데이터사이언스(김창훈)

Lee Jun Hyeok (wnsx0000@gmail.com)

April 1, 2026

목차

1. Introduction to Vectors

이 수업에서는 선형대수학을 기반으로 ML/DL 알고리즘을 이해해본다. ML/DL, data science 등에서는 데이터를 벡터로 나타내어 바라보는데, 수치화된 데이터를 바라보는 가장 강력한 수학적 언어가 선형대수학이기 때문이다.

1.1. Understanding of Multiplication

matrix-vector multiplication Ax=b, 그리고 matrix-matrix multiplication AB=C에 대해 더 이해해 보자. 참고로 vector라 하면 기본적으로 column vector를 의미한다.

1.1.1. Linear Combination of Colums

연립일차방정식에서 주로 등장하는 Ax=b 꼴의 matrix-vector multiplication을 나타내는 방법은 아래와 같이 두 가지다.

1. Row-wise way(dot product)
Row-wise way는 Ax=b를 수식 그대로 coefficient matrix A와 variable vector x의 단순 곱(dot product)으로 보는 방식이다. 즉, A의 각 row가 방정식 하나의 coefficent를 나타내고, 각 방정식들에 대한 공통해를 찾는다는 관점이다. 수식으로 나타내면 당연하게도 아래와 같다.

기하적으로 생각해 보면 각 방정식에 대응되는 선 또는 면 등을 좌표계에 그리고, 좌표계에서 일치하는 부분이 x인 것으로 이해할 수 있다.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

2. Column-wise way
Column-wise way는 Ax=b를 A의 column들에 대한 linear combination(일차결합)으로 보는 방식이다(linear combination of columns of A). 수식으로 나타내면 아래와 같다. ML/DL 관점에서는 Ax=b를 이런 방식으로 나타내는 것이 intuition을 기르기에 좋고, ai 관련 논문이나 서적 등에서도 이 방식을 주로 사용한다.

기하적으로 생각해 보면 각 column vector들을 좌표계에 나타내고, 이것들을 적절히 scalar배 해서 b를 만드는 것으로 이해할 수 있다.

$$x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix} + x_3 \begin{bmatrix} a_{13} \\ a_{23} \\ a_{33} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

column-wise way를 사용하면 Ax=b를 b에 대한 linear combination으로 이해할 수 있다는 장점도 있고, Ax=b의 차원이 달라지는 등의 변화가 생겼을 때에도 vector 하나만 추가해 주면 되므로 수식적인 이해와 수정이 용이하다는 장점도 있다. 또한 각 column vector들을 그리기만 하면 되므로 좌표계에 나타낼 때도 편리하다. 반면 row-wise way의 경우 각 방정식을 선이나 면 등의 형태로 좌표계에 나타내야 하는데, 특히 차원이 높아질수록 그리거나 이해하기 어렵다. 이에 따라 본 수업에서는 vector나 데이터셋 등을 나타낼 때 column-wise way를 사용한다.

column-wise way를 연립방정식 $Ax = b$에 적용해 보면, 임의의 vector b에 대해 Ax=b의 해가 존재하는지는 각 column vector들의 linear combination으로 b를 나타낼 수 있는지로 생각할 수 있다. 다시 말해, b가 임의의 n차원 vector일 때 column vector들의 부분집합이 해당 vector space의 basis라면 임의의 b에 대해서 해가 존재한다.

1.1.2. Linear Combination of Rows

앞에서 설명한 것처럼 Ax는 linear combination of columns of A이다. 동일한 방식으로 더 생각해 보면 $x^T A = b^T$는 linear combination of rows of A인 것으로 이해할 수 있다. 물론 이때 b는 $A^T x = b$이므로 $Ax = b$에서의 b와는 다른 값이다.

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

$$x_1 \begin{bmatrix} a_{12} & a_{12} & a_{13} \end{bmatrix} + x_1 \begin{bmatrix} a_{21} & a_{22} & a_{23} \end{bmatrix} + x_1 \begin{bmatrix} a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

물론 당연하게도 $x^T A = b^T$는 $A^T x = b$와 같은데, 이는 단순히 A를 transpose한 것이므로 linear combination of rows of A로 이해할 수 있다.

즉, 정리하면 Ax꼴은 A의 column vector들의 linear combination으로, xA 꼴은 A의 row vector들의 linear combination으로 이해할 수 있다.

1.2. Matrix Multiplication

1.2.1. Matrix Multiplication

column vector, row vector에 대한 linear combination의 측면에서 matrix multiplication을 이해해보자. 우선 기본적으로 matrix A, B, C에 대한 matrix multipliaction AB=C는 A, B, C의 각 원소 a, b, c에 대해서 아래와 같이 나타낼 수 있다.

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

이때 C의 각 row와 column은 A와 B를 활용한 linear combination으로도 나타낼 수 있다. 당연하게도 matrix multiplication을 생각해 보면 C의 각 row는 대응되는 A의 row와 B에 대한 multiplication, C의 각 column는 A와 대응되는 B의 column에 대한 multiplication이다. 즉, A의 i번째 row를 a_i^r, B의 i번째 column을 b_i^c, C의 i번째 row 또는 column을 c_i^r, c_i^c이라 하면 아래의 수식이 성립한다.

$$Ab_i^c = c_i^c, \quad a_i^r B = c_i^r$$

다시 말해, AB=C에서 C의 각 column은 A의 column vector의 linear combination이고, C의 각 row는 B의 row vector의 linear combination이다.

Ax가 A column의 linear combination으로 나타낼 수 있다는 것은 Ax가 A의 column space에 포함된다는 것이고, 이를 수식으로 나타내면 $Ax \in C(A)$이다. 또한 multiplication은 결합법칙이 성립하므로 당연하게도 $ABCx \in C(A)$이다. 단순히 $BCx = x'$으로 묶는 것으로 증명된다.

1.2.2. Matrix Multiplication 다르게 나타내기

1. Column과 Row의 곱으로 나타내기
앞에서 언급한 것처럼 $n \times m$ matrix A와 $m \times k$ matrix B의 multiplication $AB = C$는 아래와 같이 나타낼 수 있다.

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

이때 A의 i번째 column을 a_i^c, B의 i번째 row를 b_i^r라고 하면 아래와 같이 나타낼 수도 있다. 수식적으로는 이런 방식에서 특정 원소를 구하는 계산을 나타내보면 위의 수식과 같고, 직관적으로도 각 곱의 결과는 $n \times k$ 행렬이므로 이를 모두 더하면 AB와 같다.

$$AB = \sum_{k=1}^{m} a_k^c b_k^r$$

이런 식의 표기는 A의 각 열 또는 B의 각 행의 중요도에 따라 연산 결과에서 반영 비율을 바꿔보는 등의 접근를 가능하게 한다.

2. Submatrix의 곱으로 나타내기
matrix를 여러 submatrix 또는 블럭으로 나누고, 각 블럭을 성분으로 취급해 matrix multiplication을 계산할 수 있음.

$$AB = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{22}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$AB = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \cdot \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] = \left[\begin{array}{c|c} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ \hline A_{21}B_{11} + A_{22}B_{21} & A_{22}B_{12} + A_{22}B_{22} \end{array}\right]$$

2. Linear Equation

2.1. Elimination

2.1.1. Elimination

1. Elimination
Elimination(소거법)은 선형대수학에서 연립일차방정식의 해를 구하는 가장 기본적인 방법이면서, 행렬의 다양한 속성을 이해하는 데에 활용되는 기법이다.

Gaussian Elimination(가우스 소거법)은 matrix에 elementary operation을 적용해 Row Echelon Form(행 사다리꼴)으로 만드는 elimination이고, Gauss-Jordan Elimination(가우스-조던 소거법)은 동일하게 elementary operation을 적용해 RREF(Reduced Row Echelon Form, 기약행사다리꼴)로 만드는 elimination이다.

row echelon form은 아래의 조건 중 1, 2번 조건만 만족시키는 matrix이고, RREF는 아래의 조건을 모두 만족시키는 matrix이다. 이때 Pivot(피벗)은 각 row에서 처음으로 0이 아닌 원소이다. 당연하게도 matrix A를 row echelon form으로 변형했을 때 pivot의 개수는 A의 rank와 같다.

1. 0이 아닌 성분을 가지는 row는 모든 성분이 0인 행보다 위쪽에 위치한다.
2. 0이 아닌 성분을 가지는 row의 pivot은 위쪽 row의 pivot보다 오른쪽에 위치한다.
3. pivot이 존재하는 column에서 pivot을 제외한 다른 모든 원소의 값은 0이다 (pivot의 위아래가 모두 0이다.).
4. 모든 privot의 값은 1이다.

2. Elementary Operation and Elementary Matrix
matrix의 row[column]에 대한 아래의 세 연산을 Elementary Row[Column] Operation(기본행[열]연산)이라 한다. 또한 row와 column에 대한 이 연산들을 통틀어 Elementary Operation이라 한다.

1. A의 두 row[column]을 교환하는 것.
2. A의 한 row[column]에 0이 아닌 scalar를 곱하는 것.
3. A의 한 row[column]에 다른 row[column]의 scalar 배를 더하는 것.

Elementary Matrix(기본행렬)은 identity matrix I_n에 elementary operation을 한 번 적용하여 얻은 matrix이다. elementary operation을 적용하는 것은 그 matrix에 대응되는 elementary matrix(동일한 elementary operation을 I_n에 적용한 것)를 곱하는 것과 같고, 이에 따라 elementary operation을 적용하는 것을 수식적으로 나타낼 수 있다.

이때 elementary row operation을 적용하는 것은 elementary matrix를 왼쪽에 곱하는 것과 같고, elementary column operation을 적용하는 것은 elementary matrix를 오른쪽에 곱하는 것과 같다.

세 번째 elementary operation에 대한 matrix를 Elemination Marix(소거행렬)로, 첫 번째 elementary operation에 대한 matrix를 Permutation Matrix(치환행렬)로 부르기도 한다.

elementary row operation은 row space를 보존하지만, column space는 보존하지 않는다. 또한 이는 elementary matrix를 왼쪽에 곱하는 것과 같으므로 null space를 보존한다. 반면 elementary column operation은 column space는 보존하지만 row space 및 null space는 보존하지 않는다. 물론 두 operation 모두 rank는 보존한다.

3. 연립일차방정식의 풀이
Ax=b 꼴의 연립일차방정식은 Augmented Matrix(첨가행렬) $(A|b)$에 대해 elimination을 적용해 RREF로 변형하는 것으로 풀 수 있다.

1. matrix의 각 row에 대해 위에서 아래로 내려가면서 elementary row operation을 적용해 pivot의 값이 1이고, pivot과 동일한 column의 성분들 중 아래쪽에 있는 것들을 0으로 만든다.
2. 이후 아래에서 위로 올라가면서 privot과 동일한 column의 성분들 중 위쪽에 있는 것을 0으로 만든다.

4. 연립일차방정식의 해의 존재 여부 판단
연립일차방정식 $Ax = b$의 해가 임의의 b에 대해 존재하는지는 아래의 방법 중 하나로 판단할 수 있다.

1. A가 invertible이면 해가 존재한다. 또한 이 경우 $x = A^{-1}b$으로 해를 계산할 수 있다.
2. $rank(A) = rank(A|b)$이면 해가 존재한다. 이 경우 A column의 linear combination으로 b를 만들 수 있다.
3. $(A|b)$를 RREF로 나타냈을 때 모순인 수식이 나오는 행이 있으면 해가 존재하지 않는다.

연립일차방정식을 풀 때, elimination 대신 직관적으로 column의 linear combination으로 나타낼 수 있는지 확인할 수도 있는데, 이런 직관을 가지면 좋다고 한다.

2.2. Rank

2.2.1. Rank

1. Rank
Matrix의 Rank는 해당 matrix의 column space의 dimension이다. 즉, linear independent한 column의 개수이다.

임의의 matrix A의 rank는 A^T의 rank와 같다. 즉, rank는 해당 matrix의 row space의 dimension이고, linear independent한 row의 개수로 구할 수도 있다. 정리하면 **rank는 linear independent한 row 또는 column의 개수로 계산할 수 있다.** 이때 하나의 matrix에 대해서 independent한 column의 개수와 independent한 row의 개수가 동일하므로, row와 column 중 더 짧은 쪽보다 항상 rank가 작다.

elementary operation은 rank를 보존한다. 이에 따라 임의의 matrix에 elemination을 적용해 pivot이 존재하는 column의 개수로 rank를 구할 수 있다.

2. Full Column/Row Rank
임의의 $m \times n$ matrix A에 대해 column/row와 rank가 같은 경우를 Full Column/Row Rank라고 한다.

1) Full Column Rank
모든 column들이 independent한 경우이다. 즉, pivot이 모든 column에 존재한다.

이 경우 $Ax = b$에서 pivot variable이 열의 개수와 같고, free variable이 없다. free variable이 없기도 하고, dimension theorm에 의해 N(A)는 영공간이다. $x_{\text{complete}} = x_{\text{particular}}$ 이다. b가 A의 linear combination인 것을 고려하면 $n < m$인 경우 항상 해가 존재하는 것은 아니고, $n = m$인 경우에는 invertible이므로 해가 존재한다.

2) Full Row Rank
모든 row들이 independent한 경우이다. 즉, pivot이 모든 row에 존재한다.

이 경우 $Ax = b$에서 모든 b에 대해 해가 존재한다. $n > m$인 경우 rank(column space의 dimension)가 m이고 column은 길이 m의 vector이므로 colum space는 R^m이다. 이에 따라 임의의 길이 m vector b를 column의 linear combination으로 만들 수 있으므로 항상 해가 존재한다. $n = m$인 경우 invertible이므로 해가 존재한다.

2.2.2. Singular Matrix

1. Singular Matrix
column 또는 row가 independent하지 않은 matrix를 $n \times n$ Singular Matrix(특이 행렬)라고 하고, dependent한 matrix를 Non-singular Matrix라고 한다.

rank가 n이 아니면 singular, n이면 non-singular이다. 즉, singular matrix는 non-invertible이고, non-singular matrix는 invertible이다. invertible과 determinant가 0이 아닌 것은 필요충분이므로, determinant가 0이면 singular matrix, determinant가 0이 아니면 non-singular matrix이다.

정리하면, **singular인 것과 non-invertible인 것과 determinant가 0인 것은 동치 명제**이다.

2. Singular와 Non-singluar의 곱
$n \times n$ singular matrix A와 $n \times n$ non-singular matrix B의 곱 $AB = C$은 항상 singular matrix이다. 아래와 같이 여러 가지 방법으로 증명이 가능하다.

1) determinant는 행렬 곱을 보존하므로, 둘 중 하나의 determinant가 0이면 $\det(AB) = \det(A)\det(B) = 0$이다.

2) 행렬을 선형변환으로 생각해 보면, 둘 중에 하나라도 일대일대응이 아니면 AB도 일대일대응이 아니게 된다.

3) C의 각 column는 A의 column에 대한 linear combination이므로 independent할 수 없다. $rank(AB) = min(rank(A), rank(B))$인 것으로도 이해할 수 있다.

4) 귀류법으로도 접근이 가능하다. C가 non-singular라고 하면 차원정리에 의해 Cx=0인 0이 아닌 x가 존재하지 않는다. A가 singular이므로 $Ay = 0$인 0이 아닌 y가 존재하고, B는 non-singular이므로 $y = Bx$인 0이 아닌 x가 존재한다. $ABx = Ay = 0 = Cx$이므로 모순이다.

2.3. Invertibility

2.3.1. Invertibility

1. Invertibility
$n \times n$ 행렬 A에 대해서 $AB = BA = I$인 $n \times n$ 행렬 B를 A의 Inverse Matrix(역행렬)이라고 하고, A^{-1}로 표기한다. 또한 어떤 matrix가 inverse matrix가 존재하면 Invertible(가역)이라고 한다.

$n \times n$ matrix의 invertibility는 아래와 같은 기준으로 판단이 가능함.

1. rank가 n이면 invertible임.
2. determinant가 0이 아니면 invertible임.
3. Ax=0을 만족시키는 0이 아닌 벡터 x가 존재하면 non-invertible임.

$n \times n$ matrix A가 invertible하기 위한 필요충분조건은 A의 rank가 n인 것임. 즉, non-singular matrix인 것과 invertible, rank가 n인 것은 동치 명제임.

$n \times n$ matrix A가 invertible이면 rank가 n이고 nullity가 0이므로, 연립일차방정식 Ax=b는 유일한 해 $A`^{1}b$를 가짐. 또한 연립일차방정식의 해가 유일하면 nullity가 0인 것이므로 A는 invertible임.

2. Inverse Matrix 구하기
invertible인 matrix A의 inverse matrix는 $(A|I)$에 elementary row operation을 적용해 구할 수 있다. A에 해당하는 부분을 항등행렬로 만들면 결과로 나오는 $(I|B)$에서 B가 A의 inverse matrix이다.

이때 elementary row operation을 적용하는 것은 아래와 같이 elementary matrix를 곱하는 것과 같으므로, $EA = I$, $E = B$이다. 즉, E 자체가 A의 inverse matrix이다.

$$E_n \cdots E_2 E_1 (A|I) = E(A|I) = (EA|E) = (I|B)$$

3. Inverse Matrix의 성질
inverse matrix의 성질로는 아래와 같은 것들이 있다.

1. 정의에 따라 $AA^{-1} = A^{-1}A = I$가 성립한다.
2. $(AB)^{-1} = B^{-1}A^{-1}$이 성립한다.

2.4. Transpose와 Permutation

2.4.1. Transpose

1. Transpose
$n \times m$ matrix A에 대한 Transpose(전치)는 A의 row와 column을 뒤바꾸는 연산으로, A^T와 같이 표기한다. 즉, $(A^T)_{ij} = Aji$이다.

transpose는 아래와 같은 성질을 가진다.

