 더하여 하나의 스칼라 값을 얻는 방식입니다.
- **스칼라와 행렬의 연산**:
    - 스칼라와 행렬 간의 연산도 가능합니다. 스칼라 c와 행렬 A가 주어졌을 때, cA는 행렬 A의 모든 성분에 스칼라 c를 곱하여 계산합니다.

2.2 행렬과 벡터의 곱셈

- **행렬과 벡터 간의 곱셈**:

- ◦ 행렬과 벡터의 곱셈은 행렬과 벡터가 같은 차원을 가질 때 가능합니다. 예를 들어, $m \times n$ 행렬 A와 n차원 벡터 x가 있을 때, 그 결과는 m차원 벡터가 됩니다.

- **벡터 내적**:
    - 두 벡터 간 내적은 두 벡터의 대응 성분을 곱하고 그 값을 모두 더한 결과로 나타내며, 스칼라 값이 됩니다:

$$y^T x = y_1 x_1 + y_2 x_2 + \cdots + y_n x_n$$

2.3. 행렬의 전치 (Transpose of a Matrix)

- **전치 행렬 (Transpose)**:
    - 행렬A의 전치 행렬 A^T는 행과 열을 서로 바꾸는 연산입니다.
    - A_{ij}는 행렬 A의 i번째 행과j번째 열에 위치한 원소를 나타냅니다.
    - 전치 행렬에서는 이 원소가 j번째 행과 i번째 열로 바뀝니다. 즉, 전치 연산을 통해 A_{ij}가 A^T_{ji}로 변환됩니다.
    - **예시**:
        - 주어진 행렬 A:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

        - 이 행렬의 전치 행렬 AT는:

$$A^T = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix}$$

        - 그림 2.1에서 이 변환 과정을 도식적으로 설명하고 있습니다.

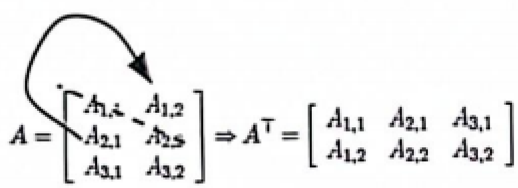


그림 2.1: 행렬의 전치는 주대각을 기준으로 행렬을 반사한 것과 같다.

- **스칼라 전치**: 스칼라 값의 경우, 이는 단일 값이므로 전치 연산은 무의미하며, 스칼라의 전치는 스칼라 자신과 동일합니다. 즉, $a^T = a$.

3. 행렬 곱셈 (Matrix Multiplication)

- **행렬 곱셈의 정의**:
  - 두 행렬 A와 B가 있을 때, 이들의 곱 $C = AB$는 다음과 같은 방식으로 계산됩니다.
  - A가 $m \times n$행렬, B가 $n \times p$행렬일 경우, 그 결과 행렬 C는 $m \times p$크기를 가집니다.
  - 행렬 C의 각 원소 C_{ij}는 행렬 A의 i번째 행과 B의 j번째 열의 각 성분을 곱한 후 더한 값으로 계산됩니다. 즉:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

  - 이 공식을 더 풀어서 설명하면, C_{ij}는 A의 i번째 행의 성분들과 B의 j번째 열의 성분들을 대응시키고 각각 곱한 후 합한 값입니다.
- **행렬 곱의 예시**:
  - 2×2 행렬 A와 B의 곱을 예시로 들면:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

    - 행렬 곱 $C = AB$의 각 성분은 다음과 같이 계산됩니다:

$$C_{11} = a_{11}b_{11} + a_{12}b_{21}$$

$$C_{12} = a_{11}b_{12} + a_{12}b_{22}$$

$$C_{21} = a_{21}b_{11} + a_{22}b_{21}$$

$$C_{22} = a_{21}b_{12} + a_{22}b_{22}$$

  - 이 방식으로 두 행렬의 곱이 성분별로 계산됩니다.
- **행렬 곱의 성질**:
  - 행렬 곱셈은 **비교환 법칙**을 따르지 않습니다. 즉, 일반적으로 $AB \neq BA$입니다. 이는 두 행렬의 곱이 서로 순서에 따라 다른 결과를 만들 수 있음을 의미합니다.
  - 그러나 행렬 곱셈은 **결합 법칙**과 **분배 법칙**을 만족합니다.
    - 결합 법칙:

$$A(BC) = (AB)C$$

- 분배 법칙:

$$A(B+C) = AB + AC$$

3.1 성분별 곱셈 (Element-wise Product 또는 Hadamard Product)

- **요소 곱 (Element-wise Product)**:
  - 행렬의 곱셈에는 성분별로 곱하는 연산 방식도 있습니다. 이를 아다마르 곱 (Hadamard product)라고도 부르며, 두 행렬 A와 B의 대응되는 위치의 성분을 각각 곱하여 새로운 행렬을 만듭니다.

    AA

    BB
  - 요소 곱을 수식으로 표현하면 다음과 같습니다:

    $$(A \circ B)_{ij} = A_{ij} \times B_{ij}$$
  - 요소 곱은 기계 학습이나 신경망 계산에서 자주 사용되는 연산 방식입니다.

3.2 벡터의 내적 (Dot Product)

- **벡터 내적**:
  - 벡터 간의 내적은 벡터 x와 y의 각 성분을 곱한 후 그 값들을 모두 더하는 방식으로 계산됩니다.
  - 수식으로는 다음과 같습니다:

    $$x^T y = \sum_i x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$
  - 결과는 하나의 스칼라 값이 되며, 이는 벡터 간의 유사성을 측정할 때 유용하게 사용됩니다.