1. $(A+B)^T = A^T + B^T$이다.
2. $(AB)^T = B^T A^T$이다. 당연하게도 $(ABC)^T = C^T B^T A^T$이다.
3. $(A^T)^{-1} = (A^{-1})^T$이다. $A^T(A^T)^{-1} = I$와, $AA^{-1} = I$를 transpose하고 비교해 증명할 수 있다.

2. Symmetric Matrix
Symmetric Matrix(대칭행렬)는 대각성분을 기준으로 각 성분의 값이 대칭인 square matrix이다. 즉, $A = A^T$이면 symmetric matrix이다.

또한 transpose를 사용해 inner product와 outer product를 정의할 수 있다. $n \times 1$ vector x와 y에 대해 inner product는 $x \cdot y = x^T y$로 정의되고(결과는 1×1), outer product는 xy^T로 정의된다(결과는 $n \times n$).

당연하게도 $A^T A$와 LDL^T 꼴의 matrix는 symmetric matrix이다.

2.4.2. Permutation Matrix

Permutation Matrix는 identity matrix의 row를 순서만 바꾼 matrix이다. 즉, identitiy matrix에 두 row를 바꾸는 elementary operation을 한 번 이상 적용한 matrix이다. 이에 따라 $n \times n$ identity matrix의 경우 $n!$개의 permutation matrix들이 존재한다.

$n \times n$ permutation matrix P는 아래의 성질을 가진다.

1. $P^{-1} = P^T$이다.
2. $P^T P = I$이다.

당연하게도 permutation matrix를 어떤 행렬의 왼쪽에 곱하는 것은 해당 permutation matrix에 해당하는 elementary operation들을 적용하는 것과 같다.

Diagonal matrix는 대각성분을 제외한 모든 성분이 0인 square matrix이다.

2.5. Decomposition

Decomposition(분해) 또는 Factorization(인수분해)는 하나의 matrix를 더 간단하거나 특정 성질의 matrix들로 나누는 기법이다. 여러 가지 decomposition들이 존재하는데, 여기에서는 CR decomposition, LU decomposition에 대해 알아보자. 추후에 eigen decomposition, spectral decomposition, SVD도 다룬다.

2.5.1. CR Decomposition

CR(Colum-Row) Decomposition은 임의의 $m \times n$ matrix A를 두 행렬 C와 R로 나누는 decomposition이다. 즉, $A = CR$ 꼴로 나눈다. 이때 C의 column은 A의 column space의 basis이고(basis of C(A)), R의 row는 A의 row space의 basis이다(basis of $C(A^T)$). 즉, **CR Decomposition은 column space와 row space의 basis를 찾아내는 분해**이다.

CR decomposition의 과정은 아래와 같다.

1. 첫 번째 matrix C를 구성한다. 첫번째 column은 A의 첫번째 column을 그냥 넣는다. 두번째 column은 넣은 첫번째 column으로 만들 수 있는지 판단하고, 만들 수 없으면 그냥 넣고 만들 수 있으면 넣지 않는다.
2. 두 번째 matrix R을 구성한다. 첫번째 matrix의 column들을 활용해 기존 matrix을 생성할 수 있도록 구성한다.

이때 A의 각 column이 C의 linear combination인 것을 고려하면 R을 채우기 쉽다. 즉, 다른 column들로 만들 수 있는 column이 아니면 하나만 1이고 나머지는 0인 column이 들어가게 되고, 다른 column들로 만들 수 있다면 해당 linear combination에 대응되는 값들이 들어가게 된다.

$$\text{Example} \quad A = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix} \quad \begin{matrix} \text{Columns 1 and 2 of } A \text{ go directly into } C \\ \text{Column 3} = \mathbf{2}\,(\text{Column 1}) + \mathbf{1}\,(\text{Column 2}) \end{matrix}$$

$$A = \begin{bmatrix} 1 & 2 & \mathbf{4} \\ 1 & 3 & \mathbf{5} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & \mathbf{2} \\ 0 & 1 & \mathbf{1} \end{bmatrix} = CR \qquad \begin{matrix} 2 \text{ columns in } C \\ 2 \text{ rows in } R \end{matrix}$$

(↓ 2 1 above the second matrix)

또는 아래와 같이 좀 더 명확한 방법도 존재한다.

1. elimination으로 A에 대한 RREF를 구한다.
2. 첫 번째 matrix C를 구성한다. A의 column 중 RREF에서 privot이 있는 위치의 것들을 가져와 순서대로 C를 구성한다.
3. 두 번째 matrix R을 구성한다. RREF에서 0으로만 이루어진 row를 모두 제거한 것을 R로 한다.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 7 & 10 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad A = CR = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

2.5.2. LU Decomposition

1. LU Decomposition

LU Decomposition은 $m \times n$ matrix A를 lower triangle인 $m \times m$ matrix L과 upper triangle인 $m \times n$ matrix U로 나누는 decomposition이다. 즉, $A = LU$ 꼴로 나눈다.

한 row에 scalar배 해서 다른 row에 곱하는 elementary operation를 사용하면 A를 row echelon form으로 만들 수 있는데, 이게 U이다. 그리고 해당 elimination에 사용된 elementary matrix의 inverse matrix를 모두 곱한 것이 L이 된다. 즉, $(E_k \cdots E_1)A = U$, $A = (E_k \cdots E_1)^{-1}U$, $L = (E_k \cdots E_1)^{-1} = E_1^{-1} \cdots E_k^{-1}$이다.

row에 scalar배 해서 다른 row에 곱하는 elementary matrix의 inverse matrix는, row 값에 곱하는 scalar인 Multiplier(승수)의 부호를 바꾼 것과 같다. 당연하게도 동일한 row에 multiplier의 음수를 곱해 더해야 원래의 값이 나온다. 이때 elimination은 위쪽 row들부터 순차적으로 수행되고, $L = E_1^{-1} \cdots E_k^{-1}$에서는 그 역순으로 수행되므로 아래쪽 row들부터 연산되어 서로의 연산에 영향을 주지 않는다. 즉, L의 성분으로는 대응되는 elementary operation들에 대한 multipier의 부호만 바꾼 것들을 그냥 넣어주면 된다.

$$E = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad E^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$E_3E_2E_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad L = E_1^{-1}E_2^{-1}E_3^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & -3 & 1 \end{bmatrix}$$

즉, LU decomposition은 아래와 같은 방법으로 구할 수 있다.

1. A에 row에 scalar배 해서 다른 row에 곱하는 elementary operation을 적용해 U를 만든다.
2. elementary operation에 사용된 multiplier의 부호를 바꿔 대응되는 위치에 넣어 L을 만든다.

2. PA=LU

invertible matrix에 대해 LU decomposition을 적용할 때 항상 한 row의 scalar배를 다른 row에 더하는 elementary operation으로 decomposition이 가능한 것은 아니다. 특히 pivot에 해당하는 위치의 성분이 0인 경우에 두 row를 바꾸는 elementary operation을 적용해야 할 수 있는데, permutation matrix를 곱하는 것으로 이를

적용할 수 있다. permutation matrix들은 해당 elimination 중간에 곱할 수도 있고, elimination 시작 전에 모두 적용할 수도 있는데 당연하게도 후자가 더 간결하다.

즉, 아래와 같이 작성할 수 있다.

$$P_n \cdots P_2 P_1 A = PA = LU$$

3. LU Decomposition의 장점
LU decomposition으로 matrix A를 L과 U로 나누는 것은 아래의 장점이 존재한다. 또한 두 개의 행렬로 나누지만 lower/upper triangle matrix이므로 A와 거의 동일한 storage를 사용할 수 있다.

1. A의 행렬식을 간단히 구할 수 있다.
2. 연립일차방정식 $Ax = b$를 더 쉽게 풀 수 있다. 특히 L과 U에 대한 연립일차방정식은 L에 대해서는 전방 대입을, U에 대해서는 후방 대입해서 풀 수 있다. 즉, 앞/뒤부터 순차적으로 대입하기만 하면 x를 구할 수 있다.
3. Ax=b 연산에 elimination을 사용하는 것은 $x = bA^{-1}$를 구하는 것에 비해 computation을 줄일 수 있다.

$n \times n$ matrix A에 대한 elimination cost를 생각해보자. 곱하고 더하는 것을 단위로 생각한다면 각 pivot의 아래쪽을 전부 0으로 만드는 연산은 해당 row를 제외한 다른 아래쪽 row 모두와 연산하므로 $\sum_1^n k^2 - k = \frac{1}{3}n^3 + ...$이다.

또한 Ax=b에서 b에 대해 동일한 elimination을 수행하는 것을 생각해보면 이에 대한 연산은 각 원소를 scalar 배 해서 그 아래 원소들에 더하는 것이므로 $(n-1) + (n-2) + \cdots + 1$이고, 후방 대입하는 것은 구하려는 변수 외의 값들을 사용해 전부 뺀 뒤에 한 번의 나누기를 하는 것이므로 $1 + \cdots + (n-1) + n$이다. 이 둘을 더하면 n^2이다.

즉, 정리하면 A와 b 각각에 대한 elimination에 $\frac{1}{3}n^3$과 n^2의 연산량이 필요하다. Ax=b는 $x = A^{-1}b$로도 구할 수 있는데, A^{-1}를 구하는 데에는 $2n^3$의 연산량이 필요하다고 한다. 즉, elimination을 활용하는 LU decomposition이 연산량 측면에서 더 효율적이다.

Lower Triangle Matrix(하삼각행렬)는 대각성분 위쪽 성분들이 값이 모두 0인 $n \times n$ matrix이고, Upper Triangle Matrix는 대각성분 아래쪽 성분들이 값이 모두 0인 $n \times n$ matrix이다. lower/upper triangle matrix의 determinant는 모든 대각성분의 값을 곱해 구할 수 있다.

LU에서 U를 diagonal matrix D로 쪼개 L 뿐만 아니라 U의 대각성분도 1로 만드는 LDU Decomposition이라는 decomposition도 존재한다. 즉, $A = LDU$로 분해한다.

3. Vector Space

3.1. Vector Space

3.1.1. Vector Space

1. Vector Space
Vector Space(벡터공간)는 scalar 곱과 vector 합에 대해 닫혀 있는 vector의 집합이다. 즉, 아래의 조건을 만족시키는 두 연산이 정의된 집합 V는 vector space이다.

1. 모든 $x \in V$, $y \in V$에 대해 $x + y \in V$이다.
2. 모든 scalar a와 모든 $x \in V$에 대해 $ax \in V$이다.

정의에 의해 영벡터만 존재하는 집합도 vector space이고, 이를 zero vector space(영벡터공간, 영공간)이라고 한다. zero vector space의 dimension은 0이다.

2. Subspace
어떤 vector space에 대한 Subspace(부분공간)은 해당 vector space에서 정의하는 scalar 곱과 vector space에 대해 닫혀 있는 부분집합이다. 즉, 아래의 조건을 만족시키는 vector space의 부분집합 W는 해당 vector space

의 subspace이다.

1. 모든 $x \in W$, $y \in W$에 대해 $x + y \in W$이다.
2. 모든 scalar a와 모든 $x \in W$에 대해 $ax \in W$이다.
3. W는 V와 동일한 0 vector를 포함한다.

matrix A의 column을 span해 생성된 vector space는 Column Space(열공간)라고 하고, $C(A)$로 표기한다. 마찬가지로 row를 span해 생성된 vector space는 Row Space(행공간)라고 하고, $R(A) = C(A^T)$로 표기한다.

정의에 의해 0 vector만 존재하는 vector space는 모든 vector space의 subspace이다.

vector space에 대한 엄밀한 정의가 존재하기는 하지만, 여기에서는 이 정도로만 정리한다.

참고로, R^n은 n개의 실수 값을 속성으로 하는 vector들의 대한 vector space이다.

matrix A에 대해 $Ax = b_1, Ax = b_2$는 해가 존재하고 $Ax = b_3$는 해가 존재하지 않으려면, 당연하게도 b_1과 b_2은 A의 column space에 존재하고 b_3는 존재하지 않으면 된다. 가장 쉬운 예시는 A의 각 열이 b_1, b_2인 것이고, b_3는 b_1과 b_2의 linear combination으로 만들 수 없는 상황이다.

참고로 P와 L이 vector space V의 subspace일 때, 당연하게도 '$P \cup L$은 subspace이다'는 거짓이고, '$P \cap L$은 subspace이다'는 참이다. vector 몇 개로 증명이 가능하다.

matrix들에 대한 vector space는 주로 M으로 표기한다. upper triangle matrix에 대한 vector space P, symmetric matrix에 대한 vector space S, $P \cap S$인 diagonal matrix에 대한 vector space 모두 M의 subspace이다.

3.1.2. Column Space and Null Space

1. Column Space
$m \times n$ matrix A의 Column Space(C(A))는 $Ay = x$를 만족시키는 vector x의 집합인 vector space이다. 즉, A의 column의 linear combination으로 생성되는 vector space로, R^n의 subspace이다.

column space가 subspace임은 vector 합과 scalar 곱에 대해 닫혀 있는 것으로 쉽게 증명할 수 있다.

당연하게도 Ax=b가 해를 가지려면 b는 A의 column space에 존재해야 한다.

A의 column space의 dimension은 A의 rank와 같다. 즉, $dim(C(A)) = rank(A)$이다.

2. Null Space
$m \times n$ matrix A의 Null Space(N(A))는 $Ax = 0$을 만족시키는 vector x의 집합인 vector space이다. 이때 A의 null space는 R^m의 subspace이다.

null space가 subspace임은 vector 합과 scalar 곱에 대해 닫혀 있는 것으로 쉽게 증명할 수 있다. 이때 x는 $n \times 1$ vector이므로 null space는 R^n의 subspace이다.

Ax=0을 만족하는 x($x \neq 0$)가 존재하면 A는 non-invertible이다. 즉, null space가 zero vector space가 아니면 A는 non-invertible이다. 이는 A column의 linear combination으로 0이 만들어질 수 있다는 것이고, linear independent하지 않으므로 당연하다.

$m \times n$ matrix A가 가지는 null space의 dimension은 A의 n-rank와 같다. 즉, $dim(N(A)) = n - rank(A)$이다.

참고로, 'Ax=b를 만족시키는 x의 집합이 항상 subspace이다.'는 거짓이다. 뒤에서 살펴볼 것처럼 subspace인 경우도 있고, 그렇지 않은 경우도 있다.

3.2. Complete Solution

3.2.1. Special/Particular Solution of Ax=b

Ax=b의 complete solution을 찾아보자.

1. Pivot Variable vs. Free Variable
elimination을 적용했을 때 row echelon form에서 pivot이 존재하는 column을 Pivot Column, pivot이 존재하지 않는 column을 Free Column이라고 한다. 당연하게도 pivot column과 동일한 위치의 column들의 집합은

column space의 basis이다.

이에 따라 Ax=b에서 x에 해당하는 각 원소를 다음과 같이 구분할 수 있다.

- pivot column에 대응되는 x의 원소는 Pivot Variable이라고 한다.

  matrix A의 rank는 row echelon form으로 변형했을 때 pivot의 개수와 같으므로, pivot variable의 개수는 rank와 같다.

- free column에 대응되는 x의 원소는 Free Variable이라고 한다. 이때 free variable은 뒤에서 다루는 것처럼 해를 구할 때 어떤 값이든 될 수 있기 때문에 free이다.

  뒤에서 다룰 것처럼, matrix A의 nullity는 그 basis인 special solution의 개수와 같고, speical solution의 개수는 free variable의 개수와 같으므로, free variable의 개수는 nullity와 같다.

pivot variable의 개수와 free variable의 개수를 dimension theorm으로 이해할 수 있다.

2. Special/Particular Solution
연립일차방정식 Ax=b에 대해서 Ax=b를 만족시키는 특정한 해를 Particular Solution, Ax=0을 만족시키는 특별한 해를 Special Solution이라고 한다. 이때 special solution은 free variable만큼 존재하며 그 집합은 A null space의 basis가 된다.

elimination의 elementary operation들은 null space를 바꾸지 않는다(참고로, column space는 바꾼다.). 즉, special solution은 elimination을 적용한 뒤에도 구할 수 있다. 이에 따라 연립방정식 Ax=0에 대한 **special solution은 다음과 같은 과정을 거쳐 구할 수 있다.**

1. A에 elimination을 적용해 RREF 또는 row echelon form인 matrix R로 변형한다.
2. $Rx = 0$에서 free variable들 중 하나에만 1, 나머지는 0을 대입해 각 경우에 대한 x를 구하면 그게 special solution이다. free variable이 하나인 경우 1을 대입하면 된다. pivot column의 linear combination으로 free column을 만들 수 있으므로, free variable에 임의의 값을 넣어도 Rx=0을 만족시키는 x를 항상 찾을 수 있다.

   이에 따라 free variable 개수만큼의 speical solution이 존재하게 된다.

   이때 RREF로 변형했으면 1을 대입하는 free variable에 대응되는 free column의 원소에서 부호만 바꾸면 special solution이므로 더 간단하다.