3.3 결합 및 분배 법칙의 예시

- **결합 법칙**:
  - 행렬 곱셈은 결합 법칙을 따릅니다. 즉:

    $$A(BC) = (AB)C$$

    이 공식은 행렬 곱셈의 순서를 바꿔도 같은 결과가 나온다는 의미입니다.

    $$A(BC) = (AB)C \, A(BC) = (AB)C$$

- **분배 법칙**:
  - 행렬 곱셈은 또한 분배 법칙도 만족합니다. 즉:

    $A(B + C) = AB + AC$
    이 공식은 행렬의 합을 먼저 계산하거나, 각 행렬을 따로 곱한 후 더해도 결과가 같다는 것을 의미합니다.

$$A(B + C) = AB + AC\,A(B + C) = AB + AC$$

3.4 행렬 곱셈의 성질

- **전치 연산과 행렬 곱셈의 관계**:
  - 행렬 곱 AB의 전치 행렬은 개별 행렬 A와 B의 전치 행렬을 역순으로 곱한 것과 동일합니다.
  - 즉:

    $(AB)^T = B^T A^T$
  - 이 공식은 행렬의 전치 연산과 곱셈 연산이 결합되는 방식을 보여줍니다.
  - 이 성질은 행렬 계산에서 중요한 역할을 하며, 특히 대칭 행렬을 다룰 때 유용하게 사용됩니다.
- **벡터 내적의 성질**:
  - 벡터 x와 y의 내적에서, 두 벡터를 전치한 결과는 동일한 스칼라 값을 반환합니다.
  - 즉:

    $x^T y = y^T x$
  - 이는 스칼라 값의 교환 법칙에 해당하며, 두 벡터의 순서가 바뀌어도 내적의 값은 변하지 않는다는 것을 의미합니다.

4.1 연립방정식의 표현

- 연립방정식은 선형대수에서 중요한 응용 중 하나입니다.
  - 일반적으로 행렬 A와 벡터 b가 주어졌을 때, $Ax = b$라는 형태로 연립방정식을 표현할 수 있습니다:

$$Ax = b$$

- 여기서 A는 $m \times n$ 크기의 행렬이고, x는 미지의 벡터입니다. b는 주어진 결과 벡터로, 각 성분 b_i는 방정식의 우변 값에 해당합니다.
- 이 식을 더 명시적으로 표현하면, 다음과 같은 연립방정식의 형태로 나타낼 수 있습니다:

$$\begin{gathered} A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n = b_1 \\ A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n = b_2 \\ \vdots \\ A_{m,1}x_1 + A_{m,2}x_2 + \cdots + A_{m,n}x_n = b_m \end{gathered}$$

- 이는 연립방정식의 각 성분을 풀어내는 방식입니다. 행렬과 벡터를 이용하면 연립방정식을 간결하게 나타낼 수 있습니다.

4.2 단위행렬과 역행렬 (Identity Matrix and Inverse Matrix)

- **단위행렬 (Identity Matrix)**:
  - 단위행렬 I는 대각 성분이 1이고, 나머지 성분은 모두 0인 행렬입니다.

    II
  - 크기가 $n \times n$인 단위행렬은 다음과 같이 표현됩니다:

$$Ix = x$$

  - 즉, 어떤 벡터에 단위행렬을 곱하면 그 벡터는 변하지 않으며, 이는 곱셈에서 1을 곱한 것과 같은 역할을 합니다.
- **역행렬 (Inverse Matrix)**:
  - 역행렬 A^{-1}은 행렬 A에 대해, $AA^{-1} = I$ 를 만족하는 행렬입니다.
  - 즉, 역행렬을 곱하면 단위행렬을 반환하게 되며, 이는 곱셈에서 역수를 구하는 것과 동일한 역할을 합니다.
  - 연립방정식 $Ax = b$ 를 역행렬을 이용해 풀 수 있습니다:

$$x = A^{-1}b$$

  - 여기서 A^{-1}은 A의 역행렬로, 이 역행렬이 존재하는 경우에만 방정식을 풀 수 있습니다. 역행렬이 존재하지 않는 경우, 방정식은 해가 없거나 무수히 많은 해를 가질 수 있습니다.

4.3 역행렬의 존재와 계산

- **역행렬의 존재**:
    - 모든 행렬에 역행렬이 존재하는 것은 아닙니다. 행렬 A에 대해 역행렬 A^{-1}이 존재하려면 특정 조건이 충족되어야 하며, 그 조건이 충족되지 않으면 역행렬을 구할 수 없습니다.
    - 이 부분에서는 A^{-1}의 존재 조건과 관련된 여러 이론들이 소개됩니다. 특히 **방정식**을 풀 때 역행렬을 사용하는데, $Ax = b$에서 $x = A^{-1}b$ 형태로 풀 수 있다는 점을 강조합니다.
- **역행렬 계산 방법**:
    - 실제로 역행렬을 구하는 방법은 다양한 알고리즘을 사용할 수 있습니다. 대부분의 소프트웨어는 이를 자동으로 처리해 주지만, 기본적으로 알고리즘을 이해하는 것이 중요합니다.
    - 예를 들어, **가우스-조던 소거법**(Gauss-Jordan elimination)이나 **LU 분해법** 등을 이용해 역행렬을 구하는 방법이 있습니다.
- **단위행렬**:
- 단위행렬은 역행렬 계산의 기본으로 사용됩니다. 아래에서는 단위행렬의 예시가 주어져 있으며, 이 행렬은 대각선 성분이 모두 1이고 나머지 성분이 0인 형태입니다.
- $n \times n$ 크기의 단위행렬은 다음과 같이 표현됩니다:

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4.4 일차종속성과 생성공간

- **일차종속 (Linear Dependence)**:
    - 일차종속이란, 하나의 벡터가 다른 벡터들의 선형 결합으로 표현될 수 있는 경우를 의미합니다. 즉, $v_1, v_2, \ldots, v_n$이 있을 때, 특정 벡터가 다른 벡터들의 선형 결합으로 나타낼 수 있으면 그 벡터들은 일차종속입니다.
    - 만약 벡터x가 벡터들의 선형 결합으로 표현되지 않는다면, 그 벡터들은 **일차독립**이라고 합니다.
- **생성공간 (Span)**:
    - 여러 벡터들의 선형 결합을 통해 형성된 공간을 **생성공간** 또는 **Span**이라고 합니다. 생성공간은 주어진 벡터들이 이루는 모든 점들의 집합으로, 선형대수에서 중요한 개념 중 하나입니다.

- 수식으로는 다음과 같이 표현됩니다:

  $$z = ax + (1 - a)y$$

- 여기서 a는 스칼라 계수이고, x와 y는 벡터입니다. 이 식은 x와 y의 선형 결합을 나타내며, 이러한 벡터들이 이루는 공간을 생성공간이라고 부릅니다.

- 열공간 (Column Space)와 **행공간 (Row Space)**:
  - 행렬의 각 열이 이루는 공간을 **열공간**이라고 하며, 이는 벡터의 일차결합으로 얻을 수 있는 모든 점들의 집합을 의미합니다.
  - 반대로, 행렬의 각 행이 이루는 공간을 **행공간**이라고 합니다.
  - 예를 들어, 행렬 A의 열벡터들이 일차결합으로 나타내는 공간이 열공간이며, 이는 행렬의 구조를 분석하는 데 중요한 역할을 합니다.

4.5 일차종속과 연립방정식의 관계

- **일차종속과 연립방정식**:
  - 연립방정식 $Ax = b$의 해가 존재하려면, 벡터 b가 행렬 A의 열공간에 속해야 합니다. 즉, 열공간의 벡터들이 b를 선형 결합으로 표현할 수 있어야 해가 존재합니다.
  - 만약 b가 열공간에 속하지 않으면, 방정식은 해가 없으며 **일차종속성**이 문제가 됩니다.
  - 또한, $m > n$인 경우, 즉 행의 수가 열의 수보다 많을 때, 해가 존재하지 않거나, 해가 유일하지 않을 가능성이 큽니다. 이는 선형대수에서 자주 나타나는 상황으로, 차원이 큰 공간에서의 해의 유일성을 확인하는 데 중요합니다.
- **일차독립성의 중요성**:
  - 벡터들이 일차독립일 경우, 연립방정식의 해는 고유하며 유일합니다. 반대로, 일차종속이 있을 경우 해가 무수히 많거나 해가 존재하지 않게 됩니다.

4.6 방정식 해의 성질

- **유일한 해의 조건**:
  - 연립방정식 $Ax = b$에서 해가 유일하려면 행렬 A의 열벡터들이 일차독립이어야 합니다.
  - 즉, A의 열벡터들이 서로 선형 결합으로 표현되지 않을 때, 방정식은 유일한 해를 가집니다.
- **무수히 많은 해의 경우**:

- A의 열벡터들이 일차종속일 경우, 방정식의 해는 무수히 많을 수 있습니다. 이는 방정식이 하나 이상의 해를 가지며, 어떤 경우에는 모든 벡터가 해가 될 수 있음을 의미합니다.

4.7 일차종속성(Linear Dependence)

- **일차종속성**은 벡터 집합에서 특정 벡터가 다른 벡터들의 선형 결합으로 표현될 수 있을 때 발생합니다. 즉, 하나의 벡터가 다른 벡터들의 가중합으로 나타날 수 있으면, 그 벡터 집합은 **일차종속**이라고 부릅니다.
- 이를 **공식적으로** 정의하면 다음과 같습니다:
    - 주어진 벡터 $v_1, v_2, \ldots, v_n$ 중 일부 벡터가 나머지 벡터들의 선형 결합으로 표현될 수 있을 때, 이 벡터들은 **일차종속**입니다.
    - 이 개념은 벡터들이 서로 독립적이지 않다는 것을 의미하며, 그 집합에 불필요한 중복이 있음을 나타냅니다.
- 만약 벡터 집합이 일차종속이 아니라면, 즉 벡터들 사이에서 선형 결합이 이루어지지 않는다면 그 벡터 집합은 **일차독립**입니다.
    - 일차독립인 벡터 집합은 각 벡터가 고유한 방향을 나타내며, 다른 벡터로 대체할 수 없는 상태를 의미합니다.
- **일차종속과 행렬**:
    - $Ax = 0$ 형태의 동차 연립방정식에서 A가 일차독립이면, 이 방정식은 유일한 해를 가집니다.
    - 만약 A가 일차종속일 경우, 해는 무수히 많거나 존재하지 않게 됩니다.

4.8 정방행렬(Square Matrix)과 역행렬

- **정방행렬 (Square Matrix)**:
    - 정방행렬은 행과 열의 개수가 동일한 $n \times n$ 크기의 행렬을 의미합니다.
    - 정방행렬의 중요한 성질 중 하나는 역행렬의 존재 조건입니다. 역행렬이 존재하려면 그 행렬이 **정칙**이어야 합니다. 정칙 행렬은 모든 행 또는 열이 일차독립일 때 성립합니다.
- **역행렬**:
    - 행렬 A의 역행렬이 존재하려면, A가 정칙 행렬이어야 하며, 그 조건은 A가 **일차독립인 열벡터**를 가져야 한다는 것입니다.

- 이때, 역행렬 A^{-1}는 다음과 같이 정의됩니다:

  $$A^{-1}A = I$$

- 이 식은 역행렬을 통해 원래 행렬로 되돌릴 수 있음을 의미합니다.

- **특이행렬(Singular Matrix)**:
  - 만약 행렬 A가 일차독립 조건을 만족하지 않으면, 즉 벡터들이 일차종속일 경우, 그 행렬은 **특이행렬**이라 불리며 역행렬을 가질 수 없습니다.
  - 특이행렬의 경우, $Ax = b$와 같은 방정식을 풀 수 없거나, 무수히 많은 해를 가질 수 있습니다.

5. 노름(Norm)

- **벡터의 크기(노름, Norm)**:
  - 노름은 벡터의 크기를 측정하는 함수입니다. 기계 학습에서는 벡터의 크기를 통해 거리나 길이를 계산할 때 노름을 자주 사용합니다.
  - L^p **노름**은 다음과 같은 수식으로 정의됩니다:

    $$\|x\|_p = \left(\sum |x_i|^p\right)^{\frac{1}{p}}$$

  - 여기서 p는 노름의 종류를 결정하는 파라미터로, 보통 $1, 2, \infty$와 같은 값이 사용됩니다.

- **유클리드 노름 (Euclidean Norm)**:
  - $p = 2$일 때, L^2 노름은 **유클리드 거리**와 같으며, 가장 널리 사용되는 노름입니다. 이는 2차원 또는 3차원 공간에서 벡터 간의 실제 거리를 측정할 때 사용하는 방식과 동일합니다.
  - L^2 노름은 다음과 같이 간단하게 표현될 수 있습니다:

    $$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

  - 이는 벡터의 각 성분의 제곱을 모두 더한 뒤, 그 결과의 제곱근을 구하는 방식으로, 벡터의 길이를 나타냅니다.

- L^1 **노름**:
  - $p = 1$일 때의 L^1 노름은 벡터의 각 성분의 절댓값을 모두 더한 값입니다.
  - 예를 들어, L^1 노름은 다음과 같이 계산됩니다:

    $$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$

$$\|x\|_1 = \sum |x_i|$$

- L^1 노름은 벡터의 크기를 절댓값으로 계산하며, 기계 학습에서는 정규화 과정에서 자주 사용됩니다. L1 노름의 특징은 벡터의 모든 성분이 0이 아닌 경우라도 작은 값이라도 추가적으로 고려된다는 것입니다.

- L^∞ **노름** (최대 노름, Max Norm):
    - L^∞ 노름은 벡터의 성분 중 가장 큰 절댓값을 사용하는 노름입니다.
    - 이는 다음과 같이 정의됩니다:

    $$\|x\|_\infty = \max |x_i|$$

    - 이 노름은 벡터에서 가장 큰 성분이 그 벡터의 크기를 결정하는 방식입니다. 이는 벡터의 특정 성분이 다른 성분들보다 상대적으로 더 중요한 경우 유용하게 사용됩니다.

- **프로베니우스 노름 (Frobenius Norm)**:
    - 프로베니우스 노름은 행렬의 크기를 측정할 때 사용되며, 행렬의 각 성분을 제곱한 값을 모두 더한 뒤, 그 결과의 제곱근을 취한 값입니다.
    - 이는 다음과 같이 정의됩니다:

    $$\|A\|_F = \sqrt{\sum_{i,j} |A_{ij}|^2}$$

    - 프로베니우스 노름은 L^2 노름과 비슷하게 행렬의 각 성분을 고려하여 그 크기를 계산합니다.

- **벡터 내적과 노름의 관계**:
    - 두 벡터 x와 y의 내적을 다음과 같은 공식으로 나타낼 수 있습니다:

    $$z = \|x\|\|y\| \cos \theta$$

    - 여기서 θ는 두 벡터 사이의 각도입니다. 이 공식을 통해 두 벡터 사이의 각도와 크기에 따라 내적을 계산할 수 있습니다.

6. 특별한 종류의 행렬과 벡터

이 부분에서는 **대각행렬**과 **대칭행렬** 같은 특별한 종류의 행렬과 벡터에 대해 설명하고 있습니다.

- **대각행렬 (Diagonal Matrix)**:
    - 대각행렬은 대각선 성분을 제외한 나머지 성분이 모두 0인 행렬을 말합니다.

- 대각행렬의 예시는 다음과 같이 정의됩니다:

  $$D = \mathrm{diag}(d_1, d_2, \ldots, d_n)$$

- 대각 성분만이 중요한 역할을 하며, 나머지 성분은 계산에 영향을 미치지 않습니다.
- 대각행렬은 계산이 단순하고 효율적이기 때문에 여러 가지 수학적 문제나 응용에서 자주 사용됩니다.

- **대칭행렬 (Symmetric Matrix)**:
  - 대칭행렬은 행렬의 성분이 대각선을 기준으로 대칭적인 행렬을 의미합니다. 즉, 행렬 A가 대칭행렬이면 $A = A^T$가 성립합니다.
  - 대칭행렬은 수학적으로 매우 중요한 성질을 가지며, 선형대수에서 자주 등장합니다.