이렇게 구한 special solution들은 free variable 위치에 대해 하나만 1이고 나머지는 0이므로, 서로 linear indepedent이다. 또한 $Ax' = 0$을 만족시키는 임의의 x'은 special solution의 linear combination으로 나타낼 수 있으므로 special solution 집합의 span은 null space이다. 이는 pivot variable의 값은 free variable의 값들에 의해 유일하게 결정되므로, x'의 free variable 값과 같도록 special solution들의 linear combination을 구성하면 pivot variable의 값 또한 유일하게 결정되기 때문이다. 즉, **special solution의 집합은 null space의 basis**이다.

$$\boldsymbol{A}=\begin{bmatrix} p & p & f & p & f \\ | & | & | & | & | \end{bmatrix} \quad \boldsymbol{R}=\begin{bmatrix} \mathbf{1} & 0 & a & 0 & c \\ 0 & \mathbf{1} & b & 0 & d \\ 0 & 0 & 0 & \mathbf{1} & e \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{s}_1=\begin{bmatrix} -a \\ -b \\ \mathbf{1} \\ 0 \\ \mathbf{0} \end{bmatrix} \quad \boldsymbol{s}_2=\begin{bmatrix} -c \\ -d \\ \mathbf{0} \\ -e \\ \mathbf{1} \end{bmatrix}$$

Ax=b에 대한 **particular solution은** $Ax = b$ **또는** $Rx = b'$**에서 free variable 전부에 0을 대입하고 연립방정식을 풀어서 구할 수 있다.** pivot variable의 linear combination으로 모든 free column을 만들 수 있으므로, Ax=b의 해가 존재한다면 모든 free variable에 0을 대입했을 때 particular solution을 찾을 수 있다. 특히 RREF로 elimination을 했다면 pivot variable 값이 b의 값과 같아지므로 매우 단순하다.

3. Rank와의 관계
$m \times n$ matrix A에 대해 rank가 r일 때, n, m, r의 대소관계에 따라 Ax=b의 해가 어떻게 존재하는지를 살펴보자. 이는 앞서 살펴본 내용에 의하면 당연하다.

- r=m=n인 경우, Ax=b는 항상 유일한 해를 가진다.
- r=n<m인 경우(full column rank), Ax=b는 해가 없거나, 유일한 해를 가진다.

- n>m=r인 경우(full row rank), Ax=b는 항상 무한한 해를 가진다.
- r<m, r<n인 경우, Ax=b는 해가 없거나, 무한한 해를 가진다.

본 수업에서는 Ax=0에서 0에 기본행렬을 곱해도 0이므로 elimination을 적용해도 null space가 바뀌지 않는다고 하는데, 각 elementary operation에 대해 null space가 바뀌지 않는다는 것으로 증명할 수도 있다.

3.2.2. Complete Solution of Ax=b

1. Complete Solution
Ax=b에 대한 Complete Solution(x_c)은 Ax=b를 만족시키는 모든 해의 집합으로, 다음과 같이 하나의 particular solution(x_p)와 special solution($s_1, s_2 \cdots$)의 linear combination(x_n. null space의 임의의 vector)으로 나타낼 수 있다.

$$x_{\mathrm{c}} = x_{\mathrm{p}} + x_{\mathrm{n}} = x_{\mathrm{p}} + (c_1 s_1 + c_2 s_2 + \cdots)$$

이는 particular solution x_{p}에 대해 $Ax_{\mathrm{p}} = b$가 성립하고, special solution의 linear combination x_n에 대해 $Ax_n = 0$이 성립하므로, 이 둘을 연립해 $A(x_p + x_n) = b$로 정리할 수 있기 때문이다.

영벡터가 아닌 b에 대해 Ax=b의 complete solution은 영벡터를 포함하지 않으므로 항상 subspace가 아니다. 영벡터가 포함된다면 A0=b이므로 모순이다.

2. Complete Solution 구하기
Ax=b에 대한 complete solution은 앞아서 다룬 것처럼 다음과 같이 구할 수 있다. 물론 해가 없는 경우에는 구할 수 없다.

1. Ax=b에 대한 augmented matrix [A|b]에 elimination을 적용해 RREF 또는 row echelon form R로 변형한다.
2. free variable들 중 하나에만 1, 나머지는 0을 대입해 $Rx = 0$에서 각 경우에 대해 special solution을 구한다.
3. 모든 free variable에 0을 대입해 $Rx = b'$에서 particular solution을 구한다.
4. special solution의 linear combination과 particular solution을 더해 complete solution을 얻는다.

또는 '프리드버그 선형대수학'에서 다뤘던 것처럼 다음과 같은 방법으로도 구할 수 있다.

1. Ax=b에 대한 augmented matrix [A|b]에 elimination을 적용해 RREF 또는 row echelon form으로 변형한다.
2. 각 pivot variable에 매개변수를 부여한다.
3. 각 free variable들을 해당 매개변수로 나타낸다.
4. 매개변수에 대해 정리해 complete solution을 얻는다.

   결과로 도출된 complete solution에서 매개변수가 곱해져 있지 않은 vector는 particular solution이고, 매개변수가 곱해져 있는 vector는 special solution이다.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} -2t_1 + 2t_2 + 3 \\ t_1 - t_2 + 1 \\ t_1 \\ 2t_2 + 2 \\ t_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 0 \\ 2 \\ 0 \end{pmatrix} + t_1 \begin{pmatrix} -2 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + t_2 \begin{pmatrix} 2 \\ -1 \\ 0 \\ 2 \\ 1 \end{pmatrix}$$

3.3. Basis&Dimension

3.3.1. Linaer Independence

어떤 vector들의 집합 $\{v_1, v_2, \cdots, v_n\}$에 대해서 다음 수식을 만족시키는 0이 아닌 scalar $c_1, c_2, \cdots, c_n$이 존재

하지 않으면 이 집합은 Linear Independent하다고 한다. 즉, 해당 vector들의 linear combination으로 영벡터를 만들 수 없으면 linear independent하다.

$$c_1v_1 + c_2v_2 + \cdots + c_nv_n = 0$$

영벡터를 만들 수 있다는 것은 해당 집합 내에서 일부 vector들의 linear combination으로 다른 vector를 만들 수 있다는 것이다. 다시 말해, linear independent하다는 것은 해당 집합 내에 불필요한 vector가 존재하지 않는다는 것으로 이해할 수 있다.

vector $v_1, \cdots, v_n$으로 column을 구성한 $m \times n$ matrix A를 생각하자. 해당 vector 집합이 linear indepedent한 것(rank가 n)과 N(A)가 zero vector space인 것(nullity가 0)은 필요충분이다. 이에 따라 A에 대해 elimination을 적용해 rank를 계산하여 independent 여부를 확인할 수 있다.

3.3.2. Span

어떤 vector들의 집합 $\{v_1, v_2, \cdots, v_n\}$에 대해서 해당 vector들의 linear combination으로 vector space S를 생성할 수 있다. 이때 해당 vector들의 집합을 span하여 S를 생성했다고 하고, 이때의 vector space를 Span(생성공간)이라고 한다.

어떤 vector 집합 β를 span했을 때 vector space V가 생성되는지는, 그 vector space에 속하는 임의의 벡터 v를 β의 linear combination으로 나타낼 수 있음을 보이면 된다. 이때 elimination을 적용해 RREF로 나타내면 더 단순할 수 있다.

어떤 vector 집합을 span해서 생성한 집합은 항상 vector space이다.

예를 들어, matrix A column들의 집합을 span하면 A의 column space가 만들어진다.

3.3.3. Basis&Dimension

어떤 vector들의 집합 $\beta = \{v_1, v_2, \cdots, v_n\}$이 1) linear indepedent이면서 2) span했을 때 vector space V를 생성한다면, 이때 β는 V의 Basis(기저)라고 한다.

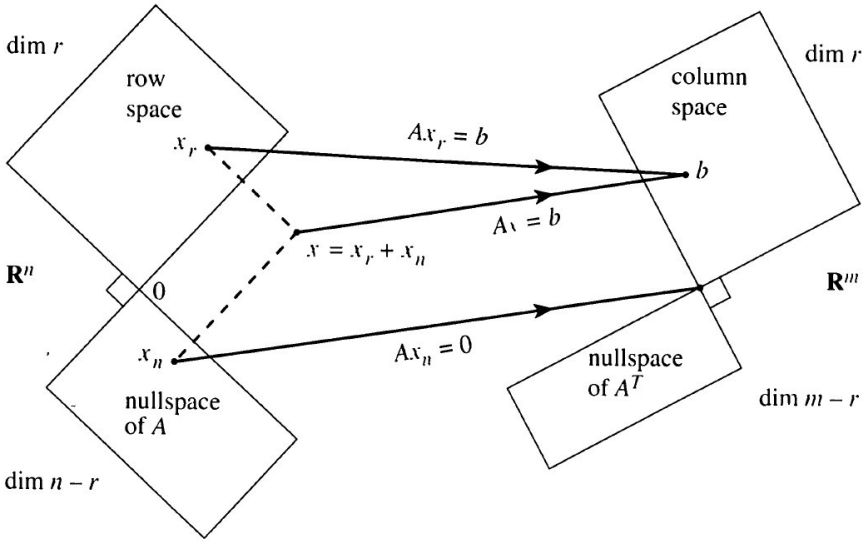
이 조건을 만족시키는 vector 집합은 여러 개일 수 있으므로 당연하게도 어떤 vector space에 대한 basis는 유일하지 않은데, 동일한 vector space에 대해서라면 모든 basis는 그 원소의 개수가 전부 같다.

2. Dimension
vector space V의 basis가 n개의 vector로 이루어질 때 n을 V의 Dimension(차원)이라고 한다.

3.4. Four Fundamental Spaces

3.4.1. Four Fundamental Spaces

연립일차방정식 Ax=b를 생각하자, rank가 r인 $m \times n$ matrix A는 다음 그림과 같이 4가지 fundamental subspace로 이해할 수 있다. orthogonality에 대한 내용은 뒤에서 따로 다룬다.



1. Column Space ($C(A)$)
Column Space는 앞에서 다룬 것처럼 길이 m vector인 column들의 linear combination으로 생성되는 subspace이다. 이때 길이 m인 vector들이므로 R^m의 subspace이다.

column space는 pivot variable에 대응되는 위치에 있는 column(RREF의 pivot column과는 다름.)들의 집합을 basis로 가진다. dimension은 r이다.

2. Null Space ($N(A)$)
Null Space는 앞에서 다룬 것처럼 Ax=0을 만족시키는 길이 n vector들로 구성된 subspace로, 이에 따라 R^n의 subspace이다.

null space는 special solution들의 집합을 basis로 가진다. dimension은 n-r이다.

3. Row Space ($C(A^T)$)
Row Space는 길이 n vector인 row들의 linear combination으로 생성되는 subspace이다. 이때 길이 n인 vector들이므로 R^n의 subspace이다.

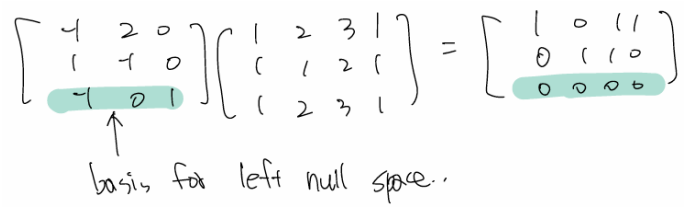
row space는 A에 대한 elimination을 통해 얻은 R에서 전부 0이 아닌 row의 집합을 basis로 가진다. dimension은 r이다(column space와 같다.).

4. Null Space of A^T ($N(A^T)$)
A^T의 Null Space 또는 Left Null Space는 $A^T x = 0$을 만족시키는 길이 m vector들로 구성된 subspace로, 이에 따라 R^m의 subspace이다. A^T의 null space는 $A^T y = 0$, $y^T A = 0$을 만족시키는 y의 집합이므로 left null space라고 부른다.

left null space의 basis는 다음과 같은 방법으로 구할 수 있다. dimension은 m-r이다.

- A^T null space의 basis를 직접 구한다.
- $[A|I]$에 elimination을 적용해 $[R|E]$로 변형한 뒤(EA=R 성립), R에서 모든 성분이 0인 row와 동일한 위치에 있는 E의 row들의 집합을 구성하면 그게 basis이다.

  이때 E는 invertible이므로 각 row는 independent하고, 전부 0인 row는 m-r개이므로 이렇게 뽑은 row들은 basis가 된다.



4. Orthogonality

4.1. Orthogonality

4.1.1. Orthogonality

1. Orthogonality of Vector
두 vector x, y에 대해서 inner product $x \cdot y = x^T y$가 0 또는 1이면 이 두 vector는 Orthogonal(직교)하다고 한다.

또한 x 또는 y가 영벡터인 경우에도 inner product가 0인데, 이 경우에도 orthogoanl하다고 한다. 즉, 영벡터는 모든 vector와 orthogonal하다.

이때 서로 orthogonal하면서 norm이 모두 1이면 Orthonormal하다고 한다.

영벡터가 아닌 두 vector가 orthogonal하면 linear independent이다.

2. Orthogoanlity of Vector Space
두 vector space S, T에 대해 모든 $v \in S$가 $w \in T$에 대해 orthogonal하면 S와 T는 orthogonal하다고 한다. 이는 두 vector space의 basis끼리 orthogonal한지 비교하는 것으로 판단할 수 있다.

당연하게도 zero vector space는 임의의 vector space와 orthogonal하다.

3. Row Space and Null Space
A의 row space는 A의 null space와 orthogonal하다. null space에 속하는 임의의 vector는 Ax=0을 만족시키고, 이에 따라 A의 모든 row와의 inner product가 0이므로 당연하다.

이때 이 두 subspace는 R^n에 대해 Orthogonal Complement(직교여공간) 관계에 있다. 즉, row space와 null space는 서로가 자신의 모든 vector와 orthogonal한 모든 vector를 포함하는 subspace이다.

또한 row space의 dimension은 n-r이고 null space의 dimension은 r이므로 이 두 subspace는 R^n을 완전하게 구성한다. 즉, 임의의 vector $x \in R^n$는 $x = x_{\text{row}} + x_{\text{null}}$과 같이 쪼갤 수 있다. 이를 A에 넣으면 $Ax = A(x_{\text{row}} + x_{\text{null}}) = Ax_{\text{row}} = b$이다.

이때 임의의 vector $b \in C(A)$는 오직 하나의 row space 상 vector와 대응된다. 이에 대한 증명은 간단하다. 어떤 두 vector $x_1, x_2 \in C(A^T)$에 대해 $Ax_1 = Ax_2 = b$가 성립한다고 가정하면 $A(x_1 - x_2) = 0$이므로 $x_1 - x_2 \in N(A)$인데, $x_1 - x_2$는 그 자체로 linear combintion이므로 $x_1 - x_2 \in C(A^T)$이다. 즉, row space와 null space의 intersection으로는 영벡터만 있으므로 모순이다.

4. Column Space and Left Null Space
A의 column space는 A의 left null space와 orthogonal하다.

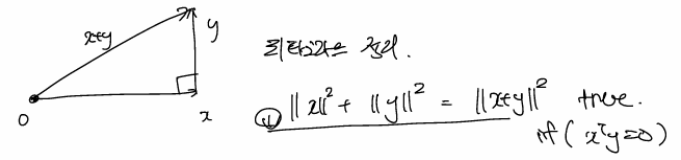
column space의 임의의 vector x_1에 대해서는 $Ay = x_1$이 성립하고, left null space의 임의의 vector x_2에 대해서는 $x_2^T A = 0$이 성립한다. $x_2^T A = 0$의 양변에 y를 곱하면 $x_2^T Ay = x_2^T x_1 = 0$이다.

Norm은 vector의 크기 또는 길이를 의미하는 개념이다. 구하는 방식에 따라 여러 가지 norm이 존재한다. 가장 흔하게 사용되고, 직선거리를 나타내는 L2 norm은 $||x||_2 = \sqrt{\sum_{i-1}^{n} |x_i|^2}$로 계산된다. 즉, 각 성분의 제곱을 전부 더한 뒤 루트를 씌운 것이다.

orthogonal하다는 것은 서로 수직하다는 의미이다. 피타고라스 정리를 예시로 생각해보자. 다음 그림과 같이 x와 y가 수직이라면 $||x||^2 + ||y||^2 = ||x + y||^2$가 성립한다. x와 y 각각의 성분을 $x_1, \cdots, x_n$, $y_1, \cdots, y_n$이라고 했을 때, $||x||^2 = x_1^2 + \cdots + x_n^2 = x^T x$이 성립한다. 이에 따라 피타고라스 정리는 다음 수식과 같이 정리된다.

$$x^T x + y^T y = (x + y)^T (x + y) = x^T x + x^T y + y^T x + y^T y$$

정리하면 $x^T y + y^T x = 0$이 성립하는데, 계산해보면 이 두 항의 값은 $x_1 y_1 + \cdots + x_n y_n$으로 같은 것을 알 수 있다. 즉, 수직인 두 vector x, y에 대해서 $x^T y = 0$이 성립한다.



당연하게도 intersection(교집합)이 존재하는 두 vector space는 orthogonal하지 않다.

4.1.2. Orthogonal Matrix

1. Orthogonal Matrix
Orthogonal Matrix는 orthonormal vector들로 column을 구성한 matrix이다. 즉, orthogonal matrix Q의 각 column을 $q_1, \cdots, q_n$이라고 하면 다음 수식이 성립한다. 이때의 vector들이 orthonormal이기는 하지만 관습적으로 orthogonal matrix라고 부른다고 한다.

$$q_i q_j = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

orthogonal matrix는 rotation matrix이다. orthogonal matrix Q를 생각하면, $(Qx)^T(Qx) = x^T Q^T Q x = x^T x$이므로 Q는 norm을 보존하고, $(Qx)^T(Qy) = x^T Q^T Q y = x^T y$이므로 Q는 각도(inner product)를 보존한다. 즉, 길이와 각도가 보존되는 변환이므로 이는 회전이다.

2. Orthogonal Matrix의 성질
orthogonal matrix에 대해 다음 성질들이 성립한다.

- $m \times n$ matrix Q에 대해 $n \times n$ matrix인 $Q^T Q$는 identity matrix이다. 곱해보면 당연하다.
- Q가 square matrix이면, $Q^T = Q^{-1}$이다. $Q^T Q = I$가 성립하므로 당연하다.

  예를 들어, permutation matrix와 그 transpose를 곱하면 identity matrix가 되는데, 이는 permutation matrix가 orthogonal matrix인 것으로도 설명이 가능하다.
- Q의 column space로 projection하는 projection matrix P는 QQ^T이다. 정리해보면 당연하다. 또한 만약 Q가 square matrix이면 $P = I$이다.

  Q가 $n \times n$ square matrix이면서 orthogonal matrix이면 R^n에서 R^n으로 보내는 matrix이므로, projection해도 이미 동일한 공간 안에 있고, 이에 따라 projection matrix가 I인 것으로도 이해할 수 있다.

orthogonal matrix가 구체적으로 왜 좋을까? 한 예시로, orthogonal matrix의 성질에 따라 뒤에서 설명할 least square problem을 더 쉽게 풀 수 있다. least square problem을 projection으로 풀 때 Ax=b를 $A^T A\hat{x} = A^T b$로 변형하는데, A가 orthogonal matrix라면 $A^T A\hat{x} = \hat{x} = A^T b$이다. 즉, 우변을 계산하기만 하면 된다.

4.1.3. Gram Schmidt Process

Gram Schmidt Process(그람 슈미트 과정)는 임의의 matrix W의 column들을 적절히 조정해서 orthogonal matrix Q로 변환하는 방법이다. 앞서 다룬 것처럼 matrix가 orthogonal이면 더 좋은 경우가 많다.

gram schmidt process에서는 projection을 활용해 orthogonality가 유지되도록 vector를 하나씩 추가한다. vector a, b에 존재할 때 a로 b를 projection한 vector는 p이고, a(p)에 orthogonal인 vector는 error인 $e = b - p$와 같다. 뒤에서 다룰 것처럼 projection matrix로 계산하면 $p = \frac{a^T b}{a^T a} a$이므로 $e = b - \frac{a^T b}{a^T a} a$이다.

이에 따라 matrix $W = [v_1, v_2, v_3, \cdots]$을 orthogonal matrix $Q = [q_1, q_2, q_3, \cdots]$로 변환하는 gram schmidt process는 다음과 같은 과정으로 수행된다.

1. v_1을 q_1으로 한다.
2. $i > 1$일 때 q_i는 $q_1, \cdots, q_{i-1}$들과 모두 orthogonal인 vector이어야 하므로, projection에 의한 error를 반복해 구하는 것으로 얻을 수 있다. 즉, q_i를 $q_1, \cdots, q_{i-1}$ 각각에 projection한 vector를 $p_1, \cdots, p_{i-1}$라 하면 $q_i = v_i - p_1 - \cdots - p_{i-1}$이다.

   예를 들어, q_3는 다음과 같이 구할 수 있다.

$$q_3 = v_3 - \frac{q_1^T v_3}{q_1^T q_1} q_1 - \frac{q_2^T v_3}{q_2^T q_2} q_2$$

이때 $\frac{q_1^T v_3}{q_1^T q_1}$가 scalar이므로 $q_1, \cdots, q_n$은 $v_1, \cdots, v_n$의 linear combination으로 표현되고, 이에 따라 W와 Q의 column space는 같다.

4.2. Projection

4.2.1. $A^T A$ 활용하기

$m \times n$ A에 대해 실제로 Ax=b를 푸는 상황을 공학적으로 생각해 보면, column은 고정되고 여러 개의 입력값들이 각 row에 들어오게 된다. 즉, $m > n$인 A에 대해 계산해야 하고, 항상 해가 존재하지는 않는다. 이런 경우 $Ax = b$를 $A^T Ax = A^T b$로 변형해 문제를 푸는 것이 유리하다. 뒤에서 설명할 것처럼, 이는 A의 column space에 b를 projection하고, 이에 따라 해가 존재하지 않는 원래 문제에 대해 가장 좋은 근사해를 구할 수 있다.

또한 $A^T A$는 다음과 같은 성질을 가진다. 특히 세 번째 성질이 중요하다.

- $A^T A$는 square matrix이다.
- $A^T A$는 symmetric matrix이다.
- $A^T A$와 A의 null space는 같다. 이에 따라 A가 full column rank를 가진다면 nullity가 0이고, $A^T A$의 nullity도 0이 되어 invertible이다.

  반대로 A가 full column rank를 가지지 않는다면 $A^T A$는 non-invertible이다.

마지막 성질은 1. $N(A^T A)$의 임의의 vector가 $N(A)$에 속하는가, 그리고 2. $N(A)$의 임의의 vector가 $N(A^T A)$에 속하는가를 보여 증명할 수 있다.

1번은 양변의 왼쪽에 x^T를 곱하고 정리하면 $(Ax)^T Ax = u^T u = 0$이고, inner product가 0이 되는 vector는 영벡터밖에 없으므로 $Ax = 0$이 성립하는 것으로 증명할 수 있다. 2번은 $Ax = 0$인 x를 $A^T A$에 넣어 쉽게 증명할 수 있다.

4.2.2. Projection

Ax=b의 해가 없다면, b를 C(A)로 projection하여 $A\hat{x} = p$를 구할 수 있다.

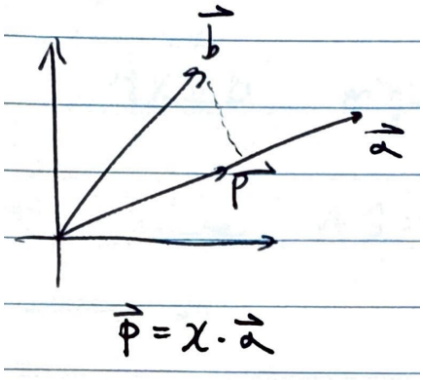
1. 벡터에 대한 projection
벡터 a와 b가 존재할 때, b에 대응되는 projection matrix를 곱해 b를 a로 projection할 수 있다. projection matrix는 아래와 같다. 이는 scalar x에 대해 $p = xa$라고 하고, a에 orthogonal인 vector err를 b-p로 정의해 $a^T(b-p) = 0$을 정리하여 구할 수 있다. 정리해 보면 $a^t a$는 상수이므로 $x = \frac{a^T b}{a^t a}$, $p = xa = ax = a\frac{a^T b}{a^T a} = \frac{aa^T}{a^T a} b$ 이다.

$$P = \frac{aa^T}{a^T a}, p = \frac{aa^T}{a^T a} b$$

projection matrix P에 대해 아래의 성질이 존재한다. 또한 수식적으로 봤을 때 b를 scalar배하면 p도 동일한 만큼 scalar배 되지만, a는 scalar배 해도 p는 변하지 않는다.

- $rank(P) = 1$이다. $a^T a$는 scalar이고 aa^T는 $n \times n$ matrix가 되는데, 모든 column이 a의 linear combination으로 만들어지므로 당연하다.
- $C(P)$는 a의 span이다. 이 또한 P의 모든 column이 a의 linear combination으로 만들어지므로 당연하다.
- $P^T = P$이다.
- $P^2 = P$이다. 이는 수식적으로도 증명이 되고, projection을 했는데 또 하는 것은 변화가 없으므로 당연하다.

어떤 matrix가 projection matrix인지는 $P^2 = P$, $P^T = P$가 성립하는지로 보일 수 있다.



2. 평면에 대한 projection
평면에 대한 projection도 직선에 대한 projection과 유사하게 계산할 수 있다.

어떤 평면에 대해 기저가 $\{a_1, a_2\}$이고, 기저로 열을 구성한 행렬 A가 있다고 하자. 정사영된 벡터 p를 해당 평면의 기저로 표현하면, $p = x_1a_1 + x_2a_2 = Ax$와 같이 나타낼 수 있다. 이때 $a_1^T(b - Ax) = a_0^T(b - Ax) = 0$ 이므로, 정리하면 $A^T(b - Ax) = 0$이다. 이를 정리하면 projection matrix는 아래와 같다.

$$P = A(A^TA)^{-1}A^T, p = A(A^TA)^{-1}A^Tb.$$

여기에서도 $P^2 = P, P^T = P$가 성립한다.

3. Subspace로의 Projection
이를 더 확장하여 임의의 subspace(또는 vector space)에 대한 projection을 생각해보자.

어떤 subspace에 vector b를 projection한 vector가 p라고 하면, p는 subspace의 basis로 나타낼 수 있다. 또한 해당 subspace의 basis가 $a_1, \cdots, a_n$이라고 하면 이걸로 column을 구성한 matrix A를 생각할 수 있다.

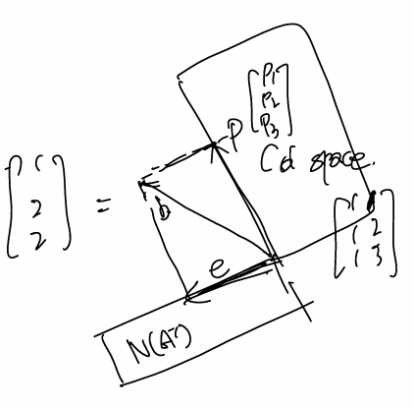
즉, $p = Ax$로 나타낼 수 있고, a_i에 대해 $a_i \cdot (b - Ax) = 0$이 성립하므로 $A^T(b - Ax) = 0$로 나타낼 수 있다. 이때 당연하게도 $e = (b - Ax)$는 fundamental space 중 $N(A^T)$에 포함된다. 즉, $C(A)$와 orthogonal이다.

이제 $A^T(b - Ax) = 0$를 정리하면 $A^TAx = A^Tb$이다. 이때 A를 basis로 구성했으므로 A^TA는 invertible이다. 즉, $x = (A^TA)^{-1}A^Tb$로 정리할 수 있다. 만약 A를 구성할 때 basis가 아니라 linear dependent한 어떤 vector의 집합을 사용했다면 x에 대해 정리할 수 없으므로 변수 설정이 잘못된 것이다.

이를 대입하면 $p = Ax = A(A^TA)^{-1}A^Tb$이다. 즉, subspace로의 projection matrix P는 평면으로의 projection에서와 동일하게 $P = A(A^TA)^{-1}A^T$이다. 참고로 이때 $P = A(A^TA)^{-1}A^T$의 괄호는 풀고 싶게 생겼지만, A가 invertible하다는 보장이 없으므로(심지어 square matrix가 아닐 수도 있다.) 불가능하다.

당연하게도 b를 A에 대해 projection할 때, $b \in C(A)$이면 그대로 b이다. 또한 b가 A에 직교하면 영벡터가 된다. 단순 대입으로 쉽게 증명할 수 있다. $b \in C(A)$이면 $Ac = b$라고 하자. $Pb = A(A^TA)^{-1}A^TAc = A(A^TA)^{-1}(A^TA)c = Ac = b$이다. 또한 b가 A에 직교하면 A의 모든 basis와 직교하고, $b \in N(A^T)$이므로 $A^Tb = 0$이다. 이것도 대입해보면 $Pb = A(A^TA)^{-1}A^Tb = 0$이므로 영벡터인 것을 알 수 있다.

projection에서 $e = b - p$였다. 즉, $b = p + e$이다. fundamental space에서 보면 $C(A)$의 p와 $N(A^T)$의 e가 더해져서 공간 외부의 b를 만드는 것을 알 수 있다. 즉, projection matrix P를 곱하면 외부의 b에서 p가 된다. 이때 b에 어떤 matrix Q를 곱해 $e \in N(A^T)$로도 projection할 수 있다. 즉, $e = Qb$이다. 정리하면 $b = Pb + Qb$, $Q = I - P$이다. 즉, Q는 identity matrix에서 P를 뺀 것이다. 추가로 이런 Q도 projection matrix이므로 앞에서 다룬 성질이 성립한다.



참고로 projection matrix의 eigen value는 0 또는 1이다. 해당 공간에 없는 vector들에 대해서는 0, 해당 공간에 있는 vector들에 대해서는 1이다.

4.2.3. Least Square Problem

1. Least Square Problem
Least Sqaure Problem은 평면에서 여러 점들에 대해서 오차를 최소화하는 직선을 찾는 문제이다. projection을 활용해 이를 해결할 수 있다.

이런 최적의 직선을 $f(x)$라고 하자. 각 t에 대해 $f(t) = \hat{y} = C + Dt$라고 할 수 있고, 각 점의 y값과의 차이를 오차값 error로 정의할 수 있다.

$f(t)$를 Ax=b꼴로 나타내보자. 각 점을 대입하면 $C + Dx_1 = y_1, \cdots\ C + Dx_n = y_n$과 같고, C와 D를 변수로 하는 이 연립일차방정식을 다음과 같이 Ax=b꼴로 나타낼 수 있다.

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} C \\ D \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

물론 이때 $b \in C(A)$이면 해가 존재하지만 대체로 해가 없는 경우가 많고, 그렇다면 $err = b - Ax$, $Ax + err = b$에 대해서만 해가 존재한다. 즉, $Ax = b$로 나타냈을 때 Ax가 b에 최대한 가깝도록 하는 x를 찾는 문제이다. 이는 C와 D에 대한 수식을 작성한 뒤 편미분해서 error가 최소가 되도록 하는 C와 D 값을 찾거나, projection을 통해 C와 D 값을 찾는 것으로 해결할 수 있다.

C와 D를 찾았으면 당연하게도 실제 예측값과 error를 구할 수 있다.

2. 편미분으로 풀기
각 점의 좌표를 (x_i, y_i)라고 했을 때 err의 각 성분은 $e_i = |f(x_i) - y_i|$이다. 즉 $L = e_1^2 + \cdots + e_n^2$을 최소로 하는 C와 D 값을 편미분해서 찾을 수 있다.

예를 들어, 위의 예시에서는 $C + D + e_1 = 1$, $C + 2D + e_1 = 2$, $C + 3D + e_1 = 2$이고 $L = (C + D - 1)^2 + (C + 2D - 2)^2 + (C + 3D - 2)^2$으로 정리할 수 있다. 이에 대해 편미분해 $\frac{\partial L}{\partial C}$, $\frac{\partial L}{\partial D}$가 0이 되는 C와 D를 찾으면 그게 최적의 직선이다.

3. Projection으로 풀기
Ax=b를 $A^T A\hat{x} = A^T b$로 변형하면 A의 column space로 b를 projection한 문제를 푸는 것으로 바꿀 수 있다. 앞에서 다룬 것처럼 A가 full column rank를 가지면 $A^T A$는 invertible하고, 항상 해가 존재한다. 즉, 단순히 $A^T A\hat{x} = A^T b$에 대해 해를 찾으면 C와 D를 알 수 있다.

5. Determinant

5.1. Determinant

5.1.1. Determinant

1. Determinant
$n \times n$ matrix A의 Determinant(행렬식)는 $det(A)$ 또는 $|A|$로 표기하고, 다음과 같이 계산한다.

- A가 1×1 matrix인 경우, $det(A) = A_{11}$이다.
- A가 2×2 matrix인 경우, $det(A) = A_{11}A_{22} - A_{12}A_{21}$이다. $(ad - bc)$
- A가 $n > 2$인 n에 대해서 $n \times n$ matrix인 경우, determinant는 임의의 row 또는 column에 대한 cofactor formula로 구할 수 있다.

  모든 i에 대해서 row i에 대한 cofactor formula를 하면 determinant는 아래와 같다.

$$det(A) = \sum_{j=1}^{n} (-1)^{i+j} A_{ij} det(M_{ij})$$

  또는 모든 j에 대해서 row j에 대한 cofactor formula를 하면 determinant는 아래와 같다.

$$det(A) = \sum_{i=1}^{n} (-1)^{i+j} A_{ij} det(M_{ij})$$

determinant가 0이면 그 matrix는 singular/non-invertible matrix이다. determinant는 singularity를 결정하므로 그 이름이 붙었다.

2. Cofactor
$n \times n$ matrix A를 생각하자. scalar $(-1)^{i+j} det(M_{ij})$는 A의 i행 j열 성분에 대한 Cofactor(여인수)라 한다.

cofactor를 $c_{ij} = (-1)^{i+j} det(M_{ij})$로 표기했을 때, determinant를 다음과 같이 cofactor들의 linear combination으로 나타낼 수 있다. 이를 i번째 row에 대한 Cofactor Formula(Cofactor Expansion, 여인수 전개) 또는 Laplace expansion(라플라스 전개)라고 한다.

$$det(A) = A_{i1}c_{i1} + A_{i2}c_{i2} + \cdots + A_{in}c_{in}$$

A의 i번째 row와 j번째 column을 지워서 얻은 $(n-1) \times (n-1)$ matrix를 A의 (i, j) Minor(소행렬)라고 하고, M_{ij}로 표기한다.

determinant는 그 정의상 square matrix에 대해서만 구한다.

5.1.2. Determinant의 성질

determinant는 다음과 같은 성질을 가진다. 이때 1번부터 3번까지의 성질로 나머지 성질들을 유도할 수 있다.

1. identity matrix I에 대해 $\det I = 1$이다.
2. matrix A의 두 row의 위치를 바꾼 matrix B에 대해서 $\det B = -\det A$가 성립한다.

   즉, 짝수 번 row를 교환하면 부호가 그대로, 홀수 번 row를 교환하면 부호가 바뀐다.
3. a. matrix A의 한 row에 scalar c를 곱한 matrix B에 대해 $\det B = c \det A$가 성립한다.

   이에 따라 A가 $n \times n$ matrix인 경우 scalar c에 대해 $\det cA = c^n \det A$이다.

   b. matrix A의 determinant는 한 row의 값에 따라 분할한 뒤 각각에 대한 determinant를 계산한 것과 같다. 즉, 다음이 성립한다.

$$\begin{bmatrix} a'+a & b'+b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} a' & b' \\ c & d \end{bmatrix}$$

4. matrix A가 동일한 row를 가지면, $\det A = 0$이다.

   row를 바꿔서 determinant의 부호가 바뀌어도 동일하므로 당연하다. 이때 동일한 row가 있다는 것은 full rank가 아니라는 것이고, 이에 따라 non-invertible이게 된다.