- **단위벡터 (Unit Vector)**:
  - 단위벡터는 크기가 1인 벡터를 의미합니다. 이는 노름을 사용하여 다음과 같이 표현됩니다:

    $$\|x\| = 1$$

  - 단위벡터는 방향만을 나타내며, 크기는 항상 1로 고정됩니다. 이는 기하학적 계산에서 벡터의 방향을 나타내는 데 자주 사용됩니다.

- **직교벡터**:
  - 벡터 x와 y의 내적 $x^T y = 0$일 때, 두 벡터는 서로 직교한다고 말합니다. 즉, 두 벡터가 서로 수직이라는 의미입니다. 직교하는 벡터들은 각각의 방향이 독립적이며, 서로 간섭하지 않는 방향을 나타냅니다.

- **직교 행렬(Orthogonal Matrix)**:
  - 여러 벡터가 서로 직교하며, 그 벡터들이 단위벡터일 때, 이 벡터들로 구성된 행렬을 **직교 행렬**이라고 합니다.
  - 직교 행렬 A는 다음과 같은 성질을 만족합니다:

    $$A^T A = AA^T = I$$

    여기서 I는 단위행렬(Identity Matrix)을 의미합니다. 직교 행렬의 행과 열 벡터는 서로 직교하고, 각각의 벡터는 단위벡터입니다.

- **직교 행렬의 역행렬**:

- 직교 행렬은 그 자체로 매우 특이한 성질을 가지는데, 직교 행렬의 역행렬은 그 행렬의 전치 행렬과 동일합니다.
- 즉:

$$A^{-1} = A^T$$

- 이는 직교 행렬의 중요한 성질로, 직교성을 유지하면서 행렬의 역연산이 매우 간단하게 처리될 수 있음을 의미합니다.

7. 고윳값 분해 (Eigenvalue Decomposition)

- **고윳값과 고유벡터**:
  - **고윳값**(Eigenvalue)과 **고유벡터**(Eigenvector)는 행렬을 분해하는 중요한 도구입니다.
  - 행렬 A에 대해, 벡터 v가 0이 아닌 고유벡터이고 스칼라 λ가 고윳값일 때, 다음을 만족합니다:

$$Av = \lambda v$$

  - 즉, 고유벡터 v는 행렬 A에 의해 변환된 후에도 방향이 바뀌지 않고, 크기만 λ배 만큼 변하게 됩니다. 이때의 λ가 고윳값입니다.
- **고윳값 분해(Eigenvalue Decomposition)**:
  - 행렬 A가 고유벡터와 고윳값을 가지고 있다면, 이 행렬을 고유벡터와 고윳값으로 분해할 수 있습니다.

    AA
  - 즉, 고윳값 분해는 다음과 같이 나타낼 수 있습니다:

$$A = V \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) V^{-1}$$

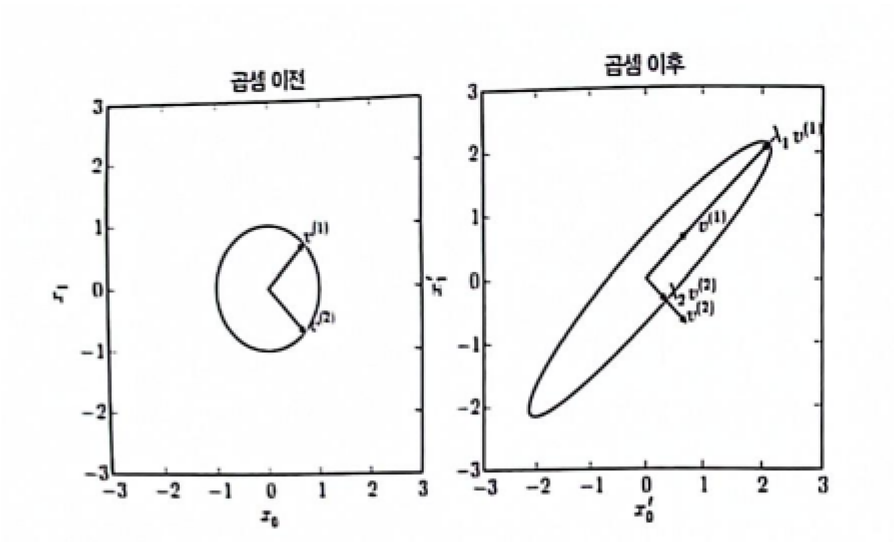
여기서 V는 고유벡터들을 모아놓은 행렬이며, $\text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$는 대각선 성분에 고윳값들이 배치된 대각 행렬입니다. V−1는 고유벡터 행렬의 역행렬입니다.

- **고윳값 분해의 의미**:
- 고윳값 분해는 복잡한 행렬을 더 간단한 대각 행렬로 변환하여 문제를 해결하는 강력한 도구입니다.
- 고유벡터는 변환 과정에서 방향을 유지하는 특별한 벡터이며, 고윳값은 이 벡터들의 크기 변화를 나타냅니다.