5. 어떤 row의 scalar배를 다른 row에 더하는 row elementary operation은 determinant를 보존한다(바꾸지 않는다.). 즉, elemination을 해도 determinant가 보존된다.

   해당 operation이 적용된 matrix는 다음과 같이 3번 성질을 사용해 두 matrix로 나눌 수 있고, scalar배 한 것은 matrix 밖으로 뺄 수 있다. 이렇게 정리해 보면 두 row가 같으므로 determinant가 0이 된다.

   $$\begin{bmatrix} a & b \\ c-la & d-lb \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} - l \begin{bmatrix} a & b \\ a & b \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

   이전에 배운 걸 생각해 봐도 만약 determinant가 보존되지 않았다면 elemination으로 invertible 여부를 판단할 수 없었을 것이고, complete soluation을 구할 수 없었을 것이다. 또한 이 성질을 이용해서 operation을 적용해 보면 2×2 matrix의 determinant가 왜 ad-bc로 계산되는지도 쉽게 증명할 수 있다.

6. A가 0으로 이루어진 row를 가지면 A의 determinant는 0이다.

   full rank가 아니므로 당연하고, 해당 row에 scalar 0이 곱해져 있는 것으로 봐도 당연하다. 또한 cofactor formula를 해 봐도 당연하다.

7. upper diagonal matrix의 determinant는 대각성분의 곱과 같다. 또한 마찬가지로 lower diagonal matrix의 determinant도 대각성분의 곱과 같다.

   한 row의 scalar배를 다른 row에 더하는 row elementary operation을 적용해 diagonal matrix로 변형하고, 3번 성질에 의해 각 row의 scalar를 앞으로 빼 보면 당연하다.

8. $\det A = 0$인 것과 A가 singular인 것, 그리고 $\det A \neq 0$인 것과 A가 non-singular인 것은 필요충분이다.

9. $\det AB = \det A \cdot \det B$이다.

   A가 invertible이면 elementary matrix의 곱으로 나타낼 수 있고, elementary matrix와의 곱에 대해서는 $\det EA = \det E \det A$가 성립하는 게 자명하므로(operation 종류별로 증명 가능하다.) 증명할 수 있다. A가 invertible이 아니면 AB도 invertible이 아니므로 성립한다.

   이에 따라 A가 invertible인 경우 $\det A^{-1} = \frac{1}{\det A}$이다. 또한 $\det A^n = (\det A)^n$이다.

10. $\det A = \det A^T$이다. 즉, 앞서 다룬 성질들은 row에 대한 것이었는데, 이는 column에 대해서도 성립한다.

    A에 대해 LU분해를 해서 보면, $\det U^T \det L^T = \det L \det U$임을 보이면 되므로 당연하다.

1번부터 3번까지의 증명은 다루지 않는다.

이런 determinant의 성질을 이용해 cofactor formula가 왜 성립하는지 직접 계산해 볼 수 있다. 예를 들어, 3×3 matrix A의 각 원소가 $a_{ij} + 0$이므로 3번 성질을 적용하면, 각 row에 하나의 원소를 제외하고는 모두 0인 matrix들(총 27개)로 분리할 수 있다. 이때 한 column이 모두 0인 matrix은 determinant가 0이므로 row와 column에 하나의 원소만 존재하는 matrix들에 대한 determinant를 계산해 더하면 된다. 즉, 아래와 같은 matrix들에 대한 determinant를 더하면 된다.

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix}, \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & 0 & a_{23} \\ 0 & a_{32} & 0 \end{bmatrix}, \begin{bmatrix} 0 & a_{12} & 0 \\ 0 & 0 & a_{23} \\ a_{31} & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & a_{12} & 0 \\ a_{21} & 0 & 0 \\ 0 & 0 & a_{33} \end{bmatrix}, \begin{bmatrix} 0 & 0 & a_{13} \\ a_{21} & 0 & 0 \\ 0 & a_{32} & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & a_{13} \\ 0 & a_{22} & 0 \\ a_{31} & 0 & 0 \end{bmatrix}$$

이 matrix들은 row를 바꾼 것이므로 0이 아닌 각 성분을 전부 곱한 뒤 부호만 맞춰주면 된다. 즉, $\det A = a_{11}a_{22}a_{33} - a_{11}a_{32}a_{23} + a_{21}a_{12}a_{33} - a_{21}a_{12}a_{33} + a_{21}a_{32}a_{13} - a_{31}a_{22}a_{13} = a_{11}C_{11} + a_{12}C_{12} + a_{13}C_{13}$가 된다.

$n \times n$ matrix에 대해서는 이렇게 나뉜 matrix가 총 $n!$개 이므로 복잡도도 그 $n!$이다. 또한 **각 row에서 column이 겹치지 않게 0이 아닌 성분을 하나씩 고르고, diagoanl matrix를 만들기 위해 row를 바꾸는 횟수로 부호를 정하는 것으로도 determinant를 구할 수 있음을 알 수 있다.** 즉, 다음과 같은 matrix B의 determinant는 각 row에 대해 column 4, 3, 2, 1 또는 column 3, 2, 1, 4의 성분을 사용하고, 각각 부호는 +, -이므로 determinant가 0이다. 이렇게 invertibility를 판정할 수도 있다.

$$\det B = \det(\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}) = 1 - 1 = 0$$

5.2. Determinant의 활용

5.2.1. Inverse Matrix와 Determinant

1. Cofactor Matrix와 Adjoint Matrix
$n \times n$ matrix A에 대한 Cofactor Matrix C는 다음과 같이 A에 대한 cofactor로 구성된 matrix이다.

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix}$$

Adjoint Matrix(수반 행렬)는 cofactor matrix를 transpose한 matrix이다.

$$\text{adj}(A) = C^T$$

2. Inverse Matrix와 Determinant
invertible matrix A에 대한 inverse matrix는 다음과 같이 determinant와 cofactor(adjoint) matrix로도 구할 수 있다.

$$A^{-1} = \frac{1}{\det A}\text{adj}(A) = \frac{1}{\det A}C^T$$

이는 $\det A \cdot I = AC^T$가 성립함을 보여 증명할 수 있다. 대각성분에 대해서는 cofactor formula를 생각해보면 당연하다. 대각성분 외의 성분들에 대해서는 B라는 가상의 matrix를 생각해 증명할 수 있다. B는 A와 동일한 성분을 가지는 $n \times n$ matrix인데, k번째 row만 어떤 i($i \neq k$)번째 row와 성분이 동일한 marix이다. k번째 row에 대해 cofactor formula를 적용해 B의 determinant를 구해 보면 $\det B = \sum_{j=1}^{n} a_{ij}C_{kj}$이다. 그런데 B는 두 row가 같은 matrix이므로 determiant가 0이고, 이에 따라 $\sum_{j=1}^{n} a_{ij}C_{kj} = 0$이다. 즉, 서로 다른 row의 성분과 cofactor를 사용해 계산한 값은 항상 0이 된다.

5.2.2. Cramer's Rule

Cramer's Rule은 A가 invertible일 때 Ax=b의 해를 determinant를 사용해 구하는 방법이다. vector x의 각 성분을 $x_1, \cdots, x_n$이라 하고, matrix B_i는 i번째 column을 vector b로 교체한 matrix라 하면 각 성분은 다음과 같이 구할 수 있다.

$$x_i = \frac{\det B_i}{\det A}$$

이는 간단히 증명할 수 있다. A가 invertible이므로 $x = A^{-1}b = \frac{1}{\det A}C^Tb$로 정리할 수 있으므로, $1 \leq k \leq n$인 k에 대해 $\sum_{j=1}^{n} C_{jk}b_j$의 값을 확인하면 된다. B_i는 i번째 column을 제외하고는 A와 성분이 같으므로 $\det B_i$를

i번째 column에 대해 구하는 것을 생각하면, $\det B_i = b_1C_{11} + b_2C_{21} + \cdots + b_nC_{1n} = \sum_{j=1}^{n} C_{jk}b_j$이다. 즉, x의 각 성분 x_i를 $\frac{\det B_i}{\det A}$로 구할 수 있다.

x의 성분만큼 determinant를 구해야 하므로 복잡도가 높고 비효율적이어서 실제로는 잘 사용하지 않는다고 한다.

5.2.3. Volume과 Determinant

matrix A의 determinant의 절댓값 $|\det A|$는 A의 row들이(당연히 column이어도 가능하다.) 구성하는 도형의 Volume(부피)이다.

이는 앞서 다룬 determinant의 여러 성질을 사용해 확인할 수 있다. 우선 두 vector (1,0), (0,1)에 대해서 살펴보면 $|\det A|$가 이 두 vector가 만드는 정사각형의 volume임은 자명하고, 이 vector 각각을 scalar배 해도 자명하다. 또한 임의의 두 vector (a, b), (c, d)가 만드는 평행사변형의 volume을 구해 봐도 자명하다.

한 row의 scalar배를 다른 row에 더하는 것은 해당 도형을 평행이동하는 것과 같다. 임의의 좌표에 있는 도형을 평행이동해서 원점으로 이동시키는 계산을 해보면 당연하다. 즉, 한 row의 scalar배를 다른 row에 더하는 것은 volume을 보존하므로 **elemination을 적용한 뒤 determinant를 구해 volume을 구할 수 있다.**

참고로 3차원 vector 3개가 이루는 삼각뿔의 부피는, 해당 vector들로 구성된 matrix의 determinant로 volume을 구하고 $\frac{1}{6}$을 곱하면 된다.

6. Eigen value and Eigen Vector

6.1. Eigen value and Eigen Vector

6.1.1. Eigen value and Eigen Vector

1. Eigen value and Eigen Vector
어떤 $n \times n$ matrix A에 대해서 $Av = \lambda v$인 scalar λ가 존재하는 영벡터가 아닌 vector v A의 Eigen Vector(고유벡터)라고 하고, 이때의 scalar λ를 eigen vector v에 대응되는 A의 Eigen Value(고윳값)라 한다. 즉, eigen vector는 A를 곱해도 그 방향이 달라지지 않는 특별한 vector이다.

또한 어떤 eigen value에 대해 존재하는 모든 eigen vector들과 영벡터에 대한 집합을 Eigen Space(고유공간)라고 한다. 이는 당연하게도 밑에서 다루는 것처럼 $A - \lambda I$의 null space이다.

참고로, $n \times n$ matrix A가 diagonalizable한 것과 A의 eigen space의 direct sum(직합)이 R^n인 것은 필요충분이다. 즉, 어떤 $n \times n$ diagonalizable matrix A에 대한 R^n(정의역)은 A의 eigen space들로 나누어 생각할 수 있고, 어떤 eigen value에 대한 eigen space에 속하는지에 따라 해당 vector가 어떻게 R^n으로(치역) 보내지는지를 알 수 있다. 더 구체적으로는, 임의의 vector x가 eigen vector로 구성된 R^n의 basis $v_1, \cdots, v_n$으로 표현될 때, 다음 수식과 같이 eigen value만큼씩이 각 eigen vector에 곱해진다.

$$Ax = A(a_1v_1 + a_2v_2 + \cdots + a_nv_n) = \lambda_1a_1v_1 + \lambda_2a_2v_2 + \cdots + \lambda_na_nv_n$$

뒤에서 다루는 것처럼 eigen value와 eigen vector의 활용은 문제를 훨씬 단순화한다는 점에 그 의미가 있다.

2. Eigen value/vector 구하기
$Av = \lambda v$를 정리하면 $(A - \lambda I)v = 0$이므로 eigen vector v는 $A - \lambda I$의 null space에 존재하는 vector이고, 다시말해 $A - \lambda I$의 special solution을 구하면 그게 eigen vector이다.

이때 eigen vector는 영벡터가 아니므로, eigen vector가 존재하려면 null space가 영공간이 아니어야 하고, 다시말해 $\det(A - \lambda I) = 0$가 성립해야 한다. 이에 따라 $\det(A - tI) = 0$을 만족시키는 t를 찾으면 그게 λ이다. 이때 matrix A에 대해서 다항식 $f(t) = det(A - tI)$을 A의 Characteristic Polynomial(특성다항식)이라 하고, $det(A - tI) = 0$을 Characteristic Equation(특성방정식)이라 한다.

정리하면, 다음과 같은 과정을 거쳐 eigen value/vector를 구할 수 있다.

1. characteristic equation이 0이 되게 하는 scalar를 구하면 그게 eigen value이다.

2. eigen value를 $A - \lambda I$에 대입해 얻은 matrix의 null space의 vector를 구하면 그게 eigen vector이다.

   이는 $A - \lambda I$에 대한 special solution을 구하는 것으로도 이해할 수 있다. 즉, $A - \lambda I$에 elimination을 적용한 뒤 free variable 중 하나에만 1, 나머지에는 0을 대입해 해당 null space(eigen space)의 basis를 찾아 구할 수 있다.

3. 대수적 중복도와 기하적 중복도
characteristic equation이 $f(t)$인 선형연산자(또는 행렬)의 고윳값 λ에 대해서, $(t-\lambda)^k$가 $f(t)$의 인수가 되도록 하는 가장 큰 자연수 k를 λ의 중복도(multiplicity) 또는 대수적 중복도라고 한다. 즉, 인수분해했을 때 해당 λ의 차수를 의미한다. diagonalization했을 때 Λ에 각 λ는 대수적 중복도만큼 나타난다.

어떤 λ에 대한 eigen space의 dimension을 해당 λ에 대한 기하적 중복도라고 한다. 즉, 기하적 중복도는 eigen space에서 independant한 eigen vector들의 개수로, $A - \lambda I$의 nullity이다.

$n \times n$ matrix A의 모든 eigen value의 대수적 중복도를 합하면 n이다. 또한 임의의 eigen value에 대해 기하적 중복도는 대수적 중복도보다 작고, 각 eigen value에 대응되는 eigen vector들의 집합이 각각 independent하면 이들을 합집합해도 independent하다(λ가 다른 두 eigen vector는 서로 independent하다.). 즉, 모든 eigen value에 대해 대수적 중복도만큼의 원소를 가지는 eigen space의 basis가 존재하면 diagonalization이 가능하고, 이런 basis를 모두 합치면 R^n에 대한 basis이다.

서로 다른 eigen value들에 대한 eigen vector들은 서로 independent하다. 이는 쉽게 증명할 수 있다. $Av_1 = \lambda_1 v_1, Av_2 = \lambda_2 v_2$이고 $\lambda_1 \neq \lambda_2$일 때 scalar a에 대해 $v_1 = av_2$라고 가정하자. $aAv_2 = a\lambda_1 v_2, Av_2 = \lambda_1 v_2 = \lambda_2 v_2$이므로 $\lambda_1 = \lambda_2$으로 모순이다.

projection matrix P를 곱해 b를 어떤 공간에 projection한 Pb를 생각하자. 당연하게도 이때 b가 해당 공간에 원래 있었다면 b는 eigen vector이고, 해당 공간에 원래 있지 않았다면 b는 eigen vector가 아니다.

eigen vector는 그 정의상 영벡터가 아니다. 만약 eigen vector가 영벡터일 수 있다면 eigen value가 무한히 존재하게 된다.

eigen vector는 영벡터가 아닌 vector지만, eigen value는 0일 수 있다. A가 singular matrix라면 0을 eigen value로 가진다.

6.1.2. Eigen Value/Vector 관련 성질들

eigen value/vector 관련 성질들은 다음과 같다.

- matrix A의 eigen value의 합은 A의 trace(대각합, 대각성분의 합.)와 같고, eigen value의 곱은 determinant와 같다.
- matrix A의 대각성분에 scalar a만큼의 값을 더하는 경우, eigen vector는 유지되고 eigen value는 a만큼 더한 값이 된다. 즉, 대각성분만 다른 경우 eigen value/vector는 한 번만 구해도 된다.

  이는 당연하게도 $Bx = (A + aI)x = Ax + ax = \lambda x + ax = (\lambda + a)x$인 것으로 보일 수 있다.
- matrix A에 대해, A^n의 eigen value는 λ^n이고 eigen vector는 A와 같다.

  x가 A의 eigen vector라고 하면 λ에 대해 $A^{100}x = \lambda^{100}x$이므로 당연하다.
- invertible matrix A에 대해, A^{-1}의 eigen value는 $\frac{1}{\lambda}$이고, eigen vector는 A와 같다.

  x가 A의 eigen vector라고 하면 $Ax = \lambda x$이므로 양번에 A^{-1}를 곱하면 $A^{-1}x = \frac{1}{\lambda}x$으로 정리할 수 있다.
- upper/lower trianlge matrix의 eigen value는 대각성분과 같다.

  matrix A가 upper/lower triangle matrix이면 $A-\lambda I$도 upper/lower triangle matrix이므로 determinant가 대각성분의 곱이 되고, 이에 따라 당연하게도 대각성분이 eigen value가 된다.
- elimination은 eigen value를 보존하지 않는다.

  아무 matrix나 elimination해서 간단히 보일 수 있다.

예를 들어, 다음과 같은 2×2 matrix A의 characteristic equation은 $(3-\lambda)^2 - 1 = \lambda^2 - 6\lambda + 8 = 0$이므로, eigen value의 합이 trace, 곱이 determinant인 것을 확인할 수 있다.

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

A와 B가 각각 $Ax = \lambda x$, $By = \alpha y$라고 하자. 이 경우 $C = A + B$의 eigen vector가 x 또는 y라는 것은 당연하게도 거짓이다. 실제로 대부분의 경우 eigen vector가 겹치지 않는다고 한다.

다음과 같은 matrix Q를 생각하자. 이는 vector를 90도 돌리는 rotation matrix이다. 기하적으로 생각했을 때 이런 rotation matrix는 vector를 돌리므로(방향이 바뀌므로) 실수인 eigen value가 존재할 수 없고, eigen vector는 항상 복소수이다.

$$Q = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

6.1.3. Similarity

$n \times n$ matrix A, B에 대해서 어떤 matrix M에 대해 $B = M^{-1}AM$이 성립하면 A와 B는 Similar(닮음)하다고 한다. 이는 동일한 linear transform을 서로 다른 basis로 바라본 것을 의미한다.

similar한 matrix들은 eigen value의 개수와 값이 서로 같고, 이에 따라 trace와 determinant가 모두 같다. 또한 동일한 linear transform을 나타내므로 rank와 nullity도 같다.

이때 eigen vector는 다를 수 있다. 만약 $B = M^{-1}AM$이라면, $A\lambda = \lambda v$, $Bv = M^{-1}AMv = \lambda v$, $AMv = \lambda Mv$이다. 즉, A의 eigen value는 λ으로 동일하고 eigen vector는 Mv이다.

diagonalizable한 matrix A에 대해 $S^{-1}AS = \Lambda$이므로 A와 Λ는 similar하고, eigen value가 같다.

하지만 eigen value가 같다고 반드시 similar한 것은 아니다. eigen value가 같아도 대수적 중복도와 기하적 중복도가 일치하지 않는 matrix는 eigen decomposition이 불가능하다. 당연하게도 모든 eigen value의 대수적 중복도가 1인 경우에는 항상 similar하다.

6.2. Eigen Value/Vector의 활용

6.2.1. Diagonalization

1. Diagonalization
Diagonalization(대각화)은 어떤 $n \times n$ matrix의 eigen value와 eigen vector를 사용해 diagonal matrix로 변환하는 방법이다. 서로 independent한 A의 eigen vector n개로 column을 구성한 matrix S와, S의 eigen vector의 순서와 대응되는 eigen value를 대각성분으로 가지는 diagonal matrix Λ를 생각하자. 다음 수식이 성립한다. 즉, eigen vector/value를 사용해 diagonalization이 가능하다. 또한 이렇게 A를 나누는 것을 Eigen Decomposition이라고 한다.

$$AS = S\Lambda, \;\; S^{-1}AS = \Lambda, \;\; A = S\Lambda S^{-1}$$

이 수식은 다음과 같이 계산되는 것을 생각해면 당연하다.

$$AS = \begin{bmatrix} \lambda_1 x_1 & \lambda_2 x_2 & \cdots & \lambda_n x_n \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} = S\Lambda$$

$S\Lambda S^{-1}$를 거듭제곱해보면 $A^n = S\Lambda^n S^{-1}$인데, 전에 다룬 것처럼 A를 거듭제곱하면 eigen vector는 그대로이고, eigen value는 동일하게 거듭제곱이 되는 것을 확인할 수 있다.

만약 A^k에 대해서 $k \to \infty$이면 모든 eigen value가 $|\lambda_i| < 1$인 경우 $A^k \to 0$이다. 이런 성질을 이용하면 마르코프 체인 등에서 확률이 어디에 수렵하는지 알 수 있다고 한다.

2. Diagonalizable 판정법
임의의 $n \times n$ matrix A에 대해 diagonalization이 성립하는 것은 아니고, 다음 조건을 만족할 때만 diagonalization이 가능하다. 즉, characteristic equation을 인수분해하고, 중근을 가지는 eigen value에 대해 대수적 중복도와 기하적 중복도가 같은지 확인해야 한다.

1. T의 특성다항식이 체 위에서 완전히 인수분해됨. 실수체인 경우 실수 범위에서 인수분해되어야 한다.
2. T의 각 고윳값에 대해서 대수적 중복도와 기하적 중복도가 같다. 즉, λ의 중복도가 $nullity(T - \lambda I) = n - rank(T - \lambda I)$이다.

앞서 다룬 것처럼 모든 eigen value에 대해 대수적 중복도만큼의 원소를 가지는 eigen space의 basis가 존재하면 diagonalization이 가능하고, 이런 basis를 모두 합치면 R^n에 대한 basis이다. 이는 eigen vector로 R^n에 대한 basis를 구성할 수 있는지를 보는 것으로도 이해할 수 있다.

6.2.2. Fibonacci Sequence

Fibonacci Sequence(피보나치 수열)를 diagonalization을 사용해 풀어보자.

문제를 풀기 전에 우선, $u_1, \cdots, u_n$이 n차원 vector이고, $n \times n$ matrix A는 n개의 독립한 eigen vector $x_1, \cdots, x_n$를 가진다고 해보자. 점화식 $u_{k+1} = Au_k$을 생각해보면, 각 eigen vector들이 R^n의 basis이므로 $u_0 = c_1x_1 + \cdots + c_nx_n$으로 나타낼 수 있고, A가 diagonalizable하므로 k번 반복하면 $u_k = A^ku_0 = c_1\lambda_1^kx_1 + \cdots + c_n\lambda_n^kx_n$으로 나타낼 수 있다.

이제 피보나치 수열을 풀어본다. 피보나치 수열은 $F_{k+2} = F_{k+1} + F_k$이므로 u_k를 다음과 같이 정의하자.

$$u_k = \begin{bmatrix} F_{k+1} \\ F_k \end{bmatrix}$$

그리고 지금 문제를 $u_{k+1} = Au_k$ 형태로 풀 것이므로 A는 다음과 같이 정의된다. (인덱스 주의: u_{k+1}은 F_{k+2}, F_{k+1}로 구성됨)

$$u_{k+1} = \begin{bmatrix} F_{k+2} \\ F_{k+1} \end{bmatrix} = \begin{bmatrix} F_{k+1} + F_k \\ F_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{k+1} \\ F_k \end{bmatrix} = Au_k, \quad A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

이런 A에 대해 eigen vector, value를 구해볼 수 있다. eigen value가 λ_1, λ_2라 하면, characteristic equation은 $\det(A - \lambda I) = \lambda^2 - \lambda - 1 = 0$ 이므로(부호 수정됨), eigen vector x_1, x_2는 다음과 같이 구할 수 있다.

$$x_1 = \begin{bmatrix} \lambda_1 \\ 1 \end{bmatrix}, \quad x_2 = \begin{bmatrix} \lambda_2 \\ 1 \end{bmatrix}$$

u_0 ($F_1 = 1, F_0 = 0$ 가정)를 이전에 한 것처럼 $u_0 = c_1x_1 + c_2x_2$ 꼴로 나타내면 계수를 구할 수 있다.

$$u_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{x_1 - x_2}{\lambda_1 - \lambda_2}$$

이제 일반항 u_k를 구하면 다음과 같다.

$$u_k = A^ku_0 = \frac{\lambda_1^kx_1 - \lambda_2^kx_2}{\lambda_1 - \lambda_2}$$

여기서 u_k의 두 번째 성분이 F_k이므로, 최종적으로 F_k는 다음과 같다.

$$F_k = \frac{\lambda_1^k - \lambda_2^k}{\lambda_1 - \lambda_2}$$

이때 k가 커지면 절댓값이 1보다 큰 eigen value($\lambda_1 \approx 1.618$, 황금비)에 의해 값이 결정되어 근사할 수 있다.

6.3. Spectral Decomposition

6.3.1. Symmetric Matrix

Symmetric Matrix(대칭행렬)는 $A = A^T$인 matrix이다. 원소가 실수인 $n \times n$ symmetric matrix는 다음과 같은 성질을 보인다.

- eigen value가 실수이다. 즉, characteristic equation이 n개의 근을 가진다.

  $Ax = \lambda x$가 성립할 때, 양변에 conjugate(켤레)를 취하면 $\bar{A}\bar{x} = A\bar{x} = \bar{\lambda}\bar{x}$이다. transpose하고 x를 곱해서 정리해 보면 $\lambda \bar{x}^T x = \bar{\lambda}\bar{x}^T x$이므로 $\lambda = \bar{\lambda}$이다.

  물론 이때 $\bar{x}^T x \neq 0$임을 보여야 완벽하다. 해당 inner product 값을 정리하면 다음과 같다. 이때 켤레복소수와의 곱은 $\bar{x}_i x_i = (a+bi)(a-bi) = a^2 + b^2$이므로 x가 영벡터인 경우에만 $\bar{x}x = 0$가 성립하는데, eigen vector는 영벡터가 아니므로 $\bar{x}_1 x_1 \neq 0$이다.

  $$\bar{x}^T x = \bar{x}_1 x_1 + \bar{x}_2 x_2 + \cdots + \bar{x}_n x_n$$

  참고로 A의 원소가 복소수이면 해당 조건이 성립하지 않는다.

- 항상 diagonalizable하다.

  이는 schur decomposition이라는 방법으로 증명이 가능하다고 한다.

- symmetric matrix의 eigen vector들로 구성한 R^n의 basis는 orthogonal하다. 즉, 각 column을 eigen vector들로 구성한 basis로 하는 matrix를 Q라 하면, $Q^T AQ = \Lambda$가 성립한다.

  하나의 eigen space에서는 gram schmidt process를 사용하여 orthogonal한 basiss을 얻을 수 있다. 또한 서로 다른 eigen space의 basis끼리는 항상 orthogonal하다. 이는 간단히 증명할 수 있다. 서로 다른 두 eigen value λ_1, λ_2에 대해 $Av_1 = \lambda_1 v_1$, $Av_2 = \lambda_2 v_2$라 했을 때, $\lambda_1 v_1^T v_2 = (Av_1)^T v_2 = v_1 A v_2 = \lambda_2 v_1^T v_2$이므로 $(\lambda_1 - \lambda_2) v_1^T v_2$이고, $\lambda_1 \neq \lambda_2$이므로 $v_1^T v_2 = 0$이다.

- symmetric matrix A의 pivot의 부호 개수는 eigen value의 부호 개수와 같다. 즉, 양수의 개수, 음수의 개수, 0의 개수가 모두 같다. 또한 이에 따라 determinant가 양수/음수인지에 따라 eigen value에서 양수/음수의 개수를 알 수 있다.

  symmtric matrix A는 elimination을 적용해 $A = LDL^T$ 꼴로 decompostion할 수 있다. 이때 D는 대각성분이 pivot인 diagonal matrix이고, A와 D는 congruent(합동)이다. Sylvester's Law of Inertia(실버스터의 관성 법칙)에 의하면 congruent한 matrix끼리는 eigen value의 부호 개수가 같은데, D는 이미 diagonal matrix이므로 A의 pivot과 eigen value는 부호 개수가 같다.

  이에 따라 symmetric matrix에 대해서는 positive definite인지를 판별할 때 eigen value를 직접 구하는 대신, elimination해서 pivot의 부호만 보면 된다.

6.3.2. Spectral Decomposition

1. Spectral Decomposition
Spectral Decomposition은 symmetric matrix A를 $A = Q\Lambda Q^T$꼴로 나누는 decomposition이다. 앞서 다룬 것처럼 실수 symmetric matrix에 대한 diagonalization은 항상 가능하고, $Q^T AQ = \Lambda$꼴로 나타내어진다. 즉, eigen decomposition을 symmetric matrix라는 특수 케이스에 적용한 것이다.

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

2. Rank-1 Matrix로 정리
Rank-1 matrix는 모든 column 또는 row가 하나의 vector의 상수배인 matrix로, 복잡한 matrix를 분해하는 기본 단위로 주로 사용되는 matrix이다. 이는 앞서 몇 번 다룬 것처럼 두 vector의 outer product 형태로 표현된다.

이에 따라 $A = Q\Lambda Q^T$는 다음과 같이 나타낼 수 있다. 이때 $q_i q_i^T$는 $n \times n$ matrix로, 이는 1. rank-1 matrix이고 2. projection matrix이고 3. symmetric matrix이다. 즉, symmetric matrix는 symmetric인 projection matrix

들의 합으로 나타낼 수 있다.

$$A = Q\Lambda Q^T = \begin{bmatrix} \lambda_1 q_1 & \lambda_2 q_2 & \cdots & \lambda_n q_n \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \cdots \\ q_n \end{bmatrix} = \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T + \cdots + \lambda_n q_n q_n^T$$

λ_i가 중요도 순으로 정렬되어 있고, 일부 λ_i만 있어도 충분하다면 이렇게 rank-1 matrix들로 나타냈을 때 중요도가 낮은 뒤쪽 값들은 그냥 버릴 수 있다. 이런 식으로 데이터를 압축하거나 중요한 정보만 추출하는 것이 가능하다.

또한 Q가 orthogonal matrix이므로 이렇게 rank-1 matrix로 정리한 수식에 Q의 column을 곱하면 더 간단히 정리할 수 있다. 예를 들어, Sq_1은 다음과 같이 정리가 가능하다.

$$Sq_1 = (\lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T + \cdots + \lambda_n q_n q_n^T) q_1 = \lambda_1 q_1 q_1^T q_1 = \lambda_1 q_1$$

7. SVD

7.1. Positive Definite Matrix

7.1.1. Positive Definite Matrix

1. Positive Definite Matrix
Positive Definite Matrix(양의 정부호 행렬)는 $x \neq 0$인 임의의 vector x에 대해서 $x^T Ax > 0$이 성립하는 symmetric matrix이다.

positive definite matrix에 대해서는 다음과 같은 성질이 성립한다. 또한 어떤 matrix가 positive definite matrix인지는 이 세 가지 중 하나가 성립하는 것을 보이면 되고, 이때 하나가 성립하면 나머지도 성립한다. 즉, 이 세 가지 중 하나만 성립해서 $x \neq 0$인 x에 대해서 $x^T Ax > 0$이다.

- 모든 eigen value가 양수이다.

  A에 대한 $Av = \lambda v$인 eigen vector v를 생각하자. $v^T Av = \lambda v^T v > 0$ 이어야 하는데, eigen vector는 영벡터가 아니므로 $\lambda > 0$이 성립한다. 당연하게 그 역도 성립한다.

- 모든 pivot이 양수이다.

  symmetric matrix에서는 eigen value의 부호 개수와 pivot의 부호 개수가 같으므로 당연하다.

- 해당 matrix의 왼쪽 위 끝의 원소를 포함하는 sub-matrix의 determinant가 모두 양수이다.

  한 row의 scalar배를 다른 row에 곱하는 연산은 determinant를 보존하므로, 이런 sub-matrix의 determinant를 작은 쪽부터 순차적으로 구해 보면 pivot의 부호에 따라 부호가 정해지는 것을 알 수 있다.

또한 $x^T Ax < 0$이 성립(모든 eigen value가 음수)하는 matrix는 Negative Definite Matrix(음의 정부호 행렬)이라고 하고, 등호가 붙으면(어떤 eigen value가 0) 각각 Positive Semi-definite Matrix(양의 준정부호 행렬), Negeative Semi-definite Matrix(음의 준정부호 행렬)라고 한다. 또한 eigen value에 양수와 음수가 섞여 있는 상태는 Indefinite Matrix(부정부호 행렬)라고 한다.

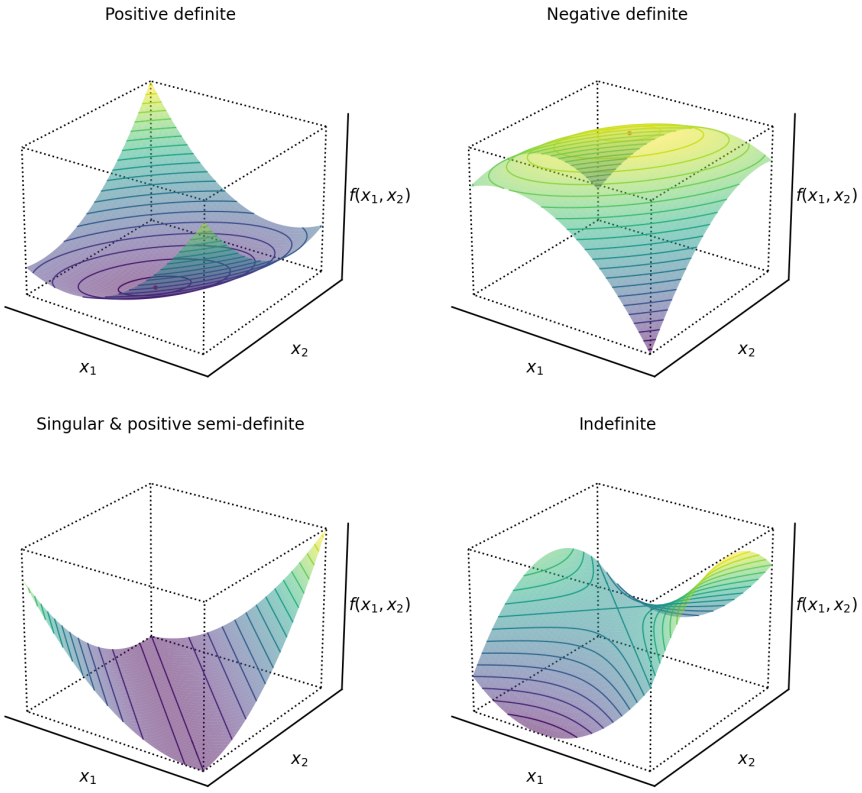
2. Positive Definite의 의미
positive definite의 정의를 나타내 보면 다음과 같이 $x_1, \cdots, x_n$에 대한 이차방정식이 0보다 크다는 것을 의미한다. 즉, 최솟값이 존재하는 경우이다. 또한 이때 변수 $x_1, \cdots, x_n$ 각각에 대한 완전제곱식 형태로 수식을 묶어 정리하면, $x_1^2, \cdots, x_n^2$의 계수가 pivot과 같다(2×2 matrix에 대해 계산해볼 수도 있고, LDL^T decomposition을 사용해 증명할 수 있다고 한다.).

$$x^T Ax = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n \end{bmatrix} = a_{11}x_1^2 + a_{22}x_2^2 + \cdots + a_{nn}x_n^2 + \cdots > 0$$

이를 2×2 matrix A에 대해 그래프를 그려보면, positive definite(밥그릇 모양), negative definite(뒤집힌 밥그릇 모양), positive semi-definite(골짜기 모양), indefinite(안장 모양) 각각에 대해 $x^T Ax$의 그래프는 다음과 같다. λ의 부호에 따라 그래프가 달라지는 것을 이해할 수 있다.

positive semi-definite의 경우 어떤 eigen value가 0이어서 해당 방향의 경우 이동해도 값이 변하지 않게 되고, 골짜기 모양의 그래프가 그려진다. indefinite의 경우 양수인 eigen value와 음수인 eigen value가 둘 다 존재해서 양의 무한대와 음의 무한대로 발산하는 상태가 공존하고, 안장 모양의 그래프가 그려진다.