- 이 분해 방법은 행렬의 고유한 성질을 분석하고 복잡한 계산을 단순화하는 데 자주 활용됩니다.
- **고윳값 분해의 정의**:
  - 행렬 A를 고유벡터와 고윳값으로 분해하는 방식은 다음과 같은 형태로 나타낼 수 있습니다:

    $$A = Q\Lambda Q^T$$

    여기서
    Q는 고유벡터들로 이루어진 직교행렬이고, Λ는 대각 행렬로, 대각선에는 고윳값들이 위치해 있습니다. Q^T는 Q의 전치행렬입니다. 이 분해는 대칭 행렬이나 특정 조건을 만족하는 행렬에 대해 수행됩니다.
- **고윳값 분해의 의미**:
  - 그림 2.3은 고유벡터 공간에서의 변환을 도식적으로 보여줍니다. 원래 공간의 원(circle)이 고유벡터 공간에서 타원(ellipse)으로 변형되며, 이 타원의 축이 고유벡터의 방향과 일치합니다. 이는 고유벡터가 변환 후에도 방향을 유지한다는 중요한 특성을 시각적으로 나타냅니다.



- 고윳값 분해는 행렬을 고유벡터 공간에서 대각 행렬로 변환하여 행렬의 특성을 쉽게 파악할 수 있게 해줍니다. 이 과정을 통해 복잡한 행렬의 구조를 더 단순하고 이해하기 쉬운 형태로 표현할 수 있습니다.
- **양의 정부호 행렬 (Positive Definite Matrix)**:
  - 고윳값이 모두 양수인 행렬을 **양의 정부호(positive definite)** 행렬이라고 합니다. 이러한 행렬은 항상 양수로 평가되는 성질을 가지며, 이는 선형대수에서 매우 중요

한 개념입니다.

- 고윳값이 음수나 0을 포함할 경우, 그 행렬은 **양의 준정부호(positive semidefinite)** 또는 **음의 정부호(negative definite)**로 구분됩니다. 이러한 구분은 행렬의 특성을 이해하고 다양한 수학적 문제를 해결하는 데 중요한 역할을 합니다.

8. 특잇값 분해 (Singular Value Decomposition, SVD)

- **특잇값 분해의 정의**:
  - 특잇값 분해는 행렬을 세 개의 행렬로 분해하는 방법으로, 다음과 같이 표현됩니다:

    $A = U\Sigma V^T$
    여기서
    U와 V는 각각 직교 행렬이고, Σ는 대각 행렬로, 대각선에는 특잇값(singular value)이 위치합니다.
- **특잇값 분해의 의미**:
  - SVD는 모든 실수 행렬에 대해 적용 가능하며, 고윳값 분해와는 달리 정방행렬이 아니더라도 사용할 수 있습니다. 이는 고유벡터와 고윳값을 이용하는 방식과는 다르지만, 행렬의 구조적 특성을 분석하는 데 매우 유용한 도구입니다.
  - U는 행렬의 열공간을 나타내고, V는 행 공간을 나타내며,Σ는 특잇값을 대각선에 갖는 대각 행렬로서 행렬의 크기와 변형을 나타냅니다.
- **SVD의 응용**:
  - 특잇값 분해는 기계 학습, 이미지 처리, 데이터 분석 등 여러 분야에서 활용되며, 특히 데이터 차원 축소와 노이즈 제거에 효과적입니다. 차원 축소 기법인 PCA(주성분분석) 역시 특잇값 분해의 응용 중 하나입니다.
- **행렬의 구조적 분석**:
  - 특잇값 분해는 행렬의 고유한 정보를 추출하고, 그 구조를 분석하는 데 도움을 줍니다. A가 가지는 특이성 또는 정칙성을 파악할 때 SVD는 매우 유용한 도구로 사용됩니다. 또한, 불안정한 시스템에서 중요한 정보와 노이즈를 구분하는 데 효과적입니다.

9. 무어-펜로즈 유사역행렬 (Moore-Penrose Pseudoinverse)

- **유사역행렬의 정의**:
  - 정방행렬이 아니거나 역행렬이 존재하지 않는 경우, 즉 비정칙 행렬에 대해서도 해를 구해야 할 때 **무어-펜로즈 유사역행렬**을 사용합니다.

- **일반적인 역행렬이 정의되지 않을 때**, 무어-펜로즈 유사역행렬을 통해 대안적인 해를 구할 수 있습니다.

- **유사역행렬을 사용하는 상황**:
  - 예를 들어, 일반적인 역행렬을 구할 수 없는 경우, 행렬 A에 대한 방정식 $Ax = y$를 풀 때 해가 없거나 무수히 많은 해가 존재할 수 있습니다. 이 경우, **유사역행렬**을 사용하여 최소 제곱 해 또는 특수한 해를 구할 수 있습니다.
  - **유사역행렬** A+는 다음과 같이 정의됩니다:

    $A^+ = \lim_{\alpha \to 0}(A^TA + \alpha I)^{-1}A^T$

  - 이 식은 역행렬이 존재하지 않는 행렬 A에 대해, 유사역행렬을 구하는 방식 중 하나입니다.

- **SVD를 이용한 유사역행렬 계산**:
  - 유사역행렬은 특잇값 분해(SVD)를 이용하여 계산할 수 있습니다. $A = U\Sigma V^T$라고 할 때, A^+는 다음과 같이 표현됩니다:

$$A^+ = V\Sigma^+U^T$$

  - 여기서 Σ^+는 Σ의 대각 성분을 역으로 취한 행렬입니다. 이 방식은 SVD를 이용해 매우 효율적으로 유사역행렬을 구할 수 있습니다.

10. 대각합 연산자 (Trace Operator)

- **대각합의 정의**:
  - 대각합(Trace)는 행렬의 주대각선 성분들의 합을 의미합니다.
  - 행렬 A의 대각합은 다음과 같이 정의됩니다:

    $\mathrm{Tr}(A) = \sum A_{ii}$

  - 대각합 연산자는 행렬의 중요한 특성 중 하나로, 여러 수학적 성질에서 자주 사용됩니다.