또한 각 column이 eigen vector인 matrix Q에 대해서 x=Qu라고 하면, $x^T Ax = u^T Q^T Q \Lambda Q^T Q u = u^T \Lambda u = \lambda_1 u_1^2 + \lambda_2 u_2^2$과 같이 각 vector 별로 하나의 항으로 정리할 수 있다. 위에서 내려다보는 것을 생각했을 때, $\lambda_1 u_1^2 + \lambda_2 u_2^2 = 1$인 경우 λ의 부호가 둘 다 양수이면 타원이 되고, λ의 부호 중 하나가 음수이면 쌍곡선이 된다.



positive definite matrix는 입력 vector와 보낸 vector의 inner product가 양수이므로 vector의 방향이 반대로 꺾이지 않도록 보내는 matrix로도 이해할 수 있다.

symmetrix matrix는 diagonalizable하고, pivot의 부호 개수와 eigen value의 부호 개수가 같다. 또한 symetric matrix에서 determinant는 pivot의 곱이므로, determinant의 부호에 따라 positive definite인지가 바뀐다. positive definite인 경우 determinant가 양수인데, 성분 값을 바꾸었을 때 determinant가 0이 되는 지점에서 positive semi-definite이 되고, 0에서 음수로 넘어가면 negative definite이거나 indefinite이 된다.

7.1.2. Positive Definite 관련 성질

positive definite는 다음과 같은 성질들을 가진다.

- $n \times n$ matrix A가 positive definite이면 A^{-1}도 positive definite이다.

  inverse matrix는 eigen value가 역수이므로 당연하다.

- $n \times n$ matrix A, B가 positive definite이면 $A + B$도 positive definite이다.

- $m \times n$ matrix A에 대해 $n \times n$ matrix A^TA는 symmetric이면서 positive semi-definite이다. 이때 A가 full column rank인 경우 $Ax = 0$을 만족시키는 $x \neq 0$인 x가 존재하지 않으므로 positive definite이 된다.

  A^TA가 symmetric인 것은 자명하다. $x^TA^TAx = (Ax)^TAx \geq 0$ 이므로 항상 positive semi-definite이다.

7.2. SVD

7.2.1. SVD

1. SVD

SVD(Singular Value Decomposition)는 임의의 $m \times n$ matrix A를 $A = U\Sigma V^T$ 꼴로 나누는 decomposition 이다. 즉, eigen decomposition을 임의의 matrix에 대해 확장한 것이다.

$$AV = U\Sigma, \;\; A = U\Sigma V^T$$

matrix A는 R^n 중 row space에서 R^m의 column space로 vector를 보낸다. 이에 따라 U, V, Σ에 대한 정의는 다음과 같다. 여기에서 $r = rank(A) = rank(A^TA) = rank(AA^T)$이다. A, A^TA와 AA^T의 rank가 같은 것은 column 또는 row의 linear combination을 생각하면 당연하다.

- A^TA의 eigen vector로 구성한 R^n의 orthonormal eigen basis $v_1, \cdots, v_n$를 생각하자. V는 이 vector들로 column을 구성한 $n \times n$ orthogonal matrix이다.

  이때 $v_1, \cdots, v_r$은 row space의 basis이고, $v_{r+1}, \cdots, v_n$은 null space의 basis이다.

- AA^T의 eigen vector로 구성한 R^m의 orthonormal eigen basis $u_1, \cdots, u_m$을 생각하자. U는 이 vector 들로 column을 구성한 $m \times m$ orthogonal matrix이다.

  이때 $u_1, \cdots, u_r$은 column space의 basis이고, $u_{r+1}, \cdots, u_m$은 left null space의 basis이다.

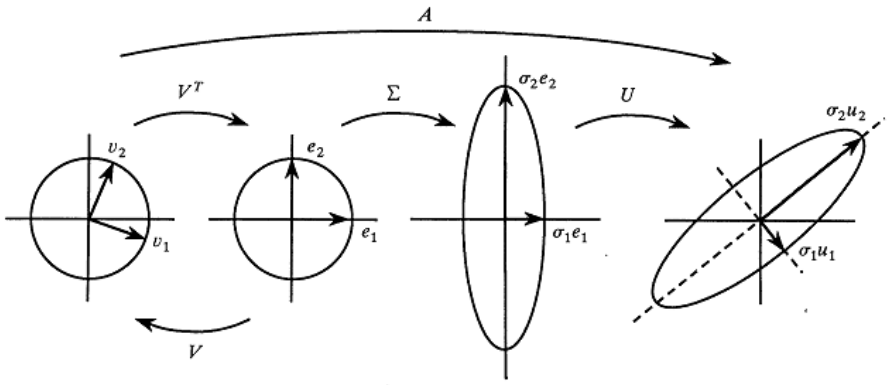
- Σ는 대각성분이 Singular Value $\sigma_1, \cdots, \sigma_n$로 구성된 $m \times n$ diagonal matrix이다. 이때 A^TA의 eigen value를 λ라 하면 singular value는 다음과 같이 정의된다. 이에 따라 r까지는 singular value가 양수이고, r+1부터는 singular value가 0이다.

$$\sigma_i^2 = \lambda_i. \;\; \sigma = \sqrt{\lambda_i}$$

  이때 $Av_i = \sigma_i u_i$가 성립한다. 즉, singular value는 R^n의 orthonormal eigen basis를 R^m의 orthonormal eigen basis로 보냈을 때 값을 scale하는(곱해서 맞추는) scalar이다. basis끼리 매핑이 존재하는 경우에는 singular value가 양의 실수값을 가지게 되고, 매핑이 존재하지 않는 경우(null space의 basis)에는 singular value가 0이 된다. 이는 앞서 다뤘던 four fundamental spaces에서 이해할 수 있다.

  SVD에서는 관례적으로 singular value를 내림차순으로 정렬해 나타낸다.

이때 U와 V는 orthonormal matrix이고, Σ는 scaling만을 수행하므로, $Ax = U\Sigma V^Tx$는 x가 V에 의해 회전되고, Σ에 의해 scaling되고, U에 의해 다시 회전되는 것으로 이해할 수 있다.



$Av_i = \sigma_i u_i$가 성립하므로, 앞서 다룬 수식이 다음과 같이 성립함을 이해할 수 있다.

$$A\begin{bmatrix} v_1 & v_2 & \cdots & v_r & \cdots & v_n \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \cdots & u_r & \cdots & u_m \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & & \cdots & & 0 \\ 0 & \sigma_2 & & \cdots & & 0 \\ & & \ddots & & & \\ \vdots & \vdots & & \sigma_r & & \vdots \\ & & & & \ddots & \\ 0 & 0 & & \cdots & & 0 \end{bmatrix}$$

2. SVD 증명

singular value는 A^TA, AA^T의 eigen value의 제곱근과 같다. 다음과 같이 SVD가 임의의 matrix에 대해 유효함을 보일 수 있다.

앞서 다룬 것처럼 임의의 $m \times n$ matrix A에 대해 A^TA를 생각하자. 이 matrix는 positive semi-definite이므로, $\lambda_i \geq 0$가 성립하여 singular value $\sigma_i = \sqrt{\lambda_i}$를 항상 정의할 수 있다. 또한 A^TA가 symmetric matrix이므로 A^TA의 eigen vector로 구성한 orthonormal eigen basis $v_1, \cdots, v_n$이 항상 존재한다.

A^TA의 eigen value를 사용해, singular value를 $\sigma = \sqrt{\lambda_i}$으로 정의하자. singular value는 r개의 양수와 $n-r$개의 0으로 구성된다.

이제 $\sigma_i u_i = Av_i$를 만족하는 orthonormal eigen basis $u_1, \cdots, u_m$이 존재함을 보이자. R^n의 basis 중 null space의 basis가 아닌 r개의 vector에 대해서는 $\sigma \neq 0$이고, $u_i = \frac{Av_i}{\sigma_i}$로 정리할 수 있다. 이때 $v_1, \cdots, v_n$은 orthonormal하므로 $u_i^T u_j = \frac{1}{\sigma_i \sigma_j} v_i^T A^T A v_j = \frac{\lambda_j}{\sigma_i \sigma_j} v_i^T v_j = 0$이다. 즉, $u_1, \cdots, u_r$는 orthogonal하고, norm으로 나누면 orthonormal하다. 또한 $AA^T u_i = \frac{AA^T Av_i}{\sigma_i} = \lambda_i \frac{Av_i}{\sigma_i} = \lambda_i u_i$이므로 AA^T의 eigen vector이다. 또한 A^TA와 AA^T의 0이 아닌 eigen value는 같다.

$u_1, \cdots, u_r$이 orthonormal eigen vector들이므로, m-r개의 orthogonal한 vector들을 gram schmidt process 등을 이용해 구하면 R^m의 orthonormal eigen basis를 찾을 수 있다.

3. SVD 값 구하기

SVD를 실제로 수행하기 위해 U, V, Σ를 구하는 방법은 다음과 같다. 이때 V와 U를 각각 A^TA와 AA^T에 대한 eigen vector를 구하는 것으로 구성할 수도 있지만, 이렇게 구하면 부호가 맞지 않는 경우가 존재하므로 둘 중에 하나에 대해 구한 뒤 $\sigma_i u_i = Av_i$임을 이용해 반대편을 구하는 게 적절하다. 이런 부호 불일치는 eigen vector는 부호가 뒤집혀도 여전히 eigen vector이기 때문에 발생한다.

1. A^TA에 대해 $v_1, \cdots, v_n$을 구하거나, AA^T에 대해 $u_1, \cdots, u_m$을 구한다. 즉, eigen value를 구한 뒤, eigen vector를 뽑아 norm으로 나눠 orthonormal eigen basis를 구성한다.
2. singular value가 0이 아닌 부분에 대해 반대편 $u_1, \cdots, u_r$ 또는 $v_1, \cdots, v_r$을 구한다. 그리고 n 또는 m에 맞춰 나머지 vector들을 적절히 구한다. AA^T, A^TA로 구하거나, 직관적으로 채우거나, gram schmidt process를 적용할 수 있다.
3. 구한 값들로 U, V, Σ를 구성한다.

또한 SVD의 공식에 따라 $A^TA = V\Sigma^2V^T$, $AA^T = U\Sigma^2U^T$와 같이 나타낼 수 있는데, 이 수식은 spectral decomposition이므로 $\sigma = \sqrt{\lambda}$이고, A^TA, AA^T의 eigen value가 같음을 확인할 수 있다.

4. Rank-1 Matrix로 정리
spectral decomposition에서와 같이, SVD를 다음과 같이 rank-1 matrix의 합으로 나타낼 수 있다.

$$A = U\Sigma V^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_n u_n v_n^T$$

각 space의 orthogonal eign basis는 각 eigen space에 대해 gram schmidt process를 수행해 구할 수 있지만, R^n과 R^m의 basis끼리 매핑됨을 보장하진 못한다. SVD는 singular value를 사용해 두 basis에 대한 매핑을 나타낸다.

SVD는 어떤 feature가 중요한지 판단하기 위한 분해이다. low rank approximation 같은 데에 사용된다.

참고로 null space의 basis도 eigen value가 0인 eigen vector이다.

7.2.2. Low Rank Approximation

singular value는 주로 크기순으로 정렬하므로, rank-1 matrix로 변환한 뒤 중요한 정보를 담고 있는 matrix들만을 남기고 나머지를 제거해 압축하거나, 중요한 matrix만을 추출할 수 있다. 이런 기법을 Low Rank Approximation이라고 한다.

low rank approximation 예시를 하나 보자. rank가 r인 $m \times n$ matrix A에 대해, rank가 $k < r$이면서 A와의 차이가 가장 작은 $m \times n$ matrix B를 찾으려고 한다. 즉, 다음 수식을 만족시키는 B를 찾아보자.

$$\text{argmin}||A - B||_F$$

여기에서 $||M||_F$는 Frobenious Norm으로, element-wise하게 제곱해서 더한 뒤 루트를 씌운 것이다. 즉, L2 norm을 matrix에서 계산한 것으로 이해할 수 있다. frobenious norm은 유니타리한 특징을 가진다. 즉, orthogonal matrix를 곱하는 것은 길이를 변화시키지 않으므로(frobenious norm을 변화시키지 않으므로), frobenious norm을 구하는 행렬을 분해했을 때 orthogonal matrix가 곱해져 있으면 지울 수 있다. 이에 따라 $B = UCV^T$라고 하면 다음과 같이 수식을 정리할 수 있다.

$$||A - B||_F = ||U(\Sigma - U^TBV)V^T||_F = ||\Sigma - C||_F$$

이는 diagonal matrix인 Σ와 가장 유사한 matrix C를 찾는 문제가 된다. 이때 오차를 최소화하려면 C의 대각성분이 아닌 값들은 전부 0이어야 하고, 대각성분에서는 가장 큰 값부터 $\sigma_1, \cdots, \sigma_k$를 가져야 한다. 즉, C는 Σ에서 대각성분의 sigular value를 k개까지만 가지고, 나머지는 모두 0인 matrix이다.

이렇게 C가 정해졌으면 $B = UCV^T$로 A와 가장 가까운 B를 구할 수 있다. 이 과정에 따라 가장 중요한 값들만 보존하고, 작은 값들은 버리게 된다.

8. Linear Transformation

8.1. Linear Transformation

8.1.1. Linear Transformation

Linear Transformation(선형변환)은 다음과 같은 조건 2가지를 만족시키는 함수이다. 즉, vector 합과 scalar 곱을 보존하는 함수이다.

1. $T(v + w) = T(v) + T(W)$
2. $T(cv) = cT(v)$

linear transformation은 두 가지 관점에서 생각해볼 수 있다. 하나는 space(좌표계. 정의역, 치역)를 고려하지 않고, Ax와 같이 단순히 값을 보내는 것이다. 또 다른 하나는 space를 고려해서, 해당 space의 basis를 가져와서 표현하는 것이다. 두 번째 관점에 의해서는 어떤 linear transformation이 어떤 함수인지를 알려면 정의역의

basis를 보내보면 된다.

예를 들어, projection matrix를 곱하는 것은 linear transformation이다.

8.2. Pseudo Inverse Matrix

8.2.1. Left/Right Inverse Matrix

어떤 marix에 대해 왼쪽에 곱해져서 identity matrix를 만들면 Left Inverse, 오른쪽에 곱해져서 identitiy matrix를 만들면 Right Inverse이다.

- left inverse는 matrix가 full column rank일 때 구할 수 있다. $(A^TA)^{-1}A^TA = I$이므로 $(A^TA)^{-1}A^T$가 left inverse이다. 이때 A^TA의 inverse matrix를 사용하므로 full column rank여야 성립한다.

  left Inverse는 A의 column space에 있는 vector에 대해서는 A의 row space의 대응되는 값으로 그대로 되돌려주고, left null space에 있는 vector는 0으로 보낸다(left null space의 vector x에 대해서 그 정의에 의해 $A^Tx = 0$인 것을 생각하면 당연하다.).

  left inverse matrix를 오른쪽에 곱한 $A(A^TA)^{-1}A^T$는 A의 column space로의 projection matrix이다. 이는 left inverse matrix가 R^m에 존재하는 임의의 vector를 A의 row space로 보내고(A의 column space에 존재하지 않는 부분은 영벡터로 가므로 없어진다.), 보낸 vector를 A로 다시 보내는 것과 같다.

  least square 문제를 풀 때 곱하던 matrix가 A에 대한 left inverse matrix이다. left inverse matrix가 A의 R^m에 존재하는 임의의 vector를 column space로 보내는 것을 생각하면 이해할 수 있다.

- right inverse는 matrix가 full row rank일 때 구할 수 있다(r=m<n). $AA^T(AA^T)^{-1} = I$이므로, $A^T(AA^T)^{-1}$가 right inverse이다. 이때 마찬가지로 AA^T의 inverse matrix를 사용하므로 full row rank여야 성립한다.

  right inverse는 R^m의 vector를 R^n의 row space로 보내되, 해당 vector를 A에 넣었을 때 원래의 vector로 돌아오도록 하는 matrix이다. 이때 right inverse는 full row rank일 때 존재하므로, left null space는 존재하지 않는다.

  right inverse matrix를 왼쪽에 곱한 $A^T(AA^T)^{-1}A$는 A의 row space로의 projection matrix이다. column space로의 projection matrix에서 transpose 여부만 바뀐 것이므로 당연하다.

$x \neq y$이고, x와 y가 row space의 vector이면 $Ax \neq Ay$이다. 즉, row space에서 보내는 매핑은 항상 일대일 대응이고, 이에 따라 $m \times n$ matrix에 대해서도 inverse matrix와 유사한 left/right inverse matrix가 정의될 수 있다. 이는 귀류법으로 간단히 보일 수 있다. 만약 $Ax = Ay$라면 $A(x-y) = 0$인데, $x-y$는 null space의 원소이다. 근데 row space와 null space의 교집합은 $\{0\}$이므로 모순이다.

left/right inverse matrix는 four fundamental spaces에 대한 diagram을 그려서 생각하면 이해가 쉽다.

8.2.2. Pseudo Inverse

1. Pseudo Inverse
Pseudo Inverse 또는 Moore-Penrose Inverse는 임의의 $m \times n$ matrix에 대해 inverse matrix와 가장 유사하게 동작하는 matrix로, inverse, left/right inverse를 아우르는 가장 일반화된 개념이다.

pseudo inverse A^+는 다음과 같이 SVD와 유사하게 정의된다. 이때 Σ^+는 SVD의 Σ에서 singular value 중 0이 아닌 것들은 역수를 취하고, 0인 것들은 그대로 둔 뒤, transpose한 matrix이다. 즉, SVD에서와 같이 A^+는 rank r까지는 $A^+u = \frac{1}{\sigma}v$로 보내고, r+1부터는 영벡터로 보낸다.

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

임의의 matrix는 row space의 vector를 column space의 vector로 일대일 매핑한다. 즉, non-invertible이어도 해당 부분으로 한정하면 일대일대응이다. pseudo inverse는 이런 매핑에서 vector를 반대로 보낸다. column space의 vector를 row space의 vector로 일대일 매핑하고, left null space에 해당하는 부분은 0으로 보내는 것으로 이해할 수 있다.

full rank인 square matrix인 경우 $A^+ = A^{-1}$이고, full column rank인 matrix에 대해서 A^+는 left inverse,

full row rank인 matrix에 대해서 A^+는 right inverse이다. 물론 full rank가 아닌 경우에도 pseudo inverse는 존재하므로, pseudo inverse는 inverse, left/right inverse의 상위 개념이다.

2. Projection Matrix인가?
A^+A는 row space로의 projection matrix이다. 이 경우 vector에서 row space에 해당하는 부분은 유지되고, null space에 해당하는 부분은 제거되므로 당연하다. A가 full column rank인 경우 null space가 영공간이므로 $A^+A = I$가 되고, A^+는 left inverse이다.

AA^+는 column space로의 projection matrix이다. 이 경우 vector에서 column space에 해당하는 부분은 유지되고, left null space에 해당하는 부분은 제거되므로 당연하다. A가 full row rank인 경우 left null space가 영공간이므로 $AA^+ = I$가 되고, A^+는 right inverse이다.

9. ML

여기에서는 머신러닝 측면의 주제들을 선형대수학적으로 다뤄본다.

예를 들어, 만약 데이터의 분포가 스파이럴, 스위스롤 형태라면 decision boundary를 어떻게 지정할까? decision boundary를 선형적으로 만들면 데이터가 잘 분류되지 않는다. 이런 문제는 선형대수학적으로 여러 접근이 가능하다. decision boundary를 비선형적으로 만들 수도 있고, 데이터를 transformation해서 linear하게 만들 수 있다. 또는 차원을 늘려서 저차원에서 확인하기 어려운 데이터를 고차원에서 확인할 수 있다.

지도학습에서 학습 데이터를 matrix로 구성했을 때, 주로 row의 길이가 데이터의 개수, column의 길이가 feature의 개수이다.

9.1. CNN

9.1.1. Natural Signal의 특징

FC, CNN, RNN, transformer 등 여러 NN(Neural Network)이 있다. natural signal은 다음과 같은 특징을 가지고, NN에 한 기법들에서는 이를 이용한다.

1. Locality of Pixel Dependencies
Locality of Pixel Dependencies는 각 데이터(픽셀)의 종속성이 locality(지역성)을 가진다는 가정이다. 즉, 각 데이터는 시간이나 위치에 대해 인접한 데이터들에게 종속성을 가진다.

FC의 측면에서는 이런 특성을 활용해 관련 있는 연결(weight)만 사용하고 그렇지 않은 부분은 제거할 수 있다. CNN의 측면에서는 각 픽셀이 인접한 픽셀에 대해 종속성이 높음을 이용해 convolution 연산을 한다.

2. Stationarity of Statistics
Stationarity(정상성)은 of Statistics는 주로 시간이나 위치에 따른 데이터의 확률 분포가 일정하다는 가정이다. 즉, 데이터에 대해서 서로 다른 시간이나 위치에서 동일한 데이터가 등장할 수 있다는 것이다. 예를 들어, 사진에서 얼굴은 오른쪽 위에 있을 수도 있고, 왼쪽 아래에 있을 수도 있다.

FC의 측면에서는 이런 특성을 활용해 특정 weight를 재사용할 수 있다. CNN의 측면에서는 filter를 사용해 feature를 추출한다.

3. Compositionality
compositionality는 feature들이 합쳐져서 하나의 high-level representation을 나타낼 수 있다는 가정이다.

FC의 측면에서는 layer를 여러 개 쌓아서 점점 feature를 high level로 변환하는 것으로 이해할 수 있다. CNN도 low level feature로부터 시작해 high level representation을 추출한다는 점에서 동일하다.

9.1.2. CNN

channel은 데이터 또는 weight에 대한 정보의 종류(ex, RGB)를 나타낸다.

1. 1D Convoluation
1D Convoluation은 입력 데이터가 1차원인 경우에 대한 convoluation을 말한다.

x의 feature 개수를 n, filter의 길이를 k라 하자. 각 row가 filter를 나타내는 weight는 $m \times k$ matrix이다. 이 filter로 convolution 연산을 하면 결과물은 filter별로 n-k+1개가 나온다.

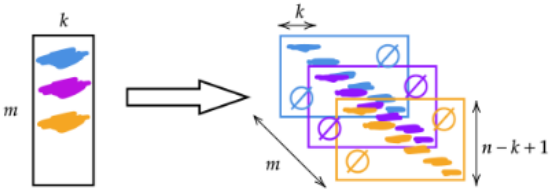


Fig 1: Illustration of 1D Convolution

The output is m (thickness) vectors of size $n - k + 1$.

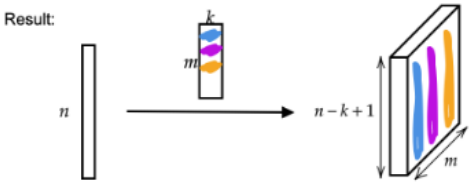


Fig 2: Result of 1D Convolution

weight matrix W와 column vector x에 대해 Wx를 생각하자. Wx에서 수행되는 W의 row와 x 사이의 연산을 내적, 정규화되지 않은 코사인 유사도, projection 등으로 부른다. 앞서 계속 다뤄왔던 것처럼 Wx는 W의 column에 대한 linear combination으로도 생각할 수 있다. 이때 x는 이미지를 벡터화한 것으로 생각해보자. 연산 이후의 결과를 이미지로 복원할 수 있다. W의 각 row도 x와 동일한 크기의 vector이므로 이미지와 동일한 해상도의 사각형으로 변환할 수 있다. 그리고 내적은 이렇게 변환한 W의 row와 입력 이미지에 대한 element-wise 곱으로도 이해할 수 있다. 즉, FC는 이미지 사이즈와 동일한 conv를 하는 것과 같다.

2. 2D Convolutation
2D Convoluation은 입력 데이터가 2차원인 경우에 대한 convoluation을 말한다. 원리는 1D와 동일한데, filter가 데이터의 아래쪽으로도 이동하며 연산한다.

9.2. Attention

9.2.1. Attention

Attention은 여러 임베딩에 대해 상호 간의 context를 반영하는 연산이다. 더 구체적으로는 현재 상태(값)을 나타내는 Query, 참조할 대상을 나타내는 Key, 실제로 사용할 값인 Value라는 세 임베딩을 사용해서, query와 key의 유사도를 계산해 그 비율만큼 value를 가중합하는 연산이다. 그 수식은 다음과 같다. 이때 Q, K, V는 query, key, value 값으로 row를 구성한 $L \times d_k$ matrix이다(L은 sequence length, d_k는 임베딩 길이).

$$Attention(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V = \text{attention value}$$

이때 어떤 입력에 대해 qkv 값은 각각 $q = W_q x$, $k = W_k x$, $v = W_v x$로 구한다. 즉, qkv는 x에 선형 변환을 적용한 것이다. 이때 q와 k에 대해서는 innver product를 계산해야 하므로 같은 dimension을 가져야 한다.

x_i에 대한 q_i, k_i, v_i를 구할 수 있다. X에 대해 구한 결과를 각 열로 하는 matrix를 $Q, K, V \in R^{d \times t}$라 한다.

이렇게 하나의 입력 sequence에 대해서 계산하는 attention은 Self Attention이고, 서로 다른 두 입력 sequence에 대한 attention은 Cross Attention이다. cross attention에서는 한 sequence의 query와 다른 sequence의 key, value에 대해 attention을 수행한다. 이에 따라 cross attention에서는 query와 key는 그 길이가 같아야 하지만,

value는 달라도 된다.

또한 attention은 multi head로 적용하기도 하는데, 이 경우 각 head별로 최종 결과를 concat하고, output weight를 거쳐 원래 길이로 복원한다.

attention은 transformer에서만 하는 건 아니고, 원래 있던 개념이다.

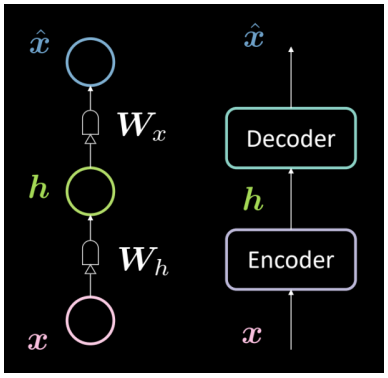
attention에는 soft attention과 hard attention이 있다. soft attention은 α_i 값들의 합이 1이도록 하는 attention이다. hard attention은 모든 α_i 중에 하나만 1이고 나머지는 0이도록 하는 attention이다. 즉, soft attention은 출력 결과에 여러 vector들의 내용을 반영하고, hard attention은 출력 결과에 하나의 vector의 내용만을 반영한다. 물론 transformer에서 hard attention은 잘 안쓴다.

9.3. Autoencoder

9.3.1. Autoencoder

1. Autoencoder
Autoencoder는 입력 데이터를 낮은 차원으로 압축(encoding)한 뒤 다시 원래대로 복원(decoding)하는 과정을 통해, 데이터의 핵심 특징을 unsupervised learning으로 스스로 학습하는 NN이다.

우선 notation, x는 어떤 image, x'은 output, z는 latent variable이다. encoder는 x를 받아 z로 변환하고, decoder는 z를 받아 x'로 변환한다. 또한 $||x - x'||_F^2$와 같이 x'와 x 사이의 차이를 loss로 해서 encoder와 decoder를 학습시킨다. autoencoder는 image의 구조와 특징을 배우게 된다.



학습된 autoencoder에서는 encoder, decoder 자체를 활용하거나, latent variable인 z값을 활용할 수 있다.

이때 차원을 축소시키는 이유는, 차원을 그대로 하거나 높인다면 모든 입력 정보를 그대로 기억했다가 넘겨주는 식으로 학습될 수 있기 때문이다. 즉, identitiy matix처럼 동작하게 될 수 있는데, 이 경우 latent variable이 유의미하지 않게 된다.

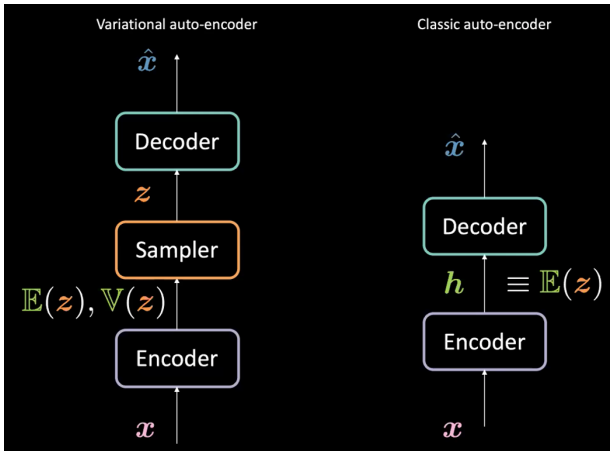
2. Denoising Autoencoder
Denoising Autoencoder는 encoder에 넣기 전에 x에 noise를 추가해 학습시키는 autoencoder이다. 이때 주로 gaussian noise를 추가한다.

BERT에서 마스킹하고 학습시키는 것처럼 학습된다고 한다.

9.3.2. VAE

VAE(Variational Autoencoder)는 입력 데이터를 확률분포(평균과 분산)로 encoding하고, 해당 확률분포에서 샘플링 한 값을 활용해 새로운 데이터를 생성하는 generative model이다.

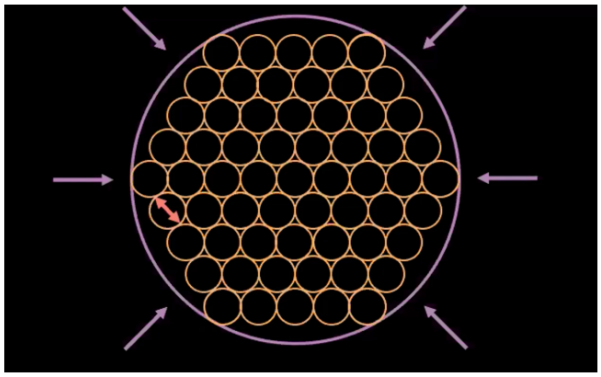
VAE는 x를 encoder에 넣어서 μ와 σ값을 뽑고(encoder의 결과를 그냥 반 잘라서 쓴다.), 이 두 값을 sampler라는 부분에 넣어 z를 도출한다. sampler는 $N(\mu, \sigma)$에서 무작위 값을 뽑는다. 확률분포만 있다면 sampling만 하면 되므로, 학습 이후에는 x가 없어도 추론이 가능하다.



이때 loss는 다음 수식과 같다. 이때 $\mathcal{L}(x, \hat{x})$은 reconsturction term으로, 입력과 예측값 사이의 차이이다. $\mathcal{L}_{KL}(N(\mu, \sigma), N(0, 1)$은 regularization term으로, 모델이 예측한 확률분포와 gaussian distribution에 대한 KL distance(분포 사이의 거리)이다.

$$\mathcal{L} = \mathcal{L}(x, \hat{x}) + \beta \mathcal{L}_{KL}(N(\mu, \sigma), N(0, 1))$$

reconstruction loss에 대해서는 각 데이터에 대한 z 분포가 멀리 떨어져 있고, 각 분포가 응집해 있어야 유리하다. 즉, 각 분포의 μ(분포 간 거리)를 멀리 떨어뜨려 놓고 σ(분포 간 응집도)가 작은 게 유리한 상황이다. 하지만 regularization term의 KL divergence 부분을 보면 μ는 0으로, σ는 1에 가까워지도록 하므로, 분포가 적당히 모아지고 덜 응집되게 한다. 이에 따라 다음 그림과 같이 최적의 분포가 만들어진다고 한다.



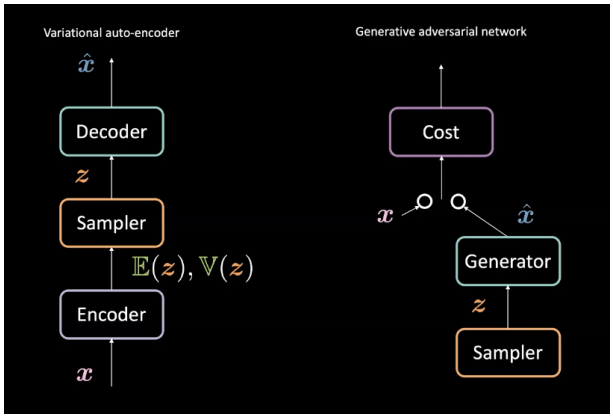
9.4. GAN

9.4.1. GAN

1. GAN

GAN(Generative Adversarial Network)은 데이터를 생성하는 generator와, 실제 데이터와 생성된 데이터를 구별하는 discriminator가 경쟁하며 학습하는 generative model이다. 주로 image generation에서 많이 사용된다.

GAN에서는 gaussian distribution에서 sampling해서 z를 얻고, decoder에 넣어서 $\hat{x}$를 얻는다. 그리고, $\hat{x}$과 x 중에 뭐가 ground truth인지 binary classification을 수행하는 discriminator를 사용한다. 즉, VAE에서 encoder 부분을 없애고, $\hat{x}$과 x를 구분하는 discriminator를 추가한 구조이다.



GAN의 학습에 사용되는 loss는 다음과 같이 generator와 discriminator 각각에 대한 loss로 구성된다. 즉, discriminator는 실제 입력은 1로, generator가 생성한 결과는 0으로 판단하도록 학습되고, generator는 generator가 생성한 결과를 discriminator가 1로 판단하도록 학습된다. 학습은 G는 고정(freeze)하고 D를 학습시키고, D를 고정하고 G를 학습시키는 것으로 진행된다.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

2. GAN 학습 도중 발생할 수 있는 문제 상황들

만약 generator에 비해 discriminator가 너무 잘 학습된 상태로 학습을 시작한다고 해보자. 그럼 generator가 의미있는 시그널을 받지 못하고 잘 학습되지 않을 것이다. 또한 discriminator에 비해 generator가 너무 잘 학습된 상태로 학습을 시작한다고 해보자. 혹은 generator가 어떤 이미지 하나만 엄청나게 잘 만들어서, 해당 이미지만 계속 생성한다고 하면.. discriminator가 전혀 맞추지 못하면서 학습되지 않을 것이다.

mode collapse가 일어날 수도 있다. Mode Collapse는 generator가 다양한 종류의 데이터를 만들어내지 못하고, discriminator를 속이기 쉬운 몇 가지 특정 데이터만 반복적으로 계속 생성하는 현상을 말한다.

참고로 binary cross entropy는 다음과 같다.

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$