- **대각합과 프로베니우스 노름(Frobenius Norm)**:
  - 프로베니우스 노름은 행렬의 성분들의 제곱합을 계산하여 그 제곱근을 구하는 방식인데, 이를 대각합을 이용해 표현할 수 있습니다:

    $\|A\|_F = \sqrt{\mathrm{Tr}(A^TA)}$

  - 이는 대각합을 이용하여 행렬의 크기(노름)를 구하는 방법을 설명하고 있습니다.

- **대각합의 성질**:
    - 대각합 연산자는 다양한 성질을 가지고 있으며, 그 중 하나는 행렬 곱셈에서 대각합이 교환 가능하다는 점입니다:

    $$\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$$

    - 이는 행렬의 곱이 이루어지는 순서에 상관없이 대각합의 값이 동일하다는 것을 보여줍니다. 이 성질은 행렬의 교환 가능성에 대한 중요한 정보를 제공하며, 행렬 이론에서 자주 사용됩니다.

11. 행렬식 (Determinant)

- **행렬식의 정의**:
    - 행렬식은 정방행렬(square matrix)에 대해 정의되며, 행렬의 크기나 변환 성질을 나타내는 스칼라 값입니다.
    - 행렬 A의 행렬식은 $\det(A)$로 표시됩니다.
- **행렬식의 의미**:
    - 행렬식은 행렬이 나타내는 변환에서 공간을 얼마나 늘리거나 축소시키는지를 나타냅니다.
    - 예를 들어, 2차원 평면에서 행렬의 행렬식이 0이 아니면, 그 행렬은 평면을 일정한 비율로 변환시키는 역할을 합니다. 반면, 행렬식이 0일 경우, 행렬은 공간을 완전히 축소시키며, 행렬이 비가역적(invertible)이므로 역행렬이 존재하지 않음을 의미합니다.
- **행렬식의 계산**:
    - 행렬식은 행렬의 대각 성분과 비대각 성분을 기반으로 계산되며, 이는 행렬의 성분이 클수록 복잡해질 수 있습니다.
    - 행렬식의 값이 0이면, 행렬은 특이행렬(singular matrix)이며, 역행렬이 존재하지 않습니다.

12. 주성분 분석 (PCA, Principal Component Analysis)

- **PCA의 정의**:
    - 주성분 분석(PCA)은 고차원의 데이터를 저차원으로 변환하면서도 데이터의 중요한 정보를 유지하는 기법입니다. 주로 데이터의 차원 축소(dimensionality reduction)에 사용됩니다.

- 이는 선형대수에서 고윳값 분해(Eigenvalue Decomposition) 또는 특잇값 분해(SVD)를 사용하여 데이터를 가장 잘 설명할 수 있는 주성분들을 찾아내는 방식입니다.

- **PCA의 원리**:
  - 주성분 분석은 고차원의 데이터에서 가장 중요한 축(Principal Axes)을 찾아내어, 그 축을 기준으로 데이터를 투영시킵니다.
  - 이는 데이터를 보다 간결하게 표현하며, 잡음(noise)을 제거하거나 데이터의 주요 특징을 더 잘 드러내는 데 사용됩니다.
  - 예를 들어, 원래 데이터가 n차원의 공간에 있다면, PCA는 그 공간에서 가장 중요한 k차원의 주성분을 찾아내어, 데이터를 저차원 공간에 표현하는 방식으로 축소합니다.

- **PCA의 수학적 표현**:
  - PCA는 주어진 데이터를 변환하여 저차원 공간에서 새로운 좌표계를 찾습니다. 이때 각 좌표축은 원래 데이터의 분산을 최대화하는 방향으로 정렬됩니다.
  - 수식으로는 다음과 같이 표현됩니다:

    $c^* = \arg\min_c \|g(c)\|$
    여기서
    $g(c)$는 저차원으로 투영된 데이터를 의미하며, L^2 노름을 최소화하는 방식으로 최적의 주성분을 찾습니다.

- **PCA의 활용**:
  - PCA는 데이터 분석, 기계 학습에서 차원 축소, 데이터 시각화, 노이즈 제거 등에 널리 사용됩니다.
  - 데이터를 저차원으로 축소하여 계산 복잡도를 낮추거나, 데이터의 특성을 더 명확하게 드러내는 데 중요한 역할을 합니다.

12.1 최적화 문제와 목적 함수

- 주어진 문제에서 최적화의 목표는 **입력 벡터** x를 **주성분 공간**으로 투영하는 **최적의 벡터** c^*를 찾는 것입니다. 이를 위해 **최소화 함수**가 사용됩니다.
- **최적화 대상 함수**는 L^2 노름을 기준으로 정의되며, 이는 주어진 함수 $g(c)$와 실제 입력 벡터 x의 차이를 최소화하려는 방식입니다.
- 수식으로는 다음과 같이 표현됩니다:

$c^* = \arg\min_c \|g(c) - g(x)\|^2$
여기서,
$g(c)$는 c벡터에 대응하는 함수로, 데이터의 차원 축소 과정에서 사용되는 함수입니다.

12.2 L^2 노름에 따른 함수의 전개

- 위의 수식 를 기반으로 L^2 노름을 적용하여, 해당 함수를 더 세부적으로 전개할 수 있습니다:

$$\|g(c) - g(x)\|^2 = (x - g(c))^T (x - g(c))$$

이는 최소화 대상을 명확히 하기 위한 수식으로, 함수의 차이를 벡터 내적을 이용해 표현한 것입니다.

- 이를 더 단순화하면 다음과 같은 형태로 나타낼 수 있습니다:

$$c^* = \arg\min_c -2z^T g(c) + g(c)^T g(c)$$

이는 최적화 문제의 목적 함수로, 각 성분 간의 관계를 나타내는 수식을 기반으로 최적의 c 값을 찾는 과정입니다.

12.3 최적화 계산 과정

- 목적 함수의 **미분**을 통해 최적값을 찾기 위한 과정이 다음 단계에서 설명됩니다:

$$c^* = \arg\min_c -2z^T Dc + c^T Dc + I_c$$

여기서 D는 투영 행렬로, 주성분 방향을 정의하는 중요한 역할을 합니다. 이 방정식은 최적화 문제를 풀기 위해 미분하여 계산한 것입니다.

- 다음 단계에서 벡터 미적분(vector calculus)을 적용하여, 최적화 문제를 풀 수 있는 방법을 제시합니다:

$$\nabla_c(-2z^T Dc + c^T Dc) = 0$$

이 과정은 함수의 **극값**을 찾기 위한 계산이며, 그 결과 최적의 c 값은 다음과 같이 정의됩니다:

$$c = D^T z$$

이는 주어진 입력 벡터 z를 주성분 공간으로 투영하는 최적의 벡터 c를 계산하는 결과입니다.

12.4 주성분 축소 및 재구성 함수

- 최종적으로, 벡터를 주성분 공간으로 투영한 후 다시 원래의 데이터 공간으로 **재구성**하는 과정을 설명합니다. 이때 **재구성 함수**는 다음과 같이 정의됩니다:

$$f(z) = D^T z$$

이는 주성분 공간으로 투영된 벡터를 다시 데이터 공간으로 변환하는 과정입니다.

- 이 과정에서 벡터의 L^2 **노름**을 최소화하는 방식으로 최적의 주성분을 찾아내고, 데이터를 가장 효율적으로 축소하여 표현합니다:

$$D' = \arg\min_D \sum \|x_i - D^T z_i\|_2^2, \text{ 단 } D^T D = I$$

이 수식은 주성분 분석의 핵심 목적 중 하나인 데이터 축소 과정에서 노이즈를 최소화하고 중요한 정보를 보존하는 역할을 합니다.

12.5 벡터 d에 대한 최적화 문제 정의

- 먼저, 주어진 문제는 $X \in \mathbb{R}^{m \times n}$ 행렬과 벡터 d가 있을 때, 특정 조건을 만족하는 d를 찾는 최적화 문제입니다.
- 주성분 분석에서 **최적의 투영 벡터** d를 구하는 문제로 시작됩니다. 이를 위해 먼저 다음과 같은 목적 함수가 정의됩니다:

$$d^* = \arg\min_d \|X - Xdd^T\|_F^2, \quad \text{단 } \|d\|_2 = 1$$

여기서, $\|\cdot\|_F$는 프로베니우스 노름(Frobenius norm)으로, 행렬의 각 성분의 제곱합을 구하는 방식입니다.

- 목적 함수는 벡터 d에 대해 **행렬을 투영하는 과정**에서 발생하는 오차를 최소화하는 문제로, 벡터 d가 단위 벡터임을 전제로 합니다.

13.6 최적화 과정의 전개

- 문제를 풀기 위해, 프로베니우스 노름을 **대각합(Trace)** 연산으로 변환할 수 있습니다.

$$d^* = \arg\min_d \mathrm{Tr}((X - Xdd^T)^T (X - Xdd^T))$$

대각합 연산을 사용하면 계산을 단순화할 수 있습니다. 이 과정을 전개하면 다음과 같은 형태로 나타납니다:

$$d^* = \arg\min_d \mathrm{Tr}(X^T X) - \mathrm{Tr}(X^T X dd^T)$$

- **대각합 성질**에 따라 위 식은 다음과 같이 단순화됩니다:

$$d^* = \arg\min_d \mathrm{Tr}(X^T X) - \mathrm{Tr}(d^T X^T X d)$$

이때 d가 최적화를 위해 어떤 역할을 하는지 파악하는 것이 중요합니다. 위 식에서 $X^T X$는 상수항이므로, 최적화 과정에서 실제로 영향을 미치는 것은 $d^T X^T X d$ 항입니다.

13.7 최종 최적화 문제

- **최종적으로**, 최적화 문제는 다음과 같은 형태로 표현됩니다:

  $$d^* = \arg\max_d \mathrm{Tr}(d^T X^T X d), \quad \text{단 } \|d\|_2 = 1$$

  이는 행렬 $X^T X$의 고유벡터를 구하는 문제로, 고윳값 분해(Eigenvalue Decomposition)를 사용해 풀 수 있습니다.

  이 최적화 문제는 행렬 $X^T X$의 가장 큰 고유값에 해당하는 고유벡터 d를 구하는 과정입니다. 따라서, **최적의 투영 벡터** d는 $X^T X$의 최대 고유값에 대응하는 고유벡터가 됩니다.

13.8 최적화의 의미

- 이 최적화 문제의 핵심은, 주성분 분석에서 가장 중요한 **주성분 방향**을 찾아내는 것입니다.
- 데이터의 분산을 가장 크게 설명하는 방향을 찾기 위해 **행렬의 고유값과 고유벡터**를 사용하여 벡터 d를 구합니다.
- 이 과정을 통해 최적의 투영 벡터를 찾을 수 있으며, 이는 **차원 축소** 또는 **데이터 압축**의 중요한 단계입니다